Multi-Aspect Mining of Complex Sensor Sequences

Takato Honda*[1,3], Yasuko Matsubara[1], Ryo Neyama[2], Mutsumi Abe[2] and Yasushi Sakurai[1]
[1]ISIR, Osaka University , [2]Toyota Motor Corporation, [3]Kumamoto University
{takato88, yasuko, yasushi}@sanken.osaka-u.ac.jp, {neyama, mutsumi_abe_aa}@mail.toyota.co.jp

***Abstract*—In recent years, a massive amount of time-stamped sensor data has been generated and collected by many Internet of Things (IoT) applications, such as advanced automobiles and health care devices. Given such a large collection of complex sensor sequences, which consists of multiple attributes (e.g., *sensor, user, timestamp*), how can we automatically find important dynamic time-series patterns and the points of variation? How can we summarize all the complex sensor sequences, and achieve a meaningful segmentation? Also, can we see any hidden user-specific differences and outliers?**

In this paper we present CUBEMARKER, an efficient and effective method for capturing such multi-aspect features in sensor sequences. CUBEMARKER performs multi-way summarization for all attributes, namely, sensors, users, and time, and specifically it extracts multi-aspect features, such as important time-series patterns (i.e., time-aspect features) and hidden groups of users (i.e., user-aspect features), in complex sensor sequences.

Our proposed method has the following advantages: (a) It is *effective*: it extracts multi-aspect features from complex sensor sequences and enables the efficient and effective analysis of complicated datasets; (b) It is *automatic*: it requires no prior training and no parameter tuning; (c) It is *scalable*: our method is carefully designed to be linear as regards dataset size and applicable to a large number of sensor sequences.

Extensive experiments on real datasets show that CUBEMARKER is effective in that it can capture meaningful patterns for various real-world datasets, such as those obtained from smart factories, human activities, and automobiles. CUBEMARKER consistently outperforms the best state-of-the-art methods in terms of both accuracy and execution speed.

***Index Terms*—Time series, IoT sensors, Tensor analysis, Automatic mining**

I. INTRODUCTION

Today, thanks to the rapid spread of small, inexpensive but high performance sensors, a massive quantity of time-stamped sensor data is generated and collected by many Internet of Things (IoT) applications, such as advanced automobiles (e.g., self-driving cars), health care devices (e.g., fitness trackers) and smart factories (e.g., Industry 4.0). In most cases, these IoT data consist of multiple sensor readings obtained from multiple users (or drivers/facilities), at every time point, that is, each entry is composed of the form *(sensor, user, timestamp)*. In this paper, we shall refer to such settings as *complex sensor sequences*. In practice, real complex sensor sequences contain various types of distinct, dynamic time-series patterns of different durations, and these patterns usually appear multiple times. Here, we shall refer to such a distinct time-series pattern as a *"regime"*. Our goal is to find important dynamic time-series patterns (i.e., regimes) and achieve multi-aspect summarization and segmentation of all the complex sensor sequences for three attributes, namely, sensors, users and time.

For example, assume that we have the automobile sensor sequences, e.g., velocity, longitudinal and lateral acceleration sensor data, obtained by multiple drivers. So, how can we find meaningful patterns/regimes with respect to three aspects: sensor, user and time? Specifically, we would like to answer the following questions: Can we see any basic driving behavior, e.g., speeding up and turning? Can we find any hidden groups of users, e.g., beginners and professionals? Is there any distinct driver-specific behavior, e.g., slowing down very carefully at an intersection?

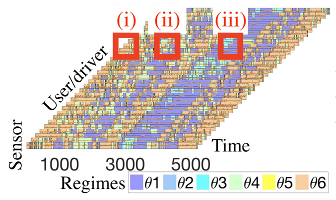
In this paper, we present CUBEMARKER, which answers all of the above questions, and provides a good summary of large collections of complex sensor sequences. Intuitively, the problem we wish to solve is as follows:

Informal Problem 1: **Given** a large collection of triplets (sensor, user, time), that is $\mathcal{X} \in \mathbb{R}^{d\times w\times n}$, which consists of d sensors in w users of duration n, **Find** a compact description of $\mathcal{X}$ that summarizes all the complex sensor sequences with respect to three aspects (i.e., sensor, user and time), automatically, in a scalable fashion.

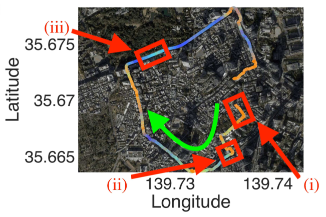
Preview of our results. Figure 1 shows some of our discoveries for automobile sensor data. The data were obtained from $d = 3$ sensors: velocity (blue line), longitudinal (yellow line) and lateral (red line) acceleration, for $w = 32$ users/drivers and $n = 5200$ time points. The experiments were performed by driving the same route (please see Figure 1 (b)). Given a set of complex automobile sensor sequences, CUBEMARKER automatically and efficiently detects important dynamic patterns, with respect to three aspects: sensor, user and time. Specifically, Figure 1 (a) shows the multi-aspect segmentation results (i.e., a set of colored segments) and the six typical time-series patterns (i.e., regimes: $\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_6$). Also, Figure 1 (b) shows representative driving behavior plotted on a map. Here, each color represents the assigned regime label, e.g., orange corresponds to regime $\boldsymbol{\theta}_6$: "slow speed", and blue corresponds to regime $\boldsymbol{\theta}_2$: "turning right".

Figure 1 (c) shows an enlarged, individual summarization for three drivers A, B, and C at three different locations, namely, (i) narrow road, (ii) intersection, and (iii) wide, multi-lane road (please also see the red rectangles in Figure 1 (a) and (b)). Here, the left columns show the original $d = 3$ dimensional sensor sequences and their segmentation results for three drivers, and the right columns show snapshots of the dashboard cameras of three drivers at each location.

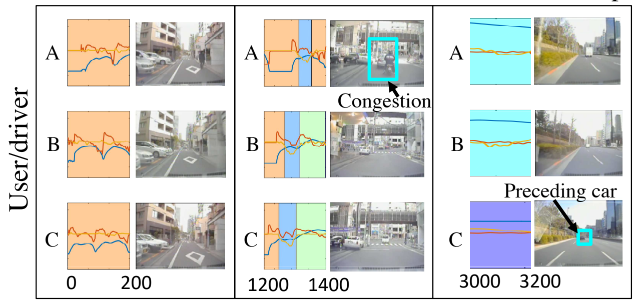
*Artificial Intelligence Research Center, The Institute of Scientific and Industrial Research (ISIR), Osaka University, Mihogaoka 8-1, Ibaraki, Osaka 567-0047, Japan



(a) Multi-aspect segmentation and summarization



(b) Representative driving behavior on a map



(c-i) Narrow road (c-ii) Intersection (c-iii) Wide road
(c) User/driver-specific behavior at three different locations

Fig. 1. Multi-aspect mining of CubeMarker for automobile sensor data. Given a set of complex sensor sequences consisting of *sensor* $\times$ *user (driver)* $\times$ *timestamp*, it detects various important patterns and user/driver-specific, individual behavior, *automatically* and *quickly*. Specifically, (a) the algorithm provides a multi-aspect segmentation result (i.e., a set of colored segments), and also, estimates six typical time-series patterns (i.e., regimes: $\boldsymbol{\theta}, \cdots, \boldsymbol{\theta}_6$). It also shows (b) the representative driving behavior plotted on a map, and (c) an enlarged, individual summarization. Here, each colored segment corresponds to a specific driving pattern/regime, e.g., the orange segments correspond to regime $\boldsymbol{\theta}_6$: "slow speed", the blue segments correspond to regime $\boldsymbol{\theta}_2$: "turning right", and the green segments correspond to regime $\boldsymbol{\theta}_4$: "speeding up". Please also see the text for more details.

TABLE I
Capabilities of approaches.

	DynaMMo	pHMM	CPRAND	AutoPlait	TICC	**CubeMarker**
Segmentation	✓	✓	-	✓	✓	✓
Regime discovery	-	✓	-	✓	✓	✓
Tensor analysis	-	-	✓	-	-	✓
Automatic	-	-	-	✓	-	✓
Multi-aspect analysis	-	-	-	-	-	✓

Most importantly, CubeMarker can capture typical and basic time-series patterns/regimes, *and* various types of distinct driver-specific behavior, *automatically* and *simultaneously*. For example, in Figure 1 (c-i), the algorithm identifies similar driving behavior for three drivers. Here, the orange segments correspond to regime $\boldsymbol{\theta}_6$: "slow speed", which means that all the drivers pay attention to their surroundings on a narrow road (e.g., pedestrians and parked cars).

In addition, the assignment of regime labels made it possible to identify the basic driving procedures at each location. In Figure 1 (c-ii), CubeMarker automatically identifies individual driving behavior when turning at an intersection. Here, drivers B and C fall into regimes $\boldsymbol{\theta}_6 \rightarrow \boldsymbol{\theta}_2 \rightarrow \boldsymbol{\theta}_4$ (i.e., slow speed $\rightarrow$ turning right $\rightarrow$ speeding up), while driver A falls into regimes $\boldsymbol{\theta}_6 \rightarrow \boldsymbol{\theta}_2 \rightarrow \boldsymbol{\theta}_6$ (i.e., slow speed $\rightarrow$ turning right $\rightarrow$ slow speed). Similarly, in Figure 1 (c-iii), the algorithm successfully identifies driver-specific patterns on a wide, multi-lane road, where drivers A and B are assigned to regime $\boldsymbol{\theta}_3$: "high speed", while driver C is assigned to regime $\boldsymbol{\theta}_3$: "middle speed" (this is because there is a preceding car, shown as a cyan rectangle in the figure). Most importantly, these driving patterns/regimes are unknown in advance, and thus the algorithm should recognize the groups of segments, each of whose pattern has similar characteristics. Also note that our method automatically determines the discontinuity/transition points of each segment.

Contributions. The main contribution of this work is the concept and design of CubeMarker, which has the following desirable properties:

1) **Effective**: It operates on large collections of complex sensor sequences and summarizes them with respect to three aspects (i.e., sensor, user, time), which enables flexible segmentation and representation. Also, we apply CubeMarker to various types of complex sensor sequences including automobiles, smart factories and health care activities.
2) **Automatic**: It is fully automatic, and so requires no prior training and no parameter tuning.
3) **Scalable**: It is carefully designed to be linear as regards input size, and thus is applicable to very large complex sensor sequences.

II. Related Work

Recently significant progress has been made on understanding the theoretical issues surrounding mining time-series [1]–[7]. Traditional approaches applied to time-series data mining include similarity search [8], auto-regression (AR), linear dynamical systems (LDS) and variants [9]–[11].

Table I illustrates the relative advantages of our method. Only CubeMarker meets all the requirements. Li et al. [11] developed DynaMMo, which is a scalable algorithm for time-evolving sequences with missing values. DynaMMo is based on a linear dynamical system, and has the ability to segment a data sequence. However, it cannot find a group of similar patterns (i.e., regimes).

Tensors are widely used in a variety of areas, such as health care [12]–[14] and social media [15]–[20]. Kolda et al. provided a powerful tool for matrix sketching methods in the context of CANDECOMP/PARAFAC [21], but it is incapable of discovering important regimes and requires parameter tuning.

In work on learning probabilistic models, hidden Markov models (HMMs) have been used in various research areas [22]–[24] including speech recognition [25] and biological data analysis [23]. Wang et al. [26] improved on the work described in [27], and presented a pattern-based hidden Markov model (pHMM). The pHMM is a new dynamical model for time-series segmentation and clustering, and provides a piecewise linear representation. However, the above model requires user-defined parameter and model structure settings. Furthermore, it does not focus on scalability.

Recently, Hallac et al. [28] proposed Toeplitz inverse covariance-based clustering (TICC), which characterizes the interdependence of different observations using a Markov random field. AutoPlait [29] focuses on the clustering of time-series patterns for multivariate time-series, however, they are not intended to cluster similar patterns extending to other users (i.e., *user-aspect* patterns).

In short, none of the existing methods focuses specifically on the automatic and multi-aspect mining of time-evolving dynamics in complex sensor sequences.

III. Problem formulation

In this section, we formally define related concepts and the problem we are trying to solve. Consider that we receive time-stamped data entries of the form *(sensor, user, time)*. We then have a collection of entries with d unique sensors and w unique users, for n time points. It is convenient to treat our sensor sequences as a 3rd-order tensor, i.e., $\mathcal{X} \in \mathbb{R}^{d \times w \times n}$, and we refer to it as a *time-series tensor.*

We want to convert a given $\mathcal{X}$ into a set of m non-overlapping segments $\mathcal{S} = \{s_1, \dots, s_m\}$ where s_i consists of the start position t_s, end position t_e and user j of the i-th segment (i.e., $s_i = \{t_s, t_e, j\}$). We also want to find a set of distinct patterns of multi-aspect segments by assigning each segment to a unified segment group, namely, a regime.

Definition 1 (Regime): Let r denote the desired number of segment groups. Each segment s is assigned to one of these groups. We define such segment groups as regimes, which are represented by a statistical model $\boldsymbol{\theta}_i$ $(i = 1, \dots, r)$.

Here, to represent the multi-aspect variation of sequences, we use a hidden Markov model (HMM). An HMM is a statistical model in which the system being modeled is assumed to be a Markov process with hidden states. It is used in many time-series pattern recognition techniques including in the speaker recognition field and the analysis of biological sequences. An HMM is composed of three probabilities: initial state probabilities $\boldsymbol{\pi} = \{\pi_i\}_{i=1}^k$, state transition probabilities $\mathbf{A} = \{a_{ji}\}_{i,j=1}^k$ and output probabilities $\mathbf{B} = \{b_i(\boldsymbol{x})\}_{i=1}^k$, where k is the number of hidden states.

Consequently, single regime dynamics can be described as a set of parameters $\boldsymbol{\theta} = \{\boldsymbol{\pi}, \mathbf{A}, \mathbf{B}\}$. We should also note that our full model parameter set $\boldsymbol{\Theta}$ consists of r regime parameter sets $\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_r$ *and* one additional matrix, namely, the *regime transition matrix* $\boldsymbol{\Delta}_{r \times r}$.

Definition 2 (Regime transition matrix): Let $\boldsymbol{\Delta}_{r \times r}$ denote a transition probability matrix of r regimes, where each element $\delta_{ij} \in \boldsymbol{\Delta}$ is the regime transition probability from the i-th regime to the j-th regime.

Here, each regime (e.g., $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$) has its own (within-regime) state transition matrix (e.g., $a_{1;ji} \in \mathbf{A}_1$, $a_{2;ji} \in \mathbf{A}_2$), and there is an upper-level transition matrix (e.g., $\delta_{12} \in \boldsymbol{\Delta}_{r \times r}$) that captures across-regime transitions (e.g., from $\boldsymbol{\theta}_1$ to $\boldsymbol{\theta}_2$).

TABLE II
Symbols and definitions.

Symbol	Definition
d	Number of sensors
w	Number of users
n	Number of timestamps
$\mathcal{X}$	Time-series tensor: $\mathcal{X} = \{\boldsymbol{X}_1, \dots, \boldsymbol{X}_w\}$
$\boldsymbol{X}$	d dimensional sensor sequence
m	Number of segments in $\mathcal{X}$
$\mathcal{S}$	Segment set in $\mathcal{X}$: $\mathcal{S} = \{s_1, \dots, s_m\}$
r	Number of regimes in $\mathcal{X}$
$\boldsymbol{\Theta}$	Model parameter set of r regimes: $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_r, \boldsymbol{\Delta}_{r \times r}\}$
$\boldsymbol{\theta}_i$	Model parameters governing i-th regime
k_i	Number of hidden states in $\boldsymbol{\theta}_i$
$\boldsymbol{\Delta}_{r \times r}$	Regime transitions: $\boldsymbol{\Delta} = \{\delta_{ij}\}_{i,j=1}^r$
$\mathcal{F}$	Segment membership: $\mathcal{F} = \{f_1, \dots, f_m\}$
$\mathcal{C}$	Candidate solution: $\mathcal{C} = \{m, r, \mathcal{S}, \boldsymbol{\Theta}, \mathcal{F}\}$
$Cost_T(\mathcal{X}; \mathcal{C})$	Total cost of $\mathcal{X}$ given $\mathcal{C}$

Definition 3 (Segment membership): Given a time-series tensor $\mathcal{X}$, let $\mathcal{F}$ be a set of m integers, $\mathcal{F} = \{f_1, \dots, f_m\}$, where f_i is the regime to which the i-th segment belongs (i.e., $1 \leq f_i \leq r$).

Finally, our goal is to find a good description of $\mathcal{X}$, i.e., the number of segments/regimes, their model parameters and positions automatically, in a scalable way. We refer to it as a *candidate solution.*

Problem 1 (Candidate solution): **Given** a time-series tensor $\mathcal{X}$ consisting of *(sensor, user, time)* triplets, **Find** a complete set of parameters, namely $\mathcal{C} = \{m, r, \mathcal{S}, \boldsymbol{\Theta}, \mathcal{F}\}$, i.e.,

- m: the number of segments
- r: the number of regimes
- $\mathcal{S} = \{s_1, \dots, s_m\}$: a set of m segments
- $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_r, \boldsymbol{\Delta}_{r \times r}\}$: model parameters of r regimes and their transition matrix
- $\mathcal{F} = \{f_1, \dots, f_m\}$: segment membership of m segments

IV. Optimization Algorithm

In the previous section, we saw how we can describe complex sensor sequences. Now, we want to determine how to estimate a candidate solution $\mathcal{C}$. Specifically, we need to answer the following two questions: (1) How can we *automatically* find an optimal solution? (2) How can we efficiently and effectively estimate $\mathcal{C}$, which describes *multi-aspect patterns*? Each question is dealt with in the following subsections.

A. Automatic time-series tensor analysis

Let us begin with the first question, namely, *how can we automatically find an optimal solution*, and this is the focus of this subsection. So, how should we determine the number of segments and regimes? How can we compress $\mathcal{C}$, and make it as compact as possible? We want to answer the question without any parameter tuning, that is, *fully automatically.* To find the solution $\mathcal{C}$, we introduce a new coding scheme, which is based on the minimum description length (MDL) principle. In short, it follows the assumption that the more we can compress the data, the more we can learn about its underlying patterns.

The total code length for $\mathcal{X}$ with respect to a given $\mathcal{C}$ can be described as follows:

$$Cost_T(\mathcal{X};\mathcal{C}) = Cost_M(\mathcal{C}) + Cost_C(\mathcal{X}|\mathcal{C}) \qquad (1)$$

Model description cost. The description cost of the model parameter set $Cost_M(\mathcal{C})$ consists of the following terms: the number of sensors d, users w, and time points n require $\log^*(d) + \log^*(w) + \log^*(n)$ bits [1], the number of segments m and the number of regimes r require $\log^*(m) + \log^*(r)$ bits, the length of each segment s, needs $\sum_{i=1}^{m-1} \log^* |s_i|$ bits, the assignment of the segments to regimes requires $m \log(r)$, the assignment of the segments to users requires $m \log(w)$, and the model parameters of r regimes need $Cost_M(\boldsymbol{\Theta})$, i.e., $Cost_M(\boldsymbol{\Theta}) = \sum_{i=1}^{r} Cost_M(\boldsymbol{\theta}_i) + Cost_M(\boldsymbol{\Delta})$. Here, $Cost_M(\boldsymbol{\theta}) = \log^*(k) + c_F \cdot (k + k^2 + 2kd)$, where c_F is the floating point cost[2]. Similarly, the regime transition requires a cost of $Cost_M(\boldsymbol{\Delta}) = c_F \cdot r^2$.

Data coding cost. Once we have decided the full parameter set $\mathcal{C}$, we can encode the data $\mathcal{X}$ using Huffman coding [30]. The encoding cost of $\mathcal{X}$ given $\mathcal{C}$ is:
$Cost_C(\mathcal{X}|\mathcal{C}) = \sum_{i=1}^{m} Cost_C(\mathcal{X}[s_i]|\boldsymbol{\Theta}) = \sum_{i=1}^{m} -\ln(\delta_{vu} \cdot (\delta_{uu})^{|s_i|-1} \cdot P(\mathcal{X}[s_i]|\boldsymbol{\theta}_u))$, where the i-th and $(i-1)$-th segments are governed by the u-th and v-th regimes, respectively. Also, $\mathcal{X}[s_i]$ is a sub-sequence of segment s_i at time-series tensor $\mathcal{X}$, and $P(\mathcal{X}[s_i]|\boldsymbol{\theta}_u)$ is the likelihood of s_i at $\mathcal{X}$. Note that $\boldsymbol{\theta}_u$ is the regime to which the segment s_i belongs.

B. Overview of multi-aspect mining

The fundamental question as regards mining time-series tensors is whether there is any underlying structure. The time-series tensor contains information about the relationship between segments from multiple viewpoints, i.e., time domains and/or users. Intuitively, we seek user-aspect groupings (i.e., groupings of multiple users for a particular activity) as well as time-aspect groupings (i.e., temporal segmentation and grouping), which reveal the underlying structure. We would like to simultaneously find multi-aspect groupings, which succinctly summarize the underlying structure in the tensor.

We introduce analytical tools for multi-aspect mining, i.e., time-aspect grouping and user-aspect grouping. The former tool tries to detect dynamic, temporal pattern transitions/changes, and we refer to it as V-Split for vertical mining, and the latter extracts individual user characteristics, namely H-Split (i.e., the horizontal splitting algorithm). These splitting algorithms, V-Split and H-Split, consist of the segment assignment for each group (expressed as a regime) and the feature extraction of the regime, and both can be performed from their own viewpoint. That is, V-Split estimates regimes from the viewpoint of time domains, and H-Split computes their regimes as individual user characteristics from the viewpoint of users.

Our proposed method, CUBEMARKER, chooses one of these tools, V-Split and H-Split, for every iteration, and the decision is made according to the total cost (Equation 1). This approach provides a flexible data summarization process, which achieves a dramatic reduction in terms of both modeling error and computation cost.

[1]Here, $\log^*$ is the universal code length for integers.

[2]We used 4×8 bits in our setting.

Algorithm 1 V-Split ($\mathcal{X}$)

```
1: Input: Time-series tensor $\mathcal{X}$
2: Output: (a) Number of segments assigned to each regime, $m_1, m_2$
3:         (b) Segment sets of two regimes, $\mathcal{S}_1, \mathcal{S}_2$
4:         (c) Model parameters of two regimes, $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta}\}$
5: Initialize models $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta}_{2\times 2}$;
6: while improving the cost do
7:     $\{m_1, m_2, \mathcal{S}_1, \mathcal{S}_2\}$ =V-Assignment ($\mathcal{X}, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta}$);
8:     $\boldsymbol{\theta}_1$ =ModelEstimation ($\mathcal{S}_1$); $\boldsymbol{\theta}_2$ =ModelEstimation ($\mathcal{S}_2$);
9:     Update $\boldsymbol{\Delta}$ from $\mathcal{S}_1, \mathcal{S}_2$;
10: end while
11: return $\{m_1, m_2, \mathcal{S}_1, \mathcal{S}_2, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta}\}$;
```

Specifically, given an original time-series tensor $\mathcal{X}$, CUBEMARKER iteratively splits $\mathcal{X}$ into multi-aspect segments while selecting the appropriate algorithm (V-Split or H-Split), and creates/estimates new regimes, as long as the total cost keeps decreasing. In short, given a time-series tensor $\mathcal{X}$, CUBEMARKER consists of the following algorithms:

1) **V-Split**: Finds *time-aspect* features and their shifting points by splitting $\mathcal{X}$ into two groups of segments (i.e., $\mathcal{S}_1, \mathcal{S}_2$) in a vertical (time) aspect and estimates their regime parameter sets $\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2$, $\boldsymbol{\Delta}$.
2) **H-Split**: Finds *user-aspect* features by splitting $\mathcal{X}$ into two segment groups $\mathcal{S}_1, \mathcal{S}_2$ in a horizontal (user) aspect and estimates regimes $\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2$, $\boldsymbol{\Delta}$.
3) **CubeMarker**: Finds a compact summary $\mathcal{C}$ of $\mathcal{X}$ by using V-Split and H-Split.

We describe the above algorithms in detail in the following subsections.

C. V-Split

Given a time-series tensor $\mathcal{X}$, our first goal is to find two groups of segments $\mathcal{S}_1, \mathcal{S}_2$ in a *time-aspect* view, and estimate their regime parameters $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$, $\boldsymbol{\Delta}$. We propose V-Split, which employs an iterative refinement framework that performs the following phases:

- *(Phase 1) V-Assignment:* Finds two sets of segments $\mathcal{S}_1, \mathcal{S}_2$ and their shifting points based on the coding cost, given the tensor $\mathcal{X}$ and the model parameters of two regimes (i.e., $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$, $\boldsymbol{\Delta}$).
- *(Phase 2) ModelEstimation:* Estimates the model parameters of two segment sets $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta}\}$ according to the new segmentation result $\mathcal{S}_1, \mathcal{S}_2$.

Algorithm 1 shows the overall procedure for V-Split. For each iteration, the algorithm tries to find the best segment sets $\mathcal{S}_1, \mathcal{S}_2$ so that it minimizes the coding cost. Then it estimates the model parameters $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta}\}$ using the segmentation results, $\mathcal{S}_1, \mathcal{S}_2$. It continues these two steps until convergence.

1) V-Assignment: Given a time-series tensor $\mathcal{X}$ and the model parameters of two regimes, $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta}\}$, we propose a one-path algorithm called V-Assignment, which efficiently detects appropriate *time-aspect* segments and their shifting points. An elementary concept that we need to introduce

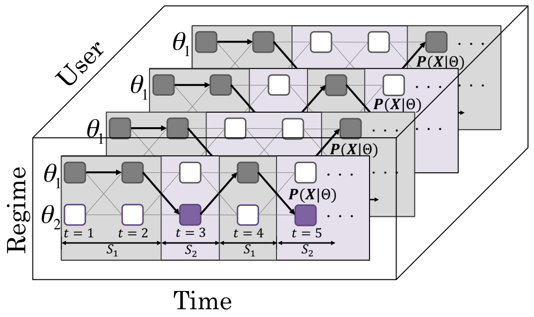


Fig. 2. Illustration of V-Assignment. Given a time-series tensor and two models $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$, our algorithm detects the regime shifting points by computing the Viterbi path that extends over all users.

is the transition diagram shown in Figure 2. We connect the transition diagrams of two regimes, $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$, compare the two regimes for each time in terms of coding cost, and estimate the pattern transition of the sequences between the given regimes. We use a dynamic programming approach, namely the Viterbi algorithm [31], to compute the coding cost $Cost_T(\mathcal{X}|\boldsymbol{\Theta}) = -\ln P(\mathcal{X}|\boldsymbol{\Theta})$. Specifically, the likelihood value $P(\mathcal{X}|\boldsymbol{\Theta})$ is computed as: $P(\mathcal{X}|\boldsymbol{\Theta}) = \max_{i=1,2}\{P(\mathcal{X}|\boldsymbol{\Theta})_i\}$, where $P(\mathcal{X}|\boldsymbol{\Theta})_i$ is the likelihood value of the regime shift to $\boldsymbol{\theta}_i$. As an example, $P(\mathcal{X}|\boldsymbol{\Theta})_1$ is computed as follows:

$$P(\mathcal{X}|\boldsymbol{\Theta})_1 = \max_{1 \le i \le k_1} \{p_{1;i}(t)\}$$

$$p_{1;i}(t) = \max \begin{cases} \delta_{21} \cdot \max_u\{p_{2;u}(t-1)\} \cdot \pi_{1;i} \cdot b_{1;i}(\boldsymbol{x}_t) \\ \text{// regime shift from } \boldsymbol{\theta}_2 \text{ to } \boldsymbol{\theta}_1 \\ \delta_{11} \cdot \max_j\{p_{1;j}(t-1) \cdot a_{1;ji}\} \cdot b_{1;i}(\boldsymbol{x}_t) \\ \text{// staying at regime } \boldsymbol{\theta}_1 \end{cases}$$

where $p_{1;i}(t)$ is the maximum probability of state i of regime $\boldsymbol{\theta}_1$ at time point t, δ_{21} is the regime transition probability from $\boldsymbol{\theta}_2$ to $\boldsymbol{\theta}_1$, $\max_u\{p_{2;u}(t-1)\}$ is the probability of the best state of $\boldsymbol{\theta}_2$ at time point $t-1$, $\pi_{1;i}$ is the initial probability of state i of $\boldsymbol{\theta}_1$, $b_{1;i}(\boldsymbol{x}_t)$ is the output probability of $\boldsymbol{x}_t$ for state i of $\boldsymbol{\theta}_1$, and $a_{1;ji}$ is the transition probability from state j to state i in $\boldsymbol{\theta}_1$. Here, at time point $t=1$, the probability for each regime $\boldsymbol{\theta}_1$ is given by $p_{1;i}(1) = \delta_{11} \cdot \pi_{1;i} \cdot b_{1;i}(\boldsymbol{x}_t)$.

2) ModelEstimation: Given the segment sets $\{\mathcal{S}_1, \mathcal{S}_2\}$, the task is to estimate the model parameters of two regimes. Note that the ModelEstimation algorithm requires the number of hidden states k for each model $\boldsymbol{\theta}$, and so how can we determine the optimal number for k? If we use a small number for k, the generated model does not fit the data, and thus the algorithm cannot find the optimal number of segments and regimes. On the other hand, if we use a large number for k, the model causes over-fitting. For the above reason, we vary $k = 1, 2, 3, ...$, and determine appropriate models to minimize the cost function: $Cost_M(\boldsymbol{\theta}) + Cost_C(\mathcal{X}[\mathcal{S}]|\boldsymbol{\theta})$. When the k number is determined, we estimate the model parameters using the BaumWelch algorithm [32]. We also need to update the regime transition probabilities $\boldsymbol{\Delta}$ to minimize the coding cost. We compute $\boldsymbol{\Delta} = \{\delta_{11}, \delta_{12}, \delta_{21}, \delta_{22}\}$ using the shifting points of segments $\{\mathcal{S}_1, \mathcal{S}_2\}$ and the model parameters $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2\}$:

Algorithm 2 H-Assignment $(\mathcal{X}, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta})$

```
1: Input: Tensor $\mathcal{X}$, model parameters $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta}\}$
2: Output: Segment sets of two regimes, $\{m_1, m_2, \mathcal{S}_1, \mathcal{S}_2\}$
3: $m_1 = 0$; $m_2 = 0$; $\mathcal{S}_1 = \emptyset$; $\mathcal{S}_2 = \emptyset$;
4: for $i = 1$ to $w$ do
5:   if $Cost_C(\mathcal{X}[i]|\boldsymbol{\theta}_2, \boldsymbol{\Delta}) > Cost_C(\mathcal{X}[i]|\boldsymbol{\theta}_1, \boldsymbol{\Delta})$ then
6:     $\mathcal{S}_1 = \mathcal{S}_1 \cup \mathcal{X}[i]$;
7:     $m_1 = m_1 + |\mathcal{X}[i]|$;
8:   else
9:     $\mathcal{S}_2 = \mathcal{S}_2 \cup \mathcal{X}[i]$;
10:    $m_2 = m_2 + |\mathcal{X}[i]|$;
11:  end if
12: end for
13: return $\{m_1, m_2, \mathcal{S}_1, \mathcal{S}_2\}$;
```

$\delta_{11} = \frac{\sum_{s \in \mathcal{S}_1} |s| - N_{12}}{\sum_{s \in \mathcal{S}_1} |s|}$, $\delta_{12} = \frac{N_{12}}{\sum_{s \in \mathcal{S}_1} |s|}$, where $\sum_{s \in \mathcal{S}_1} |s|$ represents the total length of the segments that belong to regime $\boldsymbol{\theta}_1$, and N_{12} shows the regime shift count from $\boldsymbol{\theta}_1$ to $\boldsymbol{\theta}_2$. We also compute δ_{21}, δ_{22} in a similar fashion, and omit the explanation.

Lemma 1: The V-Split algorithm takes $O(dwn(k_1 + k_2)^2)$ time.

Proof 1: The V-Assignment algorithm must compute $O(dw(k_1 + k_2)^2)$ numbers per time point, where k_1 and k_2 are the number of states of the regime $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$. The algorithm requires only a single scan to find shifting points. Thus, the time complexity of the V-Assignment is $O(dwn(k_1 + k_2)^2)$. The ModelEstimation algorithm takes $O(dwn(k_1 + k_2)^2)$ time. Consequently, the time complexity of the V-Split algorithm is $O(dwn(k_1 + k_2)^2)$.

D. H-Split

We have described how to capture the transitions of time-evolving patterns (i.e., *time-aspect* features) in a given $\mathcal{X}$. In reality, the time-series tensor will include personal differences - for example, two drivers may have the same *driving straight* patterns but their *turning* might be different. Whose patterns are different and which patterns are they? Our approach provides a powerful solution that distinguishes *user-aspect* patterns based on the model parameters. In short, this algorithm splits a given $\mathcal{X}$ into two regimes in a *user-aspect* view, and estimates the model parameters using a two-phase iteration, i.e.,

- *(Phase 1) H-Assignment:* Split $\mathcal{X}$ into two sets of segments $\mathcal{S}_1, \mathcal{S}_2$ in a *user-aspect* view, based on the model parameters of two regimes $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta}\}$.
- *(Phase 2) ModelEstimation:* Update the model parameters $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta}\}$ based on the new H-Assignment result.

The horizontal splitting can be performed in the same manner as the V-Split algorithm (we omit it due to the space limitation). The difference is that we use H-Assignment, instead of V-Assignment, for H-Split. Specifically, H-Split tries to find the best regimes in order to minimize the coding cost based on the model parameters (Phase 1). Then it estimates new model parameters $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta}\}$ based on the above grouping result (Phase 2). H-Assignment results improve ModelEstimation results, and ModelEstimation results also improve H-

Algorithm 3 CubeMarker ($\mathcal{X}$)

```
1: Input: Time-series tensor $\mathcal{X}$
2: Output: Solution $\mathcal{C} = \{m, r, \mathcal{S}, \mathbf{\Theta}, \mathcal{F}\}$
3: $Q = \emptyset$; /* $Q$: stack for segments and regimes */
4: $\mathcal{S} = \emptyset$; $m = 0$; $r = 0$; $m_0 = w$;
5: $\mathcal{S}_0 = \{(1, n, 1), ..., (1, n, w)\}$;
6: $\boldsymbol{\theta}_0$ =ModelEstimation ($\mathcal{X}[\mathcal{S}_0]$);
7: Push an entry $\{m_0, \mathcal{S}_0, \boldsymbol{\theta}_0\}$ into $Q$;
8: while stack $Q \neq \emptyset$ do
9:    Pop an entry $\{m_0, \mathcal{S}_0, \boldsymbol{\theta}_0\}$ from $Q$;
10:   $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \mathbf{\Delta}_{2\times2}\}$ = ModelSampling($\mathcal{X}[\mathcal{S}_0]$);
11:   $\{m_1^V, m_2^V, \mathcal{S}_1^V, \mathcal{S}_2^V\}$ =V-Assignment ($\mathcal{X}[\mathcal{S}_0], \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \mathbf{\Delta}$);
12:   $\{m_1^H, m_2^H, \mathcal{S}_1^H, \mathcal{S}_2^H\}$ =H-Assignment ($\mathcal{X}[\mathcal{S}_0], \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \mathbf{\Delta}$);
13:   /* Compare V-Split v.s. H-Split */
14:   if $Cost_T(\mathcal{X}[\mathcal{S}_0]; \mathcal{S}_1^H, \mathcal{S}_2^H, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2) > Cost_T(\mathcal{X}[\mathcal{S}_0]; \mathcal{S}_1^V, \mathcal{S}_2^V, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$
      then
15:      $\{m_1, m_2, \mathcal{S}_1, \mathcal{S}_2, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \mathbf{\Delta}\}$ =V-Split ($\mathcal{X}[\mathcal{S}_0]$);
16:   else
17:      $\{m_1, m_2, \mathcal{S}_1, \mathcal{S}_2, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \mathbf{\Delta}\}$ =H-Split ($\mathcal{X}[\mathcal{S}_0]$);
18:   end if
19:   /* Compare single regime $\boldsymbol{\theta}_0$ v.s. regime pair $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ */
20:   if $Cost_T(\mathcal{X}[\mathcal{S}_0]; \mathcal{S}_0, \boldsymbol{\theta}_0) > Cost_T(\mathcal{X}[\mathcal{S}_0]; \mathcal{S}_1, \mathcal{S}_2, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$ then
21:      Push entries $\{m_1, \mathcal{S}_1, \boldsymbol{\theta}_1\}, \{m_2, \mathcal{S}_2, \boldsymbol{\theta}_2\}$ into $Q$;
22:   else
23:      $r = r + 1$; $f_i = r(i = m + 1, \ldots, m_0)$;
24:      $m = m + m_0$; $\mathcal{S} = \mathcal{S} \cup \mathcal{S}_0$; $\mathbf{\Theta} = \mathbf{\Theta} \cup \boldsymbol{\theta}_0$;
25:      Update $\mathbf{\Delta}_{r\times r}$;
26:   end if
27: end while
28: return $\mathcal{C} = \{m, r, \mathcal{S}, \mathbf{\Theta}, \mathcal{F}\}$;
```

Assignment results for each iteration. H-Split repeats the above two phases until convergence.

1) H-Assignment: We propose H-Assignment, which splits $\mathcal{X}$ into two sets of segments $\mathcal{S}_1, \mathcal{S}_2$ that capture *user-aspect* features. Specifically, Algorithm 2 shows the procedure for H-Assignment. Instead of using one of the traditional clustering algorithms for all segments in $\mathcal{X}$, our algorithm, H-Assignment, can effectively extract the *user-aspect* features (e.g., user/driver characteristics for particular motions) from the time-series tensor $\mathcal{X}$. The idea is quite simple and efficient: given a time-series tensor $\mathcal{X}$, a number of regimes ($r = 2$) and model parameters $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \mathbf{\Delta}\}$, it computes the coding cost to assign segments of an user i as: $\{\mathcal{S}_{\boldsymbol{\theta}}\} = \arg\min_{\boldsymbol{\theta} \in \boldsymbol{\theta}_1, \boldsymbol{\theta}_2} Cost_C(\mathcal{X}[i] | \boldsymbol{\theta}, \mathbf{\Delta})$, where $\mathcal{X}[i] = \{s_1, s_2, \ldots\}$ is a segment set in user i.

Lemma 2: The H-Split algorithm takes $O(dwn(k_1^2 + k_2^2))$.

Proof 2: The H-Assignment algorithm computes the coding cost of all segments for two model parameters $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2\}$. Thus the iteration of the algorithm requires $O(dwn(k_1^2 + k_2^2))$ time. Also, we need to update the model parameters $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2\}$. Consequently, the algorithm requires $O(dwn(k_1^2 + k_2^2))$.

Complexity of V-Split and H-Split. As described above, the V-Split and H-Split algorithms require $O(dwn(k_1 + k_2)^2)$ and $O(dwn(k_1^2 + k_2^2))$, respectively. That is, the H-Split algorithm is $O(dwn \cdot 2k_1k_2)$ times faster than the V-Split algorithm, which means the H-Split algorithm can find the *user-aspect* features effectively and efficiently. H-Split becomes more effective as the patterns become more complex and the data size increases (please see section V for detailed experiments).

E. CubeMarker

Our final goal is to find multi-aspect patterns and their regimes automatically. We propose a stack-based algorithm, CUBEMARKER, which enables the effective optimization of the number of segments and regimes.

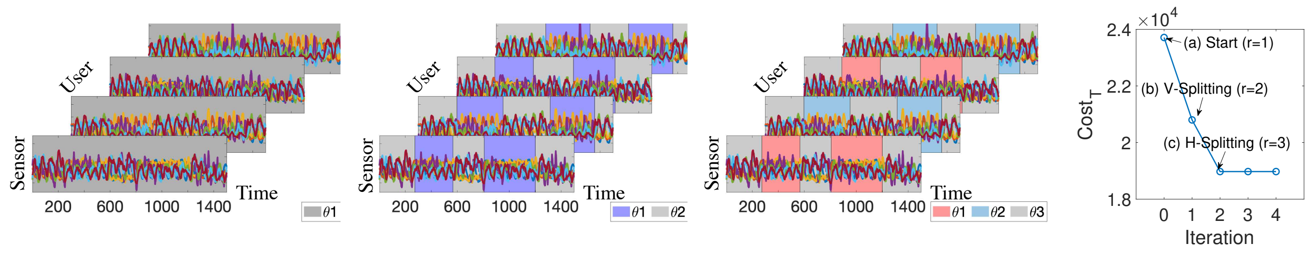
Algorithm. The CUBEMARKER algorithm is shown in detail as Algorithm 3. Our algorithm is based on the greedy approach, that is, it splits a time-series tensor $\mathcal{X}$ into segments, and creates new regimes, as long as the coding cost (Equation 1) keeps decreasing. Specifically, at each step, the algorithm pops an entry $\{m_0, \mathcal{S}_0, \boldsymbol{\theta}_0\}$ from the stack Q. The algorithm then tries to refine the current regime model parameter $\boldsymbol{\theta}_0$ using an appropriate split algorithm (V-Split or H-Split), according to the coding cost. That is, it finds a new candidate regime pair $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2\}$, and their segment sets $\{\mathcal{S}_1, \mathcal{S}_2\}$ for a given segment set $\mathcal{S}_0$. The algorithm compares the cost of the new regimes $Cost_T(\mathcal{X}; \mathcal{S}_1, \mathcal{S}_2, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$ and the cost of the original regime $Cost_T(\mathcal{X}; \mathcal{S}_0, \boldsymbol{\theta}_0)$, and if the new regimes win, it pushes them into the stack Q. Otherwise, it leaves the regime out of the stack, and outputs $\{m_0, \mathcal{S}_0, \boldsymbol{\theta}_0\}$ as the result. The results of these steps are selected based on the cost function (i.e., Equation 1). The algorithm is repeated until the holding stack Q is empty.

Model sampling. Before we start CUBEMARKER, we should initialize the model parameters $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2\}$ with some random values. The most straightforward solution would be simply to estimate the initial parameters using randomly selected segments of $\mathcal{X}$. However, this approach might cause convergence to a local minimum of the cost function, depending on the initial values. We thus propose using a sampling-based approach. The idea is that we first uniformly take several sample segments from the original $\mathcal{X}$. We then estimate the model parameters $\boldsymbol{\theta}_s$ for each sample segment s, and compute the following coding cost of all possible pairs of segments $\{\boldsymbol{\theta}_{s_1}, \boldsymbol{\theta}_{s_2}\}$. Finally, we choose the most appropriate pair $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2\} = \arg\min_{\boldsymbol{\theta}_{s_1}, \boldsymbol{\theta}_{s_2} | s_1, s_2 \in \mathcal{X}_s} Cost_C(\mathcal{X} | \boldsymbol{\theta}_{s_1}, \boldsymbol{\theta}_{s_2})$, where $\mathcal{X}_s = \{s_1, s_2, \ldots\}$ is a set of samples taken from $\mathcal{X}$.

Running example. Figure 3 shows a running example of our method with *(#1) Workout*. Starting with $r = 1$ (at iteration 0), CUBEMARKER progressively finds multi-aspect regimes. Specifically, CUBEMARKER first finds *time-aspect* pattern transitions, i.e., $\boldsymbol{\theta}_1$: shoulder workout (purple) $\rightarrow$ $\boldsymbol{\theta}_2$: arm curl (gray) in Figure 3(b). Next, CUBEMARKER captures the individual difference of $\boldsymbol{\theta}_1$: shoulder workout, and splits the regime into new regimes, i.e., $\boldsymbol{\theta}_1$: shoulder workout #1 (side raise) and $\boldsymbol{\theta}_2$: shoulder workout #2 (rowing) shown in Figure 3(c). During regime splitting, the cost is greatly reduced (see Figure 3(d)).

Lemma 3: The computation time of our method is linear with the input size.

Proof 3: For each iteration, V-Split requires $O(dwn(k_1 + k_2)^2)$ time to find the segments and regimes, and estimate the model parameters. H-Split requires $O(dwn(k_1^2 + k_2^2))$ time, and thus the time complexity is $O(\#iter \cdot dwn(k_1 + k_2)^2)$.



(a) Iter 0 (original, $r = 1$) (b) Iter 1 (V-Split, $r = 2$) (c) Iter 2 (H-Split, $r = 3$) (d) Total cost at each iteration.

Fig. 3. Running example of CUBEMARKER with *(#1) Workout* (shown in Table III). (a)-(c) CUBEMARKER finds multi-aspect regimes per iteration (i.e., $\boldsymbol{\theta}_1$: shoulder workout #1 (side raise) $\boldsymbol{\theta}_2$: shoulder workout #2 (rowing) and $\boldsymbol{\theta}_3$: arm curl) using (b) V-Split and (c) H-Split, (d) while reducing the total cost (Equation 1).

TABLE III
SUMMARY OF REAL-WORLD DATASETS.

Dataset	Data size ($d \times w \times n$)
(#1) Workout	$7 \times 182 \times 4000$
(#2) Tennis	$7 \times 100 \times 4500$
(#3) Factory	$7 \times 60 \times 3000$
(#4) Reading	$5 \times 71 \times 10000$
(#5) Free throw	$7 \times 170 \times 2000$
(#6) Automobile-Tokyo	$3 \times 171 \times 2400$
(#7) Automobile-Expressway	$3 \times 13 \times 9100$
(#8) Automobile-Togu	$3 \times 32 \times 5200$

Here, the numbers of iterations $\#iter$ and hidden states k_1, k_2 are negligible because they are small constant values. Thus, the complexity is $O(dwn)$.

V. EXPERIMENTS

To evaluate the capacity of CUBEMARKER, we carried out experiments on the eight real-world datasets shown in Table III. We normalized the values of each dataset so that they had the same mean and variance (i.e., z-normalization). Our experiments were conducted on 2.7GHz 12-core Intel Xeon E5 processor with 64GB of memory. The experiments were designed to answer the following questions:

- Q1 *Effectiveness*: How successful is our method in capturing multi-aspect patterns in given complex sensor sequences?
- Q2 *Accuracy*: How well does our method find regimes and their shifting points?
- Q3 *Scalability*: How does our method scale in terms of computation time?

A. Q1: Effectiveness

We have already provided examples of CUBEMARKER in Figure 1 and Figure 3. Here we provide other results.

1) Sport analysis: We first describe our results for *(#2) Tennis*. This dataset consists of $d = 7$ dimensional sensors, which are collected by 3-d acceleration sensors and a 4-d electromyogram (50 Hz) attached to the right arm of the user. We compared our method with state-of-the-art methods, namely, TICC [28], AutoPlait [29] and pHMM [26]. Figure 4 (a) shows the results we obtained with our method. Figure 4 (b-1) and (b-2) show the results obtained with TICC. It required two regularization parameters, β and λ, which corresponded to the number of segments and their length. Moreover, TICC needs a parameter for the number of regimes, and we gave the correct number of regimes (i.e., r=6). Figure 4 (c) shows the result obtained with AutoPlait, and Figure 4 (d-1) and (d-2) show the results obtained with pHMM. pHMM also needs two parameters, ϵ_r, ϵ_c, which corresponded to the fitting errors.

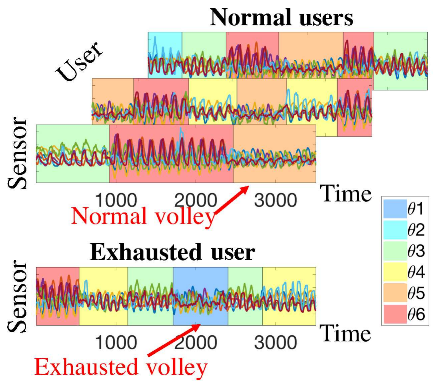
Multi-aspect pattern detection: Our method found various important regimes, i.e., $\boldsymbol{\theta}_2$: *backhand slice*, $\boldsymbol{\theta}_3$: *forehand stroke*, $\boldsymbol{\theta}_4$: *backhand stroke*, $\boldsymbol{\theta}_5$: *volley*, and $\boldsymbol{\theta}_6$: *smash*. In contrast, the competing approaches could not find such tennis strokes even when they were given the correct number of clusters and appropriate parameters. Moreover, they could not find user-specific patterns. Note that our method could identify the above multi-aspect regimes even though the dataset includes different lengths and numbers of segments without any parameters.

Physical fatigue identification: Here we show an interesting result in terms of the discovery of personal differences, namely *physical fatigue identification*. The top three sequences in Figure 4 were obtained from normal users and the bottom of the figure shows an exhausted user's sequence. We had a professional tennis player confirm that the exhausted user could not execute a volley well compared with the other strokes. The bottom of Figure 4(a) shows that our method precisely identified not only usual strokes but also unusual strokes, i.e., $\boldsymbol{\theta}_5$: *normal volley* and $\boldsymbol{\theta}_1$: *exhausted volley*, although our competitors could not detect these user-specific (*user-aspect*) differences.

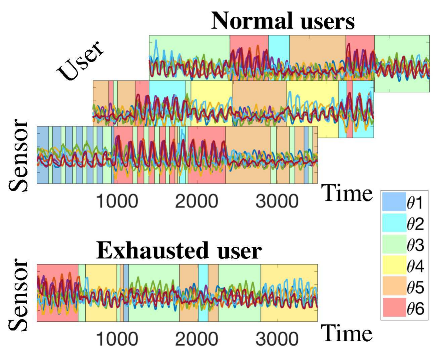
Consequently, our method can discover individual differences and similarities in $\mathcal{X}$ thanks to our multi-aspect mining.

2) Automobile driving analysis: The next example is an automobile driving analysis, where the aim is to deeply understand human driving activities, and find some usual or unusual behavior so that we can integrate their features in self-driving cars and develop advanced self-driving technology without depending entirely on attached cameras to avoid serious problems.

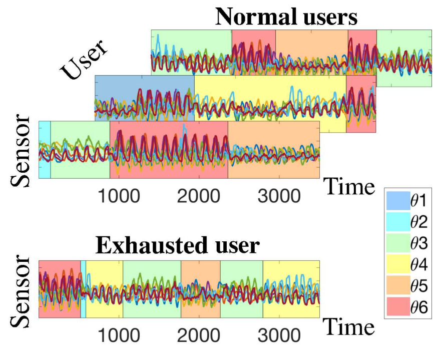
The dataset consisted of sequences of 3-dimensional vectors: velocity, longitudinal and lateral acceleration. We performed experiments on these real automobile datasets and have already shown the result for *(#8) Automobile-Togu* in Figure 1. Here, we carried out experiments on *(#7) Automobile-Expressway*, which were obtained by driving on an ordinary road ($t = 0 \sim 2000, 6000 \sim 9100$) and an urban expressway ($t = 2000 \sim 6000$).



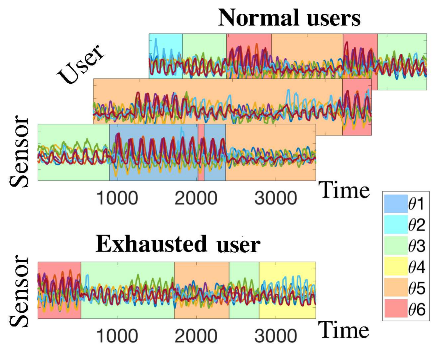
(a) **CUBEMARKER**
(**no** parameter setting)



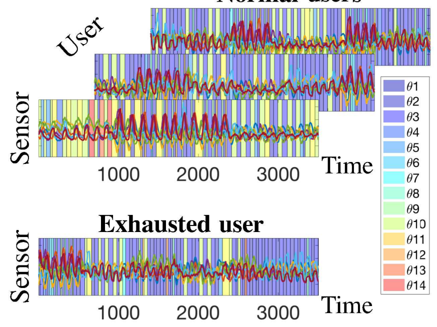
(b-1) TICC ($\beta = 100, \lambda = 1000$)
(**need** parameter setting)



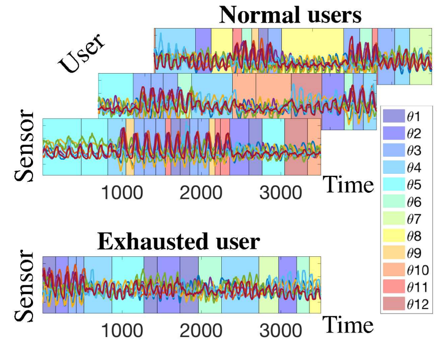
(b-2) TICC ($\beta = 600, \lambda = 1000$)
(**need** parameter setting)



(c) AutoPlait
(**no** parameter setting)



(d-1) pHMM ($\epsilon_r = 0.1, \epsilon_c = 0.8$)
(**need** parameter setting)



(d-2) pHMM ($\epsilon_r = 10, \epsilon_c = 0.8$)
(**need** parameter setting)

Fig. 4. CUBEMARKER is automatic and effective: Results with (a) our method and (b-d) those of our competitors for *(#2) Tennis* dataset consisting of $d = 7$ sensors, $w = 100$ users/players, $n = 4500$ time points. The top three sequences of the figure were obtained from normal users/players and the bottom sequence was obtained from an exhausted user/player. CUBEMARKER successfully identifies multi-aspect patterns, such as $\boldsymbol{\theta}_6$: *smash* and $\boldsymbol{\theta}_3$: *forehand stroke* regardless of the length and number of segments. Moreover, our method is able to find a user-specific stroke, i.e., $\boldsymbol{\theta}_1$: *exhausted volley*. Note that CUBEMARKER finds the correct patterns and their groups, while its competitors are very sensitive to parameter settings.

We are especially interested in the following questions: (1) Can CUBEMARKER indeed discover high-quality regimes? (2) Can CUBEMARKER capture meaningful time transitions? And (3) do different regimes reflect *user-aspect* differences? On the basis of the results, we obtained the following interesting findings.

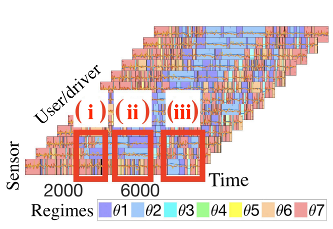
(1) Regimes are highly interpretable. In Figure 5, the segments belong to 7 regimes, which clearly reflect such driving activities as *careful driving (red area,* $\boldsymbol{\theta}_7$*)*, and *speeding up (blue area,* $\boldsymbol{\theta}_2$*)*.

(2) The regime shifting points are intuitive. For example, see the top of Figure 5(c), which shows the details of the shifting points, that is, *careful driving (red area,* $\boldsymbol{\theta}_7$*)* $\rightarrow$ *going straight (orange area,* $\boldsymbol{\theta}_6$*)* $\rightarrow$ *careful driving (red area,* $\boldsymbol{\theta}_7$*)* $\rightarrow$ *speeding up due to an interchange (blue area,* $\boldsymbol{\theta}_2$*)*. This clearly reflects the transition of driving operations, and our method finds these shifting points perfectly for all users.

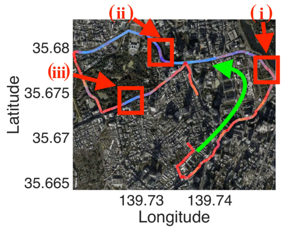
(3) The regimes capture the individual differences in driving patterns. Again, see Figure 5(c), users A and B accelerate around an interchange, while user C (the bottom of Figure 5(c)) continues with *careful driving (red area,* $\boldsymbol{\theta}_7$*)* due to a traffic sign.

As can be seen in the above, CUBEMARKER finds these multi-aspect driving patterns and their shifting points, which help us understand driving behavior.

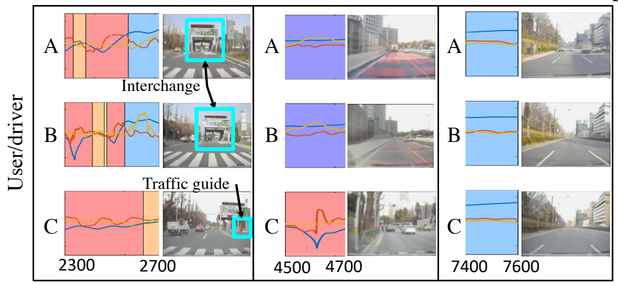
3) Factory worker monitoring: The last example is factory worker monitoring for *(#3) Factory*. Here, the aim is to monitor human workers in factories and find any unusual and user-specific behavior so that we can check the workers' mental and physical condition and avoid serious errors. Figure 6 shows our result for the dataset. This dataset consists of $d = 7$ dimensional event entries including acceleration and an electromyogram, which are monitored with a device attached to the right arm of the worker. The event activity consisted of two consecutive steps: *carrying* $\rightarrow$ *assembling* $\rightarrow \cdots$. In this situation, our aim is to monitor the worker's behavior and identify any user-specific patterns to check the condition of both worker and machine. Our method successfully captures user-specific activities, such as $\boldsymbol{\theta}_1$: *discarding defective products* and $\boldsymbol{\theta}_2$: *arm stretch*. $\boldsymbol{\theta}_1$ implies machine trouble, and $\boldsymbol{\theta}_2$ indicates worker fatigue. This can help us to avoid serious errors.



(a) Multi-aspect segmentation and summarization



(b) Representative driving behavior on a map



(c-i) Interchange (c-ii) Expressway (c-iii) Wide road
(c) User/driver-specific behavior at three different locations

Fig. 5. Multi-aspect mining for *(#7) Automobile-Expressway*: (a) Overall summarization, (b) plotted on a map, shown as (c-i)-(c-iii) in detail. CUBEMARKER successfully captures important, multi-aspect patterns and their shifting points simultaneously, automatically and quickly.

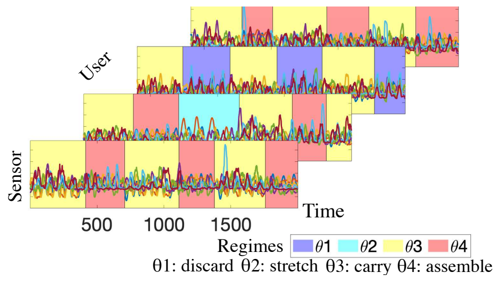


Fig. 6. Multi-aspect mining for *(#3) Factory*. CUBEMARKER successfully captures normal patterns (i.e., $\boldsymbol{\theta}_3$: *carrying* and $\boldsymbol{\theta}_4$: *assembling*) and also finds user-specific patterns (i.e., $\boldsymbol{\theta}_1$: *discarding defective product*). Moreover, CUBEMARKER finds a one-shot outlier (i.e., $\boldsymbol{\theta}_2$: *stretch arms*).

B. Q2: Accuracy

We now discuss the quality of our method in terms of segmentation, clustering and modeling accuracy. We compare our method with TICC [28], AutoPlait [29] and pHMM [26]. Moreover, to reveal the effectiveness of our multi-aspect mining, we also compare it with CUBEMARKER-V, which performs segmentation and regime identification only with V-Split. pHMM and TICC require user-defined parameters, and we varied the accuracy threshold ϵ_r from 0.1 to 100 for pHMM, and the regularization parameter β from 1 to 2000 for TICC. TICC also needs the number of clusters, and we conducted experiments on various numbers of clusters (i.e., 2, 4 and 8, shown as TICC-2, TICC-4 and TICC-8). There are no correct answers for datasets *(#6)-(#8)*, and so we applied our method to *(#1)-(#5)* in terms of pattern segmentation and segment clustering.

1) Accuracy of pattern segmentation: First, we show how accurately our method can find pattern transitions. Figure 7 shows the precision and recall scores of the competitors. Here, precision is defined as the ratio of reported correct shifting points versus the total number of reported shifts. Recall is defined as the ratio of reported correct shifts versus the total number of correct shifts. Note that we must achieve high values for both the scores in a segmentation task. That is, the closer the precision and recall results are in the top right of the figure, the more accurate the method is. In Figure 7, CUBEMARKER and AutoPlait are described as a point, since these methods do not need any parameters. Our method is very close to the ideal point.

2) Accuracy of segment clustering: Next, we discuss the quality of CUBEMARKER in terms of multi-aspect clustering accuracy. Since we know the true labels of each pattern and each sequence, we evaluate our method as a clustering problem. Specifically, we adopt a standard measure of conditional entropy (CE) from the confusion matrix (CM) of the prediction regime labels against true regime labels to evaluate the clustering quality. The CE score shows the difference between two clusters using the following equation: $CE = -\sum_{i,j} \frac{CM_{ij}}{\sum_{ij} CM_{ij}} \log \frac{CM_{ij}}{\sum_j CM_{ij}}$. Note that the ideal confusion matrix will be diagonal, in which case $CE = 0$. Figure 8 compares CUBEMARKER with our competitors in terms of CE score. Our method consistently outperforms the competitors in this task because of our multi-aspect mining approach.

C. Q3: Scalability

We now evaluate the efficiency of CUBEMARKER. Figure 9 shows the wall clock time for each dataset at $d = 100, w = 100, n = 1000$. We used $\epsilon_r = 0.1, \epsilon_c = 0.8$ for pHMM, and $\lambda = 1000, \beta = 600$ for TICC. Figure 10 also compares our method with the competitors for *(#1) Workout* in terms of computational cost when the number of sensors d, timestamps n and users w are varied. As we expected, CUBEMARKER, AutoPlait and TICC are linear with regard to data size (i.e., $slope = 1.0$ in log-log scale). However, pHMM needs $O(n^2)$ (i.e., $slope \approx 2.0$). In fact, on average CUBEMARKER is 1700 times faster than pHMM at $d = 100, w = 100, n = 1000$, and 2.3 times faster than AutoPlait. Moreover, as we mentioned in Section IV-D, CUBEMARKER finds multi-aspect patterns effectively and efficiently. Our method is 1.8 times faster than CUBEMARKER-V. Consequently, our method requires $O(dwn)$ time despite having the ability to find multi-aspect patterns and this improves the model quality.

VI. CONCLUSIONS

In this paper, we focused on the problem of the automatic mining of complex sensor sequences, and presented CUBEMARKER, which exhibits all the desirable properties:

- **Effective**: it finds meaningful patterns and groups (i.e., multi-aspect segments and regimes) in complex sensor sequences.
- **Automatic**: it needs no parameter tuning, thanks to our coding scheme.
- **Scalable**: it scales very well since it is linear as regards dataset size (i.e., $O(dwn)$).

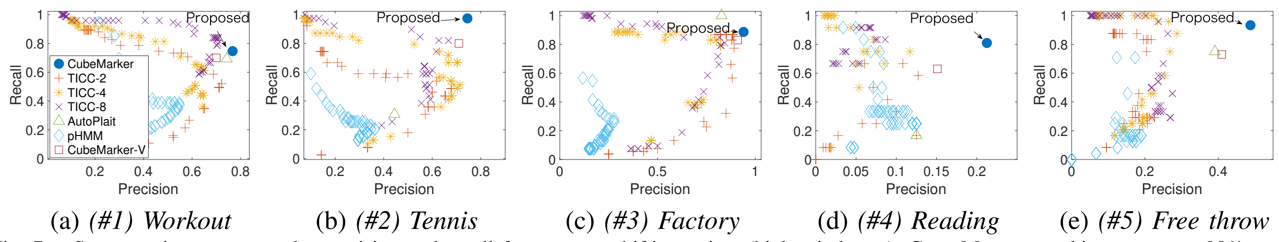


Fig. 7. Segmentation accuracy: the precision and recall for segment shifting points (higher is better). CUBEMARKER achieves at most 98% precision and 92% recall, while its competitors cannot find the correct segments.

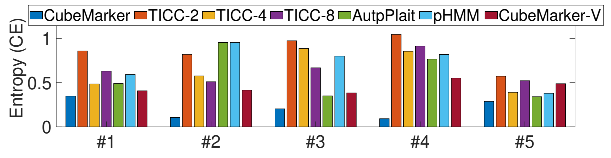


Fig. 8. Clustering accuracy (lower is better): the conditional entropy (CE) of the confusion matrix between true labels and the output results. Even here CUBEMARKER outperforms the other methods.

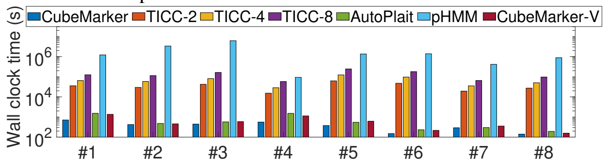


Fig. 9. Average wall clock time for each dataset at $d = 100, w = 100, n = 1000$, shown in log scale. CUBEMARKER is 2.3 times faster than AutoPlait, and 1700 times faster than pHMM.

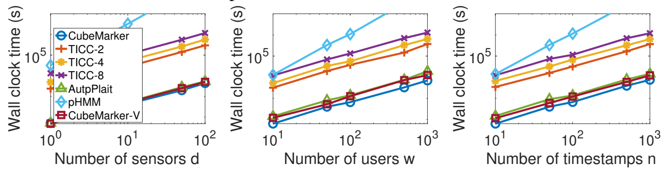


Fig. 10. Wall clock time vs. dataset size for *(#1) Workout* :CUBEMARKER is linear (i.e., *slope* = 1.0 on log-log scale), while pHMM is at least quadratic (i.e., *slope* ≈ 2.0).

We also performed our analytics on real-world IoT datasets (e.g., industries, automobiles and health care), and demonstrated the practicality of our multi-aspect mining approach.

Acknowledgement. This work was supported by JSPS KAKENHI Grant-in-Aid for Scientific Research Number JP17J11668, JP18H03245, JP17H04681, JP16K12430, PRESTO JST.

REFERENCES

[1] Y. Matsubara and Y. Sakurai, "Dynamic modeling and forecasting of time-evolving data streams," in *KDD*, 2019, pp. 458–468.
[2] X. Wu, B. Shi, Y. Dong, C. Huang, L. Faust, and N. V. Chawla, "Restful: Resolution-aware forecasting of behavioral time series data," in *CIKM*, 2018, pp. 1073–1082.
[3] G. E. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis: Forecasting and Control*, 3rd ed. Englewood Cliffs, NJ: Prentice Hall, 1994.
[4] L. Chen and R. T. Ng, "On the marriage of lp-norms and edit distance," in *VLDB*, 2004, pp. 792–803.
[5] M. Vlachos, S. S. Kozat, and P. S. Yu, "Optimal distance bounds on time-series data," in *SDM*, 2009, pp. 109–120.
[6] Y. Matsubara and Y. Sakurai, "Regime shifts in streams: Real-time forecasting of co-evolving time sequences," in *KDD*, 2016, pp. 1045–1054.
[7] S. Papadimitriou and P. S. Yu, "Optimal multi-scale patterns in time series streams," in *SIGMOD*, 2006, pp. 647–658.
[8] Y. Sakurai, M. Yoshikawa, and C. Faloutsos, "Ftw: Fast similarity search under the time warping distance," in *PODS*, Baltimore, Maryland, Jun. 2005, pp. 326–337.
[9] Z. Liu and M. Hauskrecht, "A regularized linear dynamical system framework for multivariate time series analysis," in *AAAI*, 2015, pp. 1798–1804.
[10] L. Li, B. A. Prakash, and C. Faloutsos, "Parsimonious linear fingerprinting for time series," *PVLDB*, vol. 3, no. 1, pp. 385–396, 2010.
[11] L. Li, J. McCann, N. Pollard, and C. Faloutsos, "Dynammo: Mining and summarization of coevolving sequences with missing values," in *KDD*, 2009.
[12] Y. Matsubara, Y. Sakurai, W. G. van Panhuis, and C. Faloutsos, "FUNNEL: automatic mining of spatially coevolving epidemics," in *KDD*, 2014, pp. 105–114.
[13] Y. Zhou, H. Zou, R. Arghandeh, W. Gu, and C. J. Spanos, "Non-parametric outliers detection in multiple time series A case study: Power grid data analysis," in *AAAI*, 2018.
[14] Z. Liu and M. Hauskrecht, "A personalized predictive framework for multivariate clinical time series via adaptive model selection," in *CIKM*, 2017, pp. 1169–1177.
[15] Y. Matsubara, Y. Sakurai, and C. Faloutsos, "Non-linear mining of competing local activities," in *WWW*, 2016, pp. 737–747.
[16] T. G. Kolda, B. W. Bader, and J. P. Kenny, "Higher-order web link analysis using multilinear algebra," in *ICDM*, 2005, pp. 242–249.
[17] Y. Matsubara, Y. Sakurai, C. Faloutsos, T. Iwata, and M. Yoshikawa, "Fast mining and forecasting of complex time-stamped events," in *KDD*, 2012, pp. 271–279.
[18] K. Yang, X. Li, H. Liu, J. Mei, G. T. Xie, J. Zhao, B. Xie, and F. Wang, "Tagited: Predictive task guided tensor decomposition for representation learning from electronic health records," in *AAAI*, 2017, pp. 2824–2830.
[19] H. Xu, D. Luo, and L. Carin, "Online continuous-time tensor factorization based on pairwise interactive point processes," in *IJCAI*, 2018, pp. 2905–2911.
[20] M. Kim, D. A. McFarland, and J. Leskovec, "Modeling affinity based popularity dynamics," in *CIKM*, 2017, pp. 477–486.
[21] C. Battaglino, G. Ballard, and T. G. Kolda, "A practical randomized CP tensor decomposition," in *SIAM*, 2017, pp. 876–901.
[22] R. Zhao and Q. Ji, "An adversarial hierarchical hidden markov model for human pose modeling and generation," in *AAAI*, 2018.
[23] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison, *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1999.
[24] H. Liu, L. He, H. Bai, B. Dai, K. Bai, and Z. Xu, "Structured inference for recurrent hidden semi-markov model," in *IJCAI*, 2018, pp. 2447–2453.
[25] J. G. Wilpon, L. R. Rabiner, C. H. Lee, and E. R. Goldman, "Automatic recognition of keywords in unconstrained speech using hidden Markov models," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 38, no. 11, pp. 1870–1878, 1990.
[26] P. Wang, H. Wang, and W. Wang, "Finding semantics in time series," in *SIGMOD Conference*, 2011, pp. 385–396.
[27] E. J. Keogh, S. Chu, D. Hart, and M. J. Pazzani, "An online algorithm for segmenting time series," in *ICDM*, 2001, pp. 289–296.
[28] D. Hallac, S. Vare, S. Boyd, and J. Leskovec, "Toeplitz inverse covariance-based clustering of multivariate time series data," in *KDD*, 2017, pp. 215–223.
[29] Y. Matsubara, Y. Sakurai, and C. Faloutsos, "Autoplait: Automatic mining of co-evolving time sequences," in *SIGMOD*, 2014, pp. 193–204.
[30] C. Böhm, C. Faloutsos, J.-Y. Pan, and C. Plant, "Ric: Parameter-free noise-robust clustering," *TKDD*, vol. 1, no. 3, 2007.
[31] J. G. DAVID FORNEY, "The viterbi algorithm," in *Proceedings of the IEEE*, 1973, pp. 268–278.
[32] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2006.