Automatic Sequential Pattern Mining in Data Streams

Koki Kawabata*
ISIR, Osaka University
koki88@sanken.osaka-u.ac.jp

Yasuko Matsubara*
ISIR, Osaka University
yasuko@sanken.osaka-u.ac.jp

Yasushi Sakurai*
ISIR, Osaka University
yasushi@sanken.osaka-u.ac.jp

ABSTRACT

Given a large volume of multi-dimensional data streams, such as that produced by IoT applications, finance and online web-click logs, how can we discover typical patterns and compress them into compact models? In addition, how can we incrementally distinguish multiple patterns while considering the information obtained from a pattern found in a streaming setting? In this paper, we propose a streaming algorithm, namely StreamScope, that is designed to find intuitive patterns efficiently from event streams evolving over time. Our proposed method has the following properties: (a) it is effective: it operates on semi-infinite collections of co-evolving streams and summarizes all the streams into a set of multiple discrete segments grouped by their similarities. (b) it is automatic: it automatically and incrementally recognizes such patterns and generates models for each of them if necessary; (c) it is scalable: the complexity of our method does not depend on the length of the data streams. Our extensive experiments on real data streams demonstrate that StreamScope can find meaningful patterns and achieve great improvements in terms of computational time and memory space over its full batch method competitors.

CCS CONCEPTS

• **Information systems → Data mining**;

ACM Reference Format:
Koki Kawabata, Yasuko Matsubara, and Yasushi Sakurai. 2019. Automatic Sequential Pattern Mining in Data Streams. In *The 28th ACM International Conference on Information and Knowledge Management (CIKM'19), November 3–7, 2019, Beijing, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3357384.3358002

1 INTRODUCTION

Pattern discovery in time series has attracted a considerable amount of interest because it has been widely used in numerous applications such as sensor monitoring [19, 26], online social activity [18], medical data analysis [16] and financial data analysis [27]. However, recently, the emergence of small intelligent devices has opened up new opportunities for various types of applications, such as automated factories, smart cities and connected healthcare, namely the "Internet of Things (IoT)" [10, 24]. Machines are getting in on the act. This sudden explosion of intelligent connected devices imposes new requirements on data stream mining, which include (a) real-time modeling, and (b) automatic mining. The ideal method should capture the dynamics of time-series data, efficiently and automatically over IoT data streams. It should also enable us to perform model estimation and advanced analytics, such as anomaly detection, forecasting, and pattern recognition, without any parameter tuning steps.

For example, consider a large collection of observations continuously generated by sensors in an automated factory. The most important goal is to acquire knowledge of sequence patterns reflecting various aspects of the systems such as ordinary processing and signs of accidents, which may have not appeared in the historical/training data, and also to quickly utilize this knowledge for subsequent pattern recognition. In this situation, the data sources generate data with no end in sight, making it difficult to efficiently identify meaningful (i.e., frequent, periodic and abnormal) patterns over one or multiple attributes. Given such data streams, how can we adaptively recognize newly arrived patterns as well as already known ones? How do we efficiently and exactly identify such dynamical patterns?

This paper focuses on an important time series analysis task, namely, automatic and efficient pattern discovery that can reveal all meaningful patterns, which we refer to as a *regime*, consisting of a semi-infinite time-evolving data streams. Intuitively, we design a streaming algorithm, called StreamScope, thus allowing us to deal with the following informal problem.

Informal Problem 1. ***Given*** *a data stream* $\boldsymbol{X}$*, which is composed of a series of multi-dimensional vectors, i.e.,* $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t\}$*, our challenging problem has the following components:*

- ***splitting*** $\boldsymbol{X}$ *into a set of segments by the cut points of multiple meaningful patterns,*
- ***estimating*** *the number of similar segment groups, i.e., regimes,*
- ***summarizing*** *all such dynamical patterns into model parameters,*

quickly and automatically in a streaming setting.

1.1 Preview of our results

Figure 1 shows the online pattern discovery result of StreamScope. The top of Figure 1 shows the original motion capture stream corresponding to left/right legs and arms. The data stream consists of four intuitive motion patterns: walking, stretching left, right, and both arms.

As shown in the bottom of Figure 1, our method can incrementally find these four patterns (i.e., regimes) and in terms of the time-position of each transition (i.e., the colored rectangles in the figure). More specifically, our method first discovers the motion "walking" and summarizes the feature as regime #1. After the motion has been changed to "stretching arms" at time tick 1300, our method

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

CIKM '19, November 3–7, 2019, Beijing, China

ACM ISBN 978-1-4503-6976-3/19/11…$15.00
https://doi.org/10.1145/3357384.3358002

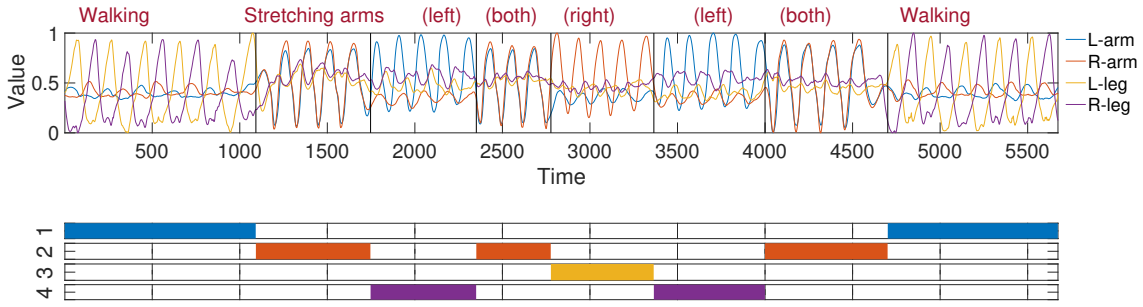


Figure 1: Online pattern discovery with StreamScope: the original motion sensor streams (top), and the output of our method (bottom). At every time tick, the method incrementally identifies intuitive motions (such as walking and stretching arms) and classifies them into groups, namely regimes. If necessary, it automatically generates a new regime with no prior knowledge.

autonomously generates regime #2 for the new pattern because the two regimes give us better summarization for the data stream. Similarly, as the kinds of stretching arms increases, regime #2 becomes insufficient for capturing the features. Thus, it employs regime #3 and #4 at time tick 3600. If a pattern is observed again (i.e., stretching both arms and walking), then it can recognize the change with the models of already estimated regimes and assign the part of the stream as a segment into the most suitable regime.

As a consequence, our method can perform incremental segmentation and dynamic modeling simultaneously, without any prior knowledge and user intervention.

1.2 Contributions

In this paper, we present an online method, namely, StreamScope, for automatic pattern discovery in co-evolving data streams. In summary, the contributions of this paper are as follows:

- *Effective:* it efficiently and correctly realizes pattern discovery in a streaming fashion. Our method can find high-level patterns, i.e., the dynamics not only within a single pattern but also between patterns.
- *Automatic:* for each pattern it can automatically and incrementally estimate the number, transitional points and model parameters. All of these characteristics are unknown in advance.
- *Scalable:* thanks to our proposed incremental update strategy, it is fast and the computational time and memory space are independent of the entire length of the input streams.

We evaluate our approach on real datasets and show that it discovers meaningful patterns effectively and efficiently.

Outline. The remainder of this paper is organized as follows. We first introduce related work, followed by problem statement, our proposed model and algorithms, experiments and conclusions.

2 RELATED WORK

Time series pattern discovery has been studied especially in relation to database and data mining [3, 6, 28, 34]. Table 1 shows the relative advantages of our method. We loosely classify the related work into two groups.

Modeling dynamics and segmentation. Traditional approaches applied to time series modeling include hidden Markov models (HMM), autoregression (AR), linear dynamical systems (LDS), and their variants [5, 15, 17]. More recent advances are based on modeling non-linear behavior on co-evolving time series [21, 23]. However, they are incapable of multiple distinct patterns in large IoT sensor data. For modeling such patterns, there has also been study on segmenting time series by their similar dynamics. Wang et al. [31] presented a pattern-based hidden Markov model (pHMM), which converts a time series into a sequence of line segments, and learns a Markov model from the sequence. Hallac et al. [11] proposed Toeplitz inverse covariance-based clustering (TICC). It is based on a Markov random field (MRF), and can find interpretable clusters, each of which captures the interdependencies between multivariate observations in a typical subsequence of that cluster. However, these methods need parameter tuning for error thresholds, model structure, the number of clusters, etc.

To achieve automatic mining, the minimum description length (MDL) principle has been applied in various contexts [14, 20, 29]. In particular, AutoPlait [20] can automatically identify the intuitive number of high-level patterns (i.e., regimes) using hierarchical HMM-based model, but unfortunately, none of above are streaming methods. The ideal method should require no parameter settings and analyze data in an online fashion.

Data mining on streams. The processing and mining of data streams have also attracted significant interest [12, 32, 33]. Ghesmoune et al. [7] provides a comprehensive overview of basic

Table 1: Capabilities of approaches. Only StreamScope meets all requirements.

	HMM/++	pHMM/TICC	AutoPlait	CluStream/++	Motif/++	ADS+	StreamScan	**StreamScope**
Data stream	-	-	-	✓	✓	-	✓	✓
Online optimization	-	-	-	✓	✓	✓	-	✓
Data compression	✓	✓	✓	✓	-	-	-	✓
Segmentation	-	✓	✓	-	✓	✓	✓	✓
Regime identification	-	-	✓	-	-	-	-	✓
Parameter free	-	-	✓	-	-	-	-	✓

clustering approaches for data streams, such as CluStream [1] and DenStream [4]. EDMStream [8] has recently proposed, which can adaptively change cluster centers to handle time-varying data distribution known as *concept drift* [24]. However, these algorithms process each data point individually and ignore temporal dependency between data, which is unsuitable for extracting the dynamics of time-evolving streams. Time series motif discovery is a well-studied research topic [13, 25, 30], which enables us to detect the frequently occurring subsequences in time series efficiently. ADS+ [34] provides a new adaptive indexing approach for big data series, where the index is built incrementally and adaptively. However, these methods use distance-based metrics (e.g., DTW and Euclidean distance), thus they cannot summarize the dynamics of the patterns into compact models. Although StreamScan [22] is able to identify variable length patterns with HMM on data streams, it requires query models for the pattern we want to find.

In conclusion, none of the above methods satisfy all the requirements, namely, anytime incremental operation, scalability on massive streams, adaptivity, and automation for intuitive pattern discovery.

3 PROBLEM FORMULATION

In this section, we formally describe the problem we want to solve for automatic stream mining of sequential patterns. Table 2 lists the symbols used in this paper. The data stream we consider is a collection of d-dimensional vectors $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t\}$, where $\boldsymbol{x}_t = \{x_i\}_{i=1}^{d}$ arrives at each time tick t.

Given a data stream $\boldsymbol{X}$, our goal is to partition it into a set of m non-overlapping segments denoted by $\mathcal{S} = \{s_1, \ldots, s_m\}$, where s_i consists of the start and end position, i.e., $s = \{t_s, t_e\}$, and m denotes the number of segments.

We also want to assign each segment s_i to any one of r groups of segments (namely *regimes*), where, r is the number of regimes. Here, the regime assignment of each segment is denoted by $\mathcal{F} = \{f(1), \ldots, f(m)\}$, where $f(i)$ is the regime index of i-th segment (i.e., $1 \leq f(i) \leq r$).

For example, in Figure 1, the data stream consists of $m = 8$ segments, each of which belongs to one of the $r = 4$ regimes, e.g., $f(1) = 1, f(2) = 2, f(3) = 4, f(4) = 2, \cdots$.

In this paper, we assume that data streams contains two levels of dynamics, namely, *within-regime* transitions and *across-regime* transitions.

To represent dynamic time-evolving pattern in a single regime (i.e., within-regime transitions), we propose using a hidden Markov model (HMM)[1] [2]. An HMM is a well-known stochastic model used in many applications such as speaker recognition and the analysis of biological sequences. It assumes the system to be a Markov process with discrete hidden states. An HMM with k states is composed of initial probabilities $\boldsymbol{\pi} = \{\pi_i\}_{i=1}^{k}$, transition probabilities $\mathbf{A} = \{a_{ij}\}_{i,j=1}^{k}$, and output probabilities $\mathbf{B} = \{b_i(\boldsymbol{x}_t)\}_{i=1}^{k}$. Consequently, a single regime dynamics can be described as a set of parameters: $\boldsymbol{\theta} = \{\boldsymbol{\pi}, \mathbf{A}, \mathbf{B}\}$.

[1] In our setting, we assume a Gaussian distribution for the output probability, which is able to handle multi-dimensional vectors at each time tick (i.e., $\mathbf{B} = \{\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i^2)\}_{i=1}^{k}$).

Table 2: Symbols and definitions.

Symbol	Definition
d	Number of dimensions
$\boldsymbol{x}_t$	d-dimensional vector at time tick t
$\boldsymbol{X}$	Co-evolving data stream, i.e., $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t\}$
m	Number of segments
$\mathcal{S}$	Segment set, i.e., $\mathcal{S} = \{s_1, \ldots, s_m\}$
$\mathcal{F}$	Segment-membership, i.e., $\mathcal{F} = \{f(1), \ldots, f(m)\}$
r	Number of regimes
$\boldsymbol{\theta}_i$	Model parameters governing i-th regime
k_i	Number of hidden states in $\boldsymbol{\theta}_i$
$\boldsymbol{\Phi}$	Regime transitions, i.e., $\boldsymbol{\Phi} = \{\phi_{ij}\}_{i,j=1}^{r}$
$\boldsymbol{\Theta}$	Model set of r regimes, i.e., $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_r, \boldsymbol{\Phi}\}$
C	Candidate solution, i.e., $C = \{m, r, \mathcal{S}, \boldsymbol{\Theta}, \mathcal{F}\}$

For the upper level, we assume that there are across-regime transitions between multiple distinct regimes $\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_r$. Here, we define a *regime transition matrix* $\boldsymbol{\Phi} = \{\phi_{ij}\}_{i,j=1}^{r}$, which consists of the regime transition probabilities ϕ_{ij} from the i-th regime $\boldsymbol{\theta}_i$ to the j-th regime $\boldsymbol{\theta}_j$. Each probability is computed as $\phi_{ij} = n_{ij} / \Sigma_{s \in \mathcal{S}_i} |s|$ where $\Sigma_{s \in \mathcal{S}_i} |s|$ indicates the total length of the segments belonging to regime $\boldsymbol{\theta}_i$, and n_{ij} shows the regime-switch count from $\boldsymbol{\theta}_i$ to $\boldsymbol{\theta}_j$. Finally, the entire model parameters can be defined as $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_r, \boldsymbol{\Phi}\}$.

Finally, we adapt all of the above components for dynamic summarization on data stream $\boldsymbol{X}$. Specifically, letting $C = \{m, r, \mathcal{S}, \boldsymbol{\Theta}, \mathcal{F}\}$ be a compact representation of $\boldsymbol{X}$, which we refer to as a candidate solution. The problem we want to solve is as follows.

Problem 1. *Given a d-dimensional data stream $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t\}$, where t is the most recent time tick, continuously optimize the candidate solution C, namely,*

- *the number of segments m and their switching positions, $\mathcal{S} = \{s_1, \ldots, s_m\}$,*
- *the number of regimes r and their model parameters, $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_r, \boldsymbol{\Phi}\}$, and*
- *the segment membership of each segment, $\mathcal{F} = \{f(1), \ldots, f(m)\}$,*

in a streaming (i.e., online) setting.

4 AUTOMATIC PATTERN MINING

In this section, we first present how we can define a good summary of co-evolving event streams, and then show our optimization problem, which consists of dynamically estimating a compact description of data streams in an automatic way.

4.1 Offline Data Compression

Here, we assume that we can access and process whole input time series $\boldsymbol{X}$ (i.e., batch processing) to simplify the discussion. Intuitively, our goal is to keep the description of $\boldsymbol{X}$ compact and reasonable using C. This approach indicates that optimal model should minimize the total description cost:

$$Cost_T(\boldsymbol{X}; C) = Cost_M(C) + Cost_C(\boldsymbol{X}|C), \quad (1)$$

where $Cost_M(C)$ is the description cost of C, and $Cost_C(\boldsymbol{X}|C)$ is the data coding cost given the model C. Thus, we apply the minimum description length (MDL) principle [9], which makes it possible to evaluate how well our model compresses $\boldsymbol{X}$. We begin by defining the two components of the total cost more concretely.

Model description cost. The model description cost is the number of bits needed to describe the model. If we use more powerful model architecture, the total cost becomes higher. For our full parameter set C, the number of dimensions, segments and regimes requires $\log^*(d) + \log^*(m) + \log^*(r)$ bits[2]. The segment membership of the regimes requires $Cost_M(\mathcal{F}) = m\log(r)$ bits. The length of each segment s needs $Cost_M(\mathcal{S}) = \sum_{i=1}^{m} \log^* |s_i|$ bits. The model parameter set Θ needs: $Cost_M(\Theta) = \sum_{i=1}^{r} Cost_M(\boldsymbol{\theta}_i) + Cost_M(\Phi)$, where a single model $\boldsymbol{\theta}$ requires the sum of $\log^*(k)$ bits for the number of hidden states, and $C_F(k + k^2 + 2kd)$ for a Gaussian HMM, i.e., $\{\boldsymbol{\pi}, \mathbf{A}, \mathbf{B}\}$. Here, C_F is the floating point cost[3]. The regime transition matrix also requires $Cost_M(\Phi) = C_F \cdot r^2$.

Finally, the total model description cost $Cost_M(C)$ is the sum of the above terms:

$$\begin{aligned} Cost_M(C) &= \log^*(d) + \log^*(m) + \log^*(r) \\ &\quad + Cost_M(\mathcal{S}) + Cost_M(\Theta) + Cost_M(\mathcal{F}) \end{aligned} \tag{2}$$

Data encoding cost. Given a full parameter set Θ, data compression using Huffman coding assigns a number of bits to each value in $\boldsymbol{X}$, which is a negative log-likelihood. The encoding cost of $\boldsymbol{X}$ given $\boldsymbol{\theta}$ is computed by: $Cost_C(\boldsymbol{X}|\boldsymbol{\theta}) = -\log P(\boldsymbol{X}|\boldsymbol{\theta})$, where $P(\boldsymbol{X}|\boldsymbol{\theta})$ is the likelihood of $\boldsymbol{X}$. Thus, the total encoding cost is:

$$\begin{aligned} Cost_C(\boldsymbol{X}|C) &= \sum_{i=1}^{m} Cost_C(\boldsymbol{X}[s_i]|\Theta) \\ &= \sum_{i=1}^{m} -\log(\phi_{vu}(\phi_{uu})^{|s_i|-1} P(\boldsymbol{X}[s_i]|\boldsymbol{\theta}_u)), \end{aligned} \tag{3}$$

where the i-th and $(i-1)$-th segments are governed by the u-th and v-th regimes, respectively. $\boldsymbol{X}[s_i]$ is a subsequence of segment s_i.

4.2 Online Optimization Objective

Next, we focus on how to evaluate the total cost $Cost_T(\boldsymbol{X}; C)$ efficiently. In a streaming setting, we cannot store and process all historical data because its size becomes extremely large as time progresses. To search for the optimal segments/regimes incrementally, we only consider the increase in the total cost when the set of new stream elements is added to $\boldsymbol{X}$. We roughly refer to this as the most recent subsequence $\boldsymbol{X}_c = \boldsymbol{X}[t' : t]$ (i.e., $1 \ll t' < t$). Assuming that the two numbers m and r have increased to m_+ and r_+, respectively, given $\boldsymbol{X}_c$ at time tick t, we define the additional description cost in Equation (1) as follows:

$$\begin{aligned} \Delta Cost_T(\boldsymbol{X}_c; C) &= \log^*(m_+) - \log^*(m) + \log^*(r_+) - \log^*(r) \\ &\quad + \Delta Cost_M(\mathcal{S}) + \Delta Cost_M(\Theta) + \Delta Cost_M(\mathcal{F}) \\ &\quad + Cost_C(\boldsymbol{X}_c|\Theta). \end{aligned} \tag{4}$$

[2]Here, $\log^*$ is the universal code length for integers.
[3]We used 4×8 bits in our setting.

Here $\Delta Cost_M(\cdot)$ represents the increase in description length for new entries in the set:

$$\Delta Cost_M(\mathcal{S}) = \sum_{i=m}^{m_+-1} \log^* |s_i|, \tag{5}$$

$$\Delta Cost_M(\Theta) = \sum_{i=r+1}^{r_+} Cost_M(\boldsymbol{\theta}_i) + C_F \cdot r_+^2 - C_F \cdot r^2, \tag{6}$$

$$\Delta Cost_M(\mathcal{F}) = m_+ \log(r_+) - m\log(r). \tag{7}$$

A key advantage of applying our techniques to data streams is that it can jointly estimate the number of regimes while processing. The above updating strategy is designed to flexibly deal with the number of regimes in $\boldsymbol{X}_c$. If we represent $\boldsymbol{X}_c$ only with existing regimes, i.e., $r_+ = r$, then the additional cost $\Delta Cost_M(\Theta) = 0$, otherwise, further cost is required corresponding to the number and complexity of the new regimes $\boldsymbol{\theta}$. This observation indicates that more complex or duplicate regimes requires a higher cost to employ. Our objective function is likely to keep the entire model structure reasonable.

5 STREAMING ALGORITHMS

We now present our proposed StreamScope for the fast and automatic pattern discovery of time-evolving data streams.

5.1 Overview

Although we have proposed our incremental coding scheme, i.e., Equation (4), it is still difficult to globally optimize the score because the number of all possible combinations of candidate segments/regimes in the whole input $\boldsymbol{X}$ becomes extremely rich. So, how efficiently can we monitor time-evolving streams while maintaining the quality of a full model set C?

To tackle this important problem, we carefully design an online optimization algorithm, namely StreamScope, which utilizes two sub-algorithms and maintains the model parameter set C. In StreamScope, we define the most recent subsequence $\boldsymbol{X}_c$ as the latest pattern we found, i.e., $\boldsymbol{X}_c = \boldsymbol{X}[t' : t]$ where t' and t show the last regime switching point and current time tick, respectively. Given $\boldsymbol{X}_c$, the decision for its model update can be decided via two procedures, namely,

(1) SegmentAssignment: This quickly searches the most suitable model/regime to represent arriving data $\boldsymbol{x}_t$ without any modification or addition of regimes.
(2) RegimeGeneration: This considers inner/outer extensions of regimes by estimating new model(s) $\boldsymbol{\theta}$ that capture dynamical patterns in $\boldsymbol{X}_c$.

Most importantly, since these procedures are completely separate from each other, we can decide how the entire model C updates by applying either of the outputs minimizing Equation (4). If $\boldsymbol{X}_c$ includes different dynamics from existing regimes, the model structure should be adapted to the dynamics even if it incurs the extra costs of model extensions. We describe StreamScope in detail after presenting the two algorithms in steps.

5.2 SegmentAssignment

As the simplest case scenario, we first assume that we know several patterns/regimes and their model parameters. Our first goal is

to identify which of the patterns is arriving at any given moment. Given a d-dimensional vector $\boldsymbol{x}_t$ at time tick t and a model parameter set of r regimes $\Theta = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_r, \Phi\}$, we want to find cut points that divide $\boldsymbol{X}$ into individual patterns (i.e., segments), so that we can minimize $Cost_C(\boldsymbol{X}|\Theta)$.

Algorithm 1 shows the overall procedure for the incremental assignment, namely, SegmentAssignment. In the first step, we compute the likelihood value $P(\boldsymbol{x}_t|\Theta)$ as follows:

$$P(\boldsymbol{x}_t|\Theta) = \max_{1\le i\le r}\{\max_{1\le j\le k_i}\{p_{t;i;j}\}\}, \tag{8}$$

$$p_{t;i;j} = \max\begin{cases}\phi_{ui}\cdot\max_{v}\{p_{t-1;u;v}\}\cdot\pi_{i;j}\cdot b_{i;j}(\boldsymbol{x}_t)\\ \phi_{ii}\cdot\max_{u}\{p_{t-1;i;u}\cdot a_{i;uj}\}\cdot b_{i;j}(\boldsymbol{x}_t)\end{cases} \tag{9}$$

where: $p_{t;i;j}$ is the maximum probability of state j of regime $\boldsymbol{\theta}_i$ at time tick t. $\pi_{i;j}$ is the initial probability of state j of $\boldsymbol{\theta}_i$. $b_{i;j}(\boldsymbol{x}_t)$ is the output probability of $\boldsymbol{x}_t$ for state j of $\boldsymbol{\theta}_i$, and $a_{i;uj}$ is the transition probability from state u to state j in $\boldsymbol{\theta}_i$. At time tick $t = 1$, the probability for each regime is given by $p_{1;i;j} = \phi_{ii}\cdot\pi_{ij}\cdot b_{i;j}(\boldsymbol{x}_1)$. Next, we retain the cut points using a candidate set $\mathcal{L} = \{L_1, L_2, \ldots\}$ for each state of all the regimes, where L_i denotes the tuple of the location t_i and the regime index w_i of the i-th cut point, which it visited (i.e., $L_i = \{t_i, w_i\}$). The entire candidate set is:

$$\mathcal{L}_{t;i;j} = \begin{cases}\mathcal{L}_{t-1;u;v}\cup\{L\} & \text{// if regime switches}\\ \mathcal{L}_{t-1;i;j} & \text{// otherwise}\end{cases} \tag{10}$$

where $\mathcal{L}_{t;i;j}$ shows the candidate set for state j of $\boldsymbol{\theta}_i$ at time tick t. We can incrementally update these likelihood and candidate cut-point sets using a dynamic programming approach that maximizes the probabilities solely from previous regimes and states.

γ–guarantee. If the regime that gives the maximum likelihood is switched to another regime, the time tick t is a cut point, i.e., a starting point of the new segment. However, this determinant cannot be optimum due to noise and uncertainness about similar patterns. To avoid a misassignment due to the temporal maximum likelihood, we propose γ–guarantee, which provides the cut-point set with a guarantee of exactness by using a time interval γ. If we find the first candidate cut point switching to the i-th regime at time tick t, the determinant cut point is obtained at time tick $t+\gamma$. The interval γ is typically set to a half the average length of segments.

Lemma 5.1. *For a given model $\Theta = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_r, \Phi\}$ and stream $\boldsymbol{X}[1 : t+\gamma]$, the exactness of the cut point set until time tick t is guaranteed at time tick $t+\gamma$.*

Proof. $P(\boldsymbol{x}_{t+\gamma}|\Theta)$ shows the maximum likelihood at time tick $t+\gamma$. SegmentAssignment computes the likelihood of every regime each with states for each time tick. At the same time, it also reports all candidate cut points without omission. Therefore, at time tick $t+\gamma$, the cut-point set giving the maximum probability $P(\boldsymbol{x}_{t+\gamma}|\Theta)$ is equivalent to a Viterbi path, that is, the optimal cut-point set. The cut point set until time tick t is thus exactly optimal at time tick $t+\gamma$. □

Algorithm 1 SegmentAssignment $(\boldsymbol{x}_t, \Theta)$

Input: (a) New vector $\boldsymbol{x}_t$ at time tick t
(b) Regime parameter set $\Theta = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_r, \Phi\}$
Output: (a) Number of segments m
(b) Segment set $\mathcal{S}$
(c) Segment membership $\mathcal{F}$

```
m ← 0; S ← ∅; F ← ∅;
/* Find candidate cut-points with Equations (9) and (10) */
for i = 1 : r do
    Compute p_{t;i;j} for state j = 1 to k_i;
    Update L_{t;i;j} for state j = 1 to k_i;
end for
if the first regime transition is found then
    t_γ ← t + γ;
end if
if t = t_γ then
    /* Add segments into the optimal regime */
    L_best ← argmax_{L_{t;i;j} | 1≤i≤r, 1≤j≤k_i} P(x_t|Θ); // Equation (8)
    for each cut point L_i = {t_i, w_i} ∈ L_best do
        Create a new segment s = {t_s, t_i};
        Add s into S; f(m+1) = w_i; m ← m+1;
        t_s ← t_i + 1;
    end for
    L_t ← ∅;
end if
return {m, S, F};
```

5.3 RegimeGeneration

In this step, our goal is to find the local optimal number of segments, regimes and their model parameters thus minimizing the cost $\Delta Cost_T(\boldsymbol{X}_c; C) = \Delta Cost_M(C) + Cost_C(\boldsymbol{X}_c; C)$. RegimeGeneration (Algorithm 2) initially regards $\boldsymbol{X}_c$ as a single regime, i.e., $m_0 = 1, \mathcal{S}_0 = \{1, |X_c|\}$, and estimates the model parameter $\boldsymbol{\theta}_0$. Given the initial parameter set $\{m_0, \mathcal{S}_0, \boldsymbol{\theta}_0\}$, it greedily tries to divide $\boldsymbol{X}_c$ by iterating the following two steps until a stack Q for keeping candidate regimes becomes empty.

(1) The algorithm first finds a new candidate pair $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \Phi\}$. Given the candidate regimes $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \Phi\}$, it iterates finding the cut points of segments, $\{\mathcal{S}_1, \mathcal{S}_2\}$, by computing the maximum likelihood (i.e., Equation (8)) and re-estimating $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \Phi\}$, while $\Delta Cost_T(\boldsymbol{X}_c[\mathcal{S}_0]|\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \Phi)$ is improved.
(2) If the total cost of applying the candidate pair is lower that than when applying only $\boldsymbol{\theta}$, the algorithm stores the new candidate set $\{m_1, \mathcal{S}_1, \boldsymbol{\theta}_1\}$ and $\{m_2, \mathcal{S}_2, \boldsymbol{\theta}_2\}$ into Q to further divide the candidate regimes. Otherwise, it updates the parameters in C based on $\{m_0, \mathcal{S}_0, \boldsymbol{\theta}_0\}$.

Note that we use the Baum-Welch algorithm whenever we undertake the parameter estimation of a single regime $\boldsymbol{\theta}$ in the above procedure. Since the Baum-Welch algorithm needs the number of hidden states k in a model $\boldsymbol{\theta}$ to be specified, we vary $k = 1, 2, \ldots$ while the cost, $\Delta Cost_T(\boldsymbol{X}[\mathcal{S}]|\boldsymbol{\theta})$, can be decreased. This approach prevents overfitting and provides a reasonable k without any prior information.

Model initialization. When we start the iterations with a candidate regime including segments $\mathcal{S}$, we uniformly take several sample segments from $\boldsymbol{X}_c[\mathcal{S}]$ and estimate the model parameters $\boldsymbol{\theta}_s$ for

Algorithm 2 RegimeGeneration (X_c)

Input: Subsequence X_c
Output: Candidate solution C

$Q \leftarrow \emptyset$; // Stack for candidate regimes
$m_0 \leftarrow 1$; $\mathcal{S}_0 \leftarrow \{1, |X_c|\}$; $\theta_0 \leftarrow$ ModelEstimation(X_c);
Push an entry $\{m_0, \mathcal{S}_0, \theta_0\}$ into Q;
while stack $Q \neq \emptyset$ **do**
  Pop an entry $\{m_0, \mathcal{S}_0, \theta_0\}$ from Q;
  $\{\theta_1, \theta_2\} \leftarrow$ ModelInitialization($X_c[\mathcal{S}_0]$); // Equation (11)
  repeat
    for $x_t \in X_c[\mathcal{S}_0]$ **do**
      $\{m^*, \mathcal{S}^*, \mathcal{F}^*\} \leftarrow$ SegmentAssignment(x_t, θ_1, θ_2, $\Phi_{2\times2}$);
    end for
    $\mathcal{S}_1 \leftarrow \{s_i \in \mathcal{S}^* | f(i) = 1, i = 1, \ldots, m^*\}$;
    $\mathcal{S}_2 \leftarrow \{s_i \in \mathcal{S}^* | f(i) = 2, i = 1, \ldots, m^*\}$;
    $\theta_1 \leftarrow$ ModelEstimation($X_c[\mathcal{S}_1]$);
    $\theta_2 \leftarrow$ ModelEstimation($X_c[\mathcal{S}_2]$);
    Update regime transition matrix $\Phi_{2\times2}$;
  until convergence;
  if $\Delta Cost_T(X_c; \mathcal{S}_1, \mathcal{S}_2, \theta_1, \theta_2) < \Delta Cost_T(X_c; \mathcal{S}_0, \theta_0)$ **then**
    /* Recursively try to split new regimes */;
    Push entries $\{m_1, \mathcal{S}_1, \theta_1\}$, $\{m_2, \mathcal{S}_2, \theta_2\}$ into Q;
  else
    $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{S}_0$; $\Theta \leftarrow \Theta \cup \theta_0$; $r \leftarrow r + 1$;
    $f(i) = r$, for each $i = m + 1, \ldots, m + m_0$; $m \leftarrow m + m_0$;
    Update $\Phi_{r\times r}$;
  end if
end while
return $C = \{m, r, \mathcal{S}, \Theta, \mathcal{F}\}$;

each of them. The most appropriate pair is chosen by computing the coding cost of all possible pairs:

$$\{\theta_1, \theta_2\} = \underset{\theta_{s_1}, \theta_{s_2} | s_1, s_2 \in \mathcal{X}}{\arg\min} Cost_C(X_c[\mathcal{S}] | \theta_{s_1}, \theta_{s_2}), \tag{11}$$

where $\mathcal{X} = \{s_1, s_2, \ldots\}$ is a set of samples taken from $X_c[\mathcal{S}]$.

5.4 StreamScope

Now, we explain how to efficiently identify regimes in a large event stream. Our final goal is to answer the questions: (a) when should we estimate model parameter θ for new regimes and confirm their necessity for an effective pattern mining? And (b) how can we obtain the compact description C at each time tick from the outputs of the two algorithms we have discussed?

Now, we address the first question. Algorithm 3 shows the overall procedure of StreamScope. Given a vector x_t and the candidate solution C, it first executes SegmentAssignment to detect regime transitions. Recall that it retains candidate transition points i.e., Equation (10). StreamScope tries to estimate new regimes by using RegimeGeneration, at the same time as the candidates are verified, which can reveal that the current regime is switching to new regime not to one of the existing regimes.

Online update of full parameter set. Here, we describe how to update C based on the output of SegmentAssignment (i.e., C_S) or RegimeGeneration (i.e., C_R). Given the two candidates, StreamScope compares the extra costs, i.e., Equation (4), then decides which candidate it applies to C.

Algorithm 3 StreamScope (x_t, C)

Input: (a) New observation x_t at time tick t
(b) Previous candidate solution $C = \{m, r, \mathcal{S}, \Theta, \mathcal{F}\}$
Output: Updated candidate solution C

$X_c \leftarrow X_c \cup x_t$;
/* (I) Assign x_t into an existing regime */
$\{m_S, \mathcal{S}_S, \mathcal{F}_S\} \leftarrow$ SegmentAssignment (x_t, Θ);
$C_S = \{m_S, r, \mathcal{S}_S, \Theta, \mathcal{F}_S\}$;
if $t = t_\gamma$ **then**
  /* (II) Generate new regime(s) */
  $C_R = \{m_R, r_R, \mathcal{S}_R, \Theta_R, \mathcal{F}_R\} \leftarrow$ RegimeGeneration (X_c);
  /* (III) Online update according to Equation (4) */
  if $\Delta Cost_T(X_c; C_S) < \Delta Cost_T(X_c; C_R)$ **then**
    for $i = 1 : m_S$ **do**
      $w \leftarrow f(i) \in \mathcal{F}_S$; $\{t_s, t_e\} \leftarrow s_i \in \mathcal{S}_S$;
      $\theta_w \leftarrow$ ModelUpdate ($X_c[t_s : t_e]$, θ_w);
    end for
    $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{S}_S$; $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{F}_S$; $m \leftarrow m + m_S$;
  else
    $w \leftarrow f(m) \in \mathcal{F}$; $\mathcal{S}_w \leftarrow \{s_i \in \mathcal{S}_R | w = f(i) \in \mathcal{F}_R\}$;
    $\theta_w \leftarrow$ ModelUpdate ($X_c[\mathcal{S}_w]$, θ_w);
    $\Theta \leftarrow \Theta \cup \Theta_R$; $r \leftarrow r + r_R - 1$;
    $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{S}_R$; $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{F}_R$; $m \leftarrow m + m_R - 1$;
  end if
  Update regime transition matrix $\Phi_{r\times r}$;
  $X_c \leftarrow X[s_m]$; $t_\gamma \leftarrow t + \gamma$;
end if
return $C = \{m, r, \mathcal{S}, \Theta, \mathcal{F}\}$;

- If StreamScope uses C_S, the regime parameters θ used in X_c are updated using their own new segments, more specifically, it incrementally computes the means, variances and probabilities for hidden states based on a series of predicted states with θ for X_c. Then, it merges new segment information $\{m_S, \mathcal{S}_S, \mathcal{F}_S\}$ with $\{m, \mathcal{S}, \mathcal{F}\}$ in C.
- If C_R is chosen, it requires renovation with regard to Θ in addition to the updates on new m_R segments. Since StreamScope monitors regime switching from the latest pattern to the next, X_c definitely includes a known dynamical pattern, which denoted by a regime θ_w in C. Therefore, the algorithm online updates θ_w using new segments $\mathcal{S}_w$ in X_c. Then, it employs the other regimes in Θ_R as members of Θ.

In both cases, the transition matrix Φ is also updated using the regime switching count in X_c.

5.4.1 Theoretical analysis. Let n be the length of a subsequence X_c and k be the maximum number of hidden states in the regimes Θ, i.e., $k = \max\{k_i\}_{i=1}^{r}$.

Lemma 5.2. *The time complexity of StreamScope is $O(drk^2 + dr^2k)$ at per time tick.*

Proof. StreamScope first runs SegmentAssignment, which updates the likelihood from all the k previous states in a regime and all the other $r - 1$ regimes for each regime and dimension. The inner state transition in a regime and regime switch require $O(dk^2)$ and $O(drk)$, respectively, thus the total time complexity is $O(drk^2 + dr^2k)$. If the algorithm estimates regime parameter θ, it iteratively performs the Baum-Welch algorithm and calculation of

the coding cost in $\boldsymbol{X}_c$. This process requires $O(\#iter \times n \times dk^2)$, where $\#iter$ is the number of iterations for RegimeGeneration. Since $\#iter$ and n are small values, which are negligible. Thus, the complexity of StreamScope is $O(drk^2 + dr^2k)$. □

Lemma 5.3. *StreamScope requires $O(drk + rk^2 + r^2)$ space per time tick.*

Proof. To maintain the cumulative likelihood for all r regimes, StreamScope maintains two arrays of $r \times k$ numbers for a single trellis structure. This computation also needs $r \times (k + k^2 + 2kd) + r^2$ space to maintain the parameters in $\boldsymbol{\Theta}$. StreamScope also requires $n \times d$ space for $\boldsymbol{X}_c$ to estimate new regime(s) $\boldsymbol{\theta}$ but the space is considered a small constant. Thus, the total memory required for StreamScope is $O(drk + rk^2 + r^2)$. □

These lemmas show that StreamScope scales quadratically with respect to the numbers r and k. However, from the experiments in Section 6, we will learn that this bound is exaggerated. In practice the numbers stay relatively small because they are decided so that they compress the data well through the effect of our proposed coding scheme Equation (4). The runtime of StreamScope therefore stays approximately constant, and is independent of the length of the data stream $\boldsymbol{X}$.

6 EXPERIMENTS

In this section, we describe the experiments we undertook to evaluate the performance of our proposed method, StreamScope, with real data streams. We designed the experiments to answer the following questions:

(1) *Q1. Effectiveness*: How successful is our method in discovering typical patterns in given input streams?
(2) *Q2. Accuracy*: How well does our method detect the cut points and regimes?
(3) *Q3. Scalability*: How does our method scale in terms of computational time?

We conducted our experiments on an Intel Xeon E5 2.7GHz with 64GB memory, running Linux. To learn initial model $\boldsymbol{\Theta}$ of StreamScope, we applied RegimeGeneration to 10% of an input data stream. We used the following three datasets.

#MoCap was obtained from the CMU motion capture database[4], which consists of 64-dimensional vectors. We used a set of 20 streams, containing about $m = 100$ segments in total. Each stream includes from 4 to 8 patterns. In our setting, we chose four dimensions corresponding to left-right arms and legs.

#Bicycle is a set of measurements from a 3-axis accelerometer fixed to a bicycle handle. The values are 25Hz and we use the absolute values for the frontal (i.e., z-axis) measurements. This dataset contains 10 streams with about $m = 200$ segments that we labeled with five driving patterns including going straight, turning and stopping on paved/unpaved roads.

#Workout consists of 7-dimensional streams of unified 3-axis acceleration and 4-dimensional EMG collected at 25Hz with an armband for smart gesture control, considering the task of distinguishing four kinds of calisthenics: arm curl, rowing, side raise and push up, and intervals between them. We used a set of 10 streams, containing about $m = 100$ segments.

The values of each dataset were normalized so that they had the same mean and variance (i.e., z-normalization).

6.1 Q1: Effectiveness

We first demonstrate the effectiveness of the proposed method in terms of capturing intuitive patterns using real datasets.

Motion sensor stream analysis. Here we show that how effectively StreamScope works on a real motion sensor stream. Figure 2 (a), (b) and (c) show outputs of our method for three streams: "*Exercise*", "*Boxing*" and "*Basketball*", respectively.

Figure 2 (a) consists of four kinds of regimes related to exercise such as walking and running. Initially, the algorithm employs regime #1 for walking and starts monitoring the data stream. If it is no longer able to capture $\boldsymbol{X}$ with only existing regime, and the dynamics have changed significantly (e.g., segment #1→#2 and segment #4→#5), it adopts the segmentation by RegimeGeneration and increases the number of regimes to maintain. Otherwise, it adopts SegmentAssignment to assign new observations into a suitable regime, then updates the regime parameters incrementally. Finally, StreamScope successfully found six segments (i.e., $m = 6$) and four regimes (i.e., $r = 4$), which corresponded to meaningful human activities. Our method also successfully divides the two event streams, Figure 2 (b) and (c), into their own unique motions such as "*punching*" and "*dribble*". Most importantly, our method needs no proior knowledge of dynamical patterns in data streams.

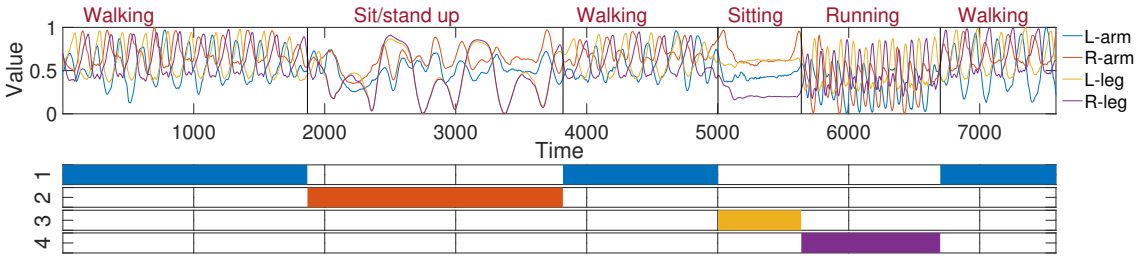
Driving pattern recognition. Figure 3 shows the original sensor stream and our online segmentation result using the *#Bicycle* dataset. StreamScope estimates regime #1/#2 for stable driving and regime #3/#4 for bumpy driving on a gravel road. Our method can automatically and precisely identify such typical regime patterns, and also find the sudden event "stopping". Note that our segmentation and pattern recognition are independent of the occurrence frequency of dynamical patterns. It is beneficial that our method can deal with unexpected situations such as the traffic accidents and heart attacks.

Analysis of fitness tracking streams. Lastly, Figure 4 shows the analytical result for the *#Workout* data stream. In this realistic scenario, users introduce their own intervals between four training motions, thus our approach can robustly recognize all the main motions (regime #1/#2/#4/#5) as well as extract the other subsequences assigned to regime #3. As shown in the figure, our method requires no user intervention to handle variable length patterns, which expands its applicability of our method.

6.2 Q2: Accuracy

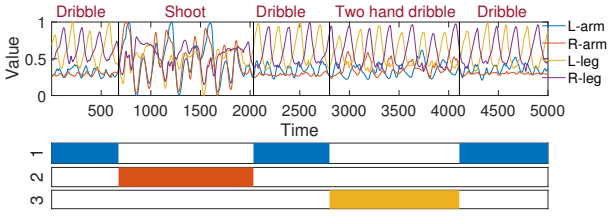
We first evaluate the quality of our method in terms of segmentation and clustering accuracy. We compare our method with pHMM [31], AutoPlait [20] and TICC [11], which are offline algorithms but state-of-the-art for simultaneous segmentation and clustering of time series. Note that pHMM requires two parameters, ϵ_r and ϵ_c, which correspond to the fitting error threshold. Therefore, we varied the two parameters from 0.1 to 10.0 to tune the quality of its segments and clusters. For tuning TICC, we varied the smoothness penalty $\beta = 0, 1, \ldots, 2000$, and set the sparsity level in the model

[4] http://mocap.cs.cmu.edu/



(a) *Exercise*

(b) *Boxing*



(c) *Basketball*

Figure 2: StreamScope is efficient: StreamScope can incrementally find regime switching (e.g., "stand up"→"walking", "sitting"→"running" in "Exercise" data stream) for a wide variety of motion data streams. It retains the latest pattern and realises its dynamics has changed in real time. All the patterns it found are compressed into a compact model, which are updated and generated while online processing.

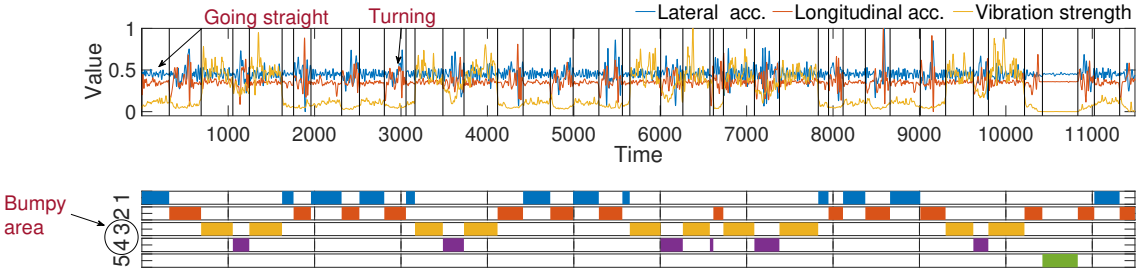


Figure 3: Driving pattern discovery of StreamScope for bicycles: Given the 3-dimensional sensor stream (top), our method finds "going straight" and "turning" on a paved/unpaved road (i.e., regimes #1/#3 and #2/#4, respectively). The approach incrementally generates regime #5 for the pattern "stopping".

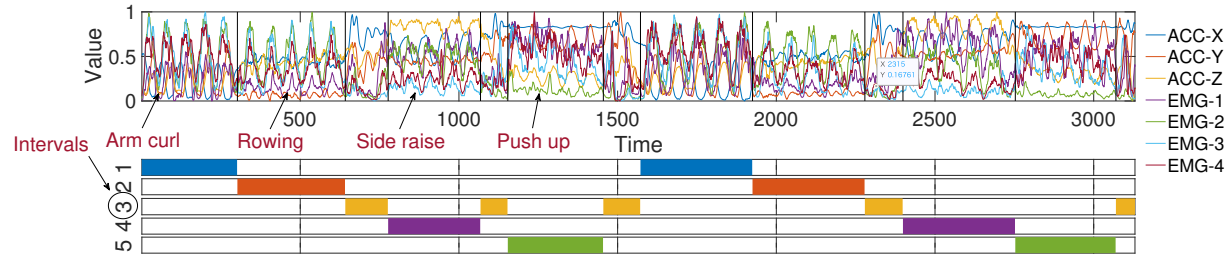


Figure 4: StreamScope can efficiently find important user activities in order, i.e., four kinds of training motions from a high dimensional data stream including 3-axis acceleration (i.e., ACC) and biological information (i.e., EMG). The algorithm also points out another pattern regime #3 (i.e., intervals) automatically.

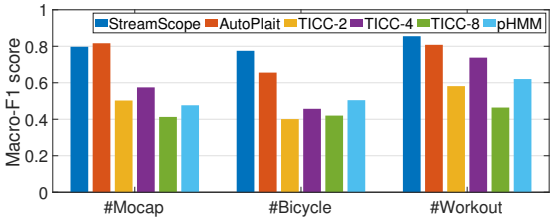


Figure 5: Segmentation accuracy with respect to macro-F_1 score: our online method performs as well as the other offline methods (higher is better).

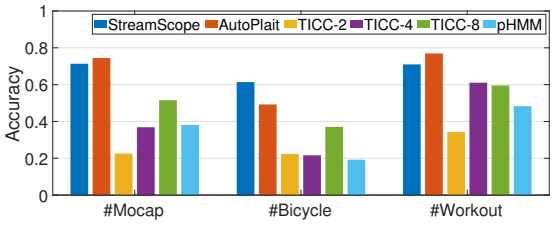


Figure 6: Clustering accuracy of StreamScope: the number of true regime labels correctly inferred with respect to the total number of observations (higher is better).

$\lambda = 10$ and window size $w = 5$. Moreover, since TICC needs to specify the number of clusters K, we also varied the numbers between 2, 4 and 8, namely TICC-K, to test its parameter sensitivity.

Segmentation accuracy. Figure 5 shows the segmentation accuracy with respect to the macro-F_1 score, which is the average of the F_1 scores for all data streams. The F_1 score is given by harmonic mean of the precision/recall defined as the ratio of reported correct cuts vs. the total number of reported/correct cuts, respectively. Overall, our proposed method outperforms the other offline methods because it can capture high-level patterns, i.e., regimes, regardless of its online approach thanks to our effective updating strategy for data stream summarization.

Clustering accuracy. Next, Figure 6 shows the clustering performance using the measure of accuracy, ACC, which is computed from the confusion matrix, CM, for prediction regime labels against true regime labels: $ACC = \sum_{i=1} CM_{ii} / \sum_{i,j=1} CM_{ij}$. The ideal confusion matrix will be diagonal, in which case $ACC = 1$. Given the incorrect number of clusters, the TICC algorithm misses important patterns, especially for *MoCap* data streams because of their variation in the number between streams. Thus, our automatic approach is effective to find multiple sequential patterns in the realistic situation where the number of patterns in streams is unknown and variable.

6.3 Q3: Scalability

We also compare the wall clock time of StreamScope with those of AutoPlait, TICC, and pHMM. The left of Figure 7 indicates the wall clock time of an experiment performed on a large *MoCap* dataset with four dimensions and various sequence lengths t. Note that the computation time of our method suddenly increases. This is because StreamScope updates the model parameter set as needed, however, it is significantly faster than its competitors. When our

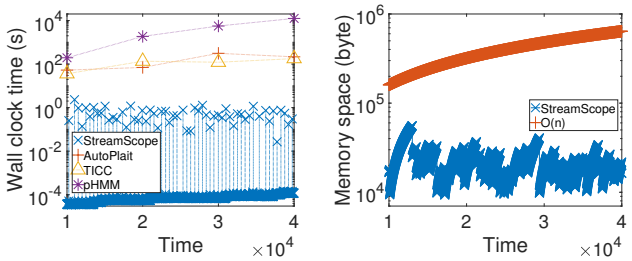


Figure 7: Scalability of StreamScope: (left) Wall clock time vs. sequence length t. it consistently surpasses its competitors. (right) Memory space consumption vs. sequence length t. it can continuously model and identify time-evolving patterns in data streams with a small space.

method does not need to update the parameters, it is also the fastest algorithm, (i.e., up to four orders of magnitude), since the likelihood computation scales constantly with the number of regimes and each state. Overall, our online algorithm is much faster than the full-batch algorithm.

The right of Figure 7 shows the space requirements of StreamScope. The requirement of our method depends on the number of regimes r and its inner states k. On the other hand, as the number of samples increases, the requirement of competitors begins to diverge to infinity; the complexity of the other competitors depends on data size n. As we expected, StreamScope only needs approximately constant space to keep track of the regime transitions, which constituted a large improvement.

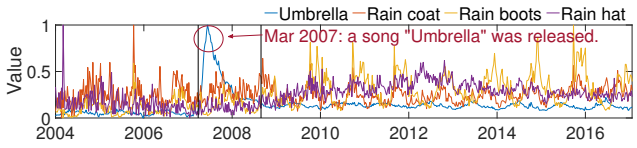
7 APPLICATIONS

In this section, we present an example of pattern discovery with our method as regards web-click activities, i.e., *GoogleTrends*[5]. The dataset we use is a multi-dimensional stream on related queries, and each dimension consists of the weekly search volumes for the query on Google over thirteen years.

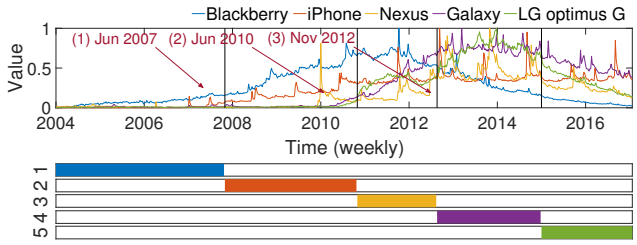
Event detection. Figure 8 (a) shows a $d = 4$ dimensional stream. Each dimension represents keywords on rain gear (i.e., umbrellas, raincoats, rain boots and rain hats). "Umbrella" is searched for throughout the year and the others exhibit yearly periodicity because of their seasonality. In March 2007, the search count for umbrella suddenly increased caused by an abnormal event, namely, that a song named "Umbrella" was released. StreamScope successfully detected this event as regime #2.

Trend discovery. Figure 8 (b) shows our result for another data stream related to some major smartphones (e.g., iPhone and Nexus). StreamScope captures the following key trends in relation to the smartphone industry caused by some latent events with the observed stream: (#1) Blackberry was used among businessmen as a pioneering smartphone, (#2) iPhone impacted and its market grew when it entered the market in June 2007, (#3) Android smartphones represented by the Nexus and Galaxy series entered the market from 2010, (#4) The competitive relationship between iOS and Android, and (#5) represents the recent global share of smartphones.

[5]https://trends.google.com/trends/



(a) *"Rain gear"*



(b) *"Smartphone"*

Figure 8: Application to real web data: StreamScope spots meaningful changeovers on *GoogleTrends*. (a) Rain gear-related keywords: it discovers one sudden event in 2007 (i.e., a hit song); (b) smartphone-related keywords: it finds five phases of smartphone market share (i.e., #1: Blackberry, #2: iPhone, #3: Android, #4: iOS vs Android, #5: Recent share).

8 CONCLUSIONS

In this paper, we have proposed a streaming algorithm, StreamScope, for automatic pattern discovery in multi-dimensional streams. As the processing progresses, our algorithm partitions time-series streams into a set of segments at points where characteristics change, and then similar segments are grouped into a regime. The advantages of StreamScope are as follows:

- It is **Effective:** our experiments with several datasets show that StreamScope successfully finds segments/regimes that match human intuition.
- It is **Automatic:** it can handle arbitrary semi-infinite data streams, and automatically maintain the number r of regimes by exploiting our novel coding scheme.
- It is **Scalable:** the computational time does not depend on the length of the input streams.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous referees for their valuable comments and helpful suggestions. This work was supported by JSPS KAKENHI Grant-in-Aid for Scientific Research Number JP19J11125, JP18H03245, JP17H04681, JP16K12430, PRESTO JST.

REFERENCES

[1] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. 2003. A Framework for Clustering Evolving Data Streams *(VLDB)*. 81–92.
[2] Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer.
[3] George E.P. Box, Gwilym M. Jenkins, and Gregory C. Reinsel. 1994. *Time Series Analysis: Forecasting and Control* (3rd ed.). Prentice Hall, Englewood Cliffs, NJ.
[4] Feng Cao, Martin Estert, Weining Qian, and Aoying Zhou. 2006. Density-based clustering over an evolving data stream with noise. In *SDM*. SIAM, 328–339.
[5] Joel Janek Dabrowski, Ashfaqur Rahman, Andrew George, Stuart Arnold, and John McCulloch. 2018. State Space Models for Forecasting Water Quality Variables: An Application in Aquaculture Prawn Farming. In *KDD*. 177–185.
[6] Ian Fox, Lynn Ang, Mamta Jaiswal, Rodica Pop-Busui, and Jenna Wiens. 2017. Contextual Motifs: Increasing the Utility of Motifs Using Contextual Data. In *KDD*. 155–164.
[7] Mohammed Ghesmoune, Mustapha Lebbah, and Hanene Azzag. 2016. State-of-the-art on clustering data streams. *Big Data Analytics* 1, 1 (2016), 13.
[8] Shufeng Gong, Yanfeng Zhang, and Ge Yu. 2017. Clustering Stream Data by Exploring the Evolution of Density Mountain. *PVLDB* 11, 4 (2017), 393–405.
[9] Peter D Grünwald, In Jae Myung, and Mark A Pitt. 2005. *Advances in minimum description length: Theory and applications*. MIT press.
[10] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Gener. Comput. Syst.* 29, 7 (2013), 1645–1660.
[11] David Hallac, Sagar Vare, Stephen Boyd, and Jure Leskovec. 2017. Toeplitz Inverse Covariance-Based Clustering of Multivariate Time Series Data. In *KDD*. 215–223.
[12] Ahsanul Haque, Zhuoyi Wang, Swarup Chandra, Bo Dong, Latifur Khan, and Kevin W. Hamlen. 2017. FUSION: An Online Method for Multistream Classification. In *CIKM*. 919–928.
[13] Eamonn J. Keogh, Selina Chu, David Hart, and Michael J. Pazzani. 2001. An Online Algorithm for Segmenting Time Series. In *ICDM*. 289–296.
[14] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. 2007. Trajectory clustering: a partition-and-group framework. In *SIGMOD*. 593–604.
[15] Lei Li, B. Aditya Prakash, and Christos Faloutsos. 2010. Parsimonious Linear Fingerprinting for Time Series. *PVLDB* 3, 1 (2010), 385–396.
[16] Zitao Liu and Milos Hauskrecht. 2017. A Personalized Predictive Framework for Multivariate Clinical Time Series via Adaptive Model Selection. In *CIKM*. 1169–1177.
[17] Alysha M De Livera, Rob J Hyndman, and Ralph D Snyder. 2011. Forecasting time series with complex seasonal patterns using exponential smoothing. *J. Amer. Statist. Assoc.* 106, 496 (2011), 1513–1527.
[18] Michael Mathioudakis, Nick Koudas, and Peter Marbach. 2010. Early online identification of attention gathering items in social media. In *WSDM*. 301–310.
[19] Yasuko Matsubara and Yasushi Sakurai. 2019. Dynamic Modeling and Forecasting of Time-evolving Data Streams. In *KDD*. 458–468.
[20] Yasuko Matsubara, Yasushi Sakurai, and Christos Faloutsos. 2014. AutoPlait: Automatic Mining of Co-evolving Time Sequences. In *SIGMOD*.
[21] Yasuko Matsubara, Yasushi Sakurai, and Christos Faloutsos. 2015. The Web as a Jungle: Non-Linear Dynamical Systems for Co-evolving Online Activities. In *WWW*.
[22] Yasuko Matsubara, Yasushi Sakurai, Naonori Ueda, and Masatoshi Yoshikawa. 2014. Fast and Exact Monitoring of Co-Evolving Data Streams. In *ICDM 2014*. 390–399.
[23] Yasuko Matsubara, Yasushi Sakurai, Willem G. van Panhuis, and Christos Faloutsos. 2014. FUNNEL: automatic mining of spatially coevolving epidemics. In *KDD*. 105–114.
[24] Gianmarco De Francisci Morales, Albert Bifet, Latifur Khan, Joao Gama, and Wei Fan. 2016. IoT Big Data Stream Mining. In *KDD, Tutorial*. 2119–2120.
[25] Abdullah Mueen and Eamonn J. Keogh. 2010. Online discovery and maintenance of time series motifs. In *KDD*. 1089–1098.
[26] Spiros Papadimitriou and Philip S. Yu. 2006. Optimal multi-scale patterns in time series streams. In *SIGMOD*. 647–658.
[27] Tobias Preis, Helen Susannah Moat, and H. Eugene Stanley. 2013. Quantifying Trading Behavior in Financial Markets Using Google Trends. *Sci. Rep.* 3 (04 2013).
[28] Yasushi Sakurai, Yasuko Matsubara, and Christos Faloutsos. 2015. Mining and Forecasting of Big Time-series Data. In *SIGMOD, Tutorial*. 919–922.
[29] Mohammad Shokoohi-Yekta, Yanping Chen, Bilson Campana, Bing Hu, Jesin Zakaria, and Eamonn Keogh. 2015. Discovery of Meaningful Rules in Time Series. In *KDD*. 1085–1094.
[30] Machiko Toyoda, Yasushi Sakurai, and Yoshiharu Ishikawa. 2013. Pattern discovery in data streams under the time warping distance. *VLDB J.* 22, 3 (2013), 295–318.
[31] Peng Wang, Haixun Wang, and Wei Wang. 2011. Finding semantics in time series. In *SIGMOD*. 385–396.
[32] Byoung-Kee Yi, N.D. Sidiropoulos, Theodore Johnson, H.V. Jagadish, Christos Faloutsos, and Alexandros Biliris. 2000. Online Data Mining for Co-Evolving Time Sequences. *ICDE* (2000), 13–22.
[33] Yunyue Zhu and Dennis Shasha. 2002. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. In *VLDB*. 358–369.
[34] Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. 2014. Indexing for interactive exploration of big data series. In *SIGMOD*. 1555–1566.