



CLaP - State Detection from Time Series

Arik Ermschhaus

Humboldt-Universität zu Berlin
Berlin, Germany
ermshaua@informatik.hu-berlin.de

Patrick Schäfer

Humboldt-Universität zu Berlin
Berlin, Germany
patrick.schaefer@hu-berlin.de

Ulf Leser

Humboldt-Universität zu Berlin
Berlin, Germany
leser@informatik.hu-berlin.de

ABSTRACT

The ever-growing amount of sensor data from machines, smart devices, and the environment leads to an abundance of high-resolution, unannotated time series (TS). These recordings encode recognizable properties of latent states and transitions from physical phenomena that can be modelled as abstract processes. The unsupervised localization and identification of these states and their transitions is the task of time series state detection (TSSD). Current TSSD algorithms employ classical unsupervised learning techniques, to infer state membership directly from feature space. This limits their predictive power, compared to supervised learning methods, which can exploit additional label information. We introduce CLaP, a new, highly accurate and efficient algorithm for TSSD. It leverages the predictive power of time series classification for TSSD in an unsupervised setting by applying novel self-supervision techniques to detect whether data segments emerge from the same state. To this end, CLaP cross-validates a classifier with segment-labelled subsequences to quantify confusion between segments. It merges labels from segments with high confusion, representing the same latent state, if this leads to an increase in overall classification quality. We conducted an experimental evaluation using 405 TS from five benchmarks and found CLaP to be significantly more precise in detecting states than six state-of-the-art competitors. It achieves the best accuracy-runtime tradeoff and is scalable to large TS. We provide a Python implementation of CLaP, which can be deployed in TS analysis workflows.

PVLDB Reference Format:

Arik Ermschhaus, Patrick Schäfer, and Ulf Leser. CLaP - State Detection from Time Series. PVLDB, 19(1): 70 - 83, 2025.
doi:10.14778/3772181.3772187

PVLDB Artifact Availability:

The source code, data, and/or other artifacts have been made available at <https://github.com/ermshaua/classification-label-profile>.

1 INTRODUCTION

The study and analysis of long-running biological, human-controlled, or physical processes, such as human activities or industrial manufacturing, is of great interest in their respective domains. The field of human activity recognition, for instance, aims to detect falls in the elderly [71]. To achieve this, human motions are conceptualized as an abstract process of distinct activities that transition

from one to another. Data acquisition and analysis workflows are employed to detect activities from mobile sensors and report falls. Similarly, in industrial manufacturing, CNC machines are monitored by pre-installed IoT devices to detect tool wear [60]. The condition of a tool can be abstracted as either normal or faulty, and wear detection is realized by recognising the transitions in-between. From a computational perspective, such phenomena can be modelled as abstract processes with discrete states and pairwise transitions between them. The core property of these processes is that the states are distinct and can be observed and measured over time by instrumentation.

Recent advances in the digitalization of measurement devices have greatly facilitated the acquisition of biological and physical process data. Sensors, such as those in smartphones, industrial machinery, or earth observation stations, continuously record high-frequency real-valued observational data, termed *time series* (TS) [34, 43, 69]. State changes in the observed processes lead to variations in the recorded signals, which form the basis for analysis. Sensor data capture the unique characteristics of states as subsequences, distinguishable by either their shape or statistical properties. These distinct parts of TS are commonly called *segments*, and each contains the recognizable properties of the observed process state. The transitions between segments, also called *change points* (CPs), mark the time points in a recording, at which the observed process changes state. The identification of segments, CPs, and state labels forms the foundation for activity recognition [18], health assessment [38], or condition monitoring in IoT [60].

Formally, the task of recovering the sequence of discrete state labels from a TS of observations is called time series state detection (TSSD). It acts as a complex, unsupervised preprocessing step between data collection and TS knowledge discovery [65]. TSSD annotates each data point in a TS with a label that corresponds to a state in the data-generating process. Consecutive stretches of the same label mark segments, while different neighbouring labels indicate CPs. TSSD requires the analysis of observations to identify signal shifts, which are assumed to result from state changes in the observed process. This creates consecutive segments of data points that are homogeneous within themselves, yet sufficiently distinct from their neighbours. These segments are further compared to each other to assign equal labels to those sharing the same state and unequal labels to those representing different states. State labels from segments are then propagated to individual data points to form the annotation, a state sequence. Contrary to supervised problems, such as TS classification (TSC), the labels in TSSD (e.g. 0,1,2) are discovered by an algorithm, abstract, and primarily used to differentiate observations based on state affiliation. They can be mapped to semantic labels (e.g. walk, jog, run) when appropriate domain knowledge is available [2].

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.

Proceedings of the VLDB Endowment, Vol. 19, No. 1 ISSN 2150-8097.
doi:10.14778/3772181.3772187

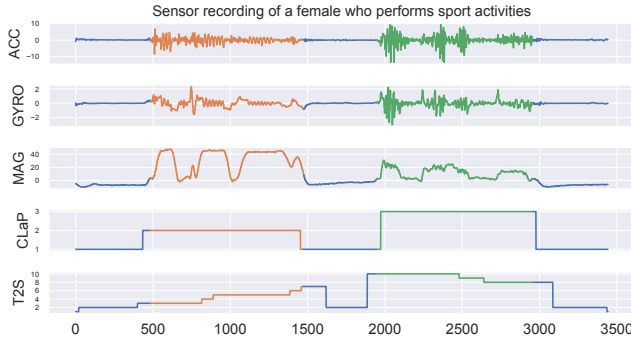


Figure 1: Top-3: human activity recording showing X-axis acceleration, gyroscope, and magnetometer measurements [18] of different motions. Activities are consecutive, differently coloured sequences. 2nd from bottom: state sequence predicted by CLaP, assigning each data point one of three possible activity labels (1 for resting, 2 for squats, and 3 for loosening legs). Bottom: state sequence from competitor method Time2State, with 10 (too many) predicted labels.

A popular approach to TSSD is a combination of TS segmentation (TSS) and clustering techniques [65]. A baseline approach could, for example, use the ClaSP (Classification Score Profile) algorithm [20] to compute a TSS and cluster the resulting segments using Time2Feat [6] to predict state labels. The central limitation of this pipeline is its reliance on distance calculations in clustering, which are known to show limited performance compared to supervised methods used in TSC [49]. Unsupervised distance-based methods determine the state membership of segments solely within the feature space, whereas supervised techniques could also utilize the label space in this computation. This issue prevails to all state-of-the-art methods [28, 33, 42, 47, 66].

We present CLaP (Classification Label Profile), a domain-agnostic TSSD algorithm that uses self-supervised change analysis to leverage the predictive power of supervised classification algorithms to the unsupervised TSSD problem. It is accurate, scalable, and capable of processing both univariate and multivariate TS. CLaP is hyper-parameter-free and learns all information from the data at hand. The routine first computes a segmentation with ClaSP, which internally uses a binary self-supervised classifier to differentiate segments based on differently shaped subsequences. CLaP then cross-validates a multi-class classifier with segment-labelled subsequences to measure the confusion between segments. This information is central to labelling segments that share the same state or not. In a subsequent merging procedure, the algorithm iteratively combines confused labels, representing the same state, using a novel merge criterion called classification gain. By design, CLaP automatically determines the number of states in a TS; an otherwise difficult to set hyper-parameter.

Unlike competing approaches, which use classical unsupervised techniques, CLaP is the first method to tackle the TSSD problem with supervised learning methods, by explicitly framing it as a self-supervised task. Segments are modelled directly in label space, which is more discriminative than feature space, used in earlier

algorithms. CLaP introduces a novel merging procedure that learns states by fusing segment labels according to their mutual confusion. This process is guided by a new score that enables fair comparisons across classification problems of differing complexity.

Figure 1 shows an example of how CLaP and a state-of-the-art competitor Time2State [66] annotate a multivariate time series (MTS) with a state sequence. The recording captures smartphone sensor readings of a 30-year-old female resting (blue), performing squats (orange), resting again (blue), loosening her legs (green), and resting again (blue) [18]. CLaP accurately identifies the transitions between activities and correctly assigns 3 distinct labels to the 5 segments (segments with the same activity are visualized as equally high flat lines; ones with different activities appear as lines at varying levels in the visualization). Time2State, in comparison, correctly identifies the resting periods, but overestimates the total number of activities being 10 separated by 12 transitions.

In summary, this paper’s contributions are:

- (1) We present CLaP, a new domain-agnostic and hyper-parameter-free TSSD algorithm that leverages self-supervised TS classification for segmentation and labelling to predict state sequences of multivariate TS. We provide technical descriptions, a computational complexity analysis, examples, and a Python implementation.
- (2) We propose two technical novelties that make TS classifiers applicable for TSSD: confused merging, a mechanism to reduce a TS state labelling to a minimal required set of labels; and classification gain, which quantifies the increase in F1 score of a classification compared to a random one.
- (3) We assess the accuracy and runtime of CLaP and six state-of-the-art competitors (Time2State, HDP-HSMM, E2USD, ClaSP2Feat, TICC, AutoPlait) on 405 TS from five benchmarks. CLaP significantly outperforms all rivals in accuracy and has the best accuracy-runtime tradeoff. It achieves the highest mutual information of 60.4% with ground truth annotations, an increase of 16.1 percentage points (pp) compared to the second-best competitor Time2State.

To foster the reproducibility of our findings, we created a supporting website [11] that contains all source codes, our evaluation framework, Jupyter notebooks for exploratory analysis, raw measurements, scores, and visualizations. The remaining paper is organized as follows: Section 2 introduces the necessary definitions and background of this work. In Section 3, we present CLaP and its technical novelties in detail. Section 4 shares the results of extensive accuracy and runtime experiments of CLaP and the competitors. In Section 5, we review related works, and Section 6 concludes.

2 DEFINITIONS AND BACKGROUND

We formally define the concepts of abstract processes, time series, subsequences, state sequences, the time series state detection (TSSD) problem, and introduce ideas of TS change analysis.

DEFINITION 1. An abstract process $P = (S, R)$ consists of one or more discrete and distinct states $s_1, \dots, s_l \in S$ that are pairwise separated by transitions $(s_i, s_j) \in R \subseteq S \times S$, with $s_i \neq s_j$.

Following the definition of Wang et al. [65], states refer to distinct phases of real-world processes observable through sensor

measurements. We assume each state has a stationary, recognizable property that persists over time and makes it distinguishable from other states. Examples include human activities or machine conditions. This definition excludes processes with homonym states, that emit indistinguishable signals, and synonym states, which cannot be recognized by observation. It also excludes states that have drifting semantics over time.

Transitions are, by definition, changes between states and constitute the links among them in processes. We assume that a transition leads to a change in the observations of a process. Gradual changes, such as trends, can be modelled as separate states. We make no assumptions about transition causes, i.e., whether they occur randomly or systematically.

For analysis, we consider measurements emitted by one or multiple sensors observing outcomes or byproducts of a process. Human activity, for instance, can be tracked by inertial measurement units (IMUs) in smartphones [43] and industrial machinery can be monitored with IoT devices [64], which results in temporal data.

DEFINITION 2. A time series (TS) T is an ordered sequence of $n \times d \in \mathbb{N}^2$ dimensional vectors $T = (\vec{t}_1, \dots, \vec{t}_n) \in \mathbb{R}^{n \times d}$ that simultaneously measures d observable outputs of a process P .

Each t_i has d dimensions, or channels, one for each sensor. Its values, also called data points or measurements, are equi-distant and ordered by time, e.g. 1 data vector is recorded every 10 milliseconds.

DEFINITION 3. Given a TS T , a subsequence $T_{s,e}$ of T with start offset s and end offset e is the d -dimensional slice of contiguous observations from T at position s to position e , i.e., $T_{s,e} = (\vec{t}_s, \dots, \vec{t}_e)$ with $1 \leq s \leq e \leq n$. The length of $T_{s,e}$ is $|T_{s,e}| = e - s + 1$.

We use the terms *subsequence* and *window* interchangeably, and refer to their length as the *width*. A state from S yields its observable properties as a subsequence with shapes or statistics distinguishable from others. We call these core structures *temporal patterns*, since they provide the necessary information to recognize the same state and distinguish it from others. Temporal patterns can drift or suddenly change over time, indicating a switch from one process state to another. Note, however, that local parts of channels contribute differently to the signal, for instance in amplitude.

DEFINITION 4. Given a TS T of size n that captures a process $P = (S, R)$, the corresponding state sequence $Q = (s_{t_1}, \dots, s_{t_n}) \in S^n$ contains the states $s_{t_i} \in S$ that are captured at time points $t_i \in T$.

A *change point* (CP) denotes an offset $i \in [1, \dots, n]$ for $t_i \in T$ that corresponds to a transition between states $s_{t_{i-1}}$ to s_{t_i} in Q , where $(s_{t_{i-1}}, s_{t_i}) \in R$ and $s_{t_{i-1}} \neq s_{t_i}$. For notational convenience, we consider the first and last values in T as CPs. We call the subsequence between two CPs a *segment* with variable size. A *segmentation* of T is the ordered sequence of CPs in T , i.e., t_{i_1}, \dots, t_{i_n} with $1 \leq i_1 < \dots < i_n \leq n$ at which the process P changes state.

DEFINITION 5. The problem of time series state detection (TSSD) is to recover the latent state sequence Q of a process P , only by analysing the time series T , emitted by P .

The TSSD problem, as defined here, is unsupervised and twofold: We need to find a segmentation of T that captures the state transitions R , as well as the distinct states S from all segments, in order

to predict a state sequence \hat{Q} . An optimal result yields a \hat{Q} that is isomorphic to Q , because an algorithm does not have access to the actual state labels. To evaluate the quality of a predicted state sequence, we can measure its alignment with the ground truth annotated by domain experts — e.g., by inspecting Covering [63] or the mutual information [51] of segments.

2.1 Self-supervised Change Analysis

To label TS with their states, CLaP computes segments of T and iteratively relabels them based on mutual similarity. Contrary to classical clustering approaches, we do not compare distances, but build on self-supervised TS change analysis, first proposed by Hido et al. [29], that we shortly introduce next.

Self-supervised learning is an unsupervised learning variant, in which the data itself is used to generate supervision labels. Consider two data sets, X_A and X_B , representing subsequences from different segments. We assign label 0 to samples from X_A and 1 to ones from X_B , enabling a binary classification evaluation using cross-validation. A TS classifier, trained on the labelled subsequences, predicts labels for unlabelled instances. k -fold cross-validation evaluates the classifier by training it on $(k - 1)$ parts of the data and testing it on the remaining one. This process repeats k times, covering all combinations, with the average of the k evaluation scores (e.g., F1-scores) representing the classifier’s predictive power. This value measures the classifier’s ability to distinguish between data sets X_A and X_B . A high score implies high dissimilarity and unique characteristics between the segments, indicating they represent different states. Conversely, a lower score indicates similarities, suggesting the segments may belong to the same state.

3 CLAP - CLASSIFICATION LABEL PROFILE

We propose the *Classification Label Profile*, short *CLaP*, a novel algorithm that formulates TSSD as a self-supervised classification problem. We first provide an overview and example of the main concept, before we explain in detail how to implement it for state detection in Subsections 3.1 to 3.3.

DEFINITION 6. Consider a process $P = (S, R)$, captured by a TS T of size $|T| = n$, and a window size w . A CLaP is a tuple (L, c) that annotates T with a label sequence $L \in \{1, \dots, k\}^{n-w+1}$ and an associated cross-validation score $c \in [0, 1]$.

The label sequence L links all the $n - w + 1$ overlapping subsequences $T_{i,i+w-1}$ with one of k labels, which are used to represent the states in S . CLaP is computed from a subsequence classification problem with k classes, whose discriminatory power is summarised in the score c . We derive L from a self-supervision mechanism (see Subsection 3.1) and calculate c as the F1-score from a 5-fold cross-validation with the ROCKET classifier [12], using the subsequences from T as data and L as artificial ground truth labels. Low scores of c indicate CLaP is not able to accurately differentiate similar from dissimilar subsequences, while high scores show that it constitutes distinguishable windows.

Figure 2 shows an example of three different CLaPs. The top part illustrates a triaxial human activity recording from a 23-year-old male, switching between horizontal (blue) and downstairs (orange) walking. The true state sequence of this recording assigns, for example, 1s to the blue data points and 2s to the orange measurements.

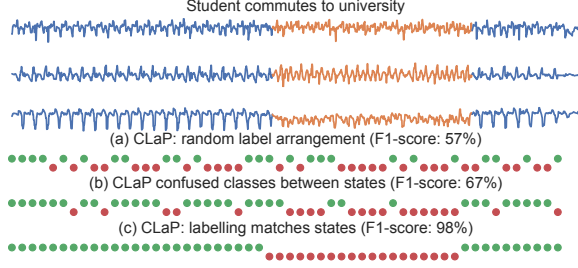


Figure 2: Top: excerpt of triaxial mobile phone acceleration recording from male student commuting to university, showing horizontal (blue) and downstairs (orange) walking motions [18]. 4th from top to bottom: three CLaPs with different labellings (green and red) and associated F1 cross-validation scores. Profiles (a to c) incrementally align more with the true state sequence, which leads to increasing F1 scores.

The bottom part displays three different CLaPs (a to c). The sequence of green and red dots shows the label configuration L , with the associated F1-score c annotated. CLaP (a) presents a random label configuration that does not match the true state sequence, resulting in poor cross-validation performance (57% F1-score). The profile in (b) somewhat aligns with Q but confuses some instances (67% F1-score). In contrast, CLaP in (c) perfectly resembles the true state sequence of the human activity recording, leading to a near-optimal performance (98% F1-score).

The true state sequence Q of a TS T constitutes a CLaP that scores the highest discriminatory power compared to all possible sequence permutations. Our implicit assumptions thereby are three-fold: (1) Process states can be conceptualized as classes of TS windows, (2) subsequences (of appropriate length) can serve as class exemplars with distinctive features, (3) and a TS classifier cross-validation score correlates with the quality of the label configuration. With these concepts, we can tackle an unsupervised problem with the predictive power of TS classification (TSC) algorithms. Much research interest continues to increase accuracy and efficacy for the task (see Middlehurst et al. [49] for a survey). However, searching a high-quality state sequence prediction \hat{Q} using CLaP is non-trivial.

The naive approach of enumerating all k^{n-w+1} candidates and choosing the highest-scoring one (according to c) is computationally inefficient and yields further challenges. For instance, the number of classes k is typically not known in advance. Also, the large collection of label combinations makes the cross-validation score incomparable for different values of k . We propose efficient and accurate solutions to prune the candidates and make their scores comparable in Subsections 3.1 and 3.2, bridging the gap to enable TSC methods to be directly applicable for TSSD. Subsection 3.3 analyses the runtime and space complexity of CLaP.

3.1 Confused Merging

The TSSD problem can be divided into a segmentation followed by clustering. We use this formulation to drastically prune the amount of potential CLaPs. First, we compute a segmentation of a TS T . This enables us to label only few segments, instead of many

Algorithm 1 Classification Label Profile

```

1: procedure CLAP( $T$ )
2:    $w \leftarrow \text{LEARN\_SUBSEQUENCE\_WIDTH}(T)$  ▷ Run SuSS
3:    $cps \leftarrow \text{COMPUTE\_SEGMENTATION}(T)$  ▷ Run ClaSP
4:    $X, L \leftarrow \text{CREATE\_DATA\_SET}(T, w, cps)$ 
5:    $y_{pred} \leftarrow \text{CROSS\_VAL\_CLF}(X, L)$  ▷ 5-fold ROCKET CV
6:   for  $i \in [1, \dots, ||cps||]$  do
7:      $merged \leftarrow false$ 
8:      $labels, conf \leftarrow \text{CALC\_CONFUSED\_LABELS}(L, y_{pred})$ 
9:     for  $(l_1, l_2) \in \text{RANK}(labels, conf)$  do
10:       $\hat{L} \leftarrow \text{REPLACE}(L, l_1, l_2)$  ▷ Merge labels
11:       $\hat{y}_{pred} \leftarrow \text{REPLACE}(y_{pred}, l_1, l_2)$ 
12:      if  $\text{CGAIN}(\hat{L}, \hat{y}_{pred}) \geq \text{CGAIN}(L, y_{pred})$  then
13:         $L, y_{pred} \leftarrow \hat{L}, \hat{y}_{pred}$  ▷ Update labels
14:         $merged \leftarrow true$ 
15:        break
16:      end if
17:    end for
18:    if  $merged \neq true$  then break ▷ Early stopping
19:  end for
20:  return  $(L, \text{SCORE}(L, y_{pred}))$  ▷ Calc. F1-score
21: end procedure

```

single data points. Secondly, we propose a novel agglomerative self-supervision mechanism, that greedily clusters confused state labels with a new bespoke merge criterion. The entire process, called *Confused Merging*, combines several design choices, that we fixed using ablations (see Subsection 4.4). For each component, we considered multiple domain-agnostic methods and chose the best-performing one. CLaP is hyper-parameter-free, automatically learns the number of k classes and stops as soon as no classes must be merged. Pseudocode is provided in Algorithm 1 and the workflow is illustrated in Figure 3.

Preprocessing: Algorithm 1 receives a TS T as user input (line 1) and starts by learning its subsequence width w (line 2), needed to compute the sliding window in CLaP. Multiple techniques are available for this task and based on the idea that temporal patterns of similar size repeat throughout TS, an assumption we share in CLaP. We choose SuSS (Subsequence Summary Statistics), one of the best-performing procedures, according to [19]. For multidimensional TS, we compute one window size per dimension and then compute their average. The CLaP algorithm continues to compute a segmentation of T with the ClaSP method [20] (line 3), which can partition multivariate TS [21] from different domains [23, 70] and has an extension for very large and streaming data [22]. This preprocessing step removes the need to label individual data points in CLaP and instead only requires assigning classes to entire segments.

Self-supervised Classification: The procedure creates an initial labelled data set from T by creating a sliding window X (width of w , stride of $\frac{w}{2}$) and labelling the resulting subsequences with their corresponding segment ranks in L , i.e. label 1 for the first segment, label 2 for the second one, label 3 for the third, and so on (line 4). Subsequences that overlap neighbouring segments with at least $\frac{w}{2}$ values are disposed. The motivation behind this labelling is twofold: it encodes the CPs (from the segmentation) as a labelling

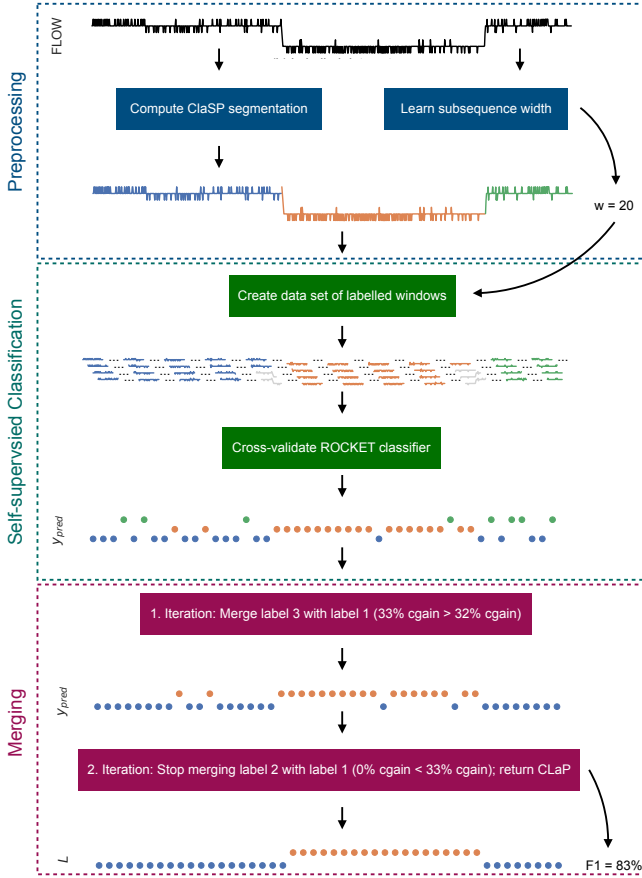


Figure 3: Workflow of CLaP for a water circuit [36]. TS captures fluid circulation flow with opened (blue and green) and closed valve (orange). Preprocessing: ClaSP segmentation divides TS into three segments, SuSS learns the subsequence width. Self-supervised Classification: labelled subsequences constitute the data set; grey ones are disposed; ROCKET classifier computes initial predicted cross-validation labels. Merging: labels are combined based on confusion as long as cgain increases. Final CLaP with artificial ground truth labels and F1-score of 83% is returned.

and first assumes all segments to capture distinct states. This enables the later merging, which fuses labels from segments sharing state. CLaP evaluates the ROCKET classifier, which is fast and accurate [4], with the labelled data set in a 5-fold cross-validation and stores the predicted labels in y_{pred} (line 5). This creates an initial measure for the quality of the labelling. Compared to the artificial ground truth L , segments representing single states are expected to mostly match. Conversely, multiple segments sharing state, are expected to confuse labels, facilitating their identification.

Merging: CLaP uses the initial labelled data set (X, L) and its cross-validated prediction labels y_{pred} to iteratively fuse segments representing the same state (lines 6–19). To achieve this, the procedure greedily merges highly confused labels, as their associated subsequences are difficult to separate. For every unique label l_1 from

L , CLaP determines its most confused counterpart l_2 , according to y_{pred} (line 8). This information is directly derived from a confusion matrix. The algorithm iterates the most confused label pairs with descending confusion to check if they can be merged (lines 9–17). Specifically, it creates temporary label vectors \hat{L} and \hat{y}_{pred} , where l_2 is replaced with l_1 (lines 10–11), and checks if their classification gain (see Subsection 3.2 for details) is greater (or equal) to the one from L and y_{pred} (line 12). In this case, the method updates the current with the merged labelling (line 13) and stops merging; otherwise, it continues with less confused label pairs. After a successful merge iteration, CLaP starts another round with updated artificial ground truth and predicted labels. Once no label pair can be further merged (line 18), the procedure stops and returns the current label configuration L and its associated cross-validation score, according to y_{pred} , constituting CLaP (line 20). The central mechanism of this process is confused label pair selection. We exploit the fact that segments sharing state are confused by the classifier, because of similar temporal patterns that are differently labelled. Hence, we use it as a selection criterion, implementing it in an agglomerative fashion, to learn the amount of states and stop as soon as possible. Classification gain (see Subsection 3.2) regularises the merging to ensure that only labels from sufficiently similar segments are fused.

Workflow: Figure 3 illustrates the confused merging algorithm for water flow circulation in a testbed [36]. The preprocessing (blue frame) divides the TS into three segments (blue, orange, green), capturing two states; namely open and closed valve. It also computes the subsequence width of 20 with SuSS. The self-supervision mechanism (green frame) creates labelled subsequences using segment ranks (1, 2, 3) and uses ROCKET to compute initial predicted cross-validation labels y_{pred} that show high confusion between label 1 and 3. Merging (purple frame) finds label 3 to have the highest overall confusion (with label 1) and fuses both, increasing classification gain (from 32% to 33%). Merging even further to one class would reduce cgain to 0%, hence the procedure stops. The resulting artificial ground truth L and its 83% F1-score constitute CLaP and correctly capture the latent states.

The procedure is hyper-parameter-free and it learns two model-parameters from the input TS: the subsequence width w and the number of classes k . By default, CLaP uses five folds for cross-validation and draws at most 1k randomly sampled, labelled subsequences from the full data set to control runtime.

3.2 Classification Gain

Confused merging relies on a merge criterion that decides if two confused classes should be combined or not. In agglomerative clustering, this is typically achieved by thresholding the distance computation. For CLaP, we could translate that into enforcing the F1-score (or e.g. cross-entropy loss) of the label configurations (Algorithm 1, line 12) to pass a minimal (maximal) value. However, this is disadvantageous for three key reasons: (a) the F1-score (also accuracy, ROC/AUC, or entropy) depends on the classifier and data; rendering threshold selection use case dependent. (b) Comparing F1-scores of classification problems with different label distributions is meaningless; more labels increase problem complexity, resulting in generally lower F1-scores. (c) The F1-score does not reflect the improvement of one labelling over another, just their total qualities.

To address these challenges, we propose a new merge criterion called *Classification Gain*, which measures the increase in F1-score over its expected random score, thereby normalizing for problem complexity. It isolates predictive power from classification difficulty, facilitating performance comparisons across different problems.

DEFINITION 7. Given a ground truth labelling $y_{true} \in U^n$ of size n , with unique classes $U \subset \mathbb{N}$, the random F1-score is defined as:

$$f1_{rand}(y_{true}) := \frac{1}{\|U\|} \sum_{l \in U} \frac{2 \cdot TP(l)}{2 \cdot TP(l) + FN(l) + FP(l)} \quad (1)$$

The random (macro) F1-score provides an unbiased estimate of classification difficulty based on the label distribution y_{true} . It can be computed efficiently using probability theory in the following way: in a random classification, we expect each instance labelled $l \in U$ to be mapped to its own (or any other) class according to its likelihood of occurrence.

$$P(y_{true} = l) := \frac{\#(y_{true} = l)}{\|y_{true}\|} \quad (2)$$

$$P(y_{true} \neq l) := 1 - P(y_{true} = l) = \frac{\#(y_{true} \neq l)}{\|y_{true}\|} \quad (3)$$

These class priors enable us to define the true positives (TPs) as the expected amount of instances that are randomly assigned to their class (Equation 4), the false negatives (FNs) as the expected number of samples that are mapped to another class (Equation 5), and the false positives (FPs) as the expected amount of misclassified examples that actually belong to a given class (Equation 6).

$$TP(l) := \#(y_{true} = l) \cdot P(y_{true} = l) \quad (4)$$

$$FN(l) := \#(y_{true} = l) \cdot P(y_{true} \neq l) \quad (5)$$

$$FP(l) := \#(y_{true} \neq l) \cdot P(y_{true} = l) \quad (6)$$

Given the random macro F1-score for y_{true} as an estimate of problem difficulty, we can decouple it from the performance of a classification to make it comparable across problems.

DEFINITION 8. Given a ground truth labelling $y_{true} \in U^n$ and associated predictions from a classifier $y_{pred} \in U^n$ of size n , the classification gain, short *cgain*, is defined as:

$$cgain(y_{true}, y_{pred}) := f1(y_{true}, y_{pred}) - f1_{rand}(y_{true}) \quad (7)$$

It reflects the improvement in macro F1-score of the classification over the expected random one. The central idea of *cgain* is to normalize F1-score with the associated classification problem complexity to make the measure comparable for different label distributions. By comparing *cgain* in CLaP (line 12), we enforce that the merge operation steadily improves classification quality, while not relying on thresholds. This addresses the problems (a) to (c) that suffer from greedily maximising performance.

Note that *cgain* is different from gini gain [8], which measures the discrepancy in purity between data sets. Classification gain measures the quality for a given classification considering its difficulty, estimated by random classification.

Example: Figure 4 (top) illustrates a satellite image TS capturing sensor observations of 3 different crops (coloured in blue, orange, green) as 9 segments. A correct segment state sequence (one label per segment) would be: 1,2,3,1,2,3,1,2,3. CLaP tries to find such sequence by merging classes from its initial sequence: 1,2,3,4,5,6,7,8,9.

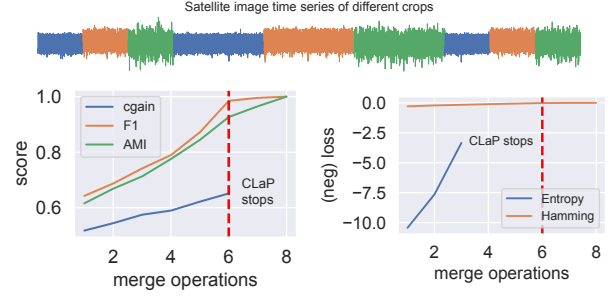


Figure 4: Top: TS capturing 9 time spans of 3 different crops (blue, orange, green) [56]. Bottom: CLaP run on TS with different merge criteria (scores and negative losses); dashed red line marks optimal amount of 6 merge operations, reducing 9 initial classes to 3. Only *cgain* accomplishes this; other measures misjudge label configuration quality and either merge too few (Entropy) or too many (F1, AMI, Hamming) classes.

In Figure 4 (bottom) we use different merge criteria in CLaP (line 12) for this computation and show their values for the merge operations. Generally, merging should increase the measures for 6 iterations (red dashed line) and then stop, as they begin to decrease. It protrudes that F1-score, adjusted mutual information score (AMI) and (negative) Hamming loss steadily increase, reducing the state sequence with 8 merges to just one state, namely 1. This is due to the fact that none of the measures considers problem complexity and overestimates label configuration quality with decreasing number of classes. Negative cross-entropy loss underestimates quality and merges only 3 times, as its value would decrease for a fourth iteration (line 12), resulting in the sequence: 1,2,3,1,2,4,1,5,6. Our proposed classification gain is the only criterion that merges exactly 6 times and would decrease in value thereafter; producing the ground truth result and stopping automatically.

3.3 Computational Complexity

The scalability of state detection algorithms is critical as the size of sensor recordings steadily grows. The complexity of CLaP mainly depends on the segmentation (Algorithm 1, line 3), self-supervised classification (line 5) and merging (lines 6-19).

For a TS T with d channels of size n , the initial computation of the window size w (line 2) with SuSS is in $O(d \cdot n \log w)$ [20]. Locating c CPs in T with ClaSP (line 3) is in $O(d \cdot n^2 \cdot c)$, which involves cross-validating a self-supervised k -nearest neighbour classifier multiple times. The entire preprocessing complexity is in $O(d \cdot n \log w + d \cdot n^2 \cdot c)$ which reduces to $O(d \cdot n^2 \cdot c)$ as $w \leq n$.

Creating the data set for self-supervised classification (line 4) is in $O(d \cdot n \cdot w)$, as it only requires reorganising T . Running a single validation of the ROCKET classifier is also in $O(d \cdot n \cdot w)$; it convolves the data set with a fixed set of 10k kernels for feature generation and uses ridge regression for classification [12]. The same complexity holds for repeating the procedure 5 times during cross-validation (line 5), which leads to a total runtime in $O(d \cdot n \cdot w)$ for the entire step. Note, that ROCKET's runtime hides five-digit constants for calculating convolutions with 10k kernels [12].

Table 1: Technical specifications of TS used in experiments.

Name	No.	TS Length	No. States (Segs.)
	TS / Dims.	Min/Med./Max	Min/Med./Max
TSSB [20]	75 / 1	240 / 3.5k / 20.7k	1 (1) / 3 (3) / 7 (9)
UTSA [26]	32 / 1	2k / 12k / 40k	2 (2) / 2 (2) / 3 (3)
HAS [18]	250 / 9	340 / 5k / 41.5k	1 (1) / 3 (4) / 12 (15)
SKAB [36]	34 / 8	745 / 1.1k / 1.3k	2 (2) / 2 (3) / 2 (3)
MIT-BIH [50]	14 / 1	57k / 258k / 650k	2 (3) / 2 (8) / 5 (16)
Total	405 / 9	240 / 4.8k / 650k	1 (1) / 3 (3) / 12 (16)

The confused merging calculates a confusion matrix (line 8) for the $\lfloor \frac{2 \cdot (n-w)}{w} \rfloor + 1$ labels from the cross-validation in $O(n)$; which only involves incrementing counts. The same complexity holds for replacing labels (lines 10–11) and calculating classification gain (line 12), which is mainly based on computing the confusion matrix for the random F1-score; counting labels and calculating likelihoods. This process is repeated, in the worst case, for $c + 1$ confused label pairs in descending order which requires $O(n \cdot c \log c)$ time. The entire process (lines 6–19) can be repeated up to c times, resulting in a total runtime complexity of $O(n \cdot c^2 \log c)$.

Considering the sum of all steps, CLaP requires $O(d \cdot n^2 \cdot c) + O(d \cdot n \cdot w) + O(n \cdot c^2 \log c)$ time, which simplifies to $O(d \cdot n^2 \cdot c + n \cdot c^2 \log c)$ for computing n state labels with maximal $c + 1$ unique classes. This runtime complexity allows for CLaP’s application to medium-sized and large TS in the batch setting. For streaming purposes, however, further advancements are needed.

The space complexity mainly depends on the data set size and confusion matrix, which is in $O(d \cdot n \cdot w + c^2)$.

4 EXPERIMENTAL EVALUATION

We evaluated the accuracy, runtime, and scalability of CLaP against six competitors on five large benchmark data sets (405 TS) to assess its performance on real-world TS data. Subsection 4.1 presents the data sets, evaluation metrics, and competitors. In Subsections 4.2 and 4.3, we present accuracy and runtime results of CLaP and its competitors. Subsections 4.4 and 4.5 analyse the results of an ablation study with different design choices for CLaP and sensitivity experiments, with varying data quality. Finally, Subsection 4.6 discusses strengths and weaknesses of CLaP and Subsection 4.7 showcases a real-world use case. All experiments were conducted on an Intel Xeon 8358 with 2.60 GHz, 2 TB RAM, 128 cores, running Python 3.8. All of our experimental results, figures, raw measurements per data set, codes, Jupyter Notebooks, and TS used in the evaluation are available on our supporting website [11] for inspection and to ensure reproducibility.

4.1 Experiment Setup

Data Sets: We evaluated CLaP and its competitors on a total of 405 small to very large-sized (240 to 650k) TS from five public segmentation benchmarks (see Table 1). Ground truth CPs and states, annotated by domain experts, are available and were used for evaluation. The benchmarks can be divided into two groups: cross-domain and single-domain use cases.

Table 2: List of competitors for experimental evaluation.

Competitor	State Detection Method
AutoPlait [47]	Hidden Markov Models
CLaP	Self-supervision
ClaSP2Feat	Self-supervision
E2USD [42]	Deep Learning
HDP-HSMM [33]	Hidden Markov Models
TICC [28]	Expectation Maximization
Time2State [66]	Deep Learning

UTSA [26] and TSSB [20] consist of 75 and 32 univariate TS, representing a diverse collection of cross-domain problem settings featuring biological, mechanical, and synthetic processes from sensor, device, image, spectrogram, and simulation signals. Conversely, HAS [18] contains 250 9-dimensional TS from smartphone sensors, capturing indoor and outdoor motion sequences from students. SKAB [36] consists of 34 8-dimensional TS from machine sensors observing a water circulation testbed with pumps, tanks, and valves, undergoing both normal and anomalous conditions. MIT-BIH [50] features 14 univariate high-resolution ECG recordings from patients with normal cardiac activity and prolonged types of arrhythmias.

All TS combined capture versatile processes from natural phenomena, humans, animals, and industrial machinery. The number of distinct states per process varies between 1 and 12, occurring in 1 to 16 segments. In 65% of all data sets, the number of states and segments match; in 15% a single state reoccurs, 9% show two reoccurrences, and the remaining 11% exhibit three to 14 reoccurrences. It is worth noting that the data sets vary greatly in problem complexity, which tends to increase with the number of states and transitions in the processes.

Evaluation Metrics: TSSD combines the segmentation and state identification tasks. We assessed both aspects separately with own scores, namely Covering [63] for segmentation and Adjusted Mutual Information (AMI) [51] for state identification.

Covering measures how well the predicted and annotated segments overlap in location, disregarding the specific state labels of the segments. This allows for the comparison of segmentations of different sizes (including empty segmentations). Let the interval of successive CPs $[t_{c_i}, \dots, t_{c_{i+1}}]$ be a segment in T , and let $segs_{pred}$ and $segs_T$ be the sets of predicted and annotated segmentations, respectively. Additionally, we set $t_{c_0} = 0$ as the first and $t_{c_{n+1}} = n + 1$ as the last CP to include the first and last segments. The Covering score reports the highest-scoring weighted overlap between predicted and annotated segmentations (using the Jaccard index) as a normalized measure in the interval $[0, \dots, 1]$, with higher scores being better (Equation 8).

$$\text{COVERING} = \frac{1}{\|T\|} \sum_{s \in segs_T} \|s\| \cdot \max_{s' \in segs_{pred}} \frac{\|s \cap s'\|}{\|s \cup s'\|} \quad (8)$$

Mutual Information (MI) reports the similarity between two different labellings of the same data. It is invariant to specific label values and allows us to quantify the agreement between predicted and annotated states, disregarding their locations. The measure is adjusted to account for chance, as the score naturally increases with a larger number of states. Let $states_{pred}$ and $states_T$ represent the

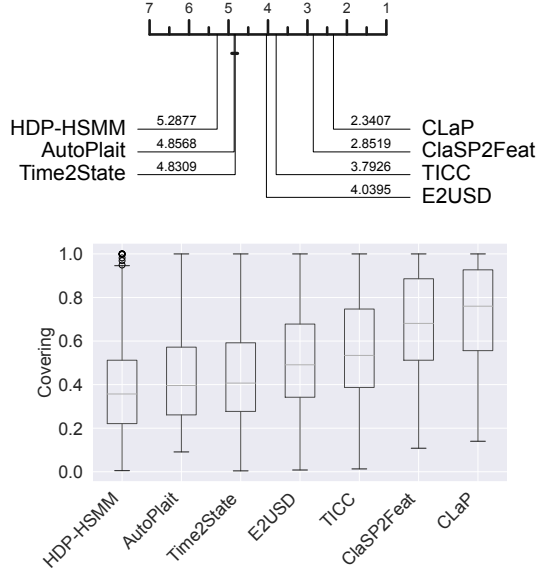


Figure 5: Covering ranks (top) and box plot (bottom) on 405 TS for CLaP (lowest rank) and six competitors.

sets of predicted and annotated states, respectively, and let H denote the entropy function. AMI measures the adjusted, weighted overlap between pairwise clusters as a normalized score within $[-1, \dots, 1]$, with higher values indicating better agreement (Equation 10).

$$MI = \sum_{s \in \text{states}_T} \sum_{s' \in \text{states}_{pred}} \frac{\|s \cap s'\|}{\|T\|} \log\left(\frac{\|T\| \cdot \|s \cap s'\|}{\|s\| \cdot \|s'\|}\right) \quad (9)$$

$$AMI = \frac{MI - E[MI]}{\text{AVG}(H(\text{states}_T), H(\text{states}_{pred})) - E[MI]} \quad (10)$$

To compare different methods across multiple data sets, we aggregate scores into a single ranking. First, we compute the rank of the score for every algorithm and TS. Then, we average the ranks per method across data sets to obtain overall mean ranks. We visualize these values with Critical Difference (CD) diagrams [15] (see, e.g., Figure 5 top) which include an assessment of statistical differences between methods. The best approaches, with the lowest average ranks, are shown to the right of the diagram. Groups of algorithms that are not significantly different in rank are connected by a bar, based on a pairwise one-sided Wilcoxon signed-rank test with $\alpha = 0.05$ and Holm correction. Besides CD diagrams, we discuss summary statistics, box plots [62], and specific examples.

Competitors: We compare CLaP against six state-of-the-art state detection competitors (see Table 2). All implementations are openly available on our supporting website [11]. As a baseline, we extract segments with ClaSP [20] and learn their labels using Time2Feat [6], which extracts and selects features from TS segments before clustering them. We call this approach ClaSP2Feat. We report results for two competitors based on Hidden Markov Models (HMMs). AutoPlait [47] uses so-called multi-level chain models to learn states and applies the minimum description length

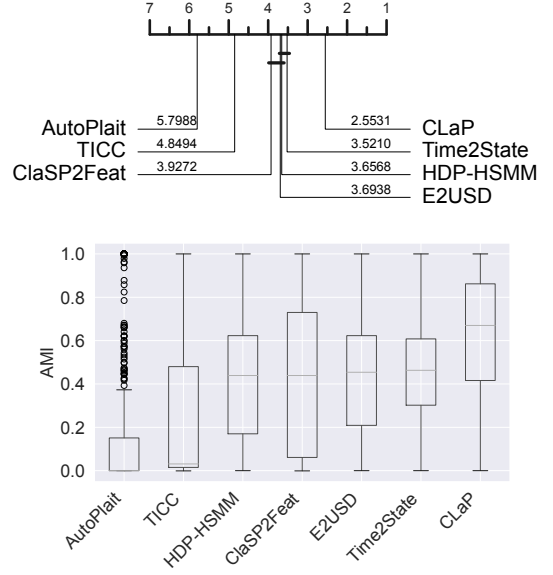


Figure 6: AMI ranks (top) and box plot (bottom) for CLaP (lowest rank) and six competitors on 405 TS.

(MDL) principle to automatically select the best-fitting model. HDP-HSMM [33] is a Bayesian extension of traditional HMMs with efficient state sampling. Another statistical method we compare against is TICC [28], which formulates state detection as a covariance-regularized maximum likelihood problem and computes it using a greedy heuristic. From deep learning, we report results for Time2State [66], which learns an encoder from TS windows using a latent state encoding loss and clusters the embeddings with a Gaussian mixture model (DPGMM) to predict states. E2USD [42] first creates a compact and informative TS embedding using Fast Fourier Transform (FFT) compression and decomposition techniques combined with contrastive learning, and then also uses DPGMM.

To prevent overfitting, we adopt the default hyper-parameters reported in the original publications or reference implementations: for HDP-HSMM we set $\alpha = 10k$ and $\beta = 20$; for TICC we use the annotated subsequence widths, $\lambda = 0.001$, $\beta = 2200$, a convergence threshold of 10^{-4} , and a maximum of 10 iterations; Time2State and E2USD set a sliding window of 256 and step sizes of 10 and 50, respectively. HDP-HSMM, TICC, Time2State, and E2USD output only state sequences, so we derive CPs by locating offsets at changing states. ClaSP2Feat and TICC require the number of states as a hyper-parameter. We set this value from the ground truth annotations. Note that it provides an information advantage to both methods, which is, however, necessary to run them. All other competitors infer the number of states automatically.

4.2 Comparative Analysis

We evaluated the average (Covering / AMI) rank of CLaP and the six competitors on all 405 TS to assess overall performance. Figures 5 and 6 (top) show the results. For both measures, CLaP (2.34 / 2.55) leads with a significant advantage, followed by ClaSP2Feat (2.85

Table 3: Summary performances for CLaP (highest scores) and six competitors on 405 TS.

Covering / AMI		
	average (in %)	median (in %)
CLaP	72.9 ± 21.6 / 60.4 ± 30.9	76.0 / 67.0
ClaSP2Feat	68.4 ± 22.9 / 44.1 ± 34.4	68.1 / 43.9
TICC	57.2 ± 24.2 / 24.4 ± 34.8	53.4 / 3.1
E2USD	50.0 ± 23.3 / 42.0 ± 27.3	49.1 / 45.4
Time2State	43.9 ± 22.6 / 44.3 ± 23.3	40.7 / 46.3
AutoPlait	45.8 ± 25.0 / 17.5 ± 32.9	39.6 / 0.0
HDP-HSMM	38.7 ± 24.1 / 41.5 ± 27.9	35.7 / 43.9

/ 3.93) and TICC (3.79 / 4.85) for Covering, and Time2State (4.83 / 3.52) as well as HDP-HSMM (5.29 / 3.66) for AMI. CLaP’s lead in Covering, compared to ClaSP2Feat, confirms the substantial advantage of confused merging over clustering, as both methods utilize ClaSP for segmentation. Interestingly, the competitors’ differences in average rank are not consistent across the measures. Considering the five benchmarks separately, CLaP also achieves first place for each collection, except in SKAB, where HDP-HSMM leads and CLaP scores third place because it detects wrongly located CPs that cannot be properly corrected by confused merging (see [11]). CLaP’s wins in Covering are not statistically significant, except for HAS. Conversely, its wins in AMI are statistically significant, except for MIT-BIH.

Looking at individual data sets, CLaP wins or ties 204 / 193 out of 405 data sets (Covering / AMI), followed by ClaSP2Feat (146 / 94) and TICC (86 / 62). Note that the Covering counts exceed the total number of data sets because of ties (see [11]). For the 37 TS with only one state, TICC achieves the best results. However, for the 154 data sets with two states and the 214 TS with three or more states, CLaP leads by a significant margin. In a pairwise comparison of CLaP against every competitor, it outperforms each method in at least 38.3% / 56.0% of cases for Covering / AMI. In summary, the rankings show that CLaP on average surpasses state-of-the-art algorithms in terms of accuracy across the five benchmarks.

The summary statistics in Figures 5 and 6 (bottom) and Table 3 confirm the rankings. On average, CLaP scores 72.9 ± 21.6 / 60.4 ± 30.9 for Covering / AMI, representing an increase of 4.5 / 16.1 pp over the second-best competitor, ClaSP2Feat / Time2State. The differences in median are even more pronounced. Across both measures and all benchmarks, CLaP achieves the highest scores, except for SKAB, which aligns with the rankings.

4.3 Runtime and Scalability

We measured the runtime of CLaP and its six competitors on the 405 TS to determine the time needed for TSSD, the accuracy-runtime tradeoff, as well as the scalability of CLaP.

Runtime: Figure 7 (left) shows the runtime distributions of the methods on all data sets. AutoPlait (median of 0.05 seconds) leads with a substantial advantage over all other competitors due to its linear runtime complexity (with respect to TS size). Additionally, the main part of its implementation uses fast C code, while all

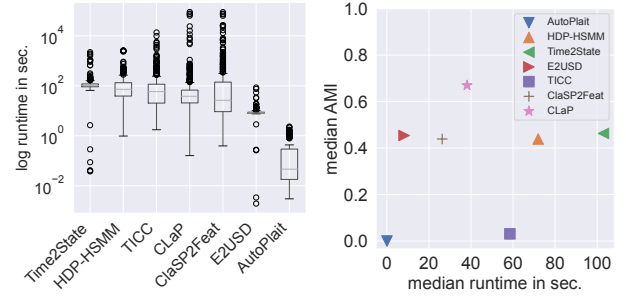


Figure 7: Runtime distribution (left) and accuracy-runtime tradeoff (right) on 405 TS for CLaP and six competitors.

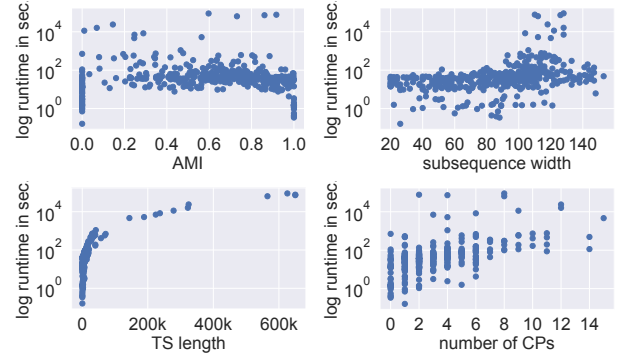


Figure 8: Scalability of CLaP considering AMI performance (top left), subsequence width (top right), TS length (bottom left), and number of CPs (bottom right).

other methods are implemented in Python. E2USD (8 seconds) and ClaSP2Feat (26 seconds) rank second and third. CLaP (38 seconds) secures fourth place. Its segmentation step takes 25 seconds, while the self-supervised classification and confused merging require 13 seconds. TICC (59 seconds) and HDP-HSMM (72 seconds) rank fifth and sixth, followed by Time2State (103 seconds) at last place. The distributions of all methods scatter widely, due to the large spans in TS lengths. CLaP is faster in median than three of its six competitors, but trades more computational costs, than e.g. AutoPlait, for higher accuracy. If runtime is a critical concern, users may choose to use the fast streaming implementation of ClaSP [22] or an even faster classifier, e.g. QUANT [14], which may, however, decrease accuracy.

Accuracy vs. Runtime: We investigated the tradeoff between accuracy and runtime of the approaches. Figure 7 (right) depicts the median runtimes vs. AMI scores. The optimal solution is a fast and accurate algorithm (top left), while a poor algorithm is slow and inaccurate (bottom right). CLaP (pink star), ClaSP2Feat (brown plus), and E2USD (red triangle) all exhibit traits of fast and accurate algorithms. However, CLaP stands out with a solution that is only 12 seconds slower but 21.6 pp more accurate. It offers, by far, the most desirable tradeoff between accuracy and runtime compared to the competitors on the considered TS.

Scalability: Figure 8 illustrates how CLaP’s (log) runtime scales concerning four different variables, namely: AMI score, subsequence width, TS length, and number of CPs. For the first two variables, we do not observe clear relationships. As expected, CLaP becomes slower with increasing TS length. For instance, it requires less than 5 minutes to process up to 20k data points and less than 3 hours for up to 200k values. Its quadratic runtime complexity with respect to TS length can be roughly observed in the experimental runtimes. For an increasing number of CPs, CLaP also requires more runtime. However, the values are more dispersed compared to TS length. In summary, CLaP produces results for TS with tens of thousands of data points in minutes. For much larger data, runtime may become an issue, as for all other methods, except AutoPlait. In future work, we will investigate TS partitioning and hierarchical merging to further improve scalability.

4.4 Ablation Study

CLaP has four main components, namely: (a) a window size selection algorithm, (b) a segmentation procedure, (c) a classifier, and (d) a merge mechanism. We evaluated different approaches for each component, fixing the others to defaults, on 20% of randomly chosen TS (21 out of 107) from TSSB and UTSA. To prevent overfitting, we excluded the remaining 86 out of 107 data sets, as well as the 298 TS from HAS, SKAB, and MIT-BIH from the ablations. The following analysis reports the summarized results. See the box plots for visualization and more evaluations on our website [11].

(a) Window Size Selection: To determine the window size for a TS in CLaP, we tested two whole-series-based methods: the most dominant Fourier frequency (FFT) and the highest autocorrelation offset (ACF) from [19]; as well as two subsequence-based algorithms: Multi-Window-Finder (MWF) [32] and Summary Statistics Subsequence (SuSS) [20]. Our experiments show no significant differences in average rank between the methods, which aligns with the results from [19]. We choose SuSS for WSS in CLaP, as it achieves the highest average scores for Covering of $80.5\% \pm 17.4\%$ and AMI of $76.1\% \pm 27.5\%$.

(b) Segmentation Procedure: We evaluated three optimization-based approaches for segmentation, namely BinSeg [61], PELT [39], and Window [61], as well as the density-based approach FLUSS [26], the density ratio estimator RuLSIF [31], and the self-supervised algorithm ClaSP [20]. For both metrics, ClaSP leads in average rank, with an insignificant advance for Covering (2.12) and a significant one for AMI (1.98). Hence, we use ClaSP in CLaP.

(c) Classifier: For self-supervised classification in CLaP, we tested six state-of-the-art classifiers according to [49], namely: the feature-based approach FreshPRINCE [48], the interval-based algorithm QUANT [14], the shapelet-based method RDST [27], the dictionary-based procedure WEASEL 2.0 [58], and two convolution-based techniques, namely ROCKET [12] and MR-Hydra [13]. Our experiments show only differences in tendency between the average ranks of the classifiers. We choose ROCKET, as it has the highest-scoring median Covering of 81.6% and AMI of 87.2%.

(d) Merge Score: To implement the merge mechanism, we evaluated four scores: our proposed cgain, F1, ROC/AUC, and AMI; we also assessed two loss functions: log and Hamming. Classification gain leads for both metrics in average rank, with an insignificant

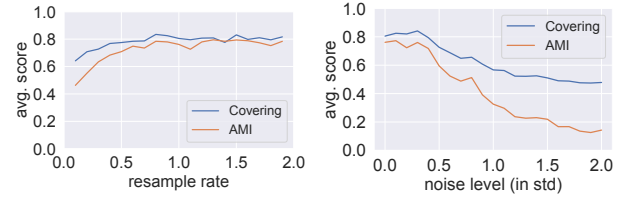


Figure 9: Sensitivity analysis of CLaP regarding sampling rate (left) and noise level (right).

advance for Covering but a significant difference to unnormalized F1 and both losses in AMI. It achieves an increase in Covering (AMI) of 2.2 (4.1) percentage points. Therefore, we use cgain in CLaP for confused merging.

In summary, the choice of window size and classification algorithm results in only negligible differences in performance, whereas the segmentation procedure and merge score have a substantial impact. This may be partially due to the interconnections between the components; for example, imprecisely located CPs can negatively influence self-supervised classification, and improper merging can easily degrade the entire state detection process.

We investigated whether CLaP’s two phases, segmentation and state identification, substantially contribute to TSSD by comparing each phase with a random baseline while keeping all other settings fixed. For segmentation, we ran CLaP with uniformly drawn CPs (without replacement), which lowered the average Covering score by 20.2 pp and AMI by 31.8 pp. For state identification, we assigned uniformly drawn labels to the segments (with replacement), causing Covering to drop by 20.5 pp and AMI by 40.9 pp. These pronounced declines show that both phases are essential to CLaP’s performance; omitting either one sharply degrades accuracy.

4.5 Sensitivity Analysis

Real-world TS data are collected at different sampling rates and often contain noise introduced by sensor artefacts or external events. We evaluated how these factors affect CLaP’s performance on the data sets from the ablation study to assess its robustness to changes in data representation.

Sampling Rate: Figure 9 (left) plots CLaP’s average Covering and AMI scores when the data sets are resampled to 10% up to 190% (in 10 pp steps) of their original size. Both metrics increase steadily from 10% to roughly 50% and then plateau. This suggests that CLaP requires a minimum temporal resolution (30% to 50% for these data sets) to identify process states accurately, but maintains stable performance at higher resolutions.

Noise Level: Figure 9 (right) shows how CLaP’s performance changes when white noise with increasing standard deviation is added to the standardised data. Performance remains quite stable up to a noise level of 0.5, after which it declines sharply. Thus, CLaP is resilient to moderate noise but struggles to differentiate process states under very high noise levels.

These experiments show that CLaP is robust to variations in sampling rate and to moderate noise. When data become excessively coarse or noisy, performance degrades gradually, indicating that the method’s performance scales with data quality.

4.6 Discussion

CLaP’s central strength is its high accuracy. The comparative analysis shows that it is significantly more accurate compared to six state-of-the-art competitors on the five benchmarks. We carefully designed this edge by splitting TSSD into a two-step process: segmentation and state identification; optimizing each part separately:

(a) Segmentation: We use ClaSP for segmentation, which is one of the most accurate TSS algorithms for univariate and multivariate [21] batch data as well as streaming TS [22]. Except for ClaSP2Feat, the other competitors either do not use a dedicated segmentation procedure (e.g. Time2State) or use one with inferior detection accuracy, such as AutoPlait.

(b) State Identification: Our proposed self-supervised classification and confused merging procedure leverages the predictive power of classification algorithms [49], which are more accurate than classical clustering strategies that do not use TS-specific feature engineering or label information, as i.e. implemented in TICC. Others rely on distance calculations [30], or use limited statistical models with assumptions (e.g. Time2State).

Both components substantially contribute to the overall performance, as the ablation study indicates for (a) and CLaP’s advance over ClaSP2Feat shows for (b). Despite its strengths, CLaP has several weaknesses. First, it assumes that subsequences can fully capture the distinctive properties of process states. This assumption holds for TS with homogeneous segments that contain similar temporal patterns, but performance degrades when intra-state diversity is high, e.g. in the presence of high noise (see Figure 9, right) or concept drift. Second, CLaP’s runtime is quadratic in both the TS length and the number of CPs. For very long sequences with many CPs, this leads to high latency and limits its direct use in streaming contexts. Although a streaming version of ClaSP exists [22], further work is required to make confused merging viable for real-time processing. Consequently, the current CLaP implementation is best suited to batch TS with homogeneous segments.

4.7 Satellite Image Data Use Case

We revisit the satellite image example from Subsection 3.2 in Figure 10. The TS (top) illustrates sensor data for 3 different crops (coloured in blue, orange, and green) across 9 segments. We computed the state sequences of CLaP, ClaSP2Feat, and Time2State (2nd from top to fourth). A well-fitting output annotates one distinct number to each observed state (e.g., 1 for blue, 2 for orange, and 3 for green). CLaP accurately identifies the boundaries of each segment and assigns correct state labels, resulting in the sequence: 1,2,3,1,2,3,1,2,3. ClaSP2Feat does not differentiate between the blue and orange segments and erroneously assigns different labels to the green segments. However, it correctly identifies some CPs. The result of Time2State is noisy and largely overestimates the total number of states. Nonetheless, it correctly labels many data points and identifies correct repetitions of segments. Figure 10 (bottom) shows the state sequences as diagrams, with nodes representing states and edges depicting transitions. The start and end states appear in green (blue, and transition probabilities are labelled. It protrudes that only CLaP’s graph is isomorphic to the ground truth.

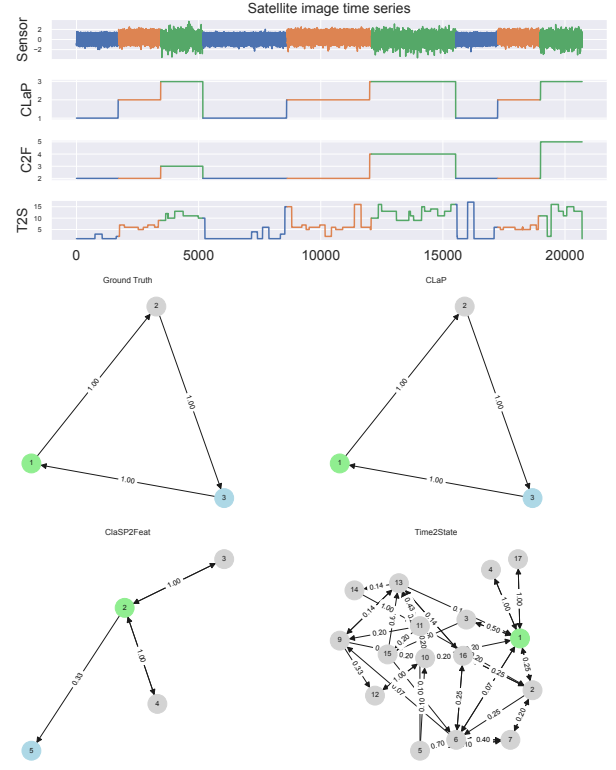


Figure 10: TS (top) shows satellite image data for 3 different crops (blue, orange, green) [56]. State sequences (2nd from top to fourth) shows results of CLaP, ClaSP2Feat, and Time2State. Bottom diagrams show graphical representations of ground truth and competitor results.

This use case demonstrates the accuracy of CLaP for long and complex time series data (20.7k data points, 9 segments, 3 states). Its results are also interpretable for human inspection.

5 RELATED WORK

The rapid growth of sensor data from IoT devices in *smart* applications, such as healthcare and factories, has driven a substantial boost in research of TS management and mining [41]. TS storage solutions include specialized databases [54], indices [17], and compression algorithms [44]. Unsupervised data analytics process the acquired data to automatically extract anomalies [7], motifs [57], or clusters [53]. This enables domain experts to make data-driven decisions, such as health professionals conducting gait analysis [73].

TSSD is a complex preprocessing step in a typical TS analysis workflow. Its main components, TS segmentation and clustering, have been researched both independently and in combination [65]. TSS can be formalized as an optimization problem, where each segment incurs a cost based on a discrepancy measure (e.g., mean-shift [52] or entropy [55]), and the number of segments [61] is penalized. Exact or approximate solvers, such as PELT [40] or Wild BinSeg [25], then compute the optimal segmentation. Adams et al. model segments as probability distributions and use a recursive

message-passing algorithm to infer the most recent CP [1]. This approach can be extended to detect short gradual changes [16], an extension of the original TSS problem [9]. A central limitation of these techniques is their dependence on domain-specific parameters, such as assumed value distributions or fitting cost functions. To overcome this, Katser et al. propose ensembling strategies, which increase robustness and accuracy [35]. Other domain-agnostic methods include FLUSS [26] and ClaSP [20], which achieve top-ranking results on recent TSS benchmarks [18, 23] and are capable of processing multivariate and streaming data. FLUSS measures the density of similar subsequences in potential segments as an arc curve, from which it extracts local minima that represent CPs [26]. ClaSP frames TSS as a collection of self-supervised subsequence classification problems and reports the segmentation with the highest cross-validation performance [20]. We use ClaSP in CLaP because it is hyper-parameter-free and makes few assumptions about segments, such as being mutually dissimilar.

After segmentation, individual segments must be assigned correct state labels, which can be achieved by clustering, for instance, by extracting equal-sized subsequences from segments, where the individual data points define the feature space. Partition-based algorithms, such as Lloyd’s algorithm [45], PAM [37], or k-Shape [53], randomly partition the data and iteratively refine it until some convergence criterion is met [30]. Specifically for TS, the data is commonly assigned to clusters based on minimal distances to their centres, such as medoids. Holder et al. evaluated the accuracy of different distance measures and found MSM [59] and TWE [46] to be good choices [30]. Different classical clustering approaches include density-based methods, e.g., DBSCAN [24] and OPTICS [3], as well as hierarchical approaches, such as BIRCH [72]. These can be applied to the raw TS segments. However, Bonifati et al. report in a recent study that extracting and selecting features from TS before clustering improves performance [6]. Specifically, they compute statistics with tsfresh [10] from the data and calculate similarity measures between sequences. Features without variance are filtered, and the remaining ones are clustered. We use this approach in our baseline ClaSP2Feat, which combines an accurate segmentation and clustering procedure and achieves top-ranking results.

Besides the independent segmentation and clustering, the literature also contains integrated TSSD approaches [65]. HMMs, for instance, can model single TS with a limited number of states and probabilities for initialization, transition, and output [5]. By design, HMM states model single data points, not prolonged events. Matsubara et al. tackle this by grouping HMM states and learning transitions between groups, which reveals their CPs. AutoPlait implements this idea by iteratively refining the segmentation to optimize a cost function [47]. HDP-HSMM uses a different approach: it incorporates explicit-duration semi-Markovian properties to model flexible HMM state durations and employs a hierarchical Dirichlet process prior to learn the number of states from the TS directly [33]. Hallac et al. propose characterizing TS using a sliding window instead of modelling single data points. Each subsequence belongs to a cluster that is characterized by a correlation network. In TICC, the assignment of subsequences and the update of cluster parameters are iteratively refined using an expectation maximization algorithm [28]. The recent method Time2State also builds a sliding window but uses it to train a deep learning encoder by

minimizing (maximizing) distances of intra-state (inter-state) subsequences. The resulting embedding is clustered using DPGMM to assign subsequences to state labels and to automatically learn their amount [66]. E2USD implements a similar deep-learning TSSD architecture, transforming each sliding window through compression, trend–seasonal decomposition and contrastive learning into a compact embedding that is subsequently clustered via DPGMM [42].

Our proposed CLaP also processes subsequences, but differs substantially from the existing methods. It is the first TSSD method that uses self-supervised learning [29] to exploit the predictive power of TS classification algorithms, which are more accurate than the unsupervised models used by the aforementioned methods that rely on classical techniques sensitive to hyper-parameters or TS characteristics. CLaP merges confused classes as long as their cgain increases, which clusters segments automatically in label space rather than in feature space.

Apart from state detection in single TS, current research also investigates the relationships among dimensions in MTS and across different data sets. Wang et al. examine high-dimensional TS in which a subset of channels must be automatically selected for accurate TSSD. They propose ISSD, an algorithm that identifies a subset of TS dimensions with high-quality states, quantified by channel set completeness, and solves this selection in an optimization problem [68]. StaCo is another algorithm designed to measure overall, partial, and time-lagged correlations between states in heterogeneous TS. It produces a state correlation matrix, from which state affiliations across data sets can be derived [67].

6 CONCLUSION

We proposed CLaP (Classification Label Profile), a new time series state detection algorithm that uses self-supervised classification techniques for segmentation and labelling. Our experimental evaluation shows that CLaP combines high-scoring design choices, leading to a statistically significant improvement in accuracy over six competitors on 405 univariate and multivariate TS from five benchmarks. CLaP also offers the best tradeoff between accuracy and runtime. It computes state labels for tens of thousands of data points in just a few minutes, providing a useful TS annotation that can be employed as input for advanced data analytics or as decision support for domain experts.

Limitations of CLaP include its sensitivity to the segmentation provided by ClaSP. While confused merging can correct false-positive CPs, it cannot relocate imprecise ones. It also has a quadratic runtime complexity with respect to TS size and the number of CPs. Some TS applications, such as predictive maintenance, require data analysis solutions capable of processing streaming data.

In future work, we plan to explore sampling, batching, and hierarchical techniques to improve the scalability of CLaP. We want to explore the incorporation of domain knowledge to CLaP to guide its TSSD. Experts often have specific information from visual inspection or access to exogenous data which could increase the quality of state detection, e.g. by indicating some CPs or state labels of single measurements. We further aim to investigate state detection in TS collections to identify recurring states and transitions across instances, potentially revealing even more semantics of the data-generating process.

REFERENCES

- [1] Ryan P. Adams and David John Cameron MacKay. 2007. Bayesian Online Change-point Detection. *arXiv: Machine Learning* (2007).
- [2] Md Atiqur Rahman Ahad, Anindya Das Antar, and Masud Ahmed. 2021. IoT Sensor-Based Activity Recognition - Human Activity Recognition. In *Intelligent Systems Reference Library*.
- [3] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. 1999. OPTICS: ordering points to identify the clustering structure. In *ACM SIGMOD Conference*.
- [4] Anthony Bagnall, Michael Flynn, James Large, Jason Lines, and Matthew Middlehurst. 2020. On the usage and performance of the hierarchical vote collective of transformation-based ensembles version 1.0 (hive-cote v1. 0). In *Advanced Analytics and Learning on Temporal Data: 5th ECML PKDD Workshop, AALTD 2020, Ghent, Belgium, September 18, 2020, Revised Selected Papers 6*. Springer, 3–18.
- [5] Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer.
- [6] Angela Bonifati, Francesco Del Buono, Francesco Guerra, Miki Lombardi, and Donato Tiano. 2023. Interpretable Clustering of Multivariate Time Series with Time2Feat. *Proceedings of the VLDB Endowment* 16 (2023), 3994–3997.
- [7] Paul Boniol, Michele Linardi, Federico Roncallo, Themis Palpanas, Mohammed Meftah, and Emmanuel Remy. 2021. Unsupervised and scalable subsequence anomaly detection in large data series. *VLDB Journal* 30 (2021), 909–931.
- [8] Leo Breiman. 2004. Technical note: Some properties of splitting criteria. *Machine Learning* 24 (2004), 41–47.
- [9] Louis Carpentier, Len Feremans, Wannes Meert, and Mathias Verbeke. 2024. Pattern-based Time Series Semantic Segmentation with Gradual State Transitions. In *SIAM International Conference on Data Mining*.
- [10] Maximilian Christ, Nils Braun, Julius Neuffer, and A. Kempa-Liehr. 2018. Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh - A Python package). *Neurocomputing* 307 (2018), 72–77.
- [11] CLaP Code and Raw Results. 2025. <https://github.com/ermshaus/classification-label-profile>.
- [12] Angus Dempster, Francois Petitjean, and Geoffrey I. Webb. 2019. ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery* 34 (2019), 1454 – 1495.
- [13] Angus Dempster, Daniel F. Schmidt, and Geoffrey I. Webb. 2022. competing convolutional kernels for fast and accurate time series classification. *Data Mining and Knowledge Discovery* 37 (2022), 1779–1805.
- [14] Angus Dempster, Daniel F. Schmidt, and Geoffrey I. Webb. 2024. QUANT: A Minimalist Interval Method for Time Series Classification. *Data Mining and Knowledge Discovery* (2024), 1–26.
- [15] Janez Demšar. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research* 7 (2006), 1–30.
- [16] Erick Draayer, H. Cao, and Yifan Hao. 2021. Reevaluating the Change Point Detection Problem with Segment-based Bayesian Online Detection. *Proceedings of the 30th ACM International Conference on Information & Knowledge Management* (2021).
- [17] Karima Echihabi, Panagiota Fatourou, Kostas Zoumpatanos, Themis Palpanas, and Houda Benbrahim. 2022. Hercules Against Data Series Similarity Search. *Proceedings of the VLDB Endowment* 15 (2022), 2005–2018.
- [18] Arik Ermshaus, Patrick Schäfer, Anthony Bagnall, Thomas Guyet, Georgiana Ifrim, Vincent Lemaire, Ulf Leser, Colin Leverger, and Simon Malinowski. 2023. Human Activity Segmentation Challenge @ ECML/PKDD’23. In *8th Workshop on Advanced Analytics and Learning on Temporal Data*.
- [19] Arik Ermshaus, Patrick Schäfer, and Ulf Leser. 2022. Window Size Selection In Unsupervised Time Series Analytics: A Review and Benchmark. *7th Workshop on Advanced Analytics and Learning on Temporal Data* (2022).
- [20] Arik Ermshaus, Patrick Schäfer, and Ulf Leser. 2023. ClaSP: parameter-free time series segmentation. *Data Mining and Knowledge Discovery* 37 (2023), 1262 – 1300.
- [21] Arik Ermshaus, Patrick Schäfer, and Ulf Leser. 2024. Multivariate Human Activity Segmentation: Systematic Benchmark with ClaSP. *9th Workshop on Advanced Analytics and Learning on Temporal Data* (2024).
- [22] Arik Ermshaus, Patrick Schäfer, and Ulf Leser. 2024. Raising the ClaSS of Streaming Time Series Segmentation. *Proceedings of the VLDB Endowment* 17, 8 (2024), 1953–1966.
- [23] Arik Ermshaus, Sunita Singh, and Ulf Leser. 2023. Time Series Segmentation Applied to a New Data Set for Mobile Sensing of Human Activities. In *EDBT/ICDT Workshops*.
- [24] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Knowledge Discovery and Data Mining*.
- [25] Piotr Fryzlewicz. 2014. Wild binary segmentation for multiple change-point detection. *The Annals of Statistics* 42, 6 (2014), 2243.
- [26] Shaghayegh Gharghabi, Chin-Chia Michael Yeh, Yifei Ding, Wei Ding, Paul R. Hidding, Samuel R. LaMunio, Andrew Kaplan, Scott E. Crouter, and Eamonn J. Keogh. 2018. Domain agnostic online semantic segmentation for multi-dimensional time series. *Data Mining and Knowledge Discovery* 33 (2018), 96 – 130.
- [27] Antoine Guillaume, Christel Vrain, and Wael Elloumi. 2021. Random Dilated Shapelet Transform: A New Approach for Time Series Shapelets. In *International Conferences on Pattern Recognition and Artificial Intelligence*.
- [28] David Hallac, Sagar Vare, Stephen P. Boyd, and Jure Leskovec. 2017. Toeplitz Inverse Covariance-Based Clustering of Multivariate Time Series Data. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2017).
- [29] Shohei Hido, Tsuyoshi Idé, Hisashi Kashima, Harunobu Kubo, and Hirofumi Matsuzawa. 2008. Unsupervised Change Analysis Using Supervised Learning. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*.
- [30] Christopher Holder, Matthew Middlehurst, and A. Bagnall. 2022. A Review and Evaluation of Elastic Distance Functions for Time Series Clustering. *Knowledge and Information Systems* 66 (2022), 765–809.
- [31] Mikhail Hushchyn and Andrey Ustyuzhanin. 2021. Generalization of change-point detection in time series data based on direct density ratio estimation. *Journal of Computational Science* 53 (2021), 101385.
- [32] Shima Imani, Alireza Abdoli, Ali Beyram, and Eamonn J. Keogh. 2021. Multi-Window-Finder: Domain Agnostic Window Size for Time Series Data. In *MileTS’21: 7th KDD Workshop on Mining and Learning from Time Series*.
- [33] Matthew J. Johnson and Alan S. Willsky. 2010. The Hierarchical Dirichlet Process Hidden Semi-Markov Model. In *Conference on Uncertainty in Artificial Intelligence*.
- [34] Ameeth Kanawaday and Aditya Sane. 2017. Machine learning for predictive maintenance of industrial machines using IoT sensor data. *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)* (2017), 87–90.
- [35] Iurii D. Katser, Viacheslav Kozitsin, Victor Lobachev, and Ivan Maksimov. 2021. Unsupervised Offline Changepoint Detection Ensembles. *Applied Sciences* 11 (2021), 4280.
- [36] Iurii D. Katser and Vyacheslav O. Kozitsin. 2020. Skoltech Anomaly Benchmark (SKAB).
- [37] Leonard Kaufman and Peter J. Rousseeuw. 1990. *Partitioning Around Medoids (Program PAM)*. John Wiley & Sons, Ltd, Chapter 2, 68–125.
- [38] Bob Kemp, Aeilko H. Zwiderman, Bert Tuk, Hilbert A. C. Kamphuisen, and Josefien J. L. Obery. 2000. Analysis of a sleep-dependent neuronal feedback loop: the slow-wave microcontinuity of the EEG. *IEEE Transactions on Biomedical Engineering* 47 (2000), 1185–1194.
- [39] Rebecca Killick, Paul Fearnhead, and Idris Arthur Eckley. 2011. Optimal detection of changepoints with a linear computational cost. *J. Amer. Statist. Assoc.* 107 (2011), 1590 – 1598.
- [40] Rebecca Killick, Paul Fearnhead, and Idris A. Eckley. 2012. Optimal detection of changepoints with a linear computational cost. *J. Amer. Statist. Assoc.* 107 (2012), 1590 – 1598.
- [41] Rajalakshmi Krishnamurthi, Adarsh Kumar, Dhanalekshmi Gopinathan, Anand Nayyar, and Basit Qureshi. 2020. An Overview of IoT Sensor Data Processing, Fusion, and Analysis Techniques. *Sensors (Basel, Switzerland)* 20 (2020).
- [42] Zhichen Lai, Huan Li, Dalin Zhang, Yan Zhao, Weizhu Qian, and Christian S. Jensen. 2024. E2Usd: Efficient-yet-effective Unsupervised State Detection for Multivariate Time Series. *Proceedings of the ACM Web Conference 2024* (2024).
- [43] Oscar D. Lara and Miguel A. Labrador. 2013. A Survey on Human Activity Recognition using Wearable Sensors. *IEEE Communications Surveys & Tutorials* 15 (2013), 1192–1209.
- [44] Panagiotis Liakos, Katia Papakonstantinou, and Yannis Kotidis. 2022. Chimp: Efficient Lossless Floating Point Compression for Time Series Databases. *Proceedings of the VLDB Endowment* 15 (2022), 3058–3070.
- [45] Stuart P. Lloyd. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory* 28 (1982), 129–136.
- [46] Pierre-François Marteau. 2007. Time Warp Edit Distance with Stiffness Adjustment for Time Series Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31 (2007), 306–318.
- [47] Yasuko Matsubara, Yasushi Sakurai, and Christos Faloutsos. 2014. Autoplait: Automatic mining of co-evolving time sequences. In *Proceedings of the ACM SIGMOD international conference on Management of data*. 193–204.
- [48] Matthew Middlehurst and Anthony Bagnall. 2022. The FreshPRINCE: A Simple Transformation Based Pipeline Time Series Classifier. In *International Conferences on Pattern Recognition and Artificial Intelligence*.
- [49] Matthew Middlehurst, Patrick Schafer, and A. Bagnall. 2023. Bake off redux: a review and experimental evaluation of recent time series classification algorithms. *Data Mining and Knowledge Discovery* 38 (2023), 1958–2031.
- [50] George B. Moody and Roger G. Mark. 2001. The impact of the MIT-BIH Arrhythmia Database. *IEEE Engineering in Medicine and Biology Magazine* 20 (2001), 45–50.

- [51] Xuan Vinh Nguyen, Julien Epps, and James Bailey. 2010. Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance. *Journal of Machine Learning Research* 11 (2010), 2837–2854.
- [52] E. S. Page. 1955. A test for a change in a parameter occurring at an unknown point. *Biometrika* 42 (1955), 523–527.
- [53] John Paparrizos and Luis Gravano. 2016. k-Shape: Efficient and Accurate Clustering of Time Series. *SIGMOD Record* 45 (2016), 69–76.
- [54] Tuomas Pelkonen, Scott Franklin, Paul Cavallaro, Qi Huang, Justin Meza, Justin Teller, and Kaushik Veeraraghavan. 2015. Gorilla: A Fast, Scalable, In-Memory Time Series Database. *Proceedings of the VLDB Endowment* 8 (2015), 1816–1827.
- [55] Piotr Sadri, Yongli Ren, and Flora D Salim. 2017. Information gain-based metric for recognizing transitions in human activities. *Pervasive and Mobile Computing* 38 (2017), 92–109.
- [56] Patrick Schäfer, Arik Ermshaus, and Ulf Leser. 2021. ClaSP - Time Series Segmentation. *Proceedings of the 30th ACM International Conference on Information & Knowledge Management* (2021).
- [57] Patrick Schäfer and Ulf Leser. 2022. Motiflets - Simple and Accurate Detection of Motifs in Time Series. *Proceedings of the VLDB Endowment* 16 (2022), 725–737.
- [58] Patrick Schäfer and Ulf Leser. 2023. WEASEL 2.0: a random dilated dictionary transform for fast, accurate and memory constrained time series classification. *Machine Learning* 112 (2023), 4763–4788.
- [59] Alexandra Stefan, Vassilis Athitsos, and Gautam Das. 2013. The Move-Split-Merge Metric for Time Series. *IEEE Transactions on Knowledge and Data Engineering* 25 (2013), 1425–1438.
- [60] Minh-Quang Tran, Hoang-Phuong Doan, Viet Q. Vu, and Lien T. Vu. 2022. Machine Learning and IoT-based Approach for Tool Condition Monitoring: A Review and Future Prospects. *Measurement* (2022).
- [61] Charles Truong, Laurent Oudre, and Nicolas Vayatis. 2020. Selective review of offline change point detection methods. *Signal Processing* 167 (2020).
- [62] John Wilder Tukey. 1977. Exploratory data analysis. *Reading/Addison-Wesley* (1977).
- [63] Gerrit JJ van den Burg and Christopher KI Williams. 2020. An evaluation of change point detection algorithms. *arXiv* (2020).
- [64] Chen Wang, Xiangdong Huang, Jialin Qiao, Tian Jiang, Lei Rui, Jinrui Zhang, Rong Kang, Julian Feinauer, Kevin A McGrail, Peng Wang, et al. 2020. Apache IoTDB: Time-series database for internet of things. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2901–2904.
- [65] Chengyu Wang, Xiong lve Li, Tongqing Zhou, and Zhiping Cai. 2024. Unsupervised Time Series Segmentation: A Survey on Recent Advances. *Computers, Materials & Continua* (2024).
- [66] Chengyu Wang, Kui Wu, Tongqing Zhou, and Zhiping Cai. 2023. Time2State: An Unsupervised Framework for Inferring the Latent States in Time Series Data. *Proceedings of the ACM on Management of Data* 1 (2023), 1 – 18.
- [67] Chengyu Wang, Yuan Yuan, Tongqing Zhou, and Zhiping Cai. 2024. Detecting State Correlations between Heterogeneous Time Series. *Proceedings of the 2024 2nd International Conference on Advances in Artificial Intelligence and Applications* (2024).
- [68] Chengyu Wang, Tongqing Zhou, Lin Chen, Shan Zhao, and Zhiping Cai. 2025. ISSD: Indicator Selection for Time Series State Detection. *Proceedings of the ACM on Management of Data* (2025).
- [69] J. H. Woollam, Jannes Munchmeyer, Frederik Tilmann, Andreas Rietbrock, Dietrich Lange, Thomas Bornstein, Tobias Diehl, Carlo Giunchi, Florian Haslinger, Dario Jozinovi'c, Alberto Michelini, Joachim Saul, and Hugo Soto. 2022. Seis-Bench—A Toolbox for Machine Learning in Seismology. *Seismological Research Letters* (2022).
- [70] Yongchao Ye, Xinting Zhu, Xuejin X. Shen, Xiaoyu Chen, Lishuai Li, and S. Joe Qin. 2024. Diffusion-based Method for Satellite Pattern-of-Life Identification. *arXiv* (2024).
- [71] Jie Yin, Qiang Yang, and Jeffrey Junfeng Pan. 2008. Sensor-Based Abnormal Human-Activity Detection. *IEEE Transactions on Knowledge and Data Engineering* 20 (2008), 1082–1090.
- [72] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996. BIRCH: an efficient data clustering method for very large databases. In *ACM SIGMOD Conference*.
- [73] Lin Zhou, Eric Fischer, Clemens Markus Brahm, Urs Granacher, and Bert Arnrich. 2023. DUO-GAIT: A gait dataset for walking under dual-task and fatigue conditions with inertial measurement units. *Scientific Data* 10 (2023).