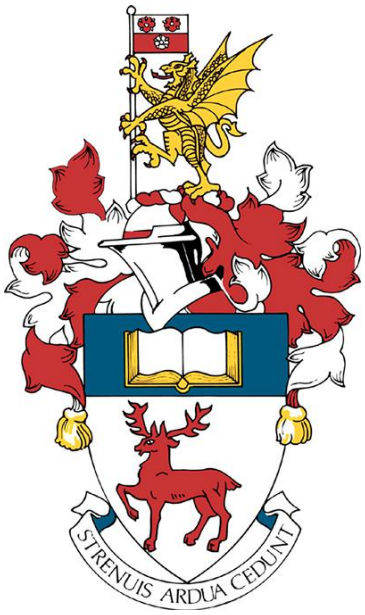
ELECTRONICS AND COMPUTER SCIENCE
FACULTY OF ENGINEERING AND PHYSICAL SCIENCES
UNIVERSITY OF SOUTHAMPTON



Daniel Roberts

11th November 2025

A Slice of Time; Segmenting Human Activity with *aeon*

Project supervisor:
Tony Bagnall
a.j.bagnall@soton.ac.uk

Second examiner:
Thanassis Tiropanis
t.tiropanis@soton.ac.uk

A project report submitted for the award of
BSc Computer Science

Dedicated to my late Dad, Jay Roberts.
I love you.

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.
- I have acknowledged all sources, and identified any content taken from elsewhere.
- I have used resources produced by others.
  - Datasets have been produced/gathered by various organisations and have been properly credited.
- I did all the work myself and have not helped anyone else.
- The material in the report is genuine, and I have included all my data/code/designs.
- I have submitted part of this work for another assessment.
  - Part of this work was submitted for the project progress report, as is permitted by regulations of this module.
- My work does involve human participants, their cells or data, or animals.
  - Ethics Approval (ERGO) Number: 101068

Contents

Acknowledgements

I would like to express my gratitude to The University for the skills and resources to produce this report.

I am also extremely grateful to my supervisor, Prof. Tony Bagnall, for his invaluable guidance, insightful feedback, and encouragement throughout this project.

I would like to acknowledge Arik Ermshaus of the *aeon* community, for his advice and recommendations of evaluation metrics.

Finally, I would like to thank my Mum, Angie Roberts, for her unwavering love and support throughout my life and academic career.

***Abstract*—Time series segmentation is a crucial step in the wider problem of human activity recognition. There are many algorithms that perform semantic segmentation of time series data, several of which are included in the open-source Python library *aeon*. How do these algorithms perform for the task of human activity recognition? Is there a better algorithm not yet included in *aeon*? This report implements a segmentation algorithm for time series data not currently present in the library; AUTOPLAIT. We evaluate it and existing *aeon* algorithms on five human activity datasets and show that it performs in line with other algorithms currently present in *aeon*, achieving similar precision, recall and, F-score, in addition to providing information on any repeating or unique patterns in the data. We conclude with a critical evaluation of AUTOPLAIT and its implementation, suggestion for further work, and a retrospective review of the project.**

1. INTRODUCTION

HUMAN activity recognition (HAR) is the problem of identifying the exercise, task or action that a person is currently performing by analysing video or motion data. It has a large variety of applications, including improving personal health and security [Mekruksavanich et al., 2022], healthcare [Wang et al., 2019], [Jobanputra et al., 2019] (particularly in care for the elderly [Broek et al., 2010], [Yin et al., 2008]), surveillance [Taha et al., 2015], sports coaching [Shan et al., 2023] and urban planning [Zheng et al., 2014]. HAR has become more common as the cost of various types of personal sensors continues to fall, and their performance continues to increase [Jobanputra et al., 2019]. With the total amount of data in the world expected to reach 394 zettabytes (10^{14} gigabytes) by 2028 [Taylor, 2024], sequential ordered data generated by these personal sensors is becoming more and more common. As such, methods for analysing this kind of temporal information are increasingly important.

Time Series Machine Learning (TSML) specializes in processing and providing insight into this type of data. *aeon*[1] is an efficient, open-source Python toolkit for many TSML tasks including classification, clustering, regression and *segmentation*. Time Series Segmentation (TSS) is an unsupervised machine learning method, in which a series of ordered data points is split into smaller 'parts' or segments, with each segment representing a behaviour or state of the underlying system. It is important to note that TSS does not assign meaning or labels to the segments it identifies. This task is a form of time series *classification*, another important tool in TSML, but out of scope for this project. TSS simply identifies the boundaries of meaningfully distinct segments [Wang et al., 2024].

A. Objectives and description

The primary goal of this project is to implement a segmentation algorithm, namely AUTOPLAIT [Matsubara et al., 2014], that is not currently present in *aeon* into the toolkit. It will be openly available for others to use and expand on, and will work seamlessly with and follow the same API as existing algorithms in *aeon*.

AUTOPLAIT is a 'fully automatic mining algorithm for co-evolving time sequences' [Matsubara et al., 2014]. The reasoning for choosing AUTOPLAIT as the TSS algorithm to implement is two-fold. As stated, the chosen algorithm must not currently exist in *aeon*, which AUTOPLAIT does not. Firstly, it is a highly cited algorithm [Tong et al., 2024], [Ermshaus et al., 2024], [Lai et al., 2024], [Guo et al., 2024] that is commonly used as a benchmark for other TSS algorithms [Ermshaus et al., 2023c]. Secondly, many TSS algorithms require the number of segments as a parameter [Gharghabi et al., 2019], [Sadri et al., 2017], [Truong et al., 2020] and this makes them difficult to use in a HAR setting, where the number of activities (segments) can vary greatly depending on the individual user or the time frame. AUTOPLAIT does not require the number of segments to be specified and therefore is promising as a method for performing HAR.

The algorithm is thoroughly evaluated for the task of human activity recognition using existing HAR datasets and compared to existing segmentation algorithms present in *aeon*.

B. Organization and structure

Within this report, we review the background and related work of human activity recognition and time series segmentation (Section 2), and review popular algorithms used to solve them (Section 3). We discuss AUTOPLAIT and related mathematical tools (Section 4). We evaluate the implementation and performance of AUTOPLAIT against segmentation algorithms currently present in *aeon* on the task of HAR using several datasets (Section 6). Finally, we present results from our experiments and draw conclusions before providing directions for further work (Sections 7 & 8), along with a retrospective review of the project.

2. BACKGROUND WORK

We begin by introducing relevant background information on HAR, techniques used to solve it, and challenges associated with it; in addition to TSML and TSS and their approaches and limitations.

A. Human Activity Recognition

Although HAR is a large problem and involves many steps, it is (ultimately) a classification task. Sensor or image data must be gathered and preprocessed, which may involve noise filtering, outlier detection, and artefact removal. Features can then optionally be extracted or engineered from the data which can then be segmented. Finally, these segments must be labelled and classified into different activities.

Methods for performing HAR broadly fall into three categories; Traditional Machine Learning based, Neural Network based and Time Series Machine Learning based.

1) Approaches: **Traditional Machine Learning** Hidden Markov Models (HMM) and variants [Li et al., 2009], [Fine et al., 1998], [Fox et al., 2009] are a type of statistical model used to describe systems with changing unobservable states over time [Rabiner, 1989]. The system being modelled is assumed to be a Markov process with k hidden states. The Markov property states that a system's future state is independent of it's history; it depends only on it's current

[1] https://www.aeon-toolkit.org/

state [Ross, 2014]. They can successfully be used for HAR [Fallmann and Kropf, 2016], [Kabir et al., 2016], however the Markov assumption can limit their accuracy.

A HMM (shown in figure 1) has the following probabilities: initial hidden state probabilities $\boldsymbol{\pi} = \{\pi_i\}_{i=1}^{k}$, state transition probabilities $\boldsymbol{A} = \{a_{ij}\}_{i,j=1}^{k}$, and emission probabilities $\boldsymbol{B} = \{b_i(\boldsymbol{x})\}_{i=1}^{k}$.

Given a model $\boldsymbol{\theta} = \{\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B}\}$ and an input sequence $\boldsymbol{X}$, the *likelihood function* $P(\boldsymbol{X}|\boldsymbol{\theta})$ is used to compute which state the HMM is in at any given time t ($1 \leq t \leq n$), where n is the length of $\boldsymbol{X}$.

$$P(\boldsymbol{X}|\boldsymbol{\theta}) = \max_{1 \leq i \leq k}\{p_i(n)\}$$
$$p_i(t) = \begin{cases} \pi_i b_i(\boldsymbol{x}_1) & \text{if } t = 1, \\ \max_{1 \leq j \leq k}\{p_j(t-1)a_{ji}\}b_i(\boldsymbol{x}_t) & \text{if } 2 \leq t \leq n. \end{cases} \quad (1)$$

The probability at time t is either the initial probability ($\pi_i b_i(\boldsymbol{x}_1)$) or is dependant on only the probabilities at time $t-1$. HMMs can be trained using a variety of different optimisation algorithms, including the Baum-Welch algorithm.

The Baum-Welch algorithm (Algorithm 1) is a special case of the Expectation-Maximisation (EM) algorithm that optimises the parameters $\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B}$ of a HMM by maximising the likelihood function. The algorithm first (randomly) initialises the parameters $\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B}$[2]. We then compute the forward and backward probabilities (expectation phase) by calculating: 1) the probability of being in state i at time t, given the observed sequence so far X_t, for all states i ($1 \leq i \leq k$), and 2) the probability of transitioning to a different state at the next time step $t+1$, given the observed sequence so far X_t. We use these probabilities to update $\boldsymbol{A}$ and $\boldsymbol{B}$ based on the observed symbols and state transitions (maximisation phase), i.e.,

$$\boldsymbol{A}_{ij} = \frac{T_{ij}}{T_i} \quad \boldsymbol{B}_{jv} = \frac{N_{jv}}{N_j} \quad (2)$$

where:

- T_{ij}: expected number of transitions from state i to state j
- T_i: expected number of transitions from state i
- N_{jv}: expected number of times observing v from state j (state j is '*emitting*' v)
- N_j: expected number of times in state j

We repeat the EM phases until we converge on an optimal solution (i.e., the likelihood function does not change significantly between iterations), or a maximum number of iterations have passed.

k-Nearest Neighbours (kNN) is a traditional supervised ML technique that uses the k closest training instances to predict the class of a new instance [Cover and Hart, 1967]. It can be used for HAR [Paul and George, 2015], however it does require labelled training data which can make gathering data time-consuming, and does not scale well to large datasets.

[2]If there is prior knowledge about the model, this can be used for initialisation instead of randomness.

Algorithm 1: BaumWelch($\mathbf{X}, k$)

Input:
(a) Observations (Time Series), $\mathbf{X}$
(b) Number of hidden states, k

Output:
(a) Initial hidden state probabilities $\boldsymbol{\pi}$
(b) State transition probabilities $\boldsymbol{A}$
(c) Emission probabilities $\boldsymbol{B}$

```
Randomly initialise θ = {π, A, B};
while P(X|θ) is improving do
    /* Expectation Phase */
    Calculate forward probabilities α;
    Calculate backward probabilities β;
    /* Maximisation Phase */
    Update A_ij and B_jv according to α and β; /*
      Equation 2 */
return {π, A, B};
```

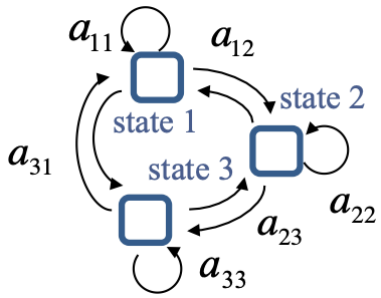


Fig. 1. Illustration of the HMM structure. Note: although not shown, the transitions a_{13}, a_{21}, a_{32} *are* included in the state transition probabilities $\boldsymbol{A}$ and are not necessarily symmetric ($a_{ij} \neq a_{ji}$). (Image source: [Matsubara et al., 2014])

A Support Vector Machine (SVM) is a supervised ML method that is particularly effective for high-dimensional data. It searches for decision boundaries in the space of the input vectors. If it is unable to find a decision boundary (i.e. the data is not linearly separable), it raises the data to a higher dimensional space and searches again. They are useful for HAR as sensor data typically has many channels (dimensions), however like kNN, they scale poorly for larger datasets [Anguita et al., 2013a].

Neural Networks Long Short-Term Memory (LSTM) is a type of neural network that uses 'memory gates' to store some of the information they process. In contrast to HMMs, LSTM networks are capable of remembering information from earlier time steps and can use this to inform their behaviour for future time steps. They can be used for HAR, but have high computational cost, making them ineffective for real-time applications [Singh et al., 2017].

Convolutional Neural Network (CNN) is a type of neural network that are effective for HAR using images or video data rather than sensor input. They can be used for both image and sensor HAR, but require thorough pre-processing which isn't

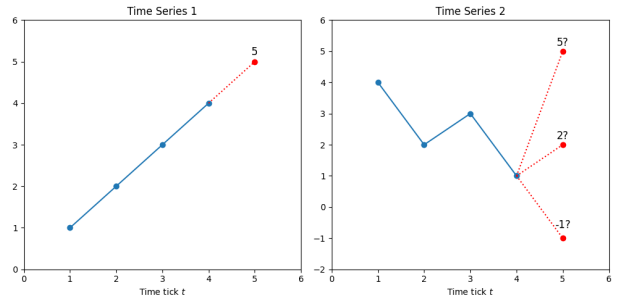


Fig. 2. Two time series with the same data values at different time ticks. TS 1 has a clear next prediction, whereas TS 2 is more difficult to predict.

always possible [Bevilacqua et al., 2019].

Time Series Machine Learning The main characteristic of TSML is temporal order. The order of the data points is equally important, if not more, than the value of the data points themselves. Consider the simple example $\boldsymbol{X} = \{1, 2, 3, 4\}$ for the task of time series regression, as shown in Figure 2. It is reasonably trivial to predict that the next value will be 5. Now consider a different order of the same data, generated by swapping positions 1 and 4, $\hat{\boldsymbol{X}} = \{4, 2, 3, 1\}$; the next value is now much less obvious to predict.

Human activities and tasks are obviously dependent on time. How can we say with any certainty that a user has been sprinting if we have exactly one heart rate reading? TSML is able to leverage the usefulness of temporal order of data readings to perform HAR. We discuss various TSML algorithms later.

2) Challenges: Although the problem of HAR may initially seem trivial, accurately determining the current task a user is performing is challenging for a variety of reasons [Chen et al., 2021].

Firstly, there is often high variability in both the data used to train a HAR approach, and the data of the user being analysed [Chen et al., 2021], [Yadav et al., 2021]. Humans perform the same activities in different ways, and this leads to variation in the data for a given activity. Consider the simple activity of writing. Data gathered from a left-handed participant is likely to be different to that of a right-handed participant, even if they write the same words in the same hand-writing style (though this is difficult to enforce). Additionally, the context in which the user is performing an activity is likely to alter any gathered data (consider the same right-handed user writing leisurely, compared to writing in a timed examination). The variation of data makes it difficult to generalise models outside of specific domains and can affect recognition accuracy, even for activities and users seen during the training process.

Secondly, collecting the large amounts of data required to train HAR models is often resource-intensive [Chen et al., 2021]. The majority of ML models (particularly neural network models) require vast amounts of data, more-so when considering the high variation in user behaviour. Annotating such large quantities of data is time-consuming, and using automatic aides can lead to inaccurate models. Furthermore, large amounts of preprocessing is often required to fuse data from various personal and/or environmental sensors used to gather data. These sources can have varying capture rates (e.g., 100Hz for accelerometer and 30fps for a video camera) making them difficult to integrate into a single feature vector.

Thirdly, many applications of HAR (healthcare, autonomous vehicles [Tammvee and Anbarjafari, 2021]) require real-time processing, which can be very computationally expensive. Certain ML models require large amounts of memory, which is constrained on low-power embedded systems such as healthcare appliances. Larger and more complex models can also take a significant amount of time to analyse new data, and any processing delays can lead to ineffective or dangerous systems.

B. Time Series Segmentation

TSS is a small but crucial step in HAR and, while training is required for the final activity classification, it is not (necessarily) required for segmentation [Wang et al., 2024].

Formally, a time series $\boldsymbol{X}$ is a sequence of ordered data points recorded at regular intervals of length n. TSS divides a TS into distinct, variable-sized, meaningful segments by identifying a set of c change points $\boldsymbol{C}$ where the TS changes behaviour.

$$\boldsymbol{X} = (t_1, t_2, t_3, \ldots, t_n)$$
$$\boldsymbol{C} = (c_1, c_2, \ldots, c_k), \quad 1 \leq b_i \leq n$$

A TS is *univariate* if all data points are real values, i.e., $t_i \in \mathbb{R}, \quad \forall i \in \{1, \ldots, n\}$. In contrast, it is *multivariate* if each data point is a d-dimensional vector, i.e., $t_i \in \mathbb{R}^d, \quad \forall i \in \{1, \ldots, n\}$, where d is the dimensionality of the time series ($d > 1$).

1) Existing algorithms: There are many existing segmentation algorithms that could be used for HAR. Some of the most popular techniques are detailed below.

Classification **S**core **P**rofile (ClaSP) [Ermshaus et al., 2023c] provides a method for segmentation by framing it as a classification problem. It has the following properties: (1) *Domain agnosticism*: the algorithm is not specialised to a particular application, and (2) *Parameter-free*: it requires no (hyper-)parameters to be set by the user.

ClaSP continually applies a binary classifier to the data, aiming to split it in to two segments with a sliding-window approach. These segments are self-similar but dissimilar to each other. The classifications (selected change points) that achieve the highest scores indicate the most probable change points in the data.

Fast **L**ow-cost **O**nline **S**emantic **S**egmentation (FLOSS) [Gharghabi et al., 2019] claims the following properties: (1) *Domain agnosticism*, (2) *Streaming*: it is able to process incoming data at a rate of up to 100 Hz, allowing real-time use, and (3) *No non-segmentable assumption*: it can ignore sections that do not seem to belong to any segment, and can leave them as 'unsegmented'.

FLUSS is an offline version of FLOSS. Both algorithms have only one parameter; the number of change points. It uses the nearest-neighbour principle within a sliding-window to

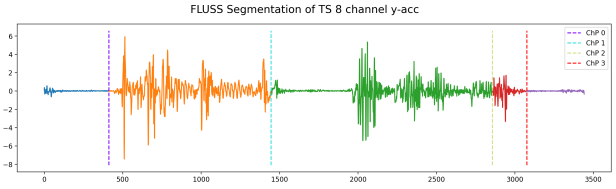


Fig. 3. Example segmentation result of one dimension of a 12-dimensional TS using FLUSS algorithm with $k = 5$ change points. Change points have been identified at $t = \{412, 1446, 2857, 3076\}$. (Data source: [Ermshaus et al., 2023a])

identify potential segments. A given point's nearest neighbour is unlikely to be from a different segment to itself; the points at which the fewest number of neighbours 'cross over' identify change points. Figure 3 shows a TS segmented using FLUSS.

Both ClaSP and FLOSS/FLUSS use a sliding-window approach to include previous data points in their computations. The window width should represent the average or maximum length of a segment in order to include only relevant data points in calculations. However, it can be very difficult to gauge an appropriate window sized based on data alone, and often domain experts are required to suggest suitable values for window width. Both algorithms are discussed in further detail in section 3.

Information **G**ain-based **T**emporal **S**egmentation (IGTS) [Sadri et al., 2017] has the objective of *Minimising entropy*: the segments found provide the largest amount of information. IGTS segments data based on the amount of information gained using Shannon entropy. It has a single parameter; the number of change points to search for. It uses a greedy approach to find change points that provide the most information gain, and continues until it has found the requested number of change points.

C. aeon TSML Toolkit

The *aeon* toolkit has been developed and maintained since January 2023, and since December 2023 it has been an affiliated project of numFOCUS; a non-profit organisation promoting reliable open-source scientific and technical computing tools. *aeon* is primarily used for classification and regression [Middlehurst et al., 2024], [Spinnato et al., 2024], [Guijo-Rubio et al., 2024], and has a wide variety of algorithms for these tasks inspired by various types of machine learning technologies, such as deep learning, shapelets, convolution and more.

Eight TSS algorithms are currently present in *aeon*; 1) ClaSP, 2) FLUSS, 3) IGTS, 4) E-Agglo [Matteson and and, 2014], 5) GGS [Hallac et al., 2019], 6) Hidalgo [Allegra et al., 2020], 7) BinSeg [Truong et al., 2020], and 8) HMM.

D. Challenges and Limitations

With the growing amount of data, labelling entire datasets is becoming more unreasonable and time-consuming; the need for unsupervised methods is clear. Evaluating different methods of segmentation is difficult due to the lack of accepted performance metrics [Lin et al., 2016]. Metrics for other ML tasks (accuracy, recall, precision, F_1, etc.) do apply to segmentation tasks, but are not always appropriate. For example, if a segmentation algorithm identifies change points that are one time-tick off for every segment (e.g. predicts 61 and 109 for ground truths 60 and 110), is that considered a failure, or is it within a certain 'tolerance' to be considered accurate? [Gharghabi et al., 2019] attempt to bridge this gap by introducing a scoring metric for segmentation.

1) Segmentation metric: To combat the brittleness of existing scoring metrics, Gharghabi et al. introduce a metric that scores the distance between a ground truth change point and the closest change point suggested by a segmentation algorithm (Algorithm 2).

Algorithm 2: ScoreSegmentation(*predChanges*, *trueChanges*, n)

Input:
(a) Extracted change points, *predChanges*
(b) Ground truth change points, *trueChanges*
(c) Length of time series, n

Output: Score: [0,1], with 1 being the best score

```
sumDiff = 0;
numChanges = length(trueChanges);
for i = 1 : numChanges do
    Find the trueChanges[j] closest to predChanges[i];
    diff = |predChanges[i] − trueChanges[j]|;
    sumDiff = sumDiff + diff;
score = 1 − sumDiff/n;
return score;
```

In addition to the issue of scoring metrics, many current algorithms for segmentation require the number of segments as a parameter [Gharghabi et al., 2019], [Sadri et al., 2017], [Truong et al., 2020]. It is very difficult to estimate the number of segments (activities) for an HAR task, meaning these algorithms often underperform. Many neural network approaches to HAR are also computationally expensive, meaning they cannot be used on low-power, memory-constrained embedded systems, preventing them from being used for real-time applications.

Additionally, numerous datasets must be used for evaluation of any TSS algorithm. The choice of datasets used to evaluate segmentation algorithms form an important part of their design. [Gharghabi et al., 2019] used 32 datasets to measure the performance of FLUSS, and [Ermshaus et al., 2023c] used 107 datasets to evaluate ClaSP. In general, at least three datasets should be used to test segmentation algorithms [Ermshaus et al., 2023c]. HAR datasets are less common than general-purpose segmentation datasets (3,343 results for 'segmentation' on kaggle[3], compared to only 339 for 'human activity') and the majority of these HAR datasets are designed for classification tasks rather than segmentation tasks.

[3]https://www.kaggle.com/datasets

E. Summary

HAR is a challenging task that benefits from accurate TSS algorithms to provide insight for a variety of different applications. A variety of ML approaches can be used to solve it, and TSML approaches can make use of the temporal order of data to gain additional information in hopes of providing a better segmentation. TSS algorithms often use a sliding-window approach to include previous data points, however picking an appropriately sized window can be challenging or require domain-specific knowledge.

This project focuses on implementing a new algorithm to the *aeon* toolkit. To address the challenges of TSS in HAR, the chosen algorithm will not require the number of segments as a parameter, allowing it to be used for HAR tasks without the need of prior knowledge on the number of activities. In the next chapter, we discuss the ClaSP and FLUSS algorithms in further detail.

3. Current Approaches

As previously mentioned, ClaSP and FLUSS are popular algorithms for performing TSS, and both employ different techniques to identify segments.

A. ClaSP

ClaSP reduces the TSS problem into a Time Series *Classification* problem. For each index i in a given TS $\boldsymbol{X}$, we compute 'how well' the sub-TS to the left of i differs from the sub-TS to the right.

1) Left/right split: The TS is partitioned into overlapping windows of size w ($[w+1, \ldots, n-w-1]$) which are used as candidate boundary points. For a given candidate point, all windows to the left of it (less than) are assigned the label 0, and all windows to the right (greater than) are assigned 1.

2) Binary classification: A binary kNN classifier is trained on the labelled windows and their features from the original TS. Many types of classifiers (most more complex than kNN) were trialled by the original authors. Not only did a simple kNN (specifically, 3-NN) classifier perform better than more complex classifiers, it is orders of magnitude faster. Full results are presented in the original paper [Ermshaus et al., 2023c].

The performance of the classifier is measured using k-Fold Cross-Validation, specifically 5-fold. The result of the validation is used as the classification score for the given index. As such, the ClaSP for the w indices at the edges of the TS will remain as 0. A higher ClaSP value at index i means that, splitting the TS at i provides left/right sides that are self-similar, but dissimilar from the other. An example ClaSP and TS can be seen in Figure 4.

3) Extracting change points: Given a ClaSP, we can extract change points using a threshold-based approach to validate if local maxima in the ClaSP are indeed change points. The details of this approach are omitted, and the interested reader is referred to the original paper [Ermshaus et al., 2023c]. An overview of the ClaSP procedure can be seen in Algorithm 3.

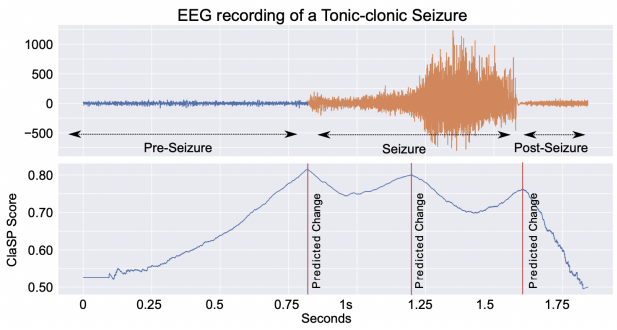


Fig. 4. Example TS and it's ClaSP. Global maxima have been identified as potential change points at $t \approx 0.8, 1.2, 1.6$s (Image source: [Ermshaus et al., 2023c])

Algorithm 3: ClaSP($\boldsymbol{X}, w$)

Input:
(a) Time Series, $\boldsymbol{X}$
(b) Subsequence length, w

Output:
(a) Classification Score Profile, $ClaSP$

```
n = length(X);
ClaSP = 0-initialised array of length n;
Y_t = 1-initialised array of length n;
W = Windows(X, w);
kNNs = KNN_Profile(W);
foreach i in [w + 1, ..., n − w − 1] do
    Y_t[i − w] = 0; /* Left window labelled as 0 */
    ClaSP[i] = Cross_Validate(kNNs, Y_t);
return ClaSP;
```

B. FLUSS

FLUSS (and later, FLOSS, for online streaming capability) identifies potential change points by producing a companion, or 'dual', TS called an *Arc Curve* (AC). The AC can then be analysed to identify (user-defined) k change points in the original TS.

1) The Arc Curve: The AC is computed using Matrix Profile Indices (*MPIndex*) which uses the Fast Fourier Transform (FFT) and a user-defined subsequence length to find the index of a subsequence that best matches the current subsequence. The identified *MPIndex*s can be visualised as 'arcs' from one subsequence to it's nearest neighbour (as shown in Figure 5). To allow multiple occurrences of the same type of segment to appear across the TS (e.g., *walking*, *running*, *walking*), nearest neighbours are only identified within a user-defined window, which should represent the average expected length of segment. Every index of the *MP* will have exactly one outbound arc, and zero-to-many in-bound arcs. The AC contains the number of arcs that spatially cross over (or above) every index i of the *MPIndex*.

2) Intuition: Suppose a TS has a change point at index i. We would expect few arcs to cross at i, as most subsequences will find their nearest neighbour *within* their own segment. We can find minima of the AC to identify where the fewest arcs

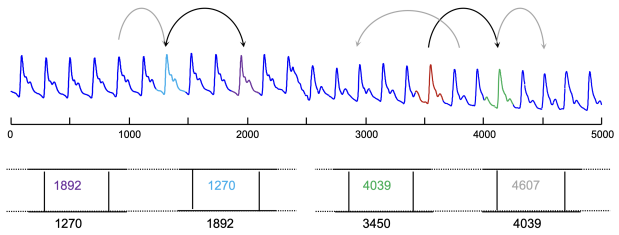
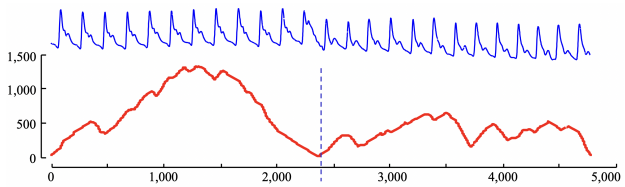
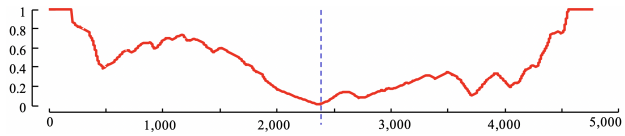


Fig. 5. Arcs from a selected subsequences to their nearest neighbour. Nearest neighbour indices can be symmetric (e.g., 1270 and 1892) but this is not generally true. An arc crossing is visible at $t \approx 3625$ (Image source: [Gharghabi et al., 2019])



(a) A TS (blue) and its corresponding AC (red).



(b) CAC of the above AC.

Fig. 6. Comparison of AC and CAC. Artificially lower crossings are visible at the edges of the AC. The CAC shows adjusted edges based on the IAC. The identified change point ($t = 2400$) remains unchanged. (Image source: [Gharghabi et al., 2019])

cross, indicating likely change points. To address the bias at the start and end of the AC, where there are fewer arcs that can cross above indices due to edges, we compare the number of observed arc crossing AC_i to the expected number of arc crossings of a completely random TS, IAC_i (the Idealised Arc Curve). The IAC forms a parabolic curve of length n and height $\frac{1}{2}n$, allowing us to compute the Corrected Arc Curve (CAC) (Algorithm 4). A comparison of an AC and it's corresponding CAC is shown in Figure 6.

3) Extracting change points: Extracting change points is as simple as identify global minima of the CAC. The number of minima to identify (and thus, the number of change points) is defined by a user parameter k. To prevent self-matches (that is, given the global minimum is at index i, the next global minimum is most likely to be at $i \pm 1$), an exclusion zone is applied, based on the subsequence length L.

The authors of the original paper go on to explain how to adapt FLUSS for online processing (FLOSS), in addition to a method to automatically determine an appropriate subsequence length L. The interested reader is referred to the original paper [Gharghabi et al., 2019].

C. Benchmarking

The techniques mentioned above are not the only methods of performing TSS, let alone HAR. The European Conference on Machine Learning (ECML) presented a challenge on HAR [Ermshaus et al., 2023a] with the aim of increasing the performance of human activity segmentation algorithms and highlighting the improvements needed for TSS to become an integral part of the HAR workflow. We recreate the results of the challenge to (1) provide a comparison benchmark and evaluation of the above approaches, in addition to two other algorithms, and (2) become familiar with the process of creating time series segmentation models with clear, existing indicators of success. We compare results with those published by the organisers.

Algorithm 4: CAC(***MPIndex***, ***L***)

Input:
(a) Matrix Profile Indices, ***MPIndex***
(b) Subsequence length, ***L***

Output:
(a) Corrected Arc Curve, CAC

```
n = length(MPIndex);
AC = CAC = nnmark = 0-initialised array of length L;
for i = 1 : L do
    j = MPIndex[i];
    nnmark[min(i,j)] = nnmark[min(i,j)] + 1;
    nnmark[max(i,j)] = nnmark[max(i,j)] - 1;
numArcs = 0;
for i = 1 : L do
    numArcs = numArcs + nnmark[i];
    AC[i] = numArcs;
IAC = parabolic curve of length n and height 1/2 n;
CAC = min(AC/IAC, 1);
Set the first and last L indices of CAC to 1;
return CAC;
```

1) Human Activity Segmentation Challenge: The challenge was to predict the location and number of boundaries in the provided 250 multivariate TS, without training or external data. The organisers provide an exploratory data analysis using five algorithms commonly used for segmentation: 1) ClaSP, 2) FLUSS (offline version of FLOSS), 3) IGTS, 4) BinSeg, and 5) GGS. The competition did not publish results for IGTS or GGS.

All of the algorithms (with the exception of ClaSP) require the number of segments as a parameter. As this was not specified for each TS, predictions were made for $n = \{2, \ldots, 6\}$ segments, and the best segmentation was taken to be the final prediction. Results are summarised in Table I[4].

4. Final Approach — AutoPlait

AutoPlait works on "*co-evolving* time series"; where each individual time series (that is, each feature or dimension of a multivariate TS) is not necessarily independent of other time series (or features). This is particularly powerful for HAR, as it enables us to analyse data from sensors placed at different parts of the body. For example, if a user claps

[4]The author of the competition suggested against the use of the metric used for the challenge, instead recommending Covering and F_1 score which was used to rank the reproduced results.

TABLE I
COMPARISON OF ALGORITHMS WITH PUBLISHED AND REPRODUCED RESULTS.

Algorithm	Competition Rank	F_1	Covering
ClaSP	1st	62.9	74.4
BinSeg	2nd	59.6	81.4
FLUSS	3rd	51.2	71.8
IGTS	N/A	45.0	67.3
GGS	N/A	67.8	85.2

their hands, the right hand time series will affect the left hand time series and vice versa.

The most valuable property of AUTOPLAIT is that it does not require parameter tuning. Many current algorithms (most of the algorithms demonstrated in 3.3.1) require the number of segments or a subsequence length as a parameter for segmentation. Although this may seem like a small and simple additional piece of data, it can be challenging to manually annotate a TS with the number of segments. AUTOPLAIT is able to automatically identify the number of segments and *regimes* within a time series.

A. Definitions

The original AUTOPLAIT paper refers to multivariate time series as a 'bundle'. To avoid confusion, we use the term multivariate time series (or simply TS) throughout the report.

Given a multivariate TS $\mathbf{X}$, we want to convert $\mathbf{X}$ into a set of m non-overlapping segments $\mathcal{S} = \{s_1, s_2, \ldots, s_m\}$ where $s_i = \{t_s, t_e\}$ consists of the start and end time-ticks of the segment. We also want to find a set of distinct overarching patterns or *regimes* of segments, such that each segment belongs to a regime.

Definition 1 (Regime). *Let $\boldsymbol{\theta}_i (i = 1, 2, \ldots, r)$ be a statistical model that represents a regime or group of segments, where r is the desired number of regimes.* [Matsubara et al., 2014]

A regime represents an activity that a human is performing, such as hand washing, cooking, running, etc. AUTOPLAIT allows *any* statistical model to be used as a regime, however, HMMs are used for their simplicity.

Definition 2 (Segment-membership). *Let $\mathcal{F} = \{f_1, f_2, \ldots, f_m\}$ be a list of m integers, where f_i is the regime of the i-th segment ($1 \leq f_i \leq r$).* [Matsubara et al., 2014]

Every segment belongs to exactly one regime, but a regime can have many segments belonging to it. In the context of HAR, a segment is a specific instance of a specific activity, and a regime is all instances of that activity. Consider the sequence of activities '*Wash hands*', '*Cook*', '*Wash hands*'; there are two regimes and three segments. Segments 1 and 3 belong to the same regime but are different instances of the hand washing activity (i.e., $\mathcal{F} = \{1, 2, 1\}$), and could last for different periods of time.

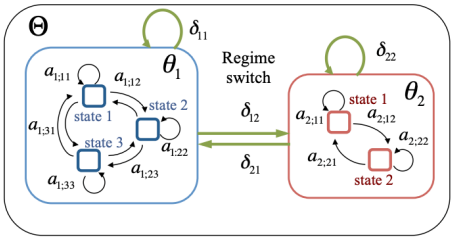


Fig. 7. Illustration of a multi-level transition model with $r = 2$ regimes (blue and red). The black and green arrows indicate intra-regime ($\mathbf{A}$) and inter-regime transitions ($\boldsymbol{\Delta}$), respectively. (Image source: [Matsubara et al., 2014])

B. Problem

Given a multivariate TS $\mathbf{X}$, we aim to find a compact description that best summarises $\mathbf{X}$, that is, 1) determine the number m of segments and their positions ($\mathcal{S} = \{s_1, s_2, \ldots, s_m\}$), 2) determine the number r of regimes and their segment memberships ($\mathcal{F} = \{f_1, f_2, \ldots, f_m\}$), and 3) estimate the model parameters of the r distinct regimes ($\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_r, \boldsymbol{\Delta}_{r \times r}\}$). The set of model parameters $\boldsymbol{\Theta}$ consists of the parameters of each regime (HMM), plus an additional *regime transition matrix* $\boldsymbol{\Delta}_{r \times r}$.

Matsubara et al. introduce multilevel transitions to HMMs through the use of a *regime transition matrix* $\boldsymbol{\Delta}$ and a multilevel chain model (MLCM). The MLCM groups states of HMMs into regimes, groups regimes into super-regimes, etc. This allows for *intra-regime* transitions using the state transition probabilities $\mathbf{A}$ and *inter-regime* transitions using the regime transition matrix $\boldsymbol{\Delta}$ (see Figure 7).

Definition 3 (Regime Transition Matrix). *Let $\boldsymbol{\Delta}_{r \times r}$ denote a transition probability matrix of r regimes, where each element $\delta_{ij} \in \boldsymbol{\Delta}$ is the regime transition probability from the i-th regime to the j-th regime.* [Matsubara et al., 2014]

A regime transition matrix $\boldsymbol{\Delta}$ shares the same properties as a state transition matrix $\mathbf{A}$: all entries are probabilities and therefore are between 0 and 1, and all rows sum up to 1.

C. Algorithm

The primary goal is to find a complete set of parameters $\mathcal{C} = \{m, r, \mathcal{S}, \boldsymbol{\Theta}, \mathcal{F}\}$ that best describes a multivariate TS $\mathbf{X}$. There are two parts to this problem: first, how do we find these parameters, and secondly, how can we tell how 'good' a set of parameters is?

1) Cost functions: A cost function is a way to measure how 'expensive' or 'good' a certain value is. In machine learning, cost functions are used to optimise models by finding the minimum cost. The cost functions used in AUTOPLAIT describe the 'goodness' of a candidate solution $\mathcal{C}$ when applied to TS $\mathbf{X}$ as $Cost_T(\mathbf{X}; \mathcal{C}) = Cost(\mathcal{C}) + Cost(\mathbf{X}|\mathcal{C})$, where $Cost(\mathcal{C})$ is the cost of describing the candidate solution $\mathcal{C}$ and $Cost(\mathbf{X}|\mathcal{C})$ is the cost of using candidate solution $\mathcal{C}$ to fully describe $\mathbf{X}$. A better solution (that is, a solution that more accurately describes $\mathbf{X}$) will have a lower cost.

2) Solution description cost: A candidate solution $\mathcal{C}$ is a set of parameters $\{m, r, \mathcal{S}, \boldsymbol{\Theta}, \mathcal{F}\}$, the cost of which is described below.

- The number of time-ticks n and the number of dimensions d require $\log^*(n) + \log^*(d)$ bits.[5]
- The number of segments m and the number of regimes r require $\log^*(m) + \log^*(r)$ bits.
- The segment-membership $\mathcal{F}$ of the r regimes requires $m \log(r)$ bits.
- The length of each segment s (in the segment set $\mathcal{S}$) requires $\sum_{i=1}^{m-1} \log^* |s_i|$ bits, where $|s_i|$ is the size of segment s_i.
- The model parameters $\boldsymbol{\Theta}$ require $Cost_M(\boldsymbol{\Theta})$ bits,

$$Cost_M(\boldsymbol{\Theta}) = \sum_{i=1}^{r} Cost_M(\boldsymbol{\theta}_i) + Cost_M(\boldsymbol{\Delta}) \quad (3)$$

A single HMM requires $\log^*(k)$ bits for the number of hidden states k, plus the model cost $\theta = \{\boldsymbol{\pi}, \mathbf{A}, \mathbf{B}\}$,

$$Cost_M(\boldsymbol{\theta}) = \log^*(k) + c_F \cdot (k + k^2 + 2kd) \quad (4)$$

where c_F is the fixed cost of representing a number as a floating point. Similarly, the cost of a regime transition matrix is $Cost_M(\boldsymbol{\Delta}) = c_F \cdot r^2$. Thus,

$$Cost(\mathcal{C}) = \log^*(n) + \log^*(d) + \log^*(m) + \log^*(r) + m\log(r) + Cost_M(\boldsymbol{\Theta}) \quad (5)$$

3) Data encoding cost: After we have the cost of a given candidate solution, we can calculate the cost of using that solution to describe a TS $\boldsymbol{X}$. Using Huffman coding, we assign a number of bits to each value in $\boldsymbol{X}$, which is the logarithm of the inverse of probability, i.e., the *negative log-likelihood* (see Equation 1). Given an individual regime $\boldsymbol{\theta}$, the cost of encoding $\boldsymbol{X}$ using $\boldsymbol{\theta}$ is:

$$Cost_E(\mathbf{X}|\boldsymbol{\theta}) = \log_2 \frac{1}{P(\mathbf{X}|\boldsymbol{\theta})} = -\ln P(\mathbf{X}|\boldsymbol{\theta}) \quad (6)$$

Given the model parameters of all r regimes of the candidate solution, and the regime transition matrix (i.e., $\boldsymbol{\Theta}$), the total cost of encoding the data in the TS is:

$$\begin{aligned} Cost_E(\mathbf{X}|\boldsymbol{\Theta}) &= \textstyle\sum_{i=1}^{m} Cost_E(\mathbf{X}[s_i]|\boldsymbol{\Theta}) \\ &= \textstyle\sum_{i=1}^{m} -\ln(\delta_{vu} \cdot (\delta_{uu})^{|s_i|-1} \cdot P(\mathbf{X}[s_i]|\boldsymbol{\theta}_u)) \end{aligned} \quad (7)$$

where the i-th and $(i-1)$-th segments are governed by the u-th and v-th regime, respectively (i.e. $f_i = u$ and $f_{i-1} = v$). $\mathbf{X}[s_i]$ is the 'sub-TS' of a segment s_i. A segment s_i consists of start and end time ticks t_s, t_e, and thus $\mathbf{X}[s_i]$ is simply all data points between the start and end of segment s_i (inclusive).

Combining the cost of encoding a solution with the cost of using that solution to describe and encode a TS gives the final cost function $Cost_T(\mathbf{X}; \mathcal{C})$:

[5] $\log^*$ is the iterated logarithm, defined as $\log^*(x) \approx \log_2(x) + \log_2 \log_2(x) + \dots$, where only positive terms are included in the sum [Rissanen, 1983].

TABLE II
SUMMARY OF SYMBOLS AND DEFINITIONS

Symbol	Definition
For Time Series	
n	Number of time-ticks (length)
d	Number of dimensions
$\mathbf{X}$	Multivariate d-dim TS
For Segments	
m	Number of segments in $\mathbf{X}$
$\mathcal{S}$	Segment set in $\mathbf{X}$, i.e., $\mathcal{S} = \{s_1, \dots, s_m\}$
$\mathcal{F}$	Segment-membership, i.e., $\mathcal{F} = \{f_1, \dots, f_m\}$
For Regimes	
r	Number of regimes in $\mathbf{X}$
$\boldsymbol{\Theta}$	Model parameters of r regimes, i.e., $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_r, \boldsymbol{\Delta}_{r\times r}\}$
$\boldsymbol{\theta}_i$	Model parameters governing i-th regime
k_i	Number of hidden states in $\boldsymbol{\theta}_i$
$\boldsymbol{\Delta}_{r\times r}$	Regime transition matrix, i.e., $\boldsymbol{\Delta} = \{\delta_{i,j}\}_{i,j=1}^{r}$
For Cost Function	
$\mathcal{C}$	Candidate solution, i.e., $\mathcal{C} = \{m, r, \mathcal{S}, \boldsymbol{\Theta}, \mathcal{F}\}$
$Cost_M(\boldsymbol{\Theta})$	Model description cost of $\boldsymbol{\Theta}$
$Cost_E(\mathbf{X}\|\boldsymbol{\Theta})$	Encoding cost of $\mathbf{X}$ given $\boldsymbol{\Theta}$
$Cost_T(\mathbf{X}; \mathcal{C})$	Total cost of $\mathbf{X}$ given $\mathcal{C}$

$$Cost_T(\mathbf{X}; \mathcal{C}) = \log^*(n) + \log^*(d) + \log^*(m) + \log^*(r) + m\log(r) + Cost_M(\boldsymbol{\Theta}) + Cost_E(\mathbf{X}|\boldsymbol{\Theta}) \quad (8)$$

4) Parameter search: AUTOPLAIT uses three algorithms to search for the optimal (least cost) set of parameters, one of which is the main algorithm itself. Firstly, 'CutPointSearch' (algorithm 5) is used to find optimal cut points (ticks at which the series changes regime) to create two *sets* of segments $\mathcal{S}_1$ and $\mathcal{S}_2$. Next, 'RegimeSplit' (algorithm 6) estimates good regime (HMM) parameters $\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2$ and $\boldsymbol{\Delta}$ for two regimes. Finally, 'AutoPlait' (algorithm 7) is used to search for the best number of regimes $(2, 3, 4, \dots)$. The results of each of these algorithms are used to produce the final set of parameters $\mathcal{C}$.

5. IMPLEMENTATION

AUTOPLAIT at its core is a probabilistic, statistical approach to TSS. In this section we discuss equations used to calculate the probabilities used in the algorithm and the Python and C code that form the concrete implementation.

A. Mathematical tools

As discussed in the previous section, the first step in AUTOPLAIT is to identify time ticks at which a TS changes between exactly *two* regimes, such that the resulting parameters have the least possible cost.

Given a TS $\boldsymbol{X}$, two regimes $\boldsymbol{\theta}_1 = \{\boldsymbol{\pi}_1, \boldsymbol{A}_1, \boldsymbol{B}_1\}$, $\boldsymbol{\theta}_2 = \{\boldsymbol{\pi}_2, \boldsymbol{A}_2, \boldsymbol{B}_2\}$ and a regime transition matrix $\boldsymbol{\Delta}_{2\times 2} = \{\boldsymbol{\delta}_{11}, \boldsymbol{\delta}_{12}, \boldsymbol{\delta}_{21}, \boldsymbol{\delta}_{22}\}$, the likelihood value $P(\boldsymbol{X}|\boldsymbol{\Theta})$ can be computed similarly to Equation 1:

$$P(\mathbf{X}|\boldsymbol{\Theta}) = \max \begin{cases} \max_{1\le i\le k_1}\{p_{1;i}(n)\} & \text{// Regime } \boldsymbol{\theta}_1 \\ \max_{1\le u\le k_2}\{p_{2;u}(n)\} & \text{// Regime } \boldsymbol{\theta}_2 \end{cases} \quad (9)$$

The likelihood value of a TS, given model parameters of two regimes, is the state of either regime (state i for $\boldsymbol{\theta}_1$ or state u for $\boldsymbol{\theta}_2$) with the highest probability. $p_{1;i}(n)$ is the maximum probability of state i at time $t = n$ (that is, the end of the TS) for regime $\boldsymbol{\theta}_1$, and similarly for state u of regime $\boldsymbol{\theta}_2$ for $p_{2;u}(n)$. The maximum probabilities are computed as follows:

$$p_{1;i}(t) = \max \begin{cases} \delta_{21} \cdot \max_v\{p_{2;v}(t-1)\} \cdot \pi_{1;i} \cdot b_{1;i}(\boldsymbol{x}_t) \\ \quad \text{// Regime switch } \boldsymbol{\theta}_2 \rightarrow \boldsymbol{\theta}_1 \\ \delta_{11} \cdot \max_j\{p_{1;j}(t-1) \cdot a_{1;ji}\} \cdot b_{1;i}(\boldsymbol{x}_t) \\ \quad \text{// Stay in regime } \boldsymbol{\theta}_1 \end{cases} \tag{10}$$

At a given time t, the maximum probability of regime $\boldsymbol{\theta}_1$ is either the probability the regime switched from regime $\boldsymbol{\theta}_2$ at time $t-1$ (first case in Equation 10) or remained in regime $\boldsymbol{\theta}_1$ at $t-1$ (second case), multiplied by the corresponding regime transition probability $\boldsymbol{\delta}_{21}$ or $\boldsymbol{\delta}_{11}$. Regime $\boldsymbol{\theta}_2$ can be computed similarly.

As an initial setting, the base cases at $t = 1$ are set to:

$$\begin{aligned} p_{1;i}(1) &= \delta_{11} \cdot \pi_{1;i} \cdot b_{1;i}(\boldsymbol{x}_1) \\ p_{2;u}(1) &= \delta_{22} \cdot \pi_{2;u} \cdot b_{2;u}(\boldsymbol{x}_1) \end{aligned} \tag{11}$$

where $\boldsymbol{x}_1$ is the first element in the TS $\boldsymbol{X}$.

To identify potential cut points in the TS $\mathcal{L} = \{l_1, l_2, \ldots, l_{m-1}\}$ where m is the number of segments and l_i is the i-th cut point ($1 \leq l_i \leq n$), we can simply keep track of time ticks t where the regime switches, as computed in Equation 10. For each state of both regimes, during the likelihood computation we have:

$$\mathcal{L}_{1;i}(t) = \begin{cases} \mathcal{L}_{2;v}(t-1) \cup \{t\} & \text{if switch } \boldsymbol{\theta}_2 \rightarrow \boldsymbol{\theta}_1 \\ \mathcal{L}_{1;j}(t-1) & \text{if stay at } \boldsymbol{\theta}_1 \end{cases} \tag{12}$$

$$\mathcal{L}_{2;u}(t) = \begin{cases} \mathcal{L}_{1;j}(t-1) \cup \{t\} & \text{if switch } \boldsymbol{\theta}_1 \rightarrow \boldsymbol{\theta}_2 \\ \mathcal{L}_{2;v}(t-1) & \text{if stay at } \boldsymbol{\theta}_2 \end{cases} \tag{13}$$

where $\mathcal{L}_{1;i}(t)$ are the cut points for regime $\boldsymbol{\theta}_1$ as suggested by state i at time tick t (and similarly for $\mathcal{L}_{2;u}(t)$ for regime $\boldsymbol{\theta}_2$) and are progressively updated accordingly to the likelihood computations in Equation 10. At time $t = n$, we select the best cut point set $\mathcal{L}_{best}$, from $\mathcal{L}_{1;i}(n)$ and $\mathcal{L}_{2;u}(n)$ for all states i and u, such that $P(\boldsymbol{X}|\boldsymbol{\Theta})$ is maximised. Algorithm 5 shows the detailed procedure for cut point search.

So far, we have assumed that the model parameters $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta}\}$, were given to search for the best cut points available. How can we estimate suitable parameters to describe the two different regimes? The goal of Algorithm 6 is to find model parameters $\boldsymbol{\Theta}$ that minimise the cost of modelling the entire TS as described in Equation 8. The problem can be solved using a iterative two-phase approach, i.e., Phase 1: Find the cut points of segments using Algorithm 5 to obtain two segment groups $\{\mathcal{S}_1, \mathcal{S}_2\}$ using model parameters $\boldsymbol{\Theta}$, and Phase 2: Update model parameters $\boldsymbol{\Theta}$, using the Baum-Welch method (see Algorithm 1), based on the new segmentation $\{\mathcal{S}_1, \mathcal{S}_2\}$.

Algorithm 5: CutPointSearch($\mathbf{X}, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta}$)

Input:
(a) Time Series, $\mathbf{X}$
(b) Model parameters of two regimes $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta}\}$

Output:
(a) Number of segments assigned to each regime m_1, m_2
(b) Segment sets of each regime $\mathcal{S}_1, \mathcal{S}_2$

```
for t = 1 : n do
    Compute p_{1;i}(t) for state i = 1, ..., k_1; /* Equations 10 & 11 */
    Compute p_{2;u}(t) for state u = 1, ..., k_2; /* Equations ?? & 11 */
    Update L_{1;i}(t) for state i = 1, ..., k_1; /* Equation 12 */
    Update L_{2;u}(t) for state u = 1, ..., k_2; /* Equation 13 */
Choose the best cut point set L_best;
t_s = 1;
foreach cut point l_i in L_best do
    Make new segment s_i = {t_s, l_i};
    if i is odd then
        Add s_i to S_1; m_1 = m_1 + 1;
    else
        Add s_i to S_2; m_2 = m_2 + 1;
    t_s = l_i;
return {m_1, m_2, S_1, S_2};
```

The model parameters $\boldsymbol{\Theta}$ must be initialised before beginning this iterative approach. A naive approach would be to randomly segment the TS (i.e., randomly select start/end points for segments), however this may converge on a local minimum of the cost function depending on the initial random selection. A more robust approach is to uniformly take several sample segments from a TS $\boldsymbol{X}$. For each sampled segment s, estimate the model parameters $\boldsymbol{\theta}_s$. We can then find the pair of segment parameters $\{\boldsymbol{\theta}_{si}, \boldsymbol{\theta}_{sj}\}$ that minimises the coding cost of the entire TS.

$$\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2\} = \arg\min_{\theta_{si}, \theta_{sj} | si, sj \in \mathcal{X}} Cost_E(\boldsymbol{X}|\boldsymbol{\theta}_{si}, \boldsymbol{\theta}_{sj}), \tag{14}$$

where $\mathcal{X} = \{s_1, s_2, \ldots\}$ is the set of sampled segments taken from $\boldsymbol{X}$.

As mentioned in Section 2, the Baum-Welch algorithm requires the number of hidden states k as a parameter for each model $\boldsymbol{\theta}$. Manually tuning and adjusting k by hand is a difficult task; a k that is too small provides a poor representation of the data (known as under-fitting), and k too large does not generalise well to unseen data (known as over-fitting). By varying $k = 1, 2, 3, \ldots$, we can determine the optimal number of hidden states such that the cost function is minimised, i.e., $Cost_M(\boldsymbol{\theta}) + Cost_C(\boldsymbol{X}[\boldsymbol{\mathcal{S}}]|\boldsymbol{\theta})$.

Finally, we generalise our approach to more than two regimes using a stack-based optimisation approach. Given a TS $\boldsymbol{X}$, we can create a trivial segmentation that is simply

Algorithm 6: RegimeSplit($\mathbf{X}$)

Input: Time Series, $\mathbf{X}$
Output:
(a) Number of segments assigned to each regime m_1, m_2
(b) Segment sets of two regimes $\mathcal{S}_1, \mathcal{S}_2$
(c) Model parameters of two regimes $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta}\}$

Initialise models $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$; /* Equation 14 */
while *cost is improving* **do**
  /* Find segments (Phase 1) */
  $\{m_1, m_2, \mathcal{S}_1, \mathcal{S}_2\}$ = CutPointSearch($\mathbf{X}, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta}$);
  /* Update parameters (Phase 2) */
  $\boldsymbol{\theta}_1 = \text{BaumWelch}(\boldsymbol{X}[\mathcal{S}_1])$;
  $\boldsymbol{\theta}_2 = \text{BaumWelch}(\boldsymbol{X}[\mathcal{S}_2])$;
  Update regime transitions $\boldsymbol{\Delta}$;
return $\{m_1, m_2, \mathcal{S}_1, \mathcal{S}_2, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta}\}$;

the entirety of $\boldsymbol{X}$ and place it on to the stack. We iteratively pop a regime from the stack and attempt to split it into two further regimes based on the cost of modelling the single larger regime against the total cost of the two smaller regimes. We add both smaller regimes to the stack if their combined cost is less than that of the larger regime they were generated from. If it is cost-effective to keep the single regime, it is added to the optimal set of regimes and the regime transition matrix $\boldsymbol{\Delta}$ is updated. AUTOPLAIT terminates once the stack is exhausted (Algorithm 7).

B. Python-specific details

[Matsubara et al., 2014] provide a C implementation of AUTOPLAIT. We employ the use of large language models (LLMs) to aid in the code translation to Python in a three-phase approach; 1) convert C code to equivalent Python code, 2) optimise Python code using vectorisation libraries (`numpy`, `numba`) and memoisation where possible, and 3) make code more Pythonic, removing C translation artifacts and link with *aeon* interface. Example segmentations generated by the final Python implementation are shown in Figure 8.

1) Literal C translation: Being based on C, Python shares many similarities in both code syntax and structure. The first step is to convert the original C implementation of AUTOPLAIT into a literal Python translation and then compare *functional* equivalence between the two implementations. Using a small subset of time series from the datasets as a test suite, we produce functionally equivalent Python code which is inefficient, messy, and difficult to maintain.

Both implementations predict the change points for a random time series from each dataset, and their computation time is recorded and summarised in Table III. The identified change points from each language are within 20 time ticks of each other, likely due to the different algorithms used to generate pseudo-random numbers in both C and Python. Similarly, the final cost of the optimal models are close for each implementation. The number of identified segments and change points is equal for both languages.

Algorithm 7: AutoPlait($\mathbf{X}$)

Input: Time Series, $\mathbf{X}$
Output: Full set of parameters $\mathcal{C}$, i.e.,
(a) Number of segments, m
(b) Number of regimes, r
(c) Segment set, $\mathcal{S} = \{s_1, \ldots, s_m\}$
(d) Model parameters of regimes, $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_r, \boldsymbol{\Delta}\}$
(e) Segment membership, $\mathcal{F} = \{f_1, \ldots, f_m\}$

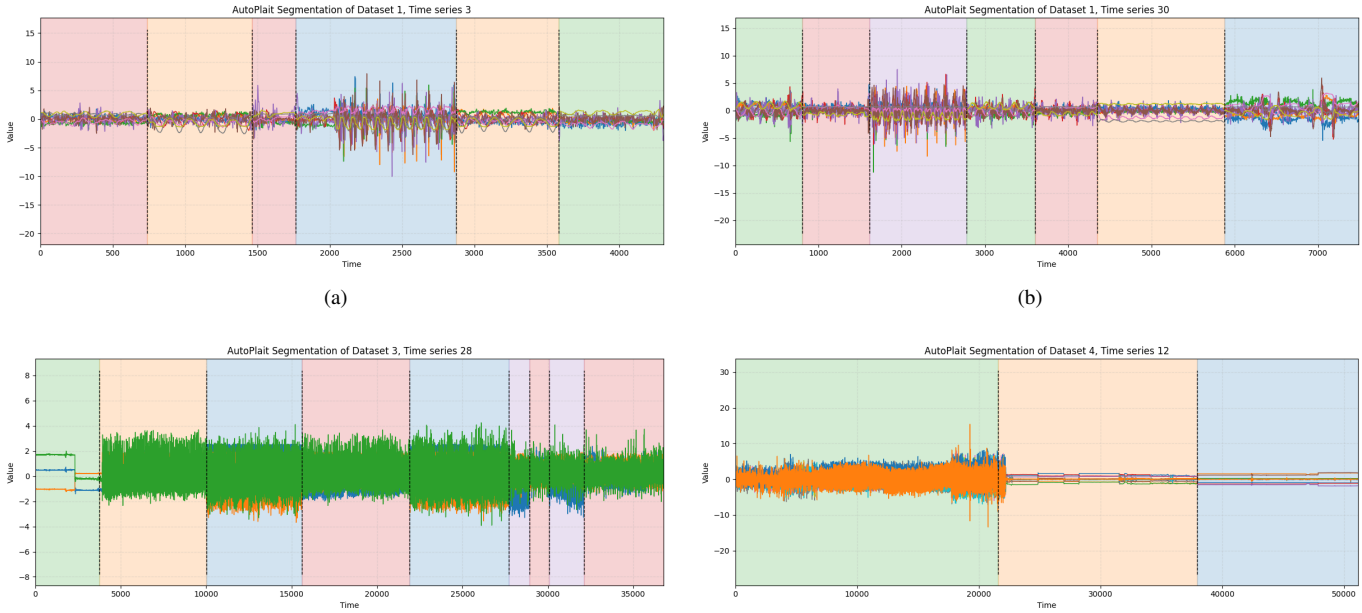
$\mathcal{Q} = \emptyset$; /* $\mathcal{Q}$: stack of $(m, \mathcal{S}, \boldsymbol{\theta})$ */
$\mathcal{S} = \emptyset$; $m = 0$; $r = 0$; $\mathcal{S}_0 = \{1, n\}$; $m_0 = 1$;
Estimate $\boldsymbol{\theta}_0$ of $\mathcal{S}_0$;
Push $(m_0, \mathcal{S}_0, \boldsymbol{\theta}_0)$ to $\mathcal{Q}$;
while $\mathcal{Q} \neq \emptyset$ **do**
  Pop an entry $(m_i, \mathcal{S}_i, \boldsymbol{\theta}_i)$ from $\mathcal{Q}$;
  $\{m_1, m_2, \mathcal{S}_1, \mathcal{S}_2, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\Delta}\}$ = RegimeSplit($\mathbf{X}[\mathcal{S}_i]$);
  Compare cost of single regime $\boldsymbol{\theta}_i$ with regime pair $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2$;
  if *regime pair is cheaper* **then**
    /* The single regime can be split into smaller regimes */
    Push $(m_1, \mathcal{S}_1, \boldsymbol{\theta}_1)$ and $(m_2, \mathcal{S}_2, \boldsymbol{\theta}_2)$ to $\mathcal{Q}$;
  else
    /* The regime cannot be split any further */
    $\mathcal{S} = \mathcal{S} \cup \mathcal{S}_i$; $\boldsymbol{\Theta} = \boldsymbol{\Theta} \cup \boldsymbol{\theta}_i$; $r = r + 1$;
    Update $\boldsymbol{\Delta}_{r \times r}$;
    Update $\mathcal{F}$;
    $m = m + m_i$;
$\boldsymbol{\Theta} = \boldsymbol{\Theta} \cup \boldsymbol{\Delta}_{r \times r}$;
return $\{m, r, \mathcal{S}, \boldsymbol{\Theta}, \mathcal{F}\}$;

TABLE III
EXECUTION TIME OF AUTOPLAIT IMPLEMENTATIONS

Dataset	C	Unoptimised	Optimised	Difference
1	32.6s	1487.9s	208.2s	$\times 45 \rightarrow \times 6$
2	4.9s	24.5s	2.3s	$\times 5 \rightarrow \times 0.5$
3	25.0s	3000+s	383.5s	$\times 120 \rightarrow \times 15$
4	61.6s	3000+s	350.9s	$\times 48 \rightarrow \times 6$
5	92.5s	3000+s	973.7s	$\times 32 \rightarrow \times 10$
Avg.	43.3s	2102.5s	383.7s	$\times 48 \rightarrow \times 9$

The large difference in execution time of the two implementations is first due to lack of proper optimisation, which we address next. Secondly, Python adds significant overhead from its higher level of abstraction, as a trade-off for its readability and simplicity, such as implicit memory management. However, the primary reason for the difference in speed is simply because C is a compiled language and Python is interpreted, and it is on average $10 - 100\times$ slower than its C counterpart.

2) Optimisation: Despite the inherent efficiency differences between compilation and interpretation, the Python implementation of AUTOPLAIT can be optimised in several ways. Firstly, we remove redundant code created by the translation process, such as memory allocation and freeing functions, which are not required in Python as memory is handled by the interpreter. Secondly, we can vectorise many operations



(a) (b) (c) (d)

Fig. 8. Optimised Python AUTOPLAIT implementation results for various test time series. Regime membership is shown by coloured background.

that use (nested) loops using `numpy`.

Many operations in AUTOPLAIT use loops over various matrices and vectors. A clear example is lines 1-5 in Algorithm 5, which iterates over n and numbers of states k_1 and k_2. Using `numpy` arrays, we can batch perform these calculations to avoid long execution times.

3) Match aeon interface: Finally, we convert the optimised Python code to match the *aeon* interface of existing segmentation algorithms. This includes providing implementations for `.fit()` and `.predict()`, and various tags that describe the capabilities of AUTOPLAIT, such as multivariate support. Additionally, we ensure there is sufficient documentation of the code, in the form of comments and docstrings, and that the syntax is as simple, idiomatic, and Pythonic as possible.

6. EXPERIMENTATION

The *aeon* toolkit contains implementations of eight TSS algorithms; 1) E-Agglo, 2) GGS, 3) Hidalgo, 4) IGTS, 5) HMM, 6) BinSeg, 7) ClaSP, and 8) FLUSS, the first 4 of which have native support for multivariate TS[6] (shown in Table IV). As AUTOPLAIT utilises HMMs within it's algorithm, we will not include the HMM segmentation algorithm in our testing.

A total of 433 TS from 5 HAR datasets are used for experimentation and are summarised in Table V. The datasets were chosen for their varying number of participants, activities, length and features (dimensionality).

1) Preprocessing: The majority of the datasets used to evaluate AUTOPLAIT are not designed specifically for TSS and focus more on classification (a more classic example of HAR). As such, they are 'converted' for use in TSS by identifying where the labelled annotations change for a given TS. For example, if $t = 149$ is labelled as 'Walking', and $t = 150$ is labelled as 'Jogging', we can identify $t = 150$ as a ground-truth change point. Similarly, formatting and refactoring are required to ensure consistency when evaluating algorithms across the datasets. Data loaders for all datasets are included in the data archive (see Appendices).

2) Univariate conversion: As mentioned above, four of the eight *aeon* implementations do not support multivariate TS data. Obviously, this immediately makes them less suitable for HAR tasks, as they are unable to take advantage of the additional information multidimensional data brings. Univariate sensor data is very limited in its usefulness and is not often collected (owing to lack of use). To overcome this, we

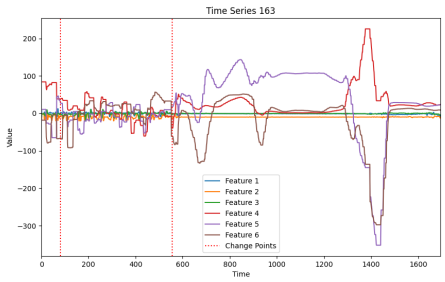
[6]Multivariate capable implementations for some of these algorithms do exist, but are not present in *aeon*. As such, we only consider official implementations currently present in the library.

TABLE IV
SUMMARY OF *aeon* ALGORITHM PROPERTIES

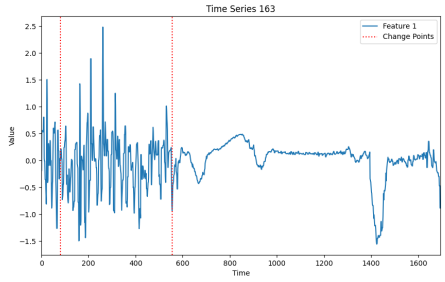
Algorithm	Multivariate support	Requires k points
AutoPlait	Yes	No
E-Agglo	Yes	No
GGS	Yes	No
Hidalgo	Yes	No
IGTS	Yes	No
BinSeg	No	Yes
ClaSP	No	Yes
FLUSS	No	Yes

TABLE V
SUMMARY OF KEY FEATURES OF DATASETS USED FOR EXPERIMENTATION

Dataset	No. TS	No. Dims	No. Activities	No. Users
1 [Ermshaus et al., 2023b]	250	6 - 9	100	16
2 [Anguita et al., 2013b]	21	274	6	21
3 [Kwapisz et al., 2011]	36	3	6	36
4 [Malekzadeh et al., 2019]	24	12	6	24
5 [Weiss, 2019]	102	6	18	51

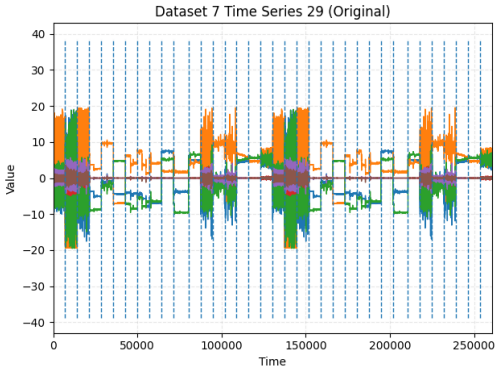


(a) Multivariate TS (6 features) with 2 change points and high variance between features.

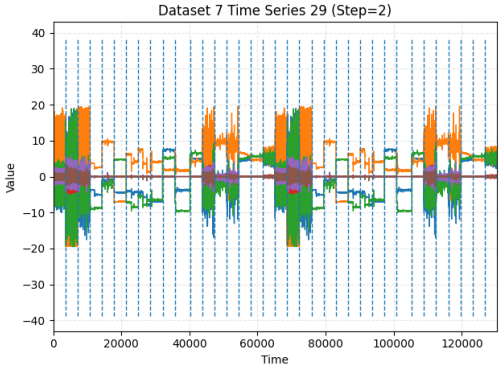


(b) Univariate TS of (a). Large spikes have been generated at the beginning of the TS, and the final segment is considerably different.

Fig. 9. Multivariate Time Series and it's univariate conversion. (a) has been heavily distorted by conversion. (Data source: dataset 1)



(a) Original TS from Datset 7 of length $n = 260676$.



(b) Sampled version of (a) with length $n = 130338$.

Fig. 10. Comparison of original and sampled TS. Change points remain representative of the structure of the data, and the shape remains very similar in both versions, with a 50% reduction in length.

employ simple averaging to reduce the dimensionality of a multivariate TS to univariate. A TS is first normalised to have a mean of 0 and standard deviation of 1, and then the mean across all dimensions is calculated for each time tick, resulting in a 1-dimensional TS of length n.

While this allows us to use univariate *aeon* algorithms on the data, it can distort the TS, potentially losing information from more important features, or even alter the shape of the data altogether (see Figure 9). However, if all features are similar, feature averaging has little affect. We discuss this considerable limitation further in a later section.

3) Missing values: All *aeon* segmentation algorithms do not support missing values in TS data. Any missing values in a given TS (primarily from dataset 5) are replaced with 0 to allow *aeon* algorithms to work with the data.

4) Subsampling: The length of time series varies greatly between and within datasets. The shortest TS is $n = 281$ and the longest is $n = 260676$; an over $90,000\%$ increase. Subsampling a TS at regular intervals shortens its length while retaining most of the information within it. To determine an appropriate step size to use for subsampling, we use Dynamic Time Warping (DTW) distance as a measure of similarity between an original TS and a sampled one. Step values of $s = 2, \ldots, 10$ are tested and 2 is chosen as it provides the closest similarity relative to the reduction in length. Time series from dataset 5 are very large and benefit from this subsampling preprocessing (see Figure 10).

A. Protocol

No parameter tuning was performed on any of the algorithms. For algorithms that require the number of change points as a parameter (BinSeg, ClaSP[7] and FLUSS), $k = 1, 2, \ldots, 6$ points are found and metrics are calculated for each k. Time Series were scaled and normalised prior to being fitted by a TSS algorithm.

B. Metrics

There are various metrics to measure the performance of ML algorithms. We use classic metrics in addition to metrics relevant to segmentation and a custom composite metric.

1) Precision, recall and F-measure: Precision and recall are perhaps the most used metrics for evaluating machine learning models, and both rely on the number of true/false positives/negatives. In the context of segmentation, a true positive (TP) is a predicted change point that is within a certain tolerance (δ) of a ground truth change point that has not already been matched to another predicted point. We use a relative tolerance of $\delta = 0.015$ or 1.5% of the TS length. A false positive (FP) is a predicted change point that does not match to any ground truth point within the tolerance window,

[7]The *aeon* implementation of ClaSP requires the number of change points as a parameter, whereas the ClaSP implementation used in Section 3.3.1 does not.

i.e., an algorithm predicted a segment that does not exist in the labelled data. Similarly, a false negative (FN) is a ground truth change point that does not match any prediction; in other words, an algorithm failed to identify a true segment. Precision for a segmentation algorithm uses the number of FPs to gauge how accurate the algorithm is. High precision indicates that an algorithm rarely identifies segments that do not exist; it is confident in any predictions it does make. In contrast, recall for a segmentation algorithm uses the number of FNs to determine how many change points were not predicted by the algorithm. High recall means few segments are missed.

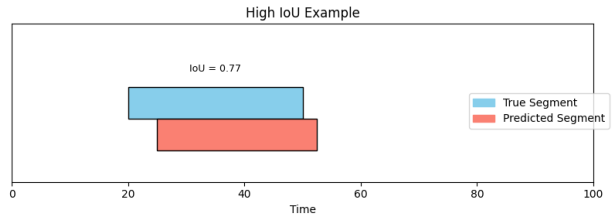
The importance of precision against recall for HAR problems depends greatly on the nature of the task. In critical situations, recall is likely to be far more important than precision. Consider a cardiac arrest monitor that identifies when a patient's heart enters abnormal activity. The consequences of missing a change point (that is, missing a cardiac arrest) are potentially fatal. Compare this with the alternative, where the system incorrectly alerts of a cardiac arrest, only to discover that the patient is perfectly healthy. In other tasks however, such as smart watch activity monitoring, a system that incorrectly identifies three different swimming strokes and a tennis match when the user is actually just jogging is extremely unhelpful, although not dangerous.

Additionally, precision is extremely unreliable when it comes to evaluating a segmentation. An algorithm that identifies a single change point will have perfect precision if that change point is correct, regardless of how many other true change points have failed to be identified. F_1 score is an instance of the more general F_β measure that calculates a balance between precision and recall and is a satisfactory metric for many machine learning tasks, particularly classification. A β value of 1 gives equal weighting to precision and recall to calculate a final metric. As discussed, recall is typically more important than precision for the task of TSS, and larger value of β allows for a higher weighting of recall to be used in calculation. We use the $F_{2.5}$ measure to provide a more reflective measure of an algorithm's usefulness for segmentation.

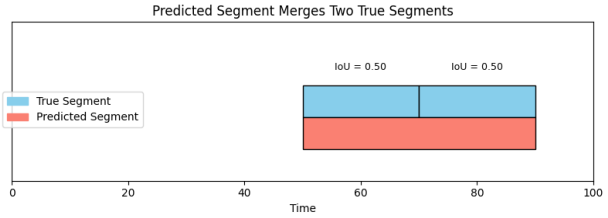
2) Intersection over union: Intersection over union (IoU) is a common metric in segmentation problems. As the name suggests, it takes the intersection of two sets of segments and divides by their union to measure the 'overlap' of the two segments to provide an evaluation of whole segments, rather than the similarity of individual change points.

IoU penalises structural mismatches, such as combining two distinct true segments into a larger predicted segment with the same change points (see Figure 11). This helps to demonstrate an algorithm's ability to identify semantically meaningful segments at varying scales. We utilise the Hungarian algorithm to identify the best one-to-one matches between predicted and ground truth segments (extracted from change points) such that the IoU between two matched segments is maximised, and then averaged across the total number of segments.

3) P_k score: P_k score measures the amounts of mismatches between the segments of points k time steps apart. Given a time tick t, P_k score checks which segments t and $t + k$ belong to; both the true labelled segments and the predicted segments, and keeps track of any disagreements. P_k score can give an indication of over-segmentation (too many change points identified) when t and $t + k$ are in different predicted segments, but the same true segment, and under-segmentation (too few change points identified) if vice versa. The value of k is set to half the average ground truth segment length, and is common practice in text segmentation problems [Beeferman et al., 1997].



(a) High IoU visualisation. Blue segment is true labelled segment (dictated by change points) and red is predicted segment (predicted by predicted points).



(b) Demonstration of structural mismatch penalty. Despite the segments having identical outer change points, the structure of the segments is different (2 true segments against 1 predicted) leading to a penalised score.

Fig. 11. Example of Intersection over union (IoU) metric. Well-matched segments score well (a) but are penalised if they do not correctly match the true structure (b).

4) FLUSS score: As mentioned previously, [Gharghabi et al., 2019] introduce a novel metric for TSS (see Algorithm 2) that we use with slight modifications. The original metric did not penalise over/under-segmentation (that is, predicting too many or too few change points). Therefore, we penalise a prediction for every *unmatched* true and predicted change point by 20% of the series length, and then normalised between 0 and 1, with 1 being a perfect score. This prevents a naive 'prediction' of every time tick from being scored as perfect. Alternatively, a balance between this metric and the F-measure can be used to address the issue of over/under-segmentation [Wang et al., 2024].

5) Covering: Used to benchmark algorithms in Section 3.3.1, covering measures how well each predicted segment 'covers' or overlaps the ground truth segments of a TS. To this end, it is a weaker version of average IoU, that is less harsh on over-segmentation and focuses more on local matches rather than global matches. It is included as a metric for consistency with the benchmarking process.

6) Composite metric: We would like a single overall 'score' for how well an algorithm performs. We use a custom composite metric calculated as the weighted sum of the above

TABLE VI
COMPOSITE WEIGHT VALUES BY METRIC

Metric	Weighting
Average IoU	5
$F_{2.5}$-Score	5
FLUSS Score	5
Precision	4
Recall	4
P_k Score	3
Covering	2

TABLE VII
DATASET 1 COMPOSITE SCORE BY ALGORITHM

Algorithm	Score	Rank
BinSeg	0.535	1st
AUTOPLAIT	0.446	2nd
ClaSP	0.441	3rd
GGS	0.415	4th
E-Agglo	0.280	5th
FLUSS	0.279	6th
IGTS	0.131	7th

averaged metrics. The final weights used are shown in Table VI, and are normalised between 0 and 1.

Precision and recall are given the same weight in the composite metric, but $F_{2.5}$-Score is given a weighting of 5, meaning that overall recall still has a greater impact on the quality of a segmentation. The average IoU and the FLUSS score are also weighted higher, as they measure global spatial agreement of true and predicted change points and structural similarity, both of which reflect accurate and meaningful segmentations. P_k score is given a weight of 3 because it focuses on local pairwise boundary errors that are not as important when evaluating a global segmentation. Covering is given a weight of 2 as it is similar to average IoU, while still providing an evaluation of how well the ground truth segments have been matched. This is the metric used to determine the best k change points for algorithms that require k as a hyperparameter.

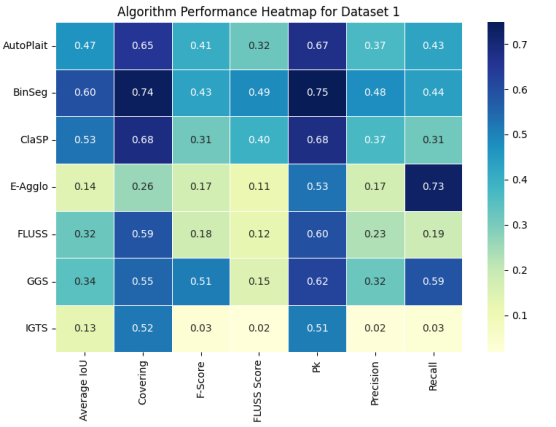
7. RESULTS & EVALUATION

Experiments were carried out on a Ryzen 9 5900X 4.5GHz with 32GB of memory, running Windows. We discuss the performance of the various algorithms on each individual dataset, and finally their overall average performance across all datasets. The Hidalgo implementation present in *aeon* failed to predict any segments for any time series across all datasets, so is omitted from results going forward.

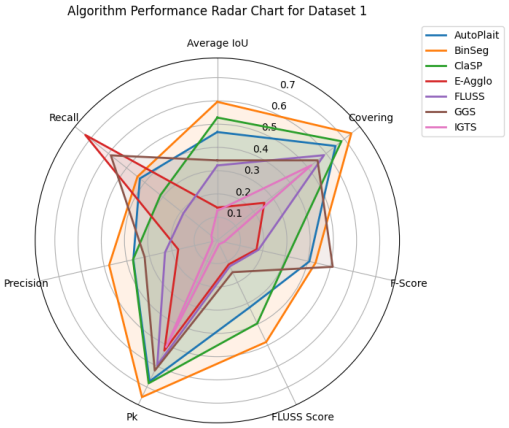
A. Dataset 1: Human Activity Segmentation Challenge

This is a HAR dataset *specifically* designed for TSS. It contains a variety of TS of varying length and number of change points, ranging from zero to nine. Time Series within this dataset have a mean length of 7720 and the number of features varies from six to nine.

The heatmap and radar chart for this dataset (Figures 12a and 12b) summarise the performance of each algorithm for all the metrics discussed. BinSeg consistently achieves the highest scores, reflected in the highest composite metric of



(a) Dataset 1 Heatmap



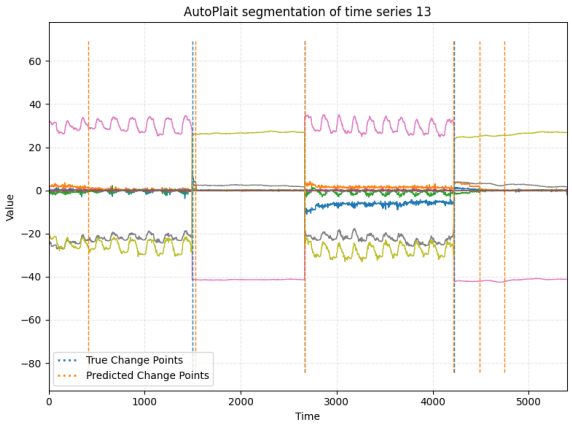
(b) Dataset 1 Radar Chart

Fig. 12. Dataset 1 Result Figures. BinSeg performs well across all metrics, closely followed by ClaSP and AUTOPLAIT.

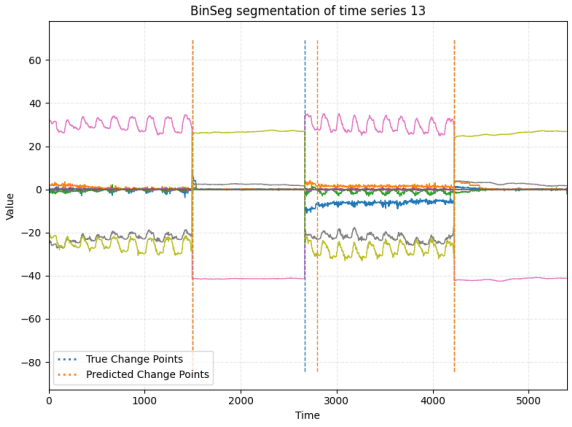
0.535 (Table VII), followed by AUTOPLAIT (0.446) and then ClaSP (0.441).

AUTOPLAIT has similar performance to BinSeg and ClaSP in all metrics except FLUSS score. BinSeg's higher FLUSS score suggests a more precise alignment of true and predicted change points, and a closer *number* of true to predicted segments; it correctly predicts the start and duration of activities better than other algorithms. However, AUTOPLAIT's over/under segmentation may identify finer changes of the human user that are not captured in the labelled true annotations of the data, such as the holding of breath or stretching, which are unlikely to be labelled as distinct activities.

BinSeg also requires the number of change points as a parameter for segmentation. As discussed earlier, we use $k = 1, \ldots, 6$ change points and select the best k (that is, the k that gives the highest composite score). As dataset 1 contains time series with at most 9 change points, it is impossible for BinSeg to over-segment a time series. In contrast, AUTOPLAIT automatically determines the number of change points to identify and thus is at risk of over-segmentation (as shown in Figure 13). This may have been a factor for the high scores of BinSeg and ClaSP (which also requires k).

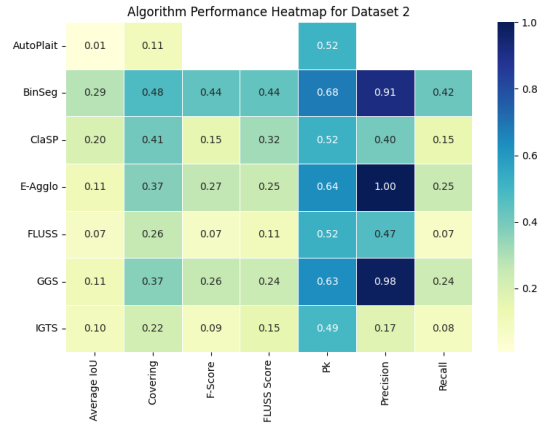


(a) AUTOPLAIT segmentation of dataset 1. The algorithm has over-segmented the data, predicted additional untrue change points at the beginning and end of the data.

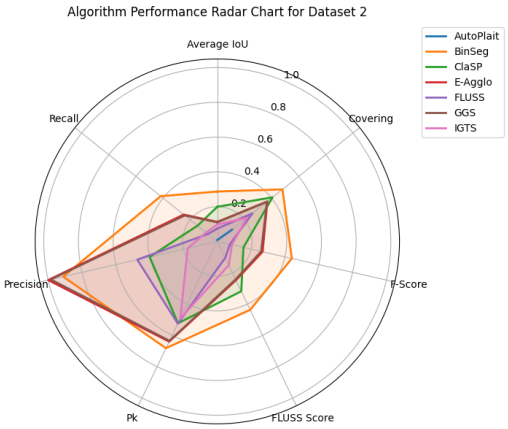


(b) BinSeg segmentation of dataset 1. BinSeg has predicted the correct number of segments, and is spatially close to the true segments.

Fig. 13. Segmentation results of AUTOPLAIT and BinSeg for time series 13 from dataset 1.



(a) Dataset 2 Heatmap. The best performing algorithms have very high precision, but low recall, suggesting under-segmentation.



(b) Dataset 2 Radar Chart

Fig. 14. Dataset 2 Result Figures

TABLE VIII
DATASET 2 COMPOSITE SCORE BY ALGORITHM

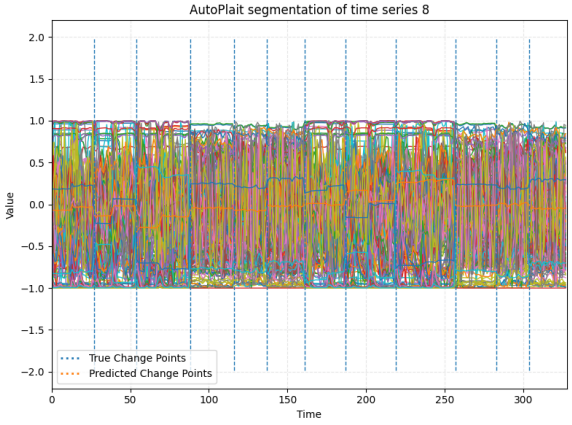
Algorithm	Score	Rank
BinSeg	0.505	1st
E-Agglo	0.385	2nd
GGS	0.378	3rd
ClaSP	0.283	4th
FLUSS	0.197	5th
IGTS	0.165	6th
AUTOPLAIT	0.064	7th

B. Dataset 2: HAR with Smartphones

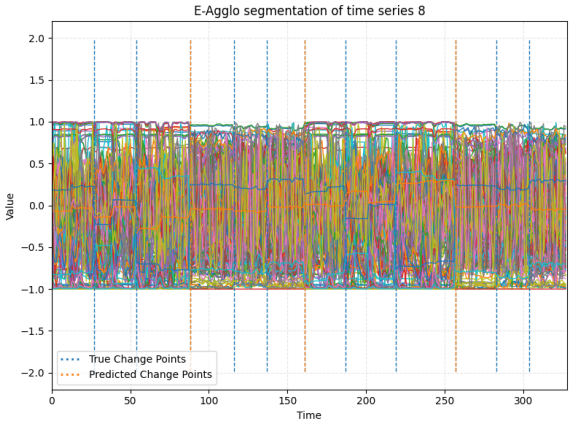
Originally a classification dataset, 'Human Activity Recognition with Smartphones' was converted into a TSS dataset by extracting change points based on labels. The dataset has been processed with Fourier transform and other signal processing techniques to generate a very large number of features (274). Time series in this dataset are short in length, with many evenly spaced change points (11 to 16).

BinSeg also performs well on this short-length high-dimensional data, with the highest composite score of 0.505, and may have been limited by the maximum value of $k = 6$ change points during the segmentation process. However, BinSeg uses only univariate data, whereas other algorithms, such as E-Agglo and AUTOPLAIT, use all 274 dimensions. Table VIII shows that BinSeg is followed by E-Agglo with a composite score of 0.385, then GGS with 0.378. AUTOPLAIT fails to identify any change points for any time series in this dataset. Modelling multivariate data using HMMs requires many parameters that scales with the number of dimensions d. Over such a short period of time, there is not enough evidence to justify splitting the time series into two regimes compared to keeping it as a single larger regime. The internal HMMs that govern the AUTOPLAIT algorithm are underfitted due to the short TS length.

HAR applications rarely use such high-dimensional data, as many personal sensors can only gather a limited amount of information, such as gyroscope and accelerometer data and heart rate. In applications where high-dimensional data is common, often feature selection and/or engineering is applied to carefully select which of the many features are most useful and should be passed to a ML algorithm. The inclusion of this dataset is to evaluate *aeon* algorithms' performance on high-dimensional data, where AUTOPLAIT performs poorly. How-



(a) AUTOPLAIT segmentation of dataset 2. No change points have been identified.



(b) E-Agglo segmentation of dataset 2. Only three change points have been identified, but all three are almost exactly correct.

Fig. 15. Segmentation results of AUTOPLAIT and E-Agglo for time series 8 from dataset 2.

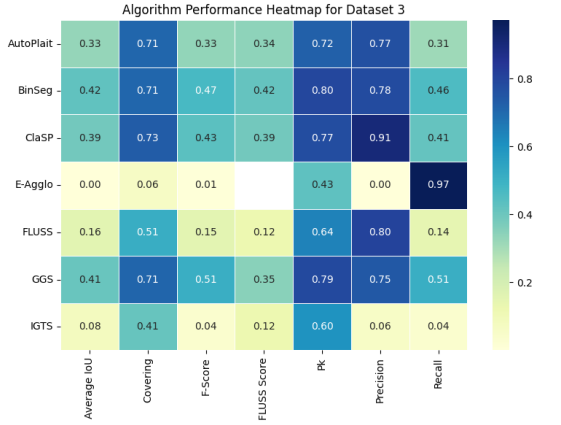
TABLE IX
DATASET 3 COMPOSITE SCORE BY ALGORITHM

Algorithm	Score	Rank
BinSeg	0.548	1st
GGS	0.541	2nd
ClaSP	0.540	3rd
AUTOPLAIT	0.461	4th
FLUSS	0.318	5th
E-Agglo	0.190	6th
IGTS	0.152	7th

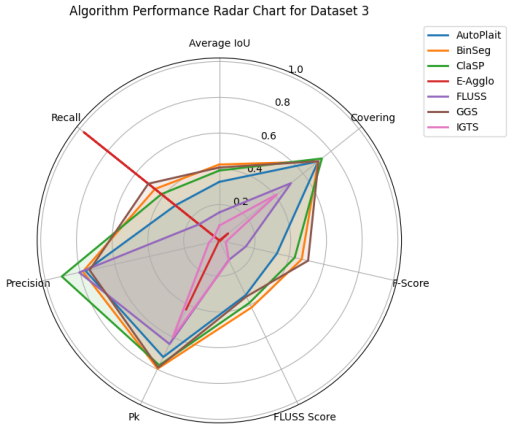
ever, AUTOPLAIT may perform better on high-dimensional datasets over longer periods of time (that is, a larger value of n).

C. Dataset 3: Human Activity Recognition

This dataset is another classification dataset that we convert for use in TSS. It contains data on 36 participants performing six different activities for specified periods of time. It contains only 3 dimensions (spatial dimensions of an accelerometer) and has an average length of 29822, with 0-23 change points per time series.



(a) Dataset 3 Heatmap



(b) Dataset 3 Radar

Fig. 16. Dataset 3 Result Figures

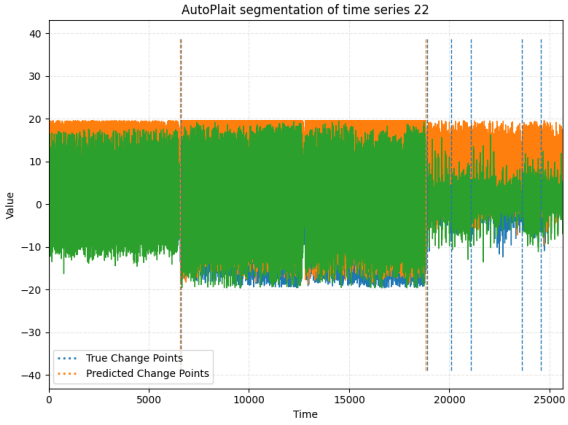
BinSeg again performs well on this data (0.548), and is almost tied with GGS at 0.541. AUTOPLAIT performs slightly worse than the top three algorithms, with the main difference in $F_{2.5}$-score impacted by low recall. This highlights AUTOPLAIT's struggle to automatically identify an appropriate number of change points in the data, and BinSeg's advantage in limiting the number of change points k to search for, which prevents over-segmentation for all time series and prevents under-segmentation for those with a true number of change points $\leq k$.

D. Dataset 5: Human Activity Classification

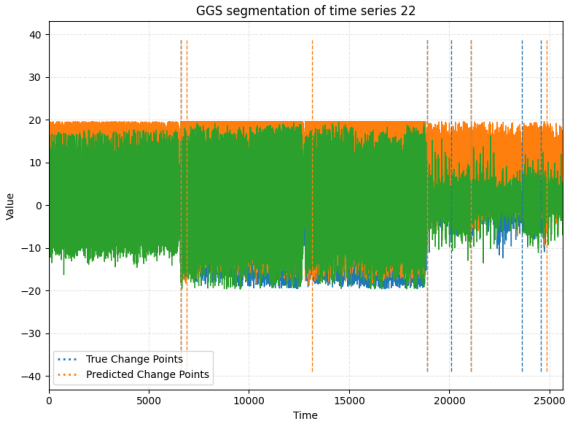
The final dataset, 'Human Activity Classification Dataset', is again converted to a TSS dataset, and consists of extremely long time series with many regularly space change points. Subsampling pre-processing was applied to this dataset for all algorithms to reduce the total length n[8].

The top 4 algorithms perform very similarly on this dataset. As shown in Figure 18a, almost all algorithms have very high precision but low recall, suggesting under-segmentation of the data. This is likely due to the very high number of change

[8]E-Agglo encountered memory issues when using the subsampled data with step $s = 2$. We use $s = 4$ to reduce the size of time series even further before passing to the E-Agglo segmenter. All other algorithms used $s = 2$.



(a) AUTOPLAIT segmentation of dataset 3. AUTOPLAIT has only identified two change points out of six total. The two it has identified are accurate.



(b) GGS segmentation of dataset 3. GGS correctly predicts a total of six change points, however their location is often incorrect.

Fig. 17. Segmentation results of AUTOPLAIT and GGS for time series 22 from dataset 3.

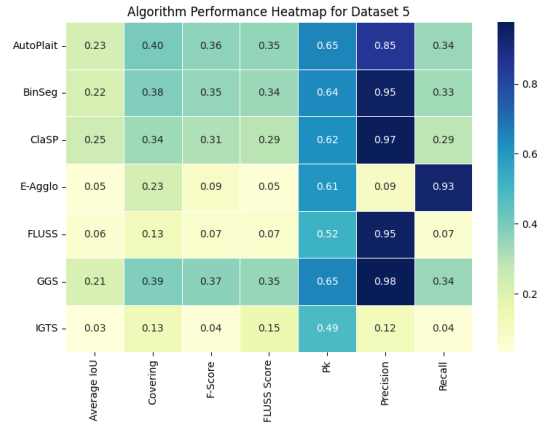
TABLE X
DATASET 5 COMPOSITE SCORE BY ALGORITHM

Algorithm	Score	Rank
GGS	0.452	1st
BinSeg	0.442	2nd
AUTOPLAIT	0.439	3rd
ClaSP	0.421	4th
E-Agglo	0.261	5th
FLUSS	0.246	6th
IGTS	0.125	7th

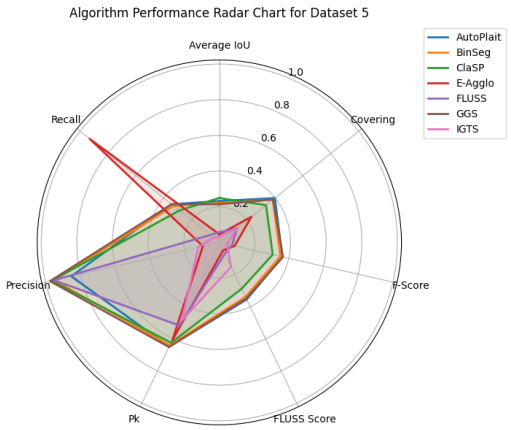
points in time series in dataset 5, and the very high length n. E-Agglo had much higher recall than precision, suggesting over-segmentation, and in fact, E-Agglo predicted change points at almost every time tick for the majority of time series in dataset 5, leading to a high recall score but an overall poor quality segmentation. The results of dataset 4 are similar to that of dataset 5.

E. Average results

Across all 5 datasets, AUTOPLAIT has similar, slightly lower performance than existing *aeon* algorithms, namely



(a) Dataset 5 Heatmap. Similarly to dataset 2, almost all datasets have very high precision but low recall, with the exception of E-Agglo which is vice versa.



(b) Dataset 5 Radar

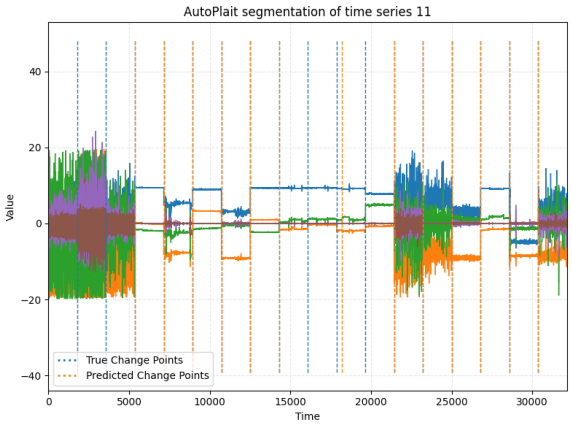
Fig. 18. Dataset 5 Result Figures

BinSeg, ClaSP and GGS, and any of these four could be used to provide insight for HAR tasks. FLUSS, E-Agglo and IGTS do not perform well on this type of human activity data and should not be used in this setting. For a more balanced evaluation, a larger limit of k change points should be used for algorithms that require it, to allow possible over/under-segmentation.

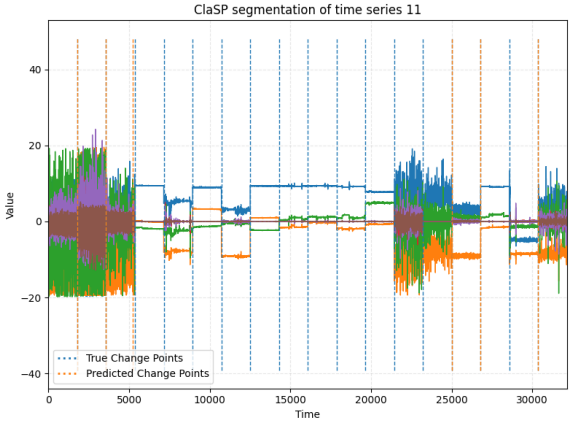
BinSeg consistently performed well across all datasets and has a final composite score of 0.510, significantly higher than second place ClaSP at 0.454. After BinSeg, the next three algorithms (ClaSP, GGS and AUTOPLAIT) are all within 0.035 of each other, showing that they perform satisfactory across the majority of test data. The bottom three algorithms (FLUSS, E-Agglo and IGTS) consistently performed poorly in testing, often predicting change points that were clustered together (FLUSS), not aligned well (IGTS) or massively over-segmenting data (E-Agglo).

8. CONCLUSION

Overall, AUTOPLAIT performs similarly to existing *aeon* algorithms for the task of HAR, but tends to over-segment data and identify additional activities that are not present in the ground truth. The additional property of pattern recognition

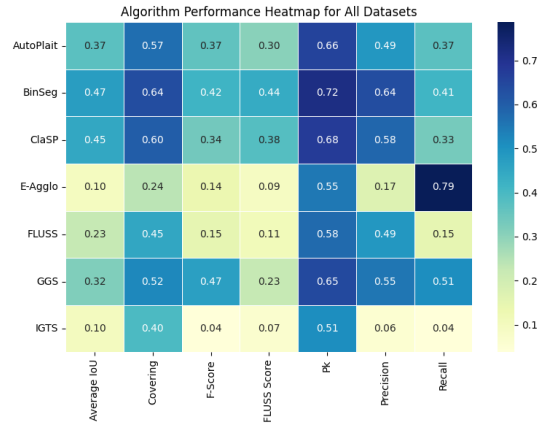


(a) AUTOPLAIT segmentation of dataset 5.

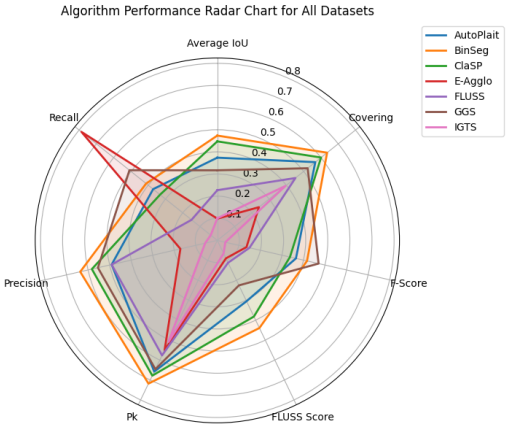


(b) ClaSP segmentation of dataset 5.

Fig. 19. Segmentation results of AUTOPLAIT and ClaSP for time series 11 from dataset 5.



(a) All Dataset Heatmap



(b) All Dataset Radar

Fig. 20. All Dataset Result Figures

TABLE XI
ALL DATASETS COMPOSITE SCORE BY ALGORITHM

Algorithm	Score	Rank
BinSeg	0.510	1st
ClaSP	0.454	2nd
GGS	0.439	3rd
AUTOPLAIT	0.420	4th
FLUSS	0.273	5th
E-Agglo	0.271	6th
IGTS	0.133	7th

provided by AUTOPLAIT's regime membership can be useful for HAR tasks, but does fundamentally change it's use for change point detection and segmentation.

The absence of a clear and consistent winning algorithm across all test datasets suggests that there is no 'one size fits all' method for TSS, and the algorithm that provides the best segmentation will depend on the length, dimensionality, statistical attributes, and various other properties of the data.

A. Limitations

The lack of multivariate capability for many *aeon* algorithms potentially limited their ability to identify the best possible segments. While univariate-only implementations such as BinSeg and ClaSP perform well on the test data, additional information provided by extra dimensionality could lead to improved accuracy and performance in segmentation tasks. Additionally, careful feature engineering or selection could have allowed all chosen algorithms to perform better compared to raw data, particularly in dataset 2 where dimensionality was very high. It is, however, difficult to find a suitable combination of features that generalise well, and doing so requires additional, manual pre-processing and domain knowledge, both of which are not always viable.

B. Further work

Despite the variety of approaches to segmentation provided by the various algorithms in *aeon*, and the different types of data used to evaluate them, no algorithm had a composite score greater than 0.600. This highlights the difficulty in automatically identifying meaningful and accurate segments from time series data, both in the context of HAR and the wider problem of TSS. Future work is required to find algorithms that can reliably identify semantically meaningful segments from time series data, or perhaps a new paradigm for segmentation entirely.

The *aeon* toolkit is constantly being updated and improved for many types of TSML, not limited to just segmentation.

The maintenance of AUTOPLAIT, extending it's integration within the *aeon* ecosystem, and improving it's efficiency are all viable future work for this algorithm. Furthermore, additional TSS algorithms can be implemented to the toolkit, such as BOCPD [Yoshizawa, 2020] that can support data streaming, unlike AUTOPLAIT, and would be an useful capability for the task of HAR.

C. Retrospective review

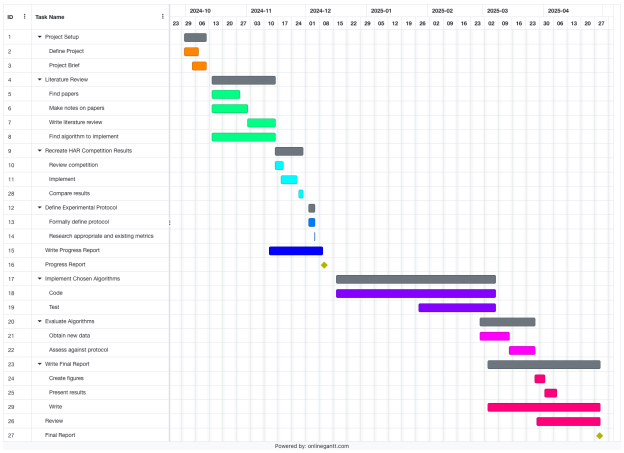
Overall, the project progressed primarily according to expectations. AUTOPLAIT is able to operate on the majority of time series within the test data, and provides meaningful, sensible change points for many of them. The Python implementation follows existing *aeon* segmentation API, and at the time of writing is under review to be contributed to the toolkit. I gained a deeper understanding on contributing to an open-source project such as *aeon*, time series segmentation and time series machine learning as a whole. I produced a report, figures, results, and code that I am happy with. However, there were complications during the project, mainly due to two reasons; 1) the semester 1 examination window, and 2) the difficulty of implementing AUTOPLAIT.

1) Examination window: As shown in the initial Gantt chart (Figure 21a), I had estimated that I would complete significant work on the implementation and testing of AUTOPLAIT during January. During this time, I had several coursework assignments and exams to prepare for, and as such this Part III Project was delayed until I had more time to dedicate to it. This could have been prevented by better planning and foresight, and by including a strategy to address this in the risk assessment (Table XII). As shown in the retrospective Gantt chart (Figure 21b), little work was actually completed during this period, since the main focus was the examination window.

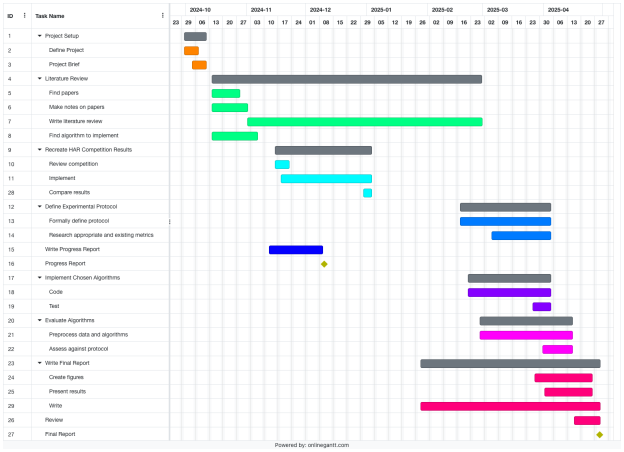
The retrospective Gantt chart also highlights the underestimation for time spent working on the literature review and background section of this report. Although it lasted many weeks in the retrospective chart, not all this time was spent dedicated solely to the literature review, but instead was several iterations of background review, based on feedback from my supervisor.

2) Implementation difficulty: Secondly, implementing the AUTOPLAIT algorithm was more difficult than anticipated. Various attempts to convert the relatively vague pseudocode included in the original paper directly to Python proved extremely difficult to capture logical equivalence. Combined with the reduced time to work on implementation as discussed above, AUTOPLAIT was not completed until late in the project cycle and prevented various other tasks from being completed, such as analysing results and creating figures. This could have been addressed by a deeper analysis of AUTOPLAIT prior to choosing it as the algorithm to implement into *aeon*, allowing a fuller understanding of the algorithm and an idea of the time needed to implement it well.

Overall, it was inaccurate to assume that the project would progress in a waterfall-type manner, but rather was more agile in nature, with several iterations of various stages of the project, such as reviewing background literature, benchmarking, testing, implementing, and creating the final report. Despite the various project management challenges, the primary objective of implementing and contributing a time series segmentation algorithm to the open-source library *aeon* and evaluating it and others for the task of HAR, was successfully achieved.



(a) Initial Gantt chart (dated 10th December 2024)..



(b) Retrospective Gantt chart (dated 11th November 2025).

Fig. 21. Comparison of initial and retrospective Gantt charts, showing the variation in planning as the project progressed.

D. Final words

HAR is a useful but challenging problem with a wide range of uses. Time series segmentation, using open-source libraries such as *aeon*, is an important pre-processing step for solving HAR. In addition to the many segmentation algorithms present in *aeon*, AUTOPLAIT performs almost as well on a variety of time series from different datasets and could be used for HAR problems. It is clear that there is no 'silver bullet' segmentation algorithm that works well for all types of time series data, but rather the choice of algorithm depends on the structure and properties of the data that requires analysis. Time series segmentation remains a difficult challenge regardless of the approach or algorithm used to solve it, and additional research into alternative techniques is suggested. The *aeon* library

TABLE XII
RISK ASSESSMENT

Risk	Description	Mitigation Strategy
Dataset Availability	Publicly available HAR datasets could have been moved or altered during the project.	Datasets were downloaded locally early in the project and stored on multiple secure local media to ensure constant available access.
Software or Hardware Failures	Local hardware or software issues could have delayed experiments or analysis.	Code and datasets were kept on local machine, with copies on secure media storage drives. Experiment results were regularly backed up (i.e., per dataset per algorithm) to avoid loss.
Implementation Errors	Translating AutoPlait from C to Python could introduce subtle logic errors.	Small changes were made incrementally, and manually inspected for each method or function. Equivalence tests were used to ensure both languages produce the same segmentation.
Algorithm Complexity	Time series of very large lengths (dataset 5) risked causing excessive computational costs in both time and space.	Subsampling (where necessary) and optimization techniques (`numpy` vectorization, memoization) were applied to improve performance.
Reproducibility	Results could be difficult to reproduce if random seeds or environment factors differed.	Random seeds were fixed where possible.
Metric Validity	Modified evaluation metrics (e.g., FLUSS score with penalties) introduced a risk of non-standard evaluation.	Modifications were documented clearly, and multiple standard metrics (Precision, Recall, IoU, Pk) were used to ensure some level of validity and standardisation.

provides a range of different algorithms for performing time series analysis, and further segmentation algorithms should be added to help tackle the problem of HAR.

GLOSSARY

AC Arc Curve. 6, 7

CAC Corrected Arc Curve. 7
CNN Convolutional Neural Network. 3

EM Expectation-Maximisation. 3

FFT Fast Fourier Transform. 6
FN False Negative. 14
FP False Positive. 13, 14

HAR Human Activity Recognition. 2–8, 12, 14–16, 18–21
HMM Hidden Markov Model. 2, 3, 8, 9, 12, 16

IAC Idealised Arc Curve. 7

kNN k-Nearest Neighbours. 3, 6

LSTM Long Short-Term Memory. 3

ML Machine Learning. 4–6, 13, 16
MLCM Multi-level Chain Model. 8

SVM Support Vector Machine. 3

TP True Positive. 13
TS Time Series. 3–16
TSML Time Series Machine Learning. 2, 4–6, 19
TSS Time Series Segmentation. 2, 4–6, 9, 12–17, 19, 20

APPENDICES

TABLE OF CONTENTS FOR CODE ARCHIVE

```
/
├── preprocessing ..... Python package
├── metrics ..... Python package
├── _autoplait.py ..... Main AUTOPLAIT algorithm
├── test_autoplait.py
├── autoplait_demo.py
├── generate_results.py ..... Run experiments
├── results_and_figures.py
├── benchmark results/
├── final results/
│   ├── dataset_metadata.csv
│   ├── average results/ .... Contains overall results
│   ├── autoplait results/ .... Contains per-dataset
│   │   results for algorithm
│   ├── binseg results/
│   ├── clasp results/
│   ├── eagglo results/
│   ├── fluss results/
│   ├── ggs results/
│   └── igts results/
├── Ethics Application Form for Secondary
│   Data Analysis
└── datasets.txt
```

The structure of the code archive submitted along with this report. Each file is briefly described below:

- `preprocessing`: Python package for loading the datasets.
- `metrics`: Python package for calculating and plotting segmentation metrics.
- `_autoplait.py`: Main implementation of AUTOPLAIT wrapped in *aeon* interface.
- `test_autoplait.py`: PyTest unit tests for `_autoplait.py`.
- `autoplait_demo.py`: Plot visualisation for several small example time series segmented using the *aeon* AUTOPLAIT implementation.
- `generate_results.py`: Code to run experiments on all algorithms for all datasets and generate individual result files.
- `results_and_figures.py`: Python file for processing and plotting results from various *aeon* algorithms.
- `benchmark results/`: Directory containing benchmarking results from Section 3.3.1.
- `dataset_metadata.csv`: Information about each time series across all datasets (e.g., ID, length, true change points).
- `average results/`: Directory containing overall results (across all datasets) for each algorithm.
- `X results/`: Directory containing results for each dataset for algorithm `X`.
- `datasets.txt`: Information about each dataset, including original source.

ORIGINAL PROJECT BRIEF

Part III Project Brief

Name: Daniel Roberts (dr1n22@soton.ac.uk)

Supervisor: Tony Bagnall (a.j.bagnall@soton.ac.uk)

Title: Segmenting Human Activity using aeon; an Open-Source Time Series Machine Learning Library.

Description:

Problem
Human Activity Recognition (HAR) is a common and highly useful task that involves automatically identifying different human activities; from basic actions like walking and running, to more complex actions such as cooking, typing and climbing stairs. HAR requires segmenting sequential, regular data gathered from various sensors into different activities or segments. This type of ordered data is called a time series, and segmentation is a type of time series machine learning (TSML).

Goals
To implement algorithm(s) into Python open-source TSML library aeon[1] that can be used for segmentation, with the ultimate goal of contributing to Prof. Liudi Jiang's research project "LOad Monitoring and Intervention System (LOMIS) to prevent diabetic foot ulceration"[2].

Scope
This project will involve researching and implementing at most two segmentation algorithms that do not currently exist in aeon. These will be tested and evaluated on datasets included in aeon, datasets available online, a new dataset generated for this project using activity sensors, and compared to existing segmentation algorithms. The novel human activity dataset will act as a case study for evaluating and presenting the results of the implemented and existing algorithms.

Timeline
The project will follow the rough timeline below:
Semester 1
1. Literature review of segmentation and selection of algorithm(s) to implement
2. Recreate results from Human Activity Segmentation Challenge[3]
3. Define standard/protocol for experiments and assessment of HAR data

Semester 2
1. Implement selected algorithm(s) into aeon
2. Evaluate and assess implemented algorithms on real HAR data generated for this project

References

[1] aeon toolkit https://github.com/aeon-toolkit/aeon/
[2] LOMIS research project https://www.southampton.ac.uk/research/projects/l-jiang-load-monitoring-intervention-system-lomis-to-prevent-diabetic-foot
[3] Human Activity Segmentation Challenge @ ECML/PKDD '23 https://ecml-aaltd.github.io/aaltd2023/papers/has_challenge_ecml.pdf

The original project brief had the aim of contributing to an engineering project to detect early signs of Type 2 Diabetes in patients by analysing data collected from a sensor inside of their shoes. AUTOPLAIT was never meant to be used by this project, nor was the project aware of this brief. Nevertheless, this algorithm (or any algorithm present in *aeon*, or another toolkit) could be used by the engineering team in their healthcare project.

MAIN BODY WORD COUNT

The Overleaf word count states that the main body of this report contains 9982 total words, 66 headers, 453 inline math and 14 display math.

REFERENCES

[Allegra et al., 2020] Allegra, M., Facco, E., Denti, F., Laio, A., and Mira, A. (2020). Data segmentation based on the local intrinsic dimension. *Scientific Reports*, 10(1):16449.

[Anguita et al., 2013a] Anguita, D., Ghio, A., Oneto, L., Parra, X., and Reyes-Ortiz, J. L. (2013a). A public domain dataset for human activity recognition using smartphones. In *The European Symposium on Artificial Neural Networks*.

[Anguita et al., 2013b] Anguita, D., Ghio, A., Oneto, L., Parra, X., and Reyes-Ortiz, J. L. (2013b). A public domain dataset for human activity recognition using smartphones. In *The European Symposium on Artificial Neural Networks*.

[Beeferman et al., 1997] Beeferman, D., Berger, A., and Lafferty, J. (1997). Text segmentation using exponential models. In *Second Conference on Empirical Methods in Natural Language Processing*.

[Bevilacqua et al., 2019] Bevilacqua, A., MacDonald, K., Rangarej, A., Widjaya, V., Caulfield, B., and Kechadi, T. (2019). *Human Activity Recognition with Convolutional Neural Networks*, page 541–552. Springer International Publishing.

[Broek et al., 2010] Broek, G., Cavallo, F., and Wehrmann, C. (2010). *AALIANCE Ambient Assisted Living Roadmap*, page 120.

[Chen et al., 2021] Chen, K., Zhang, D., Yao, L., Guo, B., Yu, Z., and Liu, Y. (2021). Deep learning for sensor-based human activity recognition: Overview, challenges, and opportunities. *ACM Comput. Surv.*, 54(4).

[Cover and Hart, 1967] Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27.

[Ermshaus et al., 2023a] Ermshaus, A., Schäfer, P., Bagnall, A., Guyet, T., Ifrim, G., Lemaire, V., Leser, U., Leverger, C., and Malinowski, S. (2023a). Human activity segmentation challenge @ ecml/pkdd'23. In Ifrim, G., Tavenard, R., Bagnall, A., Schaefer, P., Malinowski, S., Guyet, T., and Lemaire, V., editors, *Advanced Analytics and Learning on Temporal Data*, pages 3–13, Cham. Springer Nature Switzerland.

[Ermshaus et al., 2023b] Ermshaus, A., Schäfer, P., Bagnall, A., Guyet, T., Ifrim, G., Lemaire, V., Leser, U., Leverger, C., and Malinowski, S. (2023b). Human activity segmentation challenge @ ecml/pkdd'23. In *8th Workshop on Advanced Analytics and Learning on Temporal Data*.

[Ermshaus et al., 2023c] Ermshaus, A., Schäfer, P., and Leser, U. (2023c). Clasp: parameter-free time series segmentation. *Data Mining and Knowledge Discovery*, 37(3):1262–1300.

[Ermshaus et al., 2024] Ermshaus, A., Schäfer, P., and Leser, U. (2024). Raising the class of streaming time series segmentation. *Proc. VLDB Endow.*, 17(8):1953–1966.

[Fallmann and Kropf, 2016] Fallmann, S. and Kropf, J. (2016). Human activity recognition of continuous data using hidden markov models and the aspect of including discrete data. In *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, pages 121–126.

[Fine et al., 1998] Fine, S., Singer, Y., and Tishby, N. (1998). The hierarchical hidden markov model: Analysis and applications. *Machine Learning*, 32(1):41–62.

[Fox et al., 2009] Fox, E., Jordan, M., Sudderth, E., and Willsky, A. (2009). Sharing features among dynamical systems with beta processes. In Bengio, Y., Schuurmans, D., Lafferty, J., Williams, C., and Culotta, A., editors, *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc.

[Gharghabi et al., 2019] Gharghabi, S., Yeh, C.-C. M., Ding, Y., Ding, W., Hibbing, P., LaMunion, S., Kaplan, A., Crouter, S. E., and Keogh, E. (2019). Domain agnostic online semantic segmentation for multi-dimensional time series. *Data Mining and Knowledge Discovery*, 33(1):96–130.

[Guijo-Rubio et al., 2024] Guijo-Rubio, D., Middlehurst, M., Arcencio, G., Silva, D. F., and Bagnall, A. (2024). Unsupervised feature based algorithms for time series extrinsic regression. *Data Mining and Knowledge Discovery*, 38(4):2141–2185.

[Guo et al., 2024] Guo, Y., Li, Y., Zhou, S., Zhang, Z., Li, Z., and Shahidehpour, M. (2024). A data-driven three-stage adaptive pattern mining approach for multi-energy loads. *IEEE Transactions on Knowledge and Data Engineering*, 36(12):7455–7467.

[Hallac et al., 2019] Hallac, D., Nystrup, P., and Boyd, S. (2019). Greedy gaussian segmentation of multivariate time series. *Adv. Data Anal. Classif.*, 13(3):727–751.

[Jobanputra et al., 2019] Jobanputra, C., Bavishi, J., and Doshi, N. (2019). Human activity recognition: A survey. *Procedia Computer Science*, 155:698–703. The 16th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2019),The 14th International Conference on Future Networks and Communications (FNC-2019),The 9th International Conference on Sustainable Energy Information Technology.

[Kabir et al., 2016] Kabir, M. H., Hoque, M. R., Thapa, K., and Yang, S.-H. (2016). Two-layer hidden markov model for human activity recognition in home environments. *International Journal of Distributed Sensor Networks*, 12(1):4560365.

[Kwapisz et al., 2011] Kwapisz, J. R., Weiss, G. M., and Moore, S. A. (2011). Activity recognition using cell phone accelerometers. *SIGKDD Explor. Newsl.*, 12(2):74–82.

[Lai et al., 2024] Lai, Z., Li, H., Zhang, D., Zhao, Y., Qian, W., and Jensen, C. S. (2024). E2usd: Efficient-yet-effective unsupervised state detection for multivariate time series. In *Proceedings of the ACM Web Conference 2024*, WWW '24, page 3010–3021, New York, NY, USA. Association for Computing Machinery.

[Li et al., 2009] Li, L., McCann, J., Pollard, N. S., and Faloutsos, C. (2009). Dynammo: mining and summarization of coevolving sequences with missing values. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, page 507–516, New York, NY, USA. Association for Computing Machinery.

[Lin et al., 2016] Lin, J. F.-S., Karg, M., and Kulić, D. (2016). Movement primitive segmentation for human motion modeling: A framework for analysis. *IEEE Transactions on Human-Machine Systems*, 46(3):325–339.

[Malekzadeh et al., 2019] Malekzadeh, M., Clegg, R. G., Cavallaro, A., and Haddadi, H. (2019). Mobile sensor data anonymization. In *Proceedings of the International Conference on Internet of Things Design and Implementation*, IoTDI '19, pages 49–58, New York, NY, USA. ACM.

[Matsubara et al., 2014] Matsubara, Y., Sakurai, Y., and Faloutsos, C. (2014). Autoplait: automatic mining of co-evolving time sequences. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, page 193–204, New York, NY, USA. Association for Computing Machinery.

[Matteson and and, 2014] Matteson, D. S. and and, N. A. J. (2014). A nonparametric approach for multiple change point analysis of multivariate data. *Journal of the American Statistical Association*, 109(505):334–345.

[Mekruksavanich et al., 2022] Mekruksavanich, S., Jantawong, P., and Jitpattanakul, A. (2022). Recognition of complex human activities for wellness management from smartwatch using deep residual neural network. In *2022 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunications Engineering (ECTI DAMT & NCON)*, pages 350–353.

[Middlehurst et al., 2024] Middlehurst, M., Schäfer, P., and Bagnall, A. (2024). Bake off redux: a review and experimental evaluation of recent time series classification algorithms. *Data Mining and Knowledge Discovery*, 38(4):1958–2031.

[Paul and George, 2015] Paul, P. and George, T. (2015). An effective approach for human activity recognition on smartphone. pages 1–3.

[Rabiner, 1989] Rabiner, L. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

[Rissanen, 1983] Rissanen, J. (1983). A Universal Prior for Integers and Estimation by Minimum Description Length. *The Annals of Statistics*, 11(2):416 – 431.

[Ross, 2014] Ross, S. (2014). - markov chains. In Ross, S., editor, *Introduction to Probability Models (Eleventh Edition)*, pages 183–276. Academic Press, Boston, eleventh edition edition.

[Sadri et al., 2017] Sadri, A., Ren, Y., and Salim, F. (2017). Information gain-based metric for recognizing transitions in human activities. *Pervasive and Mobile Computing*, 38.

[Shan et al., 2023] Shan, S., Sun, S., and Dong, P. (2023). Data driven intelligent action recognition and correction in sports training and teaching. *Evolutionary Intelligence*, 16(5):1679–1687.

[Singh et al., 2017] Singh, D., Merdivan, E., Psychoula, I., Kropf, J., Hanke, S., Geist, M., and Holzinger, A. (2017). Human activity recognition using recurrent neural networks. pages 267–274.

[Spinnato et al., 2024] Spinnato, F., Guidotti, R., Monreale, A., and Nanni, M. (2024). Fast, interpretable, and deterministic time series classification with a bag-of-receptive-fields. *IEEE Access*, 12:137893–137912.

[Taha et al., 2015] Taha, A., Zayed, H. H., Khalifa, M. E., and El-Horbaty, E.-S. M. (2015). Human activity recognition for surveillance applications. In *Proceedings of the 7th International Conference on Information Technology*, pages 577–586.

[Tammvee and Anbarjafari, 2021] Tammvee, M. and Anbarjafari, G. (2021). Human activity recognition-based path planning for autonomous vehicles. *Signal, Image and Video Processing*, 15(4):809–816.

[Taylor, 2024] Taylor, P. (2024). Data growth worldwide 2010-2025.

[Tong et al., 2024] Tong, L., Lv, Z., and Guo, J. (2024). Application of online automated segmentation and evaluation method in anomaly detection at rail profile based on pattern matching and complex networks. *ISIJ International*, 64(10):1528–1537.

[Truong et al., 2020] Truong, C., Oudre, L., and Vayatis, N. (2020). Selective review of offline change point detection methods. *Signal Processing*, 167:107299.

[Wang et al., 2024] Wang, C., Li, X., Zhou, T., and Cai, Z. (2024). Unsupervised time series segmentation: A survey on recent advances. *Computers, Materials & Continua*, 80(2).

[Wang et al., 2019] Wang, Y., Cang, S., and Yu, H. (2019). A survey on wearable sensor modality centred human activity recognition in health care. *Expert Systems with Applications*, 137:167–190.

[Weiss, 2019] Weiss, G. (2019). WISDM Smartphone and Smartwatch Activity and Biometrics Dataset . UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5HK59.

[Yadav et al., 2021] Yadav, S. K., Tiwari, K., Pandey, H. M., and Akbar, S. A. (2021). A review of multimodal human activity recognition with special emphasis on classification, applications, challenges and future directions. *Knowledge-Based Systems*, 223:106970.

[Yin et al., 2008] Yin, J., Yang, Q., and Pan, J. J. (2008). Sensor-based abnormal human-activity detection. *IEEE Transactions on Knowledge and Data Engineering*, 20(8):1082–1090.

[Yoshizawa, 2020] Yoshizawa, G. (2020). Bayesian online change point detection for baseline shifts. *Statistics, Optimization &; Information Computing*, 9(1):1–16.

[Zheng et al., 2014] Zheng, Y., Capra, L., Wolfson, O., and Yang, H. (2014). Urban computing: Concepts, methodologies, and applications. *ACM Trans. Intell. Syst. Technol.*, 5(3).