Methods and tools for mining multivariate time series

De Gouveia da Costa Cachucho, R.E.

Chapter 2

Biclustering Multivariate Time Series

Ricardo Cachucho, Siegfried Nijssen, Arno Knobbe

in Proceedings of Intelligent Data Analysis (IDA), 2017

Abstract

Sensor networks are able to generate large amounts of unsupervized multivariate time series data. Understanding this data is a non-trivial task: not only patterns in the time series for individual variables can be of interest, it can also be important to understand the relations between patterns in different variables. In this paper, we present a novel data mining task that aims for a better understanding of the prominent patterns present in multivariate time series: multivariate time series biclustering. *This task involves the discovery of subsets of variables that show consistent behavior in a number of shared time segments. We present a biclustering method, BiclusTS, to solve this task. Extensive experimental results show that, in contrast to several traditional biclustering methods, with our method the discovered biclusters respect the temporal nature of the data. In the spirit of reproducible research, code, datasets and an experimentation tool are made publicly available to help the dissemination of the method.*

2.1 Introduction

Multivariate time series are becoming increasingly available, mostly through the rapid development of measurement systems. More and more, machinery, infrastructures and humans use sensors, collecting data synchronously over time. The continuous measuring of these systems also means that most datasets produced by them are unsupervized. These datasets contain a range of phenomena, from recurring phenomena recognizable across all the sensors, to some phenomena that are only recognizable in some of the signals or even random events that do not reoccur at all. The nature of multivariate time series data offers a big opportunity for pattern recognition. As a consequence, there is a considerable need for unsupervized methods that can provide insight in this data.

Among unsupervized tasks that have been studied in the time series field are the *segmentation* and *motif discovery* tasks [103]. These tasks aim to either identify recurring patterns in the time series, or to partition the time series into segments. Most of this earlier work is however biased towards univariate time series. Unlike the univariate setting, mining multivariate time series poses the following challenges:

- understanding the relationship between different variables is highly important;
- the relationships between different variables may alternate at different moments in time;

As a result, there is a need for new methods that identify subsets of variables with consistent behavior over subsets of time. Which methods can identify such subsets? We consider *biclustering* to be a good approach to solve this pattern recognition problem.

Biclustering is a well-known task in the literature [19, 61, 79, 51, 7, 91, 72, 42, 43]. Essentially, it aims at discovering subsets of rows with corresponding subsets of columns, such that the submatrix selected by these rows and columns satisfies a certain cohesiveness requirement. Biclustering can be seen as a generalization of clustering where not only the rows are clustered, but the columns as well. Please note that one bicluster is the result of the biclustering task. Furthermore, biclusters can overlap each other and not all rows or columns need to be included in the biclusters. However, as we will point out throughout the paper, traditional biclustering methods are not well-suited for analysing multivariate time series.

The traditional methods for biclustering ignore the temporal aspect of the data in the analysis. The possibility to shuffle the rows and include rows individually, imply that the time series will no longer be ordered. Traditional biclustering algorithms can include an arbitrary subset of time points in a bicluster; there is no guarantee that contiguous time points are included in the bicluster. As a result, very small segments of the time series, or even individual time points, may be selected in the bicluster.

We argue that discovering biclusters is also useful on time series data. Using the smartphone as an example, we know that people will walk and might cycle at some point through the day. To capture walking, we only need accelerometry data, while for cycling, we would likely to benefit from additional GPS data. However, as we will point out throughout the paper, traditional biclustering methods are not well-suited for analysing multivariate time series, as they ignore in the analysis the temporal aspect of data. They can include an arbitrary subset of time points in a bicluster; there is no guarantee that contiguous time points are included in the bicluster. As a result, very small segments of the time series may be included in the bicluster.

In this paper, we propose a new biclustering method that finds recurring patterns over time for multivariate time series. The distinguishing feature of our biclustering method is that it finds biclusters spanning sufficiently long segments of time. As a result, the biclusters that are found can be see as recurring motifs in multivariate time series data. Our algorithm includes:

- a search strategy for finding biclusters in time;
- the definition of a coherence measure to score the quality of biclusters in time;

Key advantages of our method compared to other methods include:

- the ability to deal with numeric data in a continuous scale (no discretization is needed);
- interpretability of the results.

In the following sections, we define the biclustering problem for multivariate time series, present our biclustering method and experimental results and at last a conclusion. Our experiments show promising results, both in terms of the quality of the results and scalability of the method. For reproducibility purposes, we make our code and datasets freely available. Additionally, we also developed and published an experimentation tool [13] with intuitive GUI, where all the experiments described can be easily tested and applied to various datasets.

2.2 Preliminaries

In this section, we define the main concepts of our problem: multivariate time series (Section 4.2.1), followed by the problem statement of biclustering multivariate time series (Section 4.2.3) and the definition of the similarity measure (Section 2.2.3) used both to segmentation and biclustering of multivariate time series.

2.2.1 Multivariate Time Series

We assume that we are given a multivariate time series of length n over m variables. This time series is represented in an $n \times m$ matrix $\mathbf{T}$. In this matrix, $\mathbf{T}_{ij}$ represents the measurement at time point i for variable j.

We will often need to identify parts of this data matrix. We introduce some notation to make this easier. Let $I \subseteq \{1, \ldots, n\}$ be a subset of time points and let $J \subseteq \{1, \ldots, m\}$ be a subset of variables, then $\mathbf{T}_{IJ}$ defines the following submatrix:

$$\mathbf{T}_{IJ} = \begin{pmatrix} \mathbf{T}_{I_1 J_1} & \mathbf{T}_{I_1 J_2} & \cdots & \mathbf{T}_{I_1 J_k} \\ \mathbf{T}_{I_2 J_1} & \mathbf{T}_{I_2 J_2} & \cdots & \mathbf{T}_{I_2 J_k} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{T}_{I_\ell J_1} & \mathbf{T}_{I_\ell J_2} & \cdots & \mathbf{T}_{I_\ell J_k} \end{pmatrix},$$

where I_u is the uth element in the set I, $k = |J|$ and $\ell = |I|$. Furthermore, we will use $\mathbf{T}_{i\bullet}$ as a shorthand for T_{IJ} with $I = \{i\}$ and $J = \{1, \ldots, m\}$ and $\mathbf{T}_{\bullet j}$ as a shorthand for T_{IJ} with $I = \{1, \ldots, n\}$ and $J = \{j\}$.

2.2.2 Problem Statement

As discussed in the introduction, we are interested in identifying biclusters in time series data. More formally, we define the problem at hand as follows.

Definition 3. *One* bicluster in time series data *consists of:*

- *selected segments* $\mathcal{I} = \{I_1, \ldots, I_q\}$*, where each segment consists of contiguous measurements* $I_x = \{a_x, a_x + 1, \ldots, b_x\}$*, for some* $a_x, b_x \in \{1, \ldots, n\}$ *and* $I_x \cap I_y = \varnothing$ *if* $x \neq y$*;*
- *selected variables* $J \subseteq \{1, \ldots, m\}$*,*

such that:

- *the selected segments and variables satisfy the following requirement:*

$$H(\mathcal{I}, J) < \delta,$$

where

$$H(\mathcal{I}, J) = \frac{1}{|\mathcal{I}||J|} \sum_{j \in J} \sum_{I \in \mathcal{I}} d(\mathbf{T}_{Ij}, \mathbf{T}_{\cup \mathcal{I} j}); \tag{2.1}$$

here $\cup\mathcal{I} = \cup_{I \in \mathcal{I}} I$ *and* $d(\mathbf{t}_1, \mathbf{t}_2)$ *measures how similar two time series* $\mathbf{t}_1$ *and* $\mathbf{t}_2$ *are; i.e., it is required that each selected segment is similar to the union of the other selected segments, for all chosen variables;*

- J *is maximal: no variable can be added such that the similarity constraint remains satisfied;*
- $\mathcal{I}$ *is maximal: no segment can be added such that the similarity constraint remains satisfied;*
- *each segment* $I \in \mathcal{I}$ *has a sufficient length, i.e.,* $|I| \geq \ell$.

Notice that $H(I, J)$ is set to capture the coherence for each column independently. This allows biclusters to be coherent, even if two variables do not share their distribution. This way, two variables can measure the same phenomenon in different ways, depending on the system characteristics and measuring system setup. As an example, high levels of body activity can be measured by high levels of heart rate or sinusoidal patterns of acceleration. For each selected variable in a bicluster, each selected segment is *similar* to the other selected segments for that variable, and we are interested in the largest such bicluster.

2.2.3 Probabilistic divergence score

Many alternatives are possible for the definition of the similarity, $d(\mathbf{t}_1, \mathbf{t}_2)$, between two subsequences. In this paper, we propose to measure the similarity of two subsequences by determining the divergence between the *distributions* of measurements in these subsequences.

Definition 4. *Given two probability densities* $p(x)$ *and* $p'(x)$ *over the same domain, a* divergence score $d(p, p')$ *is a score with the following properties:*

- $d(p, p') \geq 0$;
- $d(p, p') = 0$ *iff* $p(x) = p'(x)$ *for all* x.

Definition 5. *Given two time series subsequences* $\mathbf{t}_1$ *and* $\mathbf{t}_2$*, where* $\mathbf{t}_1$ *contains samples from probability density* $p(x)$ *and* $\mathbf{t}_2$ *contains samples from probability density* $p'(x)$*, a divergence score for these subsequences is an* estimate *of the divergence of these distributions, i.e.,* $d(\mathbf{t}_1, \mathbf{t}_2)$ *is an estimate of* $d(p, p')$ *as calculated from* $\mathbf{t}_1$ *and* $\mathbf{t}_2$*.*

2.3 Biclustering Multivariate Time Series

The problem formalized in the previous section is hard to solve exactly. In order to solve it, we propose a heuristic method. Our algorithm iteratively removes segments or columns until a bicluster is obtained with the desired quality. Unlike the traditional biclustering algorithm of Cheng and Church [19], our method is designed for multivariate time series, stressing solutions for the temporal aspect of multivariate time series data in a bicluster.

In our setting, given the length of most time series, removing individual rows is not a feasible approach; converging on a bicluster would take too long. Furthermore, it is unlikely that a heuristic that removes rows one by one is likely to lead to segments of good quality. For these reasons, we propose a new approach that works as follows:

1. Identify segment boundaries in the time series of each variable.
2. Combine the segment boundaries of the different attributes to obtain segment boundaries across all variables.
3. Perform a node deletion biclustering algorithm where either columns and all rows between segment boundaries are removed at the same time.

Below, we will discuss the details of this approach.

2.3.1 Time Series Segmentation

The process we propose for segmentation is detailed in Alg. 1. This algorithm computes the density differences between consecutive subsequences of time series using a sliding window approach. The user-defined parameters for this algorithm are as follows. The size of each subsequence (window) is defined by the parameter w. The window is moved over the time series in steps of size $jump$. This parameter is intuitively bounded by $1 \leq jump \leq w$. Note that by setting $jump \geq \ell$, we can ensure that segments are never of length shorter than ℓ.

Algorithm 1 Segmentation.

Input: multivariate time series $\mathbf{t}$, threshold for local maximum selection θ, window size w, jump between consecutive windows $jump, 1 \leq jump \leq w$.
$\mathcal{C} \leftarrow \{1\}$
for each column of $\mathbf{T}$ **do**
  for each $i \in \{0, 1, \ldots, \lfloor (n-w)/jump - 1 \rfloor\}$ **do**
    $s \leftarrow i \cdot jump$
    $d[i] \leftarrow d(\mathbf{t}[s+1, \ldots, s+w], \mathbf{t}[s+jump+1, \ldots, s+jump+w])$
  end for
  for each $i \in \{2, \ldots, \lfloor (n-w)/jump - 2 \rfloor\}$ **do**
    if $d[i-1] \leq d[i] \wedge d[i+1] \leq d[i] \wedge d[i] \geq \theta$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{i \cdot jump + w\}$
  end for
end for
return segment start indices $\mathcal{C}$

Having decided on how to go through the data, one needs to calculate the divergence scores between consecutive windows of data, $d(\mathbf{t}_1, \mathbf{t}_2)$. A solution for this divergence score is proposed in Section 2.3.3. After calculating the divergence score for all consecutive windows, we extract all local maxima to find the segment boundaries. The risk of using local maxima is that too many boundaries might be found, creating too many segments. For this purpose, a threshold θ for the selection of local maximum divergences is introduced. Only local maxima above θ will be selected as segment boundaries. A reasonable solution for this threshold is to normalize each variable and have the boundaries rescaled between 0 and 1 such that $0 \leq \theta \leq 1$.

Alg. 1 returns a set $\mathcal{C}$ consisting of the segment boundaries. This set is the union of all the segments found for each of the m variables. Note that we segment the variables independently of each other; this is important as in biclustering, we assume that the behavior of different variables over time may be different. To consider independency between variables, we need a univariate divergence measure. Alternatively, one could consider a multivariate version of the divergence measure. This would be suitable if the purpose would be to only segment multivariate time series ($\frac{1}{m}\sum_{1}^{m} divergence_{i,j}$). However, a multivariate divergence seems counter-intuitive in the case of biclustering, once we are looking both for subsets of segments and subsets of variables to include in the matrix $\mathbf{T}_{IJ}$. Experiments in Section 2.4.1 present the difference between univariate and multivariate divergence scores.

2.3.2 Biclustering

In this section, we present the *BiclusTS* algorithm to solve the problem of finding multiple biclusters given a multivariate time series. We will discuss the main challenge of recognizing interesting subsets of rows and columns ($\mathbf{T}_{\mathcal{I}J}$), while respecting the temporal nature of time series. We then move into the details of how to find a bicluster and how to explore the data in order to find multiple biclusters.

BiclusTS: Single node deletion

The BiclusTS algorithm is described in Algorithm 2. The algorithm assumes an initial set of segments in the data, as computed by Alg. 1. Then, a greedy process removes segments and variables (columns) that present the largest divergence to the bicluster. During this repeated process, the difference $H(\mathcal{I}, J)$ reduces monotonically until it drops below the acceptability bound δ. The remaining subset of segments $\mathcal{I}$ and columns J is returned as a bicluster. The difference measure $H(\mathcal{I}, J)$ is defined as follows:

$$H(I, J) = \frac{1}{|\mathcal{I}||J|} \sum_{j \in J} \sum_{I \in \mathcal{I}} d(\mathbf{T}_{Ij}, \mathbf{T}_{\cup \mathcal{I} j}) \tag{2.2}$$

Notice that $H(I, J)$ is a probabilistic measure, not interested in the correlation between variables. Instead, it is set to account coherence of each column independently (see Definition 2.1). This allows biclusters to be coherent, even if the behavior between variables is not correlated, as we consider desirable. This is because time series can measure the same phenomenon in different ways, depending on the system characteristics and measuring system setup. As an example, high levels of body activity can be measured both by high levels of heart rate and sinusoidal patterns of acceleration.

The requirements of our method to find a bicluster is to provide a segmented time series, composed of the time series $\mathbf{T}$ itself and the boundaries of each segment (as produced by Alg. 1). This will ensure a faster biclustering procedure and results consistent with the temporal aspects of the multivariate time series. Another requirement of Alg. 2 is a parameter δ that ensures a certain similarity for all segments within each column of the bicluster.

Algorithm 2 BiclusTS: Find One Bicluster.

Input: initial set of segments $\mathcal{I}$, acceptability threshold δ.
  let J be the set of all variables
  calculate $H(\mathcal{I}, J)$
  while $H(\mathcal{I}, J) > \delta$ **do**
    for all segments I and variables j **do**
      calculate $d(\mathbf{T}_{Ij}, \mathbf{T}_{\cup\mathcal{I}j})$
    end for
    for each segment I of $\mathcal{I}$ **do**
      calculate $\frac{1}{|J|}\sum_{j\in J} d(\mathbf{T}_{Ij}, \mathbf{T}_{\cup\mathcal{I}j})$
    end for
    for each variable j of J **do**
      calculate $\frac{1}{|\mathcal{I}|}\sum_{I\in\mathcal{I}} d(\mathbf{T}_{Ij}, \mathbf{T}_{\cup\mathcal{I}j})$
    end for
    find maximum margin divergence; remove the corresponding segment or variable
    recalculate $H(\mathcal{I}, J)$
  end while
  return $\mathbf{T}_{\mathcal{I}J}$, a bicluster that is a submatrix of $\mathbf{T}$

Finding a given number of biclusters

Having described the process of finding one bicluster, the challenge of finding a number of biclusters remains. For this task, we propose Alg. 3, which finds k non-overlapping biclusters by iteratively looking for biclusters in those segments that have not been selected yet. As input, we must have an initial segmented multivariate time series T and acceptability bound δ already introduced in Section 2.3.2. Additionally, this algorithm requires k, which is the number of potential biclusters to be found.

In the first iteration, Alg. 3 starts using all the segments $\mathcal{I}$ to find the first bicluster. After finding a bicluster, we add it to the set of biclusters $\mathcal{B}$ and remove all the segments that have been biclustered from the initial set of segments, $\mathcal{I}$. This process is iterated until no segments are left to be included in a new bicluster or the number of potential biclusters, k is reached. We use the word "potential" because there are situations where k is not reached, due to unavailable segments to bicluster, or when the acceptability bound δ is too low to produce any results.

Algorithm 3 Find k biclusters.

Input: initial set of segments $\mathcal{I}$, acceptability threshold δ, the desirable number of biclusters k.
  $\mathcal{B} \leftarrow \varnothing$, an empty set of biclusters
  while $|\mathcal{B}| \leq k$ and $\mathcal{I} \neq \varnothing$ **do**
    $\mathcal{B} \leftarrow \mathcal{B} \cup BiclusTS(\mathcal{I}, \delta)$
    Remove all segments in $\mathcal{B}$ from $\mathcal{I}$
  end while
  return $\mathcal{B}$, a set of biclusters found in matrix T

2.3.3 Density-Difference Estimation (LSDD)

An important choice that remains to be specified is which divergence score to use. Time series can assume many different shapes, depending on the phenomena and measurement system, making the comparison between subsequences a non-trivial problem. Obvious solutions for comparing time series would be two-step approaches. For instance, one could first estimate the probability density distributions (PDFs) of both subsequences, and then compare these using an f-divergence measure. As pointed out by [86], the drawback of such approaches is that good estimations will smoothen the PDFs and thus result in under-estimations of density-differences.

Instead of taking such a step-wise approach, we here propose to use a more direct approach, based on calculating a least-squares density-difference (LSDD) [86]. LSDD measures the similarity between two time series by directly estimating density-differences ($f(x)$) between time series subsequences. This method does not require a separate estimation of the time series distributions. LSDD directly estimates the density-difference between two samples, $f(x)$, by fitting a density-difference model $g_\theta(x)$ that minimizes:

$$\underset{\theta}{\operatorname{argmin}} \int \Big(g_\theta(x) - f(x)\Big)^2 dt + \lambda \theta^T \theta. \tag{2.3}$$

Note that the second term in this formula is a regularization term. The model $g_\theta(t)$ that is used to estimate the difference is a mixture model of Gaussians:

$$g(x) = \sum_{\ell=1}^{|\mathbf{c}|} \theta_\ell \exp(-\frac{||x - c_\ell||}{2\sigma^2}),$$

where $\mathbf{c}$ is a random sample of measurements in both time series $\mathbf{t}_1$ and $\mathbf{t}_2$.

To fit the model, the equivalence of Equation 2.3 with the following formula is exploited:

$$\underset{\theta}{\operatorname{argmin}} \int g_\theta^2(x)\,dx - 2\int g_\theta(x) f(x)\,dx + \lambda\theta^T\theta. \tag{2.4}$$

Given that $f(x)$ is unknown, an empirical estimate is used for the second term:

$$\sum_{\ell=1}^{|\mathbf{c}|} \frac{\theta_\ell}{|\mathbf{t}_1|} \sum_{i=1}^{|\mathbf{t}_1|} \exp\Big(-\frac{||t_{1i}-c_\ell||}{2\sigma^2}\Big) - \frac{\theta_\ell}{|\mathbf{t}_2|} \sum_{i=1}^{|\mathbf{t}_2|} \exp\Big(-\frac{||t_{2i}-c_\ell||}{2\sigma^2}\Big).$$

The resulting minimization problem can be solved analytically, as shown by the authors of LSDD [86]. Note that this model fitting procedure has two parameters: the Gaussian kernel width σ and the regularization parameter λ. The authors of LSDD propose to optimize these parameters using cross-validation. When LSDD is used on a large scale, this cross-validation becomes too demanding.

After a study of LSDD's behaviour, reported in Section 2.4.1, in which we compare different subsequences of the same time series, we found that the parameters of LSDD are very stable, i.e., the optimal choice for the parameters values does not change very often for the same time series. Thus, we propose that the parameters are estimated with cross-validation a certain number of times at the beginning of the segmentation task (Alg. 1). Then, we fix these parameters for the rest of the LSDD calculation processes, thus speeding up computational efficiency while keeping the quality as a density difference estimation measure.

LSDD is also considered now the biclustering process for the calculation of $H(\mathcal{I}, J)$. The advantage of using LSDD for computing $H(\mathcal{I}, J)$ is that it allows rich descriptions of the segments to be taken into account in the process of finding each bicluster. From the computational perspective, fixing the parameters of LSDD is beneficial to speed up the process of finding each bicluster, due to the extensive amount of calculations of LSDD in Algorithm 2. This can be seen in Alg. 2, where for all the segments and variables LSDD is used as a divergence measure between two time series subsequences (e.g. $d(\mathbf{T}_{Ij}, \mathbf{T}_{\cup\mathcal{I}j})$). Thus, making of $H(\mathcal{I}, J)$ a mean density difference score.

2.4 Experiments

In this section, we divide the experiments in two parts: segmentation and biclustering. We present experimental results of the segmentation task including how to do LSDD estimation (normalization and cross-validation of parameters) and univariate versus multivariate segmentation. The second part is about biclustering with several experiments comparing traditional biclustering with our proposed method.

We evaluate our method on four datasets, details of which are given in the table below. The datasets were selected for their length and their multivariate nature (with datasets having up to 119 variables). Except for *Accelerometry*, the datasets also have variables that can be grouped in different categories, such that each group will show considerably different behaviour. For example, the *InfraWatch* data [103] is collected from three types of sensor (each sensitive to different phenomena and time scales): strain gauges, vibration sensors, and temperature sensors.

dataset	# variables	# time points	sampling rate	duration
Accelerometry	3	176 700	85 Hz	34.6 min
Snowboarding	21	21 180	1 Hz	5.88 hrs
Running	6	951 200	100 Hz	2.64 hrs
InfraWatch	119	17 996	1/3600 Hz	749.8 days

All experiments were performed with an implementation in R. A demo tool called Bipeline [13], demonstrates this implementation and is easily accessible online (`http://fr.liacs.nl:7000/`). Using Bipeline, one can replicate experiments and can try the various settings and choices. The code is also made available (`https://github.com/kainliu/ShinyDashboard`). The four datasets mentioned above have also been made available, (in accordance with reproducible research standards) and can be found at:

- `www.openml.org/data/download/1854941/accelerometry.csv`
- `www.openml.org/data/download/1854942/infrawatch.csv`
- `www.openml.org/data/download/1854943/running.csv`
- `www.openml.org/data/download/1854944/snowboard.csv`.

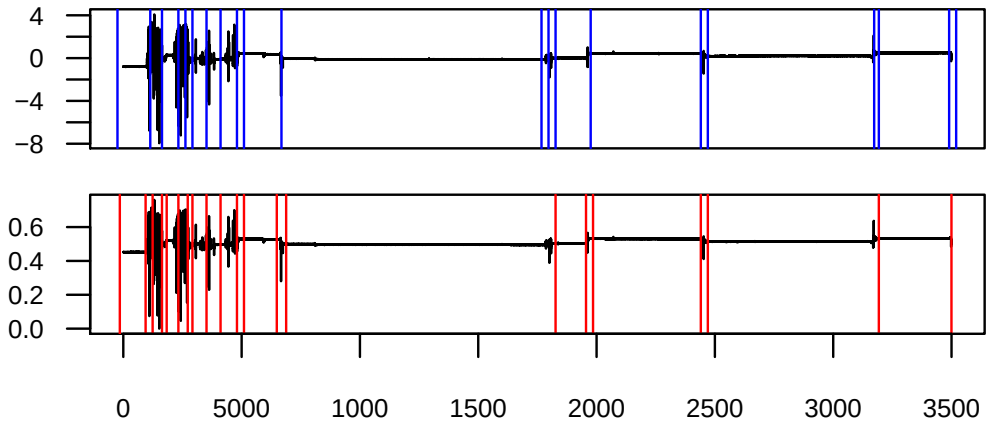

Figure 2.1: Two graphs showing the segmentation obtained on the *X-axis* variable of *Accelerometry* dataset, for the original time series (above) and the normalized one.

2.4.1 Segmentation

In Section 2.3.1, we proposed a solution to segment multivariate time series in order to bicluster them. Here, we present the experimental results that support our decisions on how to solve this segmentation task.

Normalization

Before considering aspects of the actual segmentation, we examine the effect of normalization. As was argued in Section 2.3.3, the usage of LSDD estimation as a divergence score in the multivariate setting requires normalization of each variable. To segment multivariate time series as described in Alg. 1, all the datasets were normalized and the parameter θ was fixed at a value of 0.75. Here, we study the effect of normalization on Alg. 1 results.

In this experiment, we performed comparisons on each variable, to see whether the produced segmentation is notably different, before and after normalization. These experiments demonstrated that this effect is marginal. Fig. 2.1 shows two segmentations produced on the X-axis accelerometer variable of dataset *Accelerometry*. Note that the differences in segmentations in the two settings are minimal, thus indicating that one could normalize the time series as a pre-processing step, ensuring comparable LSDD results across variables during the tasks of segmentation and biclustering.

LSDD estimation

The method we used to compare consecutive subsequences is a single-shot estimation of the difference between probability densities (LSDD), defined in Section 2.3.3. This method is based on fitting a Gaussian model where two parameters, λ and σ, need to be estimated. As was mentioned in Section 2.3.3, we estimate reasonable values for these parameters prior to the LSDD estimation for the entire time series, and then work with these fixed values. Clearly, we are trading off computational speed over accuracy of setting λ and σ always with cross-validation.

Here, the experiments consist of running Alg. 1 for all the datasets, with non-overlapping data windows of 100 samples ($w = jump = 100$), and considering two setups. In one setup, for each iteration we estimate LSDD with cross-validation. All the parameter estimations and computation times are saved. In the other setup, we cross-validate LSDD the first 100 iterations and then fix the parameters, by choosing the median σ and λ.

First, we consider the penalty in time produced by the cross-validation at each subsequence, compared to fixing the parameters at the start. For each dataset, the penalty of computing LSDD with cross-validation was calculated. For each setup of running Alg. 1, all the LSDD estimation times are summed up over all LSDD estimations per variable, and over all variables. Thus, the penalty is the time ratio of computing LSDD with cross-validation to LSDD with fixed parameters (so how many times faster the second is). In other words, how many times slower it is to perform cross-validation as opposed to fixing the parameters. The following time penalties were obtained for the four datasets:

Dataset	Penalty
Accelerometry	172
Snowboarding	62
Running	91
InfraWatch	93

Clearly, doing cross-validation at each subsequence is prohibitively expensive, while our choice to fix parameters at the start of each LSDD estimation is much more realistic.

While clearly being much faster, there is the risk of producing sub-optimal values for λ and σ. We test this end of the trade-off by examining the stability of parameter values when LSDD is estimated using cross-validation. If these values remain mostly the same throughout the LSDD estimation,

then we can safely pre-compute the values and fix them. For this experiment, we test parameter settings from a fixed range, identical to that proposed originally [86]: $\lambda \in \{0.001, 0.003, 0.01, 0.031, 0.1, 0.316, 1, 3.162, 10\}$ and $\sigma \in \{0.25, 0.5, 0.75, 1, 1.2, 1.5, 2, 3, 5\}$. For the four datasets and both parameters, the percentage of values found that deviate from the median were obtained, and averaged over all variables:

Dataset	Deviation
Accelerometry	1.2%
Snowboarding	1.5%
Running	0.7%
InfraWatch	3.1%

As seen in the table above, the LSDD parameters (λ and σ) are extremely stable. Here, we do not argue the fact that this solution to fix the parameters is a sub-optimal heuristic, when considering an alternative such as cross-validation. Our argument for the decision of fixing both λ and σ is related with the trade-off between accuracy and computational efficiency. The lost in accuracy is low, with a residual deviation from the cross-corelated parameters and on the other hand the computational efficiency gains are in orders of magnitude of at least two digits. To conclute, the parameters are extremely stable and can be safely estimated and fixed prior to LSDD estimation, with considerable efficiency gains.

Univariate or multivariate segmentation

One choice that influences the segmentation (that can be configured in our online tool) is whether to choose segment boundaries based on density differences in individual variables, or whether to compute them from the average density distances over all variables. Figs. 2.2 and 2.3 demonstrate the effect that this choice has on the boundaries produced. Clearly, segmentation over the combined density differences produces fewer boundaries, and only in locations where clear changes are visible in the majority of the time series. The multivariate segmentation, on the other hand, is much more sensitive to changes only observable in individual time series, and as a result produces more boundaries and thus smaller segments on average. Although either option is available and has its advantages and disadvantages, we feel that the univariate, more detailed approach to segmentation is more in line with the philosophy of biclustering and our goals to discover temporal phenomena in subsets of the variables. For that reason, we adopt the univariate option for all further experiments.

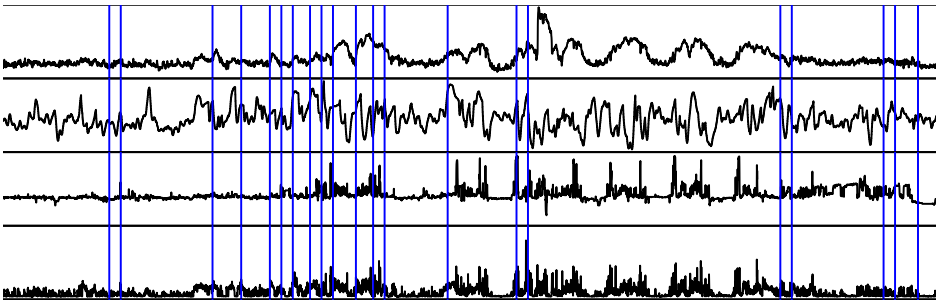

Figure 2.2: Multivariate segmentation example (*Snowboarding* data).

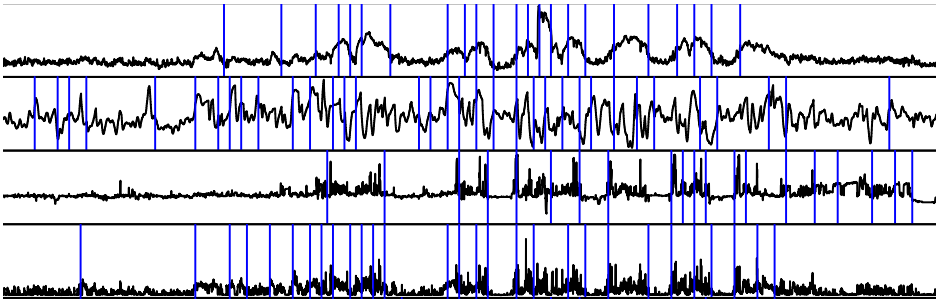

Figure 2.3: Univariate segmentation example.

2.4.2 Biclustering

As a reference point for reproduction of experiments, the parameter settings of our proposed method is presented in the table below.

Datasets	Window	Jump	Delta
Accelerometry	500	100	0.01
Snowboarding	90	30	0.01
Running	500	100	0.01
InfraWatch	120	24	0.01

For all the experimental settings, all parameters defaults can be found in the online tool. Additionally, we present results to further compare with other methods. We observe how many segments were created and the segment's average size. Notice that we want a relatively small number of segments with rather large sizes.

Datasets	Duration	Number segments	Average segment size
Accelerometry	7.55 s	14	642.5 (0.36%)
Snowboarding	81.5 s	24	81.5 (0.38%)
Running	16.7 s	103	1671.8 (0.18%)
InfraWatch	36.2 hrs	21	36.2 (0.20%)

At this point, it is important to understand the differences between the results produced by our approach and those produced by traditional biclustering algorithms, such as the Cheng & Church algorithm. In this experiment, we apply to our four datasets all the available algorithms in the `biclust` package [42] in R. This means that experiments were run to compare BiclusTS with the following biclustering algorithms: Cheng & Church (C&C) [19], the Xmotifs biclustering algorithm [72], the Plaid model [91], Bimax [79], and Questmotif [43, 72].

In the table below, we present the number of segments created by each algorithm. Please note that some algorithms are not even able to deal with large datasets (represented by -). The numbers of segments are in most case more than those resulting from BiclusTS, showing that fragmentation is a systematic problem of the traditional biclustering algorithms.

	Number segments					
Datasets	C&C	Xmotifs	Plaid	Bimax	Questmotif	BiclustTS
Accelerometry	4 034	1 863	1 641	141	3 338	14
Snowboarding	22	547	1	-	13	24
Running	3 852	3 852	9	-	34 791	103
InfraWatch	17	64	-	1	98	21

Complementary to the number of segments, one should look at the average size of the segments biclustered. The average size of the segments created by the traditional biclustering algorithms is in most cases very small. Interestingly, in two cases (indicated by the *), the biclustering algorithms produced segments stretching the entire length of the time series, thus failing to identify any meaningful segmentation into different activities. Notice that Plaid has some exceptions. Still, take as an example the *Snowboarding* dataset. For this dataset, Plaid created a bicluster containing all the data.

	Average segments size					
Datasets	C&C	Xmotifs	Plaid	Bimax	Quest	BiclustTS
Accelerometry	13.6	16.8	44.5	1.1	20.4	642.5
Snowboarding	2.8	15.1	21 180*	-	1420.4	81.5
Running	1.5	2.8	105 688	-	2.7	1671.8
InfraWatch	5.7	2.3	-	17 996*	83.1	36.2

Capturing distributions

The biclustering component of our method compares different segments, in order to cluster similar ones. Central to our approach is the choice to capture such similarity by differences in the distribution within each segment, by means of LSDD. A distribution is a fairly rich way to describe a set of measurements, and one could argue that simpler descriptions of segments could work as well, and might be more efficient also. In this experiment, rather than relying on the one-shot LSDD, we describe each segment of a time series simply by the average value, and apply biclustering to this simplified intermediate representation.

As can be expected, representing segments with only an average tends to cluster together segments that have very little in common, except for an average value that happens to be similar. Fig. 2.4 shows two examples of segments that were clustered in such an undesirable way. Note that for steady time series, this solution will actually work just fine. The problem only occurs with sinusoidal or other more complex periodic shapes, which are in fact very common in time series. The corresponding clustering by BiclusTS does not produce these meaningless groupings of segments.

Demonstration

Our method was designed to find biclusters that avoid very short segments of consecutive time points. However, having good subsets of time periods represented in the bicluster is not enough. BiclusTS also aims to capture interesting phenomena involving complex patterns. These patterns can be observed visually.

In order to show what can be expected from BiclusTS, we applied it to the *Snowboarding* dataset. With 21 variables, this dataset measures a person using a BioHarness chest sensor while riding a snowboard. The sensor measures vital signs such as heart rate, breath rate and body temperature, as

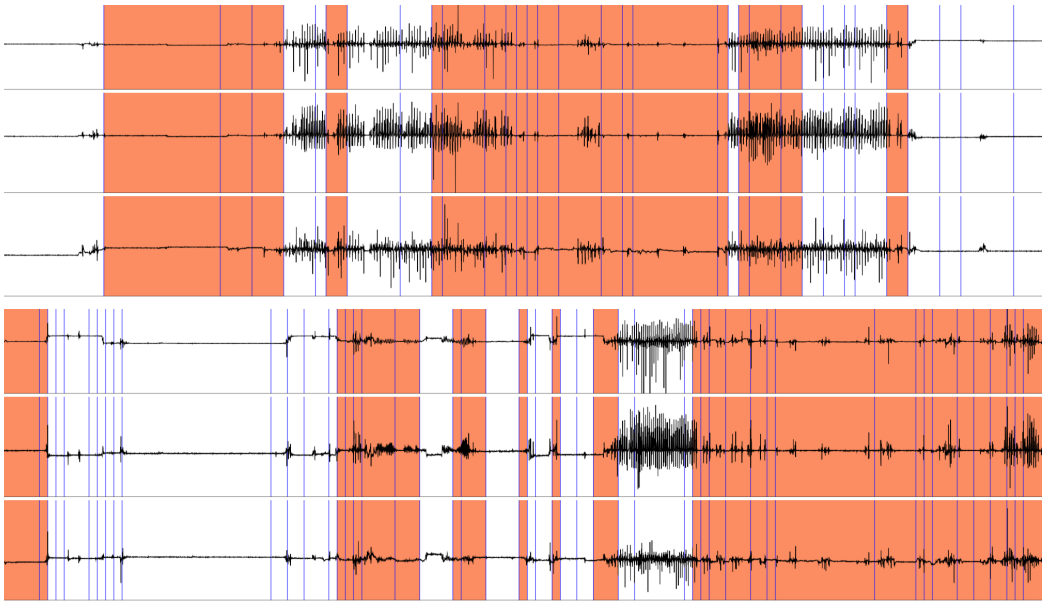

Figure 2.4: Two examples of undesirable clustering of segments.

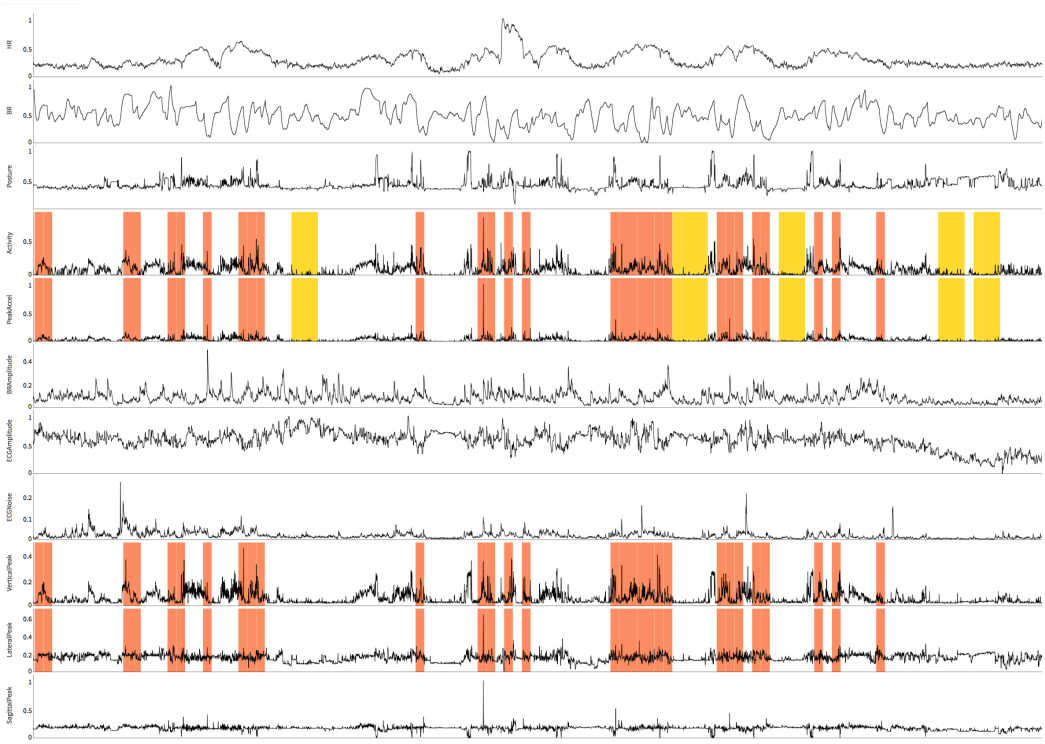

Figure 2.5: Examples of biclusters produced with BiclusTS.

well as acceleration. The dataset covers various sorts of activities common for a day of snowboarding in the high mountains.

Figure 2.5 shows two examples of biclusters related to alternating time periods of rest and downhill snowboard activities. As for the selected variables, one bicluster shows high levels of physical activity and different postures, during the periods of snowboarding. The other bicluster identifies resting periods in between snowboarding. These resting periods are recognized only using the activity levels measured by the sensor system (*Activity* and *PeakAcceleration*). As expected, our biclustering method makes use of different subsets of variables to describe different phenomena.

2.5 Related Work

Biclustering has received a lot of attention in the last decade and has become a well established task [19, 51, 61, 79, 29], with important real-world application, especially in the bioinformatics research field. The most cited biclustering algorithm is the Cheng & Church algorithm [19], making it an obvious benchmark. All these algorithms were not build to consider time series and perform badly due to the shuffling of rows without concern for the temporal nature of time series.

Adaptations for time series have been proposed [108, 62, 29], specifically for gene expression data. These adaptations focus on finding biclusters that consist of one contiguous segment in time, representing similar amounts of concentration for a given period of measurements. This means that these biclusters are mostly shaped-based and not repeatable over time. Adaptations of biclustering for general time series have not been proposed yet.

We believe our work presents novelty when by designing a generic solution to bicluster multivariate time series. The focus of our study is on defining a good optimization criterion for a new problem setting, and studying how to evaluate this evaluation criterion efficiently; developing a search algorithm for this criterion is just one of the components of our study. The difference with standard biclustering is that our data is ordered in time. In our setting, a bicluster is required to cover sufficiently long consecutive stretches of time. Also, our optimization criterion is different: it evaluates similarities in distributions. Moreover, the proposed clustering algorithm (part of a larger approach) is designed specifically to work with segments of time series and to allow the calculation of H(I, J) as described in Definition 1.

Still considering biclustering, one could also consider to carry out a comparison between our heuristic method and a complete enumeration approach. We considered this option, but initial results were such that complete enumeration was not a feasible approach for the datasets of realistic size and complexity. The proposed evaluation criterion has a high computational complexity; evaluating it for an exponential number of biclusters is not feasible within reasonable runtimes. Consider that our largest data has 951 200 time measurements, for 6 attributes. A naive enumeration of all subsets of time measurements is clearly not possible. We would also like to point out that, while exhaustive solutions to biclustering have been studied in the literature, they are not commonly used or accepted. Most common are greedy algorithms, such as the one studied in our work.

The task of motif discovery is similar to the task studied here [20, 71, 103]. In most cases, motif discovery algorithms focus on the univariate setting. Nevertheless the tasks are similar, as both motif discovery and biclustering aim at finding subsequences in a time series with a certain similarity. The difference is that motif discovery methods are mostly shape-based [71, 103]. The most cited approach [20] also uses probabilistic metrics, but requires the discretization of data for efficiency reasons. We propose to make full use of the richness of the time series by using LSDD [86] to compare segments.

A number of studies have considered the multivariate setting [88, 70, 93]. These also propose to find non-overlapping subsequences in a multivariate setting. The difference here is that [88, 93] propose a two-step approach, where the univariate setting is extended to the multivariate setting by grouping motifs using principle component analysis or coincidence rates. As for [70], they propose shape-based motif discovery in the multivariate setting, without considering the independence between variables. With our biclustering approach, we allow the discovery of subsets of both time periods and variables, without discretizing the measurements, thus allowing richer probabilistic descriptions to find more complex patterns.

One step further is to look into the problem of updating projections in a stream of tensors [87]; it incrementally maintains projections in these streams and is not capable of finding patterns that reoccur for the same sensors in different unconnected fragments of time. The optimization criterion that [87] used is a global optimization criterion that favors obtaining an as accurate representation of the data as possible, while instead, we focus on finding local patterns. Arguably, here, our problem setting compares to [87] problem setting as biclustering compares to matrix factorization: the problems are related, but have a different focus that justifies both's existence.

2.6 Conclusions and Future Work

This paper introduced the task of biclustering multivariate time series. In this context, respecting the temporal order is absolutely critical. Given that the traditional biclustering setting assumes that all samples are independent, the shuffling process of finding subsets of rows and columns will lead to useless biclusters in the time series context. We showed the importance of an algorithmic solution to bicluster multivariate time series. First, we proposed the creation of segments of sufficient length; second, we argued for the use of an LSDD divergence score [86] to ensure that for each selected variable, all selected segments have a similar distribution.

We presented an algorithm for solving this biclustering task. It consists of two stages: first, a stage in which the time series are segmented in segments of sufficient length; second, a stage in which a selection of segments and variables is made. In both the first and second stage, we used LSDD with fixed parameters. Experiments showed that LSDD produces stable results, and that the probabilistic descriptions of each segment can be used to accept, or reject a bicluster during the node deletion process.

This paper introduced the task of biclustering multivariate time series. In this context, respecting the temporal order is absolutely critical. Given that the traditional biclustering setting assumes that all samples are independent, the shuffling process of finding subsets of rows and columns will lead to useless biclusters in the time series context. With this paper, we show the importance of an algorithmic adaptation of traditional methods when biclustering multivariate time series.

We proposed two modifications of the traditional biclustering task. First, we proposed the creation of segments of sufficient length; second, we argued for the use of an LSDD divergence score [86] to ensure that for each selected variable, all selected segments have a similar distribution.

We presented an algorithm for solving this biclustering task. In this algorithm, we distinguished two stages: first, a stage in which the time series are segmented in segments of sufficient length; second, a stage in which a selection of segments and variables is made. Both in the first and in the second stage we used LSDD with fixed parameters. Experiments showed that LSDD produces stable results, and that the probabilistic descriptions of each segment can be used to accept, or reject a bicluster during the node deletion process.

As for the problem of setting multiple parameters, although the algorithms have some parameters, several of them are either fixed or estimated automatic. As for the parameters presented in Algorithms 1, 2 and 3, we have decided to fix most of them to the same value in all the experiments, except for window size, w, and jump between consecutive windows, $jump$. All the other parameters are fixed, because the results are not very sensitive to their setting. The default values can be found in the online tool (see Chapter 3). Additionally, we would like to point out that the parameters from Section 2.3.3 (θ and σ) are estimated automatically, and justified experimentally in Section 2.4.1 (LSDD estimation).

As future work, other types of biclustering in time series data can be studied. Comparisons between segments can be based on other descriptions, such as Fourier transforms with scores of spectral similarities, ARMA models coefficients and functional fits, both with scores on standard errors. These representations could be effective at capturing specific phenomena in the data that are not recognized by comparing distributions or other representations. Our algorithm can be improved as well. One could consider the use of multiple node deletion and addition to speed-up the search, and could modify it to allow for biclustering. This would be highly beneficial to both speed up the biclustering procedure and capture overlapping phenomena.