algorithms

MDPI

Article

Efficient Time-Series Clustering through Sparse Gaussian Modeling

Dimitris Fotakis [1], Panagiotis Patsilinakos [1], Eleni Psaroudaki [1,*] and Michalis Xefteris [2]

[1] School of Electrical and Computer Engineering, National Technical University of Athens, 15780 Zografou, Greece; fotakis@cs.ntua.gr (D.F.); patsilinak@mail.ntua.gr (P.P.)
[2] LIP6, Sorbonne Université, CNRS, F-75005 Paris, France; michail.xefteris@lip6.fr
* Correspondence: epsaroudaki@mail.ntua.gr; Tel.: +30-210-772-1550

Abstract: In this work, we consider the problem of shape-based time-series clustering with the widely used Dynamic Time Warping (DTW) distance. We present a novel two-stage framework based on Sparse Gaussian Modeling. In the first stage, we apply Sparse Gaussian Process Regression and obtain a sparse representation of each time series in the dataset with a logarithmic (in the original length T) number of inducing data points. In the second stage, we apply k-means with DTW Barycentric Averaging (DBA) to the sparsified dataset using a generalization of DTW, which accounts for the fact that each inducing point serves as a representative of many original data points. The asymptotic running time of our Sparse Time-Series Clustering framework is $\Omega(T^2/\log^2 T)$ times faster than the running time of applying k-means to the original dataset because sparsification reduces the running time of DTW from $\Theta(T^2)$ to $\Theta(\log^2 T)$. Moreover, sparsification tends to smoothen outliers and particularly noisy parts of the original time series. We conduct an extensive experimental evaluation using datasets from the UCR Time-Series Classification Archive, showing that the quality of clustering computed by our Sparse Time-Series Clustering framework is comparable to the clustering computed by the standard k-means algorithm.

Keywords: time series; clustering; Dynamic Time Warping; sparse Gaussian processes

Citation: Fotakis, D.; Patsilinakos, P.; Psaroudaki, E.; Xefteris, M. Efficient Time-Series Clustering through Sparse Gaussian Modeling. *Algorithms* **2024**, *17*, 61. https://doi.org/10.3390/a17020061

Academic Editor: Frank Werner

Received: 24 December 2023
Revised: 24 January 2024
Accepted: 25 January 2024
Published: 30 January 2024

1. Introduction

A time series is a sequence of observations measured successively in time. Time series are usually classified as dynamic data because their observed values evolve over time. Typical examples of time series include financial and sales data, stock prices, weather data, energy production and consumption data, biomedical measurements and biometrics data, sensor data, and mobility data through GPS detectors. As the computational and data storage capabilities increase, more (and much larger) time-series datasets become available, and the demand for efficiently processing them and using them to support forecasting and decision-making grows. Hence, in the last couple of decades, efficient processing of time series has become one of the most important and intriguing tasks in modern algorithms and data science (see, e.g., the long list of references in [1,2] for many diverse examples of time-series applications and may different approaches to efficient time-series processing and analysis).

Computational tasks of interest related to time series include regression and prediction [3], forecasting [4], and clustering [2,5]. In this work, we focus on efficient clustering of relatively long time series (with at least a few hundred data points, see also Table 1), using the very popular (but also computationally demanding) Dynamic Time Warping (DTW). Clustering time series is an important (and a daunting) computational task. On the one hand, it can be easily applied to a wide range of contexts and settings (see, e.g., ([2], Table 1)), due to its unsupervised nature. If successfully applied, it naturally leads to the discovery of interesting patterns that evolve. However, time-series data are inherently high-dimensional and complex. Even determining the similarity of two time series admits many different viewpoints (see, e.g., the very long list of dis(similarity) measures in ([2],

Tables 2 and 3)), with most dis(similarity) measures being computationally demanding. For example, computing (or approximating) the DTW distance of two time series requires quadratic time in their length, making it time-consuming for time series with more than a few thousand observations.

1.1. Clustering Time Series: Approaches and Related Work

Due to its practical significance and its applications to many different fields, there is a vast literature on efficient clustering of time series following several different approaches (e.g., hierarchical, density-based, grid-based, shape-based, feature-based, model-based) and using various dis(similarity) measures (see, e.g., the long list of references in [2,5,6]). Moreover, time-series clustering can be used as a subroutine in other data mining algorithms, such as rule discovery and indexing (see, e.g., [7] and the references therein).

At the conceptual level, there are two main formulations for time-series clustering [5]: correlation-based online clustering, where time series are clustered in real time based on their correlation; and offline clustering, where a time-series dataset is partitioned into k clusters based on a distance function that quantifies how much two time series agree with each other. There are three main different approaches to time-series similarity, namely shape-based, feature-based, or model-based [6]. In feature-based approaches, a selection of static features is extracted from each time series, and similarity reflects the proximity of the time series in the feature space. In model-based approaches, each time series is approximated by an appropriate model (e.g., by selecting appropriate parameters for a particular function form), and similarity reflects the proximity of the time series in the space of the model parameters. In this work, we focus on shape-based clustering, where the raw time series is considered, and similarity reflects how well the shapes of two time series agree with each other.

In offline shape-based clustering, the choice of the distance function is of key importance (and comprises a major challenge). The majority of shape-based time-series clustering methods are based on Dynamic Time Warping (DTW), where the distance of two time series is computed with respect to an optimal alignment of their data points (see, e.g., [8–10] and their references for the key properties and many applications of DTW). DTW is regarded as one of the most robust and accurate distance functions for time series because, in a natural, versatile, and robust way, DTW deals with differences in the time reference, the length, the time scale and/or the observation frequency of the time series at hand. However, due to the requirement for an optimal alignment of the data points, computing DTW of two time series with length T requires $O(T^2)$ time (Regarding the computational complexity of DTW, we refer an interested reader to [11], where a (large) constant factor nearly linear time approximation algorithm for the closely related edit distance is presented. Moreover, the related work section of [11] outlines a significant volume of work on the computational complexity of computing the edit distance either exactly or approximately). Thus, DTW becomes computationally expensive for time series with more than a few hundred data points. There has been a significant volume of previous work aiming at computationally efficient methods for shape-based time-series clustering with the DTW distance by managing to put aside the quadratic computational burden of computing (or approximating) DTW (see, e.g., the relevant references in [2,5,6]).

Moreover, there is previous work on sparse representation methods for time series and computationally efficient time series clustering. An interesting approach relevant to our work is that of Adaptive Piecewise Constant Approximation (APCA) [12]. APCA aims to approximate a time series using a set of constant-value segments with varying lengths, such that the total reconstruction error is minimal. The approach of Iorio et al. [13] is also conceptually similar to ours. They model a time series using P-spline smoothers and then cluster the functional objects, as summarized by the optimal spline coefficients, using the k-means algorithm and the DTW distance. Their experimental evaluation approach is also based on the Adjusted Rand Index (ARI) of the resulting clusterings.

1.2. Contribution

In this work, following an approach conceptually similar to that of [12,13], we present and evaluate experimentally a novel two-stage framework for shape-based time-series clustering with the widely used Dynamic Time Warping (DTW) distance. However, instead of constant-value segments of varying length [12] or *P*-spline smoothers [13], we resort to the richer and better-behaved space of Sparse Gaussian Processes for time-series simplification. More specifically, to mitigate the burden of DTW computation, we first use Sparse Gaussian Process Regression (SGPR) to obtain a sparse representation of each time series in the dataset, using a logarithmic (in the original length T) number of inducing data points. We then apply the k-means algorithm to the sparsified time series with an appropriate generalization of DTW. As with most previous work on the topic and for simplicity, we focus on univariate time series in the presentation and the experimental evaluation of our approach, with the understanding that extension to time series with d-dimensional observations is straightforward.

As in [12,13], our key insight is to regard each time series as a noisy realization of a functional form. To identify the function that best fits the time series at hand, we apply regression to the space of Gaussian Processes. A *Gaussian Process* (GP) is a stochastic process such that the joint distribution of every finite collection of its random variables is a multivariate Gaussian distribution [14]. Gaussian Processes extend the notion of multivariate Gaussian distributions to infinite dimensions (and thus, to distributions over functions) and are fully characterized by a mean function and a covariance (or kernel) function (see also Section 3). Building on this intuition, *Gaussian Process Regression* (GPR) applies the standard Bayesian regression approach to the space of Gaussian Processes. GPR aims to identify an optimal set of parameters for the mean function and the kernel function from a relatively small set of observations and then use the resulting Gaussian Process to predict the data point values at other points in time.

Gaussian Process Regression is conceptually simple and has many nice theoretical properties (see also Section 3.3). In practice, however, GPR can only deal with regression tasks of moderate size (with at most a few thousand data points) due to its cubic running time. As a result, several sparse approximation methods have been proposed to extend the practical applicability of GPR (see, e.g., [15] and the references therein). In this work, we resort to Sparse Gaussian Process Regression (SGPR) [16]. SGPR uses a small number of carefully selected inducing points to obtain a sparse approximation to the actual Gaussian Process with a small number of inducing data points (see also Section 3.4 and [17]). (Sparse) Gaussian Process Regression can potentially approximate any continuous target function, with the use of appropriate kernel functions, see, e.g., in [17,18]. Hence, our approach does not make use of any (implicit or explicit) assumptions on the nature of the time series.

In our *Sparse Time-Series Clustering* framework, we use SGPR and approximate a time series of length T with a sparse time series consisting of $\Theta(\log T)$ inducing points. Then, we cluster the resulting sparse dataset using the k-means algorithm with a generalization of DTW, which accounts for the fact that each inducing point serves as a representative of many original data points (see also Section 2.2). In the implementation of k-means, we use the DTW Barycentric Averaging (DBA) algorithm [19] to update the cluster representatives in each iteration.

The running time of our Sparse Time-Series Clustering framework is $O(NT\log^2 T + INk\log^2 T)$, where the first term accounts for the time complexity of Sparse Gaussian Process Regression with $\Theta(\log T)$ inducing points and the second term accounts for the running time of k-means with the DTW distance, running for a maximum of I iterations when applied to N sparse time series with $\Theta(\log T)$ inducing points each. The running time of applying the k-means algorithm to the original dataset with N time series of length T each is $O(INkT^2)$. Therefore, the asymptotic running time of our Sparse Time-Series Clustering framework is $\Theta(\max\{T^2/\log^2 T, IkT/\log^2 T\})$ times faster than the asymptotic running time of directly applying k-means to the original dataset. Intuitively, the improved

asymptotic running time of our framework is due to improved running time for computing DTW from $\Theta(T^2)$ in the original data to $\Theta(\log^2 T)$ in the sparsified data.

In addition to speeding up k-means, by computing a sparse representation of the original data set, Sparse Gaussian Process Regression tends to smoothen outliers and particularly noisy parts of the original time series, thus resulting in clusterings that are more robust to noisy observations and of higher quality.

We conduct an extensive experimental evaluation of our Sparse Time-Series Clustering framework on the datasets of the University of California (UCR) Time-Series Classification Archive [20] (see Section 5). The main finding is that Sparse Time-Series Clustering, with a logarithmic number of inducing points, computes clusterings with Adjusted Rand Index (ARI, see Section 2.3) comparable to the ARI of a baseline clustering, computed by applying the standard k-means algorithm to the original datasets (see Table 3). As for the running time, k-means runs significantly faster when applied to the sparsified dataset (see Figure 6). The most computationally demanding step of our framework is the modeling step, where we apply Sparse Gaussian Process Regression. For large datasets, the total running time of our framework is faster than applying k-means to the original dataset (see Table 5). Moreover, there are datasets for which we did not manage to run the standard k-means algorithm in our computational infrastructure, while our Sparse Time-Series Clustering framework produces good quality clusterings in reasonable running time (see Table 4). We should also note that the modeling step may run offline, once per time series, with its sparse approximation stored for any future use, and is perfectly parallelizable.

1.3. Organization

In Section 2, we introduce the generalization of DTW used in our framework and the Adjusted Rand Index (ARI), used to evaluate the quality of clusterings. Section 3 gives a brief introduction to Gaussian Processes, Gaussian Process Regression, and Sparse Gaussian Process Regression. Our framework of Sparse Time-Series Clustering and its main properties are presented in Section 4. The experimental setting and the key findings of our experimental evaluation are presented in Section 2.3. We briefly summarize our work and conclude with some directions for future work in Section 6.

2. Notation and Preliminaries

A *univariate time series* X of length T is a sequence $X = ((x_1, t_1), (x_2, t_2), \ldots, (x_T, t_T))$ of pairs where each $x_i \in \mathbb{R}$ is a data point and each $t_i \in \mathbb{R}$, with $0 \leq t_1 < t_2 < \cdots t_T$, is the point in time when x_i is observed.

2.1. Time-Series Clustering

Given a set $\mathcal{X} = \{X_1, \ldots, X_N\}$ of N time series, a *k-clustering* of $\mathcal{X}$ is a partitioning of $\mathcal{X}$ into k sets (or *clusters*) $\mathcal{X}_1, \ldots, \mathcal{X}_k \subseteq \mathcal{X}$ such that similar time series are assigned to the same set (see also ([2], Definition 1)).

In this work, we mostly focus on time series with the same number T of data points and on *shape-based clustering*, where we aim to maximize the similarity of time series in the same cluster (or to maximize the dissimilarity of time series in different clusters). Shape-based clustering is defined with respect to a shape-based *dissimilarity* (or *distance*) function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$, which is symmetric, i.e., $d(X, Y) = d(Y, X)$ for all $X, Y \in \mathcal{X}$, and satisfies $d(X, X) = 0$ for all $X \in \mathcal{X}$, but in the context of our work, may not satisfy the triangle inequality. We say that a dissimilarity function d satisfies the *triangle inequality* if for all $X, Y, Z \in \mathcal{X}$, $d(X, Z) \leq d(X, Y) + d(Y, Z)$. If a dissimilarity function is symmetric, has $d(X, X) = 0$ for all $X \in \mathcal{X}$, and satisfies the triangle inequality, we say that d is a distance function. For simplicity and clarity, we abuse the terminology and refer to dissimilarity functions d that may not satisfy the triangle inequality as distance functions. We focus on the widely used Dynamic Time Warping (DTW) distance (cf. Section 2.2), which is symmetric but does not satisfy the triangle inequality.

Given a distance function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0}$, a k-shape-based clustering (or k-clustering, for brevity) is a partitioning $\mathcal{X}$ into k clusters $\mathcal{X}_1, \ldots, \mathcal{X}_k \subseteq \mathcal{X}$, where each cluster $\mathcal{X}_j$ is associated with a representative time series C_j (which may or may not belong to $\mathcal{X}_j$), with the closest representative of each time series $X \in \mathcal{X}_j$ being C_j, i.e.,

$$\mathcal{X}_j = \left\{ X \in \mathcal{X} : d(X, C_j) = \min_{i \in [k]} \{ d(X, C_i) \} \right\}, \tag{1}$$

such that the total clustering cost, defined as

$$\text{Cost}(\mathcal{X}, k) = \sum_{j=1}^{k} \sum_{X \in \mathcal{X}_j} d(X, C_j), \tag{2}$$

is minimized.

2.2. Distance Functions

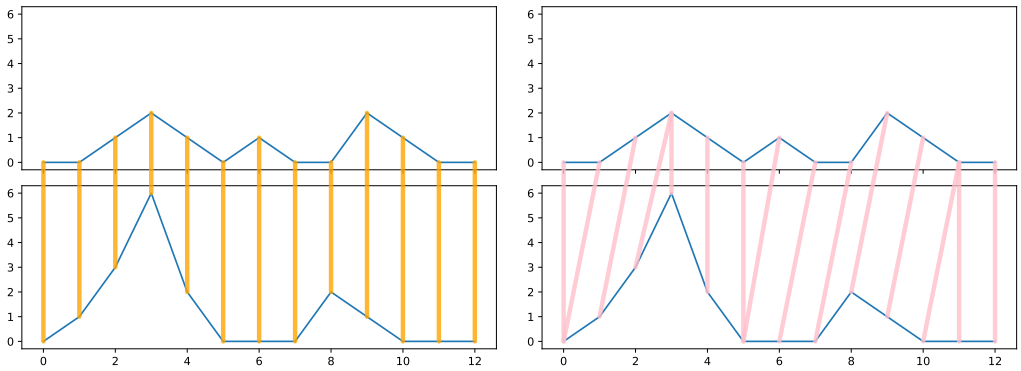
There is a very long list of possible distance functions among time series (see, e.g., ([2], Table 3)). In this work, we focus on a prominent representative of shape-based distance functions for time series, the Dynamic Time Warping (DTW) distance. For completeness, we first introduce the simpler Euclidean distance for time and then present DTW as an elastic generalization of it.

2.2.1. Euclidean Distance

The *Euclidean distance* [21] is a so-called *lockstep* distance, which can be applied only if two time series have the same number of data points, i.e., the same length. The Euclidean distance $L_2(X, Y)$ of two time series X and Y with T data points each is simply the L_2 norm of the L_2 distances between the corresponding data points (see also Figure 1a). Namely,

$$L_2(X, Y) = \sqrt{\sum_{i=1}^{T} (x_i - y_i)^2} \tag{3}$$

A generalization of the Euclidean distance, usually referred to as the *Minkowski distance* for time series, can be obtained by taking the L_p norm of the L_p distances between the corresponding data points (instead of the L_2 norm of the L_2 distances in (3)), for some fixed $p \geq 1$ or $p = \infty$. The main advantages of the Euclidean distance are that (i) it is simple and intuitive, and (ii) it can be computed in linear time in the size of the input. However, the Euclidean distance fails to deal with slight time shifts and/or periodical changes in the sampling frequency of the time series.



(**a**) Euclidean distance. (**b**) DTW distance.

Figure 1. Simple example outlining the difference between the Euclidean distance in (**a**) and the Dynamic Time Warping (DTW) in (**b**) for time series.

2.2.2. Dynamic Time Warping Distance

The *Dynamic Time Warping* (DTW) distance is a so-called *elastic* distance, which can deal with time series of different lengths, different sampling frequencies, and different time alignments. DTW seeks an optimal alignment (or warping) of the data points of two time series X and Y that minimizes the resulting pairwise Euclidean distance of the corresponding data points. DTW is regarded as one of the most robust and accurate distance functions for time series and has been extensively used in practice (see, e.g., [8–10,22] and the references therein).

The optimal alignment of two time series X and Y, with $X = ((x_1, t_1), \ldots, (x_T, t_T))$ and $Y = (y_1, \tau_1), \ldots, (y_V, \tau_V))$, which is used for the computation of DTW, is a sequence $W = ((T_1, V_1), \ldots, (T_l, V_l))$ with $l \leq T + V$ index pairs that aligns the data points of X and Y such that (i) $(T_1, V_1) = (1, 1)$ and $(T_l, V_l) = (T, V)$, i.e., the first and the last points of X and Y are aligned with each other; (ii) every point of X (resp. Y) is aligned with at least one point of Y (resp. X); and (iii) for every two consecutive index pairs (T_ℓ, V_ℓ), $(T_{\ell+1}, V_{\ell+1})$, $(T_\ell - T_{\ell+1}, V_\ell - V_{\ell+1}) \in \{(0, 1), (1, 0), (1, 1)\}$, i.e., the alignment sequence is increasing and without cross-alignments (see also Figure 1b).

Then, the DTW distance of two time series X and Y is defined as:

$$D(X, Y) = \min_{X\text{-}Y \text{ warping } W} \sqrt{\sum_{(T_\ell, V_\ell) \in W} (x_{T_\ell} - y_{V_\ell})^2} \tag{4}$$

We note that DTW is symmetric and satisfies $D(X, X) = 0$, but it does not satisfy the triangle inequality. The square of DTW can be computed in quadratic time $\Theta(TV)$, using dynamic programming based on the recursion below, which is similar to the recursion used for computing the Edit Distance of two strings:

$$D(X[i], Y[j]) = \begin{cases} \sum_{\ell=1}^{j} (x_1 - y_\ell)^2 & \text{if } i = 1 \\ \sum_{\ell=1}^{i} (x_\ell - y_1)^2 & \text{if } j = 1 \\ (x_i - y_j)^2 + & \\ \min\{D(X[i-1], Y[j-1]), D(X[i-1], Y[j]), D(X[i], Y[j-1])\} & \text{otherwise} \end{cases}$$

where $X[i] = ((x_1, t_1), \ldots (x_i, t_i))$ denotes the prefix of X consisting of X's first i data points and $Y[j] = ((y_1, \tau_1), \ldots (y_j, \tau_j))$ denotes the prefix of Y consisting of Y' first j data points.

We should note that the definition of DTW sometimes restricts the maximum number of data points that can be aligned with a single one, which leads to faster computation and possibly better practical results (see also [23]).

In this work, we focus on a generalization of DTW, referred to as β-DTW, for some fixed parameter $\beta \geq 0$, where the distance between two aligned data points (x_i, t_i) and (y_j, t_j) is computed as:

$$\delta_\beta((x_i, t_i), (y_j, t_j)) = (x_i - y_j)^2 + \beta(t_i - t_j)^2\,, \tag{5}$$

instead of simply $(x_i - y_j)^2$. Then, β-DTW is computed by the dynamic programming above by replacing the distance function $(x_i - y_j)^2$ with the more general $\delta_\beta((x_i, t_i), (y_j, t_j))$.

For $\beta = 0$, we obtain the standard DTW. As β grows larger, β-DTW penalizes the alignment of data points observed at quite different points in time. Using a moderate value of β proves useful in our Sparse Time-Series Clustering framework because the inducing points used for time-series representation can be located by Sparse Gaussian Process Regression at very different points in time. In fact, the locations of the inducing points highly depend on the shape and the variance of the regressed time series at different time intervals. Hence, the second term in $\delta_\beta(\cdot)$ serves to penalize significant differences in how two time series evolve. Moreover, as β increases above a certain (instance-dependent) threshold, β-DTW becomes a lockstep distance and essentially coincides with the Euclidean distance.

2.3. Evaluation Criteria of Sparse Time-Series Clustering

We focus on shape-based clustering, where the clustering algorithm seeks to minimize $\text{Cost}(\mathcal{X}, k)$, with $\mathcal{X}$ being a set of time series to be partitioned into a predefined number k of clusters. $\text{Cost}(\mathcal{X}, k)$ is given by (2), with respect to the β-DTW distance for every instance. To this end, we apply the k-means algorithm with the β-DTW distance for some fixed parameter $\beta \geq 0$, and the DTW Barycenter Averaging (DBA) algorithm [19] for updating the cluster representatives in each iteration. For each dataset $\mathcal{X}$, k-means is applied to the original time series in $\mathcal{X}$ and to the time series consisting of the inducing points placed by Sparse Gaussian Process Regression (SGPR) applied to each time series in $\mathcal{X}$.

A ground truth clustering is available for all our instances $(\mathcal{X}, k)$. Therefore, to evaluate the performance of our approach, we resort to the *Adjusted Rand Index* (ARI), an extensively used extrinsic clustering metric quantifying how much the clustering computed by our algorithm overlaps with the ground truth clustering. For completeness, we briefly define the Rand Index (RI) and the Adjusted Rand Index (ARI) below.

2.3.1. Rand Index

The *Rand Index* (RI) [24] quantifies the similarity of two clusterings (i.e., the ground truth clustering and the clustering computed by the algorithm) by counting the number of "correct" pairs of data points, which are both assigned either to the same or to different clusters in the two clusterings. More precisely, RI is defined as

$$\text{RI} = \frac{2\,(\text{SS} + \text{DD})}{n(n-1)}\,,$$

where n is the number of data points (and $n(n-1)/2$ is the total number of data point pairs) and $\text{SS} + \text{DD}$ is the number of "correct" pairs. Specifically, SS (resp. DD) is the number of pairs of data points that belong to the same cluster (resp. to different clusters) in both clusterings. By definition, $\text{RI} \in [0, 1]$. However, the threshold above which RI values are considered satisfactory strongly depends on the number of clusters k, with a completely random (and oblivious to the cluster sizes) clustering achieving an expected RI of $1/k$ against any given clustering.

2.3.2. Adjusted Rand Index

The *Adjusted Rand Index* (ARI) [25,26] is essentially a normalization of RI so that a random clustering obtains an ARI equal to 0. A simple and natural way to define ARI is:

$$\text{ARI} = \frac{\text{RI} - \mathbb{E}[\text{RI}]}{1 - \mathbb{E}[\text{RI}]}\,, \tag{6}$$

where $\mathbb{E}[\text{RI}]$ is computed over all random clusterings with given cluster sizes compared against the ground truth clustering. (6) defines ARI as the fraction by which the RI of the computed clustering (against a fixed ground truth clustering) outperforms the RI of a random clustering (with given cluster sizes, against the same ground truth clustering).

An equivalent definition of ARI is given in ([27], Section 2):

$$\text{ARI} = \frac{2\,(\text{SS} \cdot \text{DD} - \text{SD} \cdot \text{DS})}{(\text{SS} + \text{SD})(\text{SD} + \text{DD}) + (\text{SS} + \text{DS})(\text{DS} + \text{DD})}\,, \tag{7}$$

where SS (resp. DD) is the number of pairs of data points that belong to the same cluster (resp. to different clusters) in both clusterings and DS (resp. SD) is the number of pairs of data points they belong to different clusters (resp. to the same cluster) in the ground truth clustering and to the same cluster (resp. to different clusters) in the clustering computed by the algorithm.

A perfect agreement among two clusterings is denoted by $\text{ARI} = 1.0$, while an essentially random clustering is denoted by $\text{ARI} = 0.0$. ARI can take negative values, denoting

clusterings with an unusually high number of discordant data point pairs. However, correcting for $\mathbb{E}[\mathrm{RI}]$ in (6) implies that ARI values do not explicitly depend on the number k of clusters.

2.3.3. Evaluation

In this work, we resort to ARI to quantify the performance of the clusterings computed by our framework. More precisely, for every instance $(\mathcal{X}, k)$, with a set $\mathcal{X}$ of time series to be partitioned into a predefined number k of clusters, we calculate the ARI for the clustering computed by k-means on the original instance $(\mathcal{X}, k)$ and on the instance obtained by applying SGPR to the time series in $\mathcal{X}$.

Comparing the ARI of the two clusterings indicates the performance loss (or sometimes benefit) due to the sparsity of $\mathcal{X}$'s representation in our framework. Moreover, we use the *spread* and the *average difference* (which can be gain or loss), computed with respect to the ARI of the two algorithms on a family of instances, as a summary indicator of the performance of our Sparse Time-Series Clustering framework.

2.3.4. Average and Spread Difference of Two Clustering Methods

To compare two clustering methods for multiple instances, we need to aggregate the difference in ARI values of the two methods across all instances. To this end, we use the average difference and the second moment of the difference of the two methods' ARI values.

The *average difference* quantifies the average performance gain (or loss) of a clustering method against another one with respect to their ARI values across multiple instances:

$$\mathrm{Diff}_{(1,2)} = \sum_{i=1}^{N} \frac{\mathrm{ARI}^1(i) - \mathrm{ARI}^2(i)}{N} \tag{8}$$

We usually refer to $\mathrm{Diff}_{(1,2)}$ as $\mathrm{Gain}_{(1,2)}$, if $\mathrm{Diff}_{(1,2)} > 0$, as $\mathrm{Loss}_{(1,2)}$, if $\mathrm{Diff}_{(1,2)} < 0$.

The *spread* corresponds to the second moment of the difference between the ARI values of two clustering methods across multiple instances:

$$\mathrm{Spread} = \sum_{i=1}^{N} \frac{(\mathrm{ARI}^1(i) - \mathrm{ARI}^2(i))^2}{N} \tag{9}$$

where $\mathrm{ARI}^1(i)$ (resp. $\mathrm{ARI}^2(i)$) denotes the ARI value of algorithm 1 (resp. 2) for instance i and N is the total number of instances.

3. Gaussian Process Regression for Time Series

A typical approach to dealing with a sequence of individual data points, such as a time series, boils down to inferring a continuous function that approximately describes the entire sequence. There are a few popular approaches in this direction depending on the prior information about the model and on the complexity of the data itself (see, e.g., [28]).

If a time series can be described (or can be approximated) by a relatively simple function (i.e., a polynomial of degree d), we may use parametric fitting to estimate the function's unknown parameters from a few data points. Then, interpolation, or *regression*, can be used to essentially fill in the space between data points and to create a continuous function representation of the time series, which can be used as a way to predict new or hidden data points, as well as to reduce the size and the complexity of the time series representation. In the context of time series, we may regard regression as a supervised learning problem, where we wish to learn a continuous mapping f from the time domain to the domain of data points, given a relatively simple class of functions to select from (i.e., polynomials of degree d) and a relatively small set of data points.

Although regression may sound natural and practically appealing, in most practical applications, time series cannot be reasonably approximated by a fixed class of relatively

simple functions. Hence, in this work, we resort to (sparse) *Gaussian Process Regression* (GPR), a general approach relying on minimal assumptions about the nature of the time series.

GPR is a Bayesian nonparametric approach to regression, where instead of calculating a probability distribution over the finite set of parameters of a specific functional form, we calculate a probability distribution over all admissible functions that best approximates the time series at hand. In the following paragraphs, we briefly present the main ingredients of sparse Gaussian Process Regression (see also [14,29] for elaborate introductions to Gaussian Processes and their applications).

3.1. Gaussian Processes for Time Series

At a conceptual level, a *Gaussian Process* extends the notion of multivariate Gaussian distributions to infinite dimensions (and thus, to distributions over functions). Formally, a Gaussian Process is a stochastic process $(Z_1, \ldots, Z_t, \ldots)$ such that the joint distribution of every finite collection $(Z_{t_1}, \ldots, Z_{t_k})$ of its random variables is a *multivariate Gaussian distribution* ([14], Definition 2.1).

For this work, it is convenient to regard a Gaussian Process model as a probability distribution over continuous functions (or over time series). For simplicity and since in this work we mostly deal with real-valued time series, where the observed data points $x_i \in \mathbb{R}$, we restrict our attention to regression over functions that map points in time to real data points. Starting with the special case where a time series is evaluated in a fixed number n of points in time, for any n-dimensional vectors $\vec{x} = (x_1, \ldots, x_n)$ and $\vec{t} = (t_1, \ldots, t_n) \in \mathbb{R}^n$, a n-variate Gaussian distribution $\mathcal{N}(m(\vec{t}), k(\vec{t}, \vec{t}))$ determines the probability that the observed data points $(x_1, \ldots, x_n)$ at times $(t_1, \ldots, t_n)$ are drawn from a n-variate Gaussian distribution with mean function $m : \mathbb{R} \to \mathbb{R}$ and covariance function (a.k.a. *kernel* function) $k : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. In fact, given a time series $X = (\vec{x}, \vec{t})$, we want to compute a $m(\vec{t}) = (m(t_1), \ldots, m(t_n))$ and positive semidefinite covariance matrix $K = (k(t_i, t_j))_{i,j \in [n]})$ maximizing the probability that the observed data point vector $\vec{x}$ is drawn the n-variate Gaussian distribution $\mathcal{N}(m(\vec{t}), k(\vec{t}, \vec{t}))$.

The notion of a Gaussian Process naturally extends the idea of a n-variate Gaussian distribution to arbitrary dimensions, which allows us to regress over continuous functions and time series. A Gaussian process is defined by a *mean function* $m : \mathbb{R} \to \mathbb{R}$ and a *kernel function* $k : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. The mean function $m(\cdot)$ determines the expected value $m(t)$ of a data point $x_t \in \mathbb{R}$ at any time t. The kernel function $k(\cdot, \cdot)$ quantifies the correlation $k(t, t')$ between the observed data points x_t and $x_{t'}$ at any two times t and t'. We let $f \sim \mathcal{GP}(m, k)$ denote that the time series described by the function $f : \mathbb{R} \to \mathbb{R}$ (i.e., the values of the data points are $f(t)$ at all times t) follows a Gaussian Process $\mathcal{GP}$ with mean function m and kernel function k.

The mean function m determines the average value of data points over time and is usually normalized to 0 over the entire time horizon. The kernel function k determines the shape of the time series modeled by the Gaussian Process, in the sense that if two points in time t and t' are highly correlated (e.g., because t and t' are neighboring points in time, or because we expect that the observed data points at t and t' should be close to each other), the kernel function should favor time series f with similar values $f(t)$ and $f(t')$.

3.2. Kernel Functions for Time Series

Kernel functions play a central role in Gaussian Process Regression because they incorporate the information (and our assumptions) about the smoothness and the degree of correlation between data points in the time series that we aim to approximate. The kernel function $k : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ maps any two points in time t and t' to a real number that quantifies the expected similarity between the observed data points x_t and $x_{t'}$. In any finite dimensions n, the covariance matrix K of the corresponding n-variate Gaussian is computed by evaluating the kernel function $k(t, t')$ over all n^2 pairs of points t and t' in time. The kernel function should ensure that the resulting covariance matrix is always positive semidefinite.

A kernel function k is *stationary*, if for any two points in time t and t' and any translation u, it holds that $k(t, t') = k(t + u, t' + u)$. Specifically, the covariance $k(t, t')$ of any two points in time t and t' only depends on $t - t'$ and is invariant under translation. If the covariance $k(t, t')$ of any two points in time t and t' only depends on their distance $|t - t'|$, we say that the kernel function is *isotropic*. In the following, we discuss two widely used stationary (and isotropic) kernel functions, the *Radical Basis Function* (RBF) kernel and the *Matérn* kernel (see also ([14], Chapter 4)).

3.2.1. The Radial Basis Function Kernel

The Radial Basis Function (RBF) kernel (a.k.a. the Gaussian kernel or the Squared Exponential kernel) is defined as:

$$k_{\mathrm{RBF}(s,\ell)}(t, t') = s^2 \exp\left(-\frac{(t - t')^2}{2\ell^2}\right) \tag{10}$$

In (10), k has two hyperparameters: the *scale factor* s, which quantifies the deviation to the mean value m of a function f drawn from the corresponding Gaussian Process; and the *length scale* ℓ, which quantifies the strength of the correlation between the data points $f(t)$ and $f(t')$ of two points in time at distance $|t - t'|$. Gaussian Process Regression with the RBF kernel corresponds to Bayesian linear regression with an infinite number of basis functions. Due to its nice properties (see, e.g., ([14], Sections 4.2 and 4.3)), RBF has become the default kernel function in practical applications.

3.2.2. The Matérn Kernel

The class of Matérn kernels is a generalization of RBF kernels with an additional hyperparameter ν, which controls the smoothness of the kernel function. A Matérn kernel is defined as:

$$k_{M(s,\ell,\nu)}(t, t') = \frac{s^2 2^{\nu-1}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}|t - t'|}{\ell}\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu}|t - t'|}{\ell}\right), \tag{11}$$

where $\Gamma(\cdot)$ is the gamma function and $K_\nu(\cdot)$ is the modified Bessel function of the second kind. Larger values of ν result in smoother approximated time series, while as ν grows to ∞, the Matérn kernel becomes equivalent to the RBF kernel. We note that the cases where $\nu = 3/2$ and $\nu = 5/2$ have nice closed forms ([14], (4.17)) and are of special interest to practical applications. In this work, we use the Matérn kernel for $\nu = 3/2$.

3.3. Gaussian Process Regression

Gaussian Process Regression (GPR) applies the standard Bayesian regression approach to Gaussian Processes. In Bayesian regression, we first compute a posterior distribution on the parameters of an admissible functional form (e.g., polynomials of degree d) based on some prior information about these parameters (if available) and on the available data points. This is extended to a predictive posterior distribution, i.e., a distribution on the values of unseen data points, which is derived from the prior distribution on the parameters of the admissible functional form.

In our setting, we are given a time series $X = ((x_1, t_1), \ldots, (x_n, t_n))$ with n data points and aim to compute a posterior Gaussian Process, which provides a probability distribution over unseen data points $x(t)$ at any point in time t. We assume that each data point $x_i = f(t_i) + \epsilon_i$, where $f : \mathbb{R} \to \mathbb{R}$ is a latent function (for which do not make any particular assumptions) and $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ is an independent sample drawn from a white noise process with standard deviation σ.

For the Bayesian regression process, we make the standard assumption that the data are normalized to 0 over the entire time horizon. Hence, we consider Gaussian Processes with a zero-mean function, i.e., with mean function $m(t) = 0$ for all points in time t. Then,

the prior distribution is a n-variate Gaussian distribution conditioned on the input time series $X = ((x_1, t_1), \ldots, (x_n, t_n))$. Specifically, the distribution of $\vec{x}$ conditional on $\vec{t}$ is:

$$\vec{x} \,|\, \vec{t} \sim \mathcal{N}(0, k_\theta + \sigma^2)\,, \tag{12}$$

where $k_\theta : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a kernel function (e.g., an RBF kernel or a Matérn kernel) with hyperparameters θ and $\vec{x} = (x_1, \ldots, x_n)$ are noisy values sampled from the latent function $f : \mathbb{R} \to \mathbb{R}$ at times $\vec{t} = (t_1, \ldots, t_n)$. Due to noise, the covariance matrix of $\mathcal{N}$ is $\tilde{K}_{\theta,\sigma} = K_\theta + \sigma^2 I$, where $K_\theta = (k_\theta(t_i, t_j))_{i,j\in[n]}$ is the positive semidefinite covariance matrix computed by applying the kernel function k_θ to $\vec{t}$, σ is the standard deviation of the white noise, and I is the identity matrix.

The hyperparameters θ of the kernel function k_θ and the standard deviation of the observation noise σ are optimized based on the input time series $X = ((x_1, t_1), \ldots, (x_n, t_n))$. Hyperparameter optimization in (12) corresponds to a non-convex optimization problem, which is typically solved through gradient-based optimization techniques, such as Adam [30] and L-BFGS [31]. The running time is typically cubic, and the space requirement is typically quadratic in the size of the input data, i.e., $O(n^3)$ and $O(n^2)$ for time and space, respectively, with the computational complexity dominated by the time required to invert the covariance matrix $\tilde{K}_{\theta,\sigma}$.

We use the resulting posterior Gaussian Process $\mathcal{GP}(0, k_\theta)$ in order to estimate the values of m data points $\vec{x}' = (x'_1, \ldots, x'_m)$ observed at times $\vec{t}\,' = (t'_1, \ldots, t'_m)$, assuming that the values x'_j are sampled from the same latent function $f : \mathbb{R} \to \mathbb{R}$ used for X and thus, the joint distribution of $\vec{x}$ and $\vec{x}'$ is given by $\mathcal{GP}(0, k_\theta)$.

Therefore, the joint distribution of $\vec{x}$ and $\vec{x}'$ is an $(n+m)$-variate Gaussian distribution obtained by applying the Gaussian Process $\mathcal{GP}(0, k_\theta)$ to time vectors $\vec{t}$ and $\vec{t}\,'$. The covariance matrix $K_\theta(\vec{t}, \vec{t}\,')$ is a positive semidefinite $(n+m) \times (n+m)$ matrix with the following form:

$$K_\theta(\vec{t}, \vec{t}\,') = \begin{bmatrix} K_\theta(\vec{t}, \vec{t}) + \sigma^2 I & K_\theta(\vec{t}, \vec{t}\,') \\ K_\theta(\vec{t}\,', \vec{t}) & K_\theta(\vec{t}\,', \vec{t}\,') \end{bmatrix},$$

where the covariance (sub)matrix $K_\theta(\vec{a}, \vec{b})$ is defined as $K_\theta(\vec{a}, \vec{b}) = (k_\theta(a_i, b_j))_{a_i\in\vec{a}, b_j\in\vec{b}}$.

Conditioning on X and its posterior distribution $\mathcal{N}(0, k_\theta + \sigma^2)$, we obtain the Gaussian predictive distribution of $\vec{x}'$. Specifically, we obtain that the values of the data points $\vec{x}'$ follow a m-variate Gaussian distribution with mean value vector $\vec{m}'$ and covariance matrix K' as given below:

$$\vec{m}' = K_\theta(\vec{t}\,', \vec{t})(K_\theta(\vec{t}, \vec{t}) + \sigma^2 I)^{-1}\vec{x} \tag{13}$$

$$K' = K_\theta(\vec{t}\,', \vec{t}\,') - K_\theta(\vec{t}\,', \vec{t})(K_\theta(\vec{t}, \vec{t}) + \sigma^2 I)^{-1} K_\theta(\vec{t}, \vec{t}\,') \tag{14}$$

We note that the predicted mean value of $\vec{x}'$ in (13) is a linear combination of the input values $\vec{x}$. Equivalently, one can obtain the mean value of each x'_j as a linear combination $\sum_{i=1}^{n} \alpha_j k_\theta(t_i, t'_j)$, with coefficients $\vec{\alpha} = (K_\theta(\vec{t}, \vec{t}) + \sigma^2 I)^{-1}\vec{x}$. We also note that the covariance matrix in (14) does not directly depend on the input values $\vec{x}$ (but it depends on the points in time $\vec{t}$ when these values are observed). We refer the interested reader to ([14], Section 2.2) for more details on Gaussian Process Regression.

3.4. Sparse Gaussian Process Regression

From a conceptual viewpoint, Gaussian Process Regression (GPR) is versatile and elegant, with a simple conceptual structure and many desirable theoretical properties. In practice, however, GPR can only deal with regression tasks of moderate size, with at most a few thousand input data points, due to the cubic time complexity required for computing the posterior and the predictive posterior distributions. As a result, several sparse approximation methods have been proposed to make GPR practically applicable to settings with a medium to large number of input data points (see, e.g., [15,17] and the

references therein). These sparse GPR methods aim to represent the underlying Gaussian Process using a much smaller set of m, with $m \ll n$, inducing points, which can be learned so that they are highly informative about the actual posterior Gaussian Process. Sparse GPR methods achieve a time complexity of $O(m^2 n)$ and a space complexity of $O(m^2 + n)$ for approximating the posterior and the predictive posterior Gaussian Processes.

A standard approach to optimizing the sparse Gaussian Process is by minimizing its Kullback–Leibler (KL) divergence to the actual (and possibly intractable) Gaussian Process. In general, optimal (with respect to the KL divergence) sparse Gaussian processes do not have a closed form (as, e.g., happens with the predictive Gaussian process in (13) and (14)). Then, Variational Inference (VI) can be used to approximate the actual posterior with a variational distribution.

In this work, we use the *Variational Free Energy* (VFE) framework (a.k.a. *Sparse Gaussian Process Regression* (SGPR)), introduced by Titsias [16]. SGPR uses a small number of carefully selected inducing points, along with variational inference, to obtain a low-rank approximation (with respect to the KL divergence) to the actual Gaussian Process. In SGPR, the total number m of inducing points is chosen in advance so that the overall time and space complexity are acceptable. Their locations in time and their values are optimized so that more inducing points are located at time intervals where the time series exhibits a more complex behavior (see also Figure 2).

Other approaches to sparse GPR include treating inducing point selection as a continuous optimization problem [32] and online approaches where the sparse Gaussian Process is iteratively trained by processing each input individually [33,34]. We refer the interested reader to [17] for an elaborate treatment of Sparse Gaussian Process Regression.

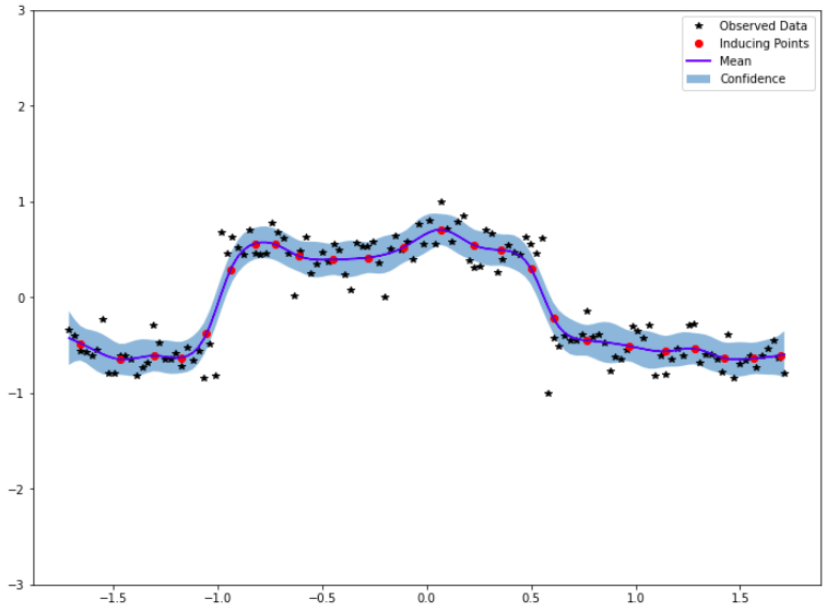


Figure 2. Sparse Gaussian Process Regression (SGPR) uses a predetermined number of inducing points (red dots) to compute a sparse approximate representation of a given time series (black stars).

4. The Sparse Time-Series Clustering Framework

The *Sparse Time-Series Clustering* (STSC) framework consists of two stages, as outlined in Algorithm 1. The input instance consists of $(\mathcal{X}, k)$, where $\mathcal{X}$ is a set of N time series to be partitioned into k clusters. The first stage is to approximate each time series $X_j \in \mathcal{X}$ with a sparse time series X'_j consisting of m inducing data points by applying Sparse Gaussian Process Regression (see Section 4.1 and Algorithm 2). The second stage is to cluster the reduced instance $(\{X'_1, \ldots, X'_N\}, k)$ using the k-means algorithm with the β-DTW distance, for some fixed parameter $\beta \geq 0$, and the DTW Barycenter Averaging (DBA) algorithm (see Section 4.2 and Algorithm 3). The outcome of the second stage is a tuple with k

representative time series, which define a k-clustering of the reduced dataset $\{X'_1, \ldots, X'_N\}$ (and the corresponding k-clustering for the original dataset $\mathcal{X}$) by (1).

Algorithm 1 Sparse Time-Series Clustering Framework

Input : Instance $\mathcal{X} = (X_1, \ldots, X_N)$ with N time series, number of clusters k
Output: k cluster representatives $(C_1, \ldots, C_k)$ (which k-cluster $\mathcal{X}$ into $(\mathcal{X}_1, \ldots, \mathcal{X}_k)$.
`framework`($\mathcal{X}$, m, seed):
  for $j \in [N]$ **do**
    $X'_j \leftarrow$ `modeling`(X_j, m) {▷ SGPR with m inducing points on each X_j ◁}
  end for
  $(C_1, \ldots, C_k) \leftarrow$ `clustering`$(\{X'_1, \ldots, X'_N\}, k,$ seed$)$
  {▷ k-means clustering of reduced instance ◁}
  return $(C_1, \ldots, C_k)$

Algorithm 2 Modeling through Sparse Gaussian Process Regression

Input : time series $X = ((x_1, t_1), \ldots, (x_T, t_T))$, number of inducing points m
Output: approximate time series $X' = ((x'_1, \tau_1), \ldots, (x'_m, \tau_m))$ with m inducing points
`modeling` $(X = ((x_1, t_1), \ldots, (x_T, t_T)), m)$:
  model $\leftarrow$ `train_SGPR`(X, m)
  {▷ apply SGPR for the given number m of inducing points ◁}
  $(x'_1, \tau_1), \ldots, (x'_m, \tau_m)) \leftarrow$ `extract_induced_points`(model, m)
  return $X' = ((x'_1, \tau_1), \ldots, (x'_m, \tau_m))$

Algorithm 3 k-Means Clustering

Input : dataset $\mathcal{X} = (X_1, \ldots X_N)$ with N time series, number k of clusters, seed for initialization
Output: $\mathcal{C} = (C_1, \ldots, C_k)$ with k cluster representatives
`clustering` $(\mathcal{X}, k,$ seed$)$:
  for $j \in [k]$ **do**
    $r \leftarrow$ `random_generator`(seed, N)
    {▷ randomly choose initial cluster representatives ◁}
    $C_j^{(0)} \leftarrow X_r$
  end for
  for $i \in [100]$ **do**
    $(C_1^{(i)}, \ldots, C_k^{(i)}) \leftarrow$ `DBA`$(\mathcal{X}, k, (C_1^{(i-1)}, \ldots, C_k^{(i-1)}))$
    {▷ update representatives with DBA ◁}
    if $(C_1^{(i)}, \ldots, C_k^{(i)}) \approx (C_1^{(i-1)}, \ldots, C_k^{(i-1)})$ **then**
      break {▷ k-means converged to a k-clustering ◁}
    end if
  end for
  return $(C_1, \ldots, C_k)$

In the following (and unless stated otherwise), we consider a dataset $\mathcal{X} = \{X_1, \ldots, X_N\}$ of N univariate time series, where each time series $X = ((x_1, t_1), \ldots, (x_T, t_T)) \in \mathcal{X}$ consists of T data points. We assume that all time series in the same dataset have the same length T (nevertheless, our framework can be applied to datasets with time series of different lengths without any modification).

4.1. Modeling through Sparse Gaussian Process Regression

In this stage, we apply the framework of Sparse Gaussian Process Regression, as outlined in Section 3, in order to obtain a sparse approximation $X' = (x'_1, \tau_1), \ldots, (x'_m, \tau_m))$ of each time series $X = ((x_1, t_1), \ldots, (x_T, t_T))$ in the original dataset $\mathcal{X}$ (see also Algorithm 2).

We use the Variational Free Energy (VFE) (a.k.a. Sparse Gaussian Process Regression (SGPR)) approach, outlined in Section 3.4, using the Matérn kernel with parameter $\nu = 3/2$. Each time series $X = ((x_1, t_1), \ldots, (x_T, t_T))$ is approximated by a sparse time series $X' = ((x'_1, \tau_1), \ldots, (x'_m, \tau_m))$ with $m \ll T$ inducing data points.

A logarithmic (in length T of the original time series) number of inducing points suffices for a reasonably good approximation of the original time series. More specifically, we use $m = \gamma \log_2 T$, for $\gamma \in \{1, 2, 3, 4, 5\}$ (and with the resulting number rounded to the closest integer), in our experimental evaluation. Thus, the time complexity of this step is $O(T \log^2 T)$, and the space required is $O(T + \log^2 T)$.

4.2. Clustering Stage

The second and final stage of our framework is to partition the dataset $\mathcal{X}' = (X'_1, \ldots, X'_N)$, consisting of N sparse time series with m inducing points each, into k-clusters using the k-means algorithm for time series, with the DTW Barycenter Averaging (DBA) algorithm [19] used for updating the cluster representatives in each iteration of k-means.

We apply k-means with the β-DTW distance, defined in Section 2.2, with $\beta = \frac{\alpha T}{m}$. We use $\alpha \in \{0, 10^{-4}, 10^{-3}, 10^{-2}\}$ in our experimental evaluation. The intuition behind setting β to a multiple of T/m is that we expect that each inducing point serves as a representative of approximately T/m points, with increasing density of inducing points in time intervals where the time series exhibits more complex behavior. Hence, we want to (mildly) penalize with a multiple of T/m cases where different time series exhibit a high density of inducing points in distant time intervals.

The time complexity of k-means with the β-DTW distance and the DBA algorithm for updating the cluster representatives is $\Theta(INkm^2) = O(INk \log^2 T)$, where I is the number of iterations and $O(m^2)$ is the time required to compute the β-DTW distance between a pair of sparse time series, and $\Theta(Nkm^2T) = \Theta(Nk \log^2 T)$ is the time complexity of a single iteration of k-means. In our experimental evaluation, we run k-means and DBA for a maximum of $I = 100$ iterations.

4.3. Time Complexity

The overall time complexity of our Sparse Time-Series Clustering framework is $\Theta(NT \log^2 T + INk \log^2 T)$, where the first term corresponds to the time complexity of the Sparse Gaussian Process Regression with $m = \Theta(\log T)$ inducing points and the second term corresponds to the running time of k-means with the β-DTW distance when applied to N sparse time series with $m = \Theta(\log T)$ inducing points each. As expected, the running time of our framework crucially depends on the number m of inducing points.

In our experimental evaluation, we compute a baseline clustering by applying Algorithm 3. The overall time complexity for computing a baseline clustering with k-means on the original dataset with N time series of T data points each is $\Theta(INkT^2)$.

Therefore, the asymptotic running time of our Sparse Time-Series Clustering framework is about $\Theta(\max\{T^2/\log^2 T, IkT/\log^2 T\})$ times faster than the asymptotic running time of directly applying k-means to the original dataset. Intuitively, the asymptotic running time of our framework is $\Omega(T^2/\log^2 T)$ times faster than the standard k-means because the improved running time for computing DTW from $\Theta(T^2)$ in the original data to $\Theta(\log^2 T)$ in the sparsified data.

5. Experimental Evaluation

5.1. Datasets

The University of California (UCR) Time-Series Classification Archive [20] is one of the most widely used and the largest labeled time-series data archives for classification,

consisting of 128 datasets. Each dataset is divided into training and test data and is accompanied by performance indicators of several algorithms with different parameter settings. In this work, we use the univariate datasets of the UCR archive in order to support the claim that our Sparse Time-Series Clustering (STSC) framework leads to clusterings with respect to the β-DTW distance of similar (or even improved) quality (compared against applying k-means to the original datasets), but with significantly improved running time. Baseline results are available for most of the UCR datasets in [22]. Nevertheless, we chose to run the baseline k-means algorithm on all the datasets used for experimental evaluation. For the scope of this work, we focused on univariate datasets and omitted datasets for which, due to computational power considerations, we were not able to run the k-means algorithm with DTW metric (see Section 5.6 for more details).

The datasets used in our experimental evaluation are synthetic, semi-synthetic, or real and originate from various domains. Each dataset is univariate and contains from 40 to 5000 time series. Although the time series within each dataset have the same length, the length varies across datasets, ranging from 60 to 1882. A concise summary of these datasets is provided in Table 1, including information such as the number of time series, the number of clusters, the length of each time series, and dataset type.

Table 1. Dataset Description.

Dataset	Size	Length	No. of Classes	Type
Adiac	781	176	37	IMAGE
ArrowHead	211	251	3	IMAGE
Beef	60	470	5	SPECTRO
BeetleFly	40	512	2	IMAGE
BirdChicken	40	512	2	IMAGE
Car	120	577	4	SENSOR
CBF	930	128	3	SIMULATED
Coffee	56	286	2	SPECTRO
Computers	500	720	2	DEVICE
CricketX	780	300	12	MOTION
CricketY	780	300	12	MOTION
CricketZ	780	300	12	MOTION
DiatomSizeReduction	322	345	4	IMAGE
DistalPhalanxOutlineAgeGroup	539	80	3	IMAGE
DistalPhalanxCorrect	876	80	2	IMAGE
DistalPhalanxTW	539	80	6	IMAGE
Earthquakes	461	512	2	SENSOR
ECG200	200	96	2	ECG
ECGFiveDays	884	136	2	ECG
FaceAll	2250	131	14	IMAGE
FaceFour	112	350	4	IMAGE
FacesUCR	2250	131	14	IMAGE
FiftyWords	905	270	50	IMAGE
Fish	350	463	7	IMAGE
GunPoint	200	150	2	MOTION
Ham	214	431	2	SPECTRO
Herring	128	512	2	IMAGE
InsectWingbeatSound	2200	256	11	SENSOR
ItalyPowerDenand	1096	24	2	SENSOR
LargeKitchenAppliances	750	720	3	DEVICE
Lightning2	121	637	2	SENSOR
Lightning7	143	319	7	SENSOR

Table 1. *Cont.*

Dataset	Size	Length	No. of Classes	Type
Meat	120	448	3	SPECTRO
MedicalImages	1141	99	10	IMAGE
MiddlePhalanxOutlineAgeGroup	554	80	3	IMAGE
MiddlePhalanxOutlineCorrect	891	80	2	IMAGE
MiddlePhalanxTW	553	80	6	IMAGE
MoteStrain	1272	84	2	SENSOR
OliveOil	60	570	4	SPECTRO
OSULeaf	442	427	6	IMAGE
PhalangesOutlinesCorrect	2658	80	2	IMAGE
Plane	210	144	7	SENSOR
ProximalPhalanxOutlineAgeGroup	605	80	3	IMAGE
ProximalPhalanxOutlineCorrect	891	80	2	IMAGE
ProximalPhalanxTW	605	80	6	IMAGE
RefrigerationDevices	750	720	3	DEVICE
ShapeletSim	200	500	2	SIMULATED
ShapesAll	1200	512	60	IMAGE
SmallKitchenAppliances	750	720	3	DEVICE
SonyAIBORobotSurface1	621	70	2	SENSOR
SonyAIBORobotSurface2	980	65	2	SENSOR
Strawberry	983	235	2	SPECTRO
SwedishLeaf	1125	128	15	IMAGE
Symbols	1020	398	6	IMAGE
SyntheticControl	600	60	6	SIMULATED
ToeSegmentation1	268	277	2	MOTION
ToeSegmentation2	166	343	2	MOTION
Trace	200	275	4	SENSOR
TwoLeadECG	1162	82	2	ECG
TwoPatterns	5000	128	4	SIMULATED
Wine	111	234	2	SPECTRO
WordSynonyms	905	270	25	IMAGE
Worms	258	900	5	MOTION

5.2. Experimental Setting

Both our Sparse Time-Series Clustering framework and the baseline, which applies k-means to the original datasets, are implemented in Python. For the application of k-means to the sparse (resp. the original) dataset, we use β-DTW (resp. the standard DTW) and DBA for updating the cluster representatives in each iteration. We use the GPyTorch library [35] for the implementation of Sparse Gaussian Process Regression and the Tslearn package [36] for the implementation of k-means. Our experiments run on an Intel(R) Xeon(R) Silver 4210 CPU (2.20 GHz) with 16 GB of RAM.

5.3. Parameter Selection and Tuning

We run our experiments with a logarithmic (in the length T of the original time series) number m of inducing points. More specifically, we run our experiments with $m = \gamma \log_2 T$ (rounded to the closest integer), for $\gamma \in \{1, 2, 3, 4, 5\}$. For the Sparse Gaussian Process Regression, we choose a constant-mean prior for the Gaussian Process and use the Matérn kernel with parameter $\nu = 1.5$. We use Adam [30] to optimize the parameters of SGPR. We use a learning rate of $1/10$ for optimizing the parameters of the Gaussian Process (i.e., the hyperparameters of the Matérn kernel and the standard deviation of the noise) and a learning rate of $1/(10\gamma)$ for optimizing the locations of the inducing points. For the initialization of SGPR's inducing point location optimization, we divide the time horizon of the dataset into m equally sized intervals. Thus, we avoid ending up with quite different inducing point locations for different initializations, which may happen due to the non-convexity of SGPR's objective and its sensitivity to different initializations.

We should highlight that SGPR's objective function is highly sensitive to Adam's learning rate. If the learning rate is too large, we end up with inducing points outside the time horizon of the original time series, while if the learning rate is too low, the learning becomes local, and inducing points tend to reflect the behavior of the time series in a small interval around them. Hence, we had to carefully select the learning rates for our experiments, with the value of $1/10$ proven to be a good and consistent choice. A summary of the parameters used in the experimental evaluation, with a short description of their role and their values, can be found in Table 2.

Table 2. Collection of parameters (with their role in our approach) and their corresponding values used in experimental evaluation. We recall that T denotes the length of the time series, N is the number of time series in the instance, and k is the number of clusters.

Parameter	Algorithm—Role	Value
learning rate	Adam, parameter optimization SGPR	$1/10$
learning rate	Adam, optimization of inducing point locations in SGPR	$1/(10\gamma)$, for $\gamma \in \{1,2,3,4,5\}$
ν	Matérn kernel, SGPR	$3/2$
m	# inducting points in SGPR	$\gamma \log_2 T$, for $\gamma \in \{1,2,3,4,5\}$
α	$(\alpha T/m)$-DTW distance for clustering	$\in \{0, 10^{-4}, 10^{-3}, 10^{-2}\}$
I	max # iterations of k-means and DBA	100

For the k-means algorithm, we use the β-DTW distance with $\beta = \alpha T/m$ (which we often denote $(\alpha T/m)$-DTW), for $\alpha \in \{0, 10^{-4}, 10^{-3}, 10^{-2}\}$ and the DTW Barycenter Averaging (DBA) [19] for updating the cluster representatives in each iteration. We run k-means and DBA for a maximum of 100 iterations each.

We execute 10 runs of Algorithm 1 for each sparse dataset and each parameter combination (we have 5 choices for γ times 4 choices for α, which makes 12 different parameter combinations for each dataset), showcasing the average CPU time and the average Adjusted Rand Index (ARI) against the ground truth clustering provided in the UCR archive. In each run, the initial set of k cluster representatives is chosen randomly. Nevertheless, the seeds were predefined for each run to ensure reproducibility and facilitate fair comparisons between different methods.

To assess our approach against the standard k-means, we opted to compute a baseline ARI for each original dataset from scratch by executing 10 runs of Algorithm 3 for each of them with the standard DTW distance (with $\beta = 0$). As for the sparse case, the initial set of k cluster representatives is chosen randomly with predefined seeds. For each original dataset, we logged the average CPU time and the average Adjusted Rand Index (ARI) against the ground truth clustering. Therefore, we ensure consistent initialization and CPU time reporting across all runs, facilitating a comprehensive comparison and evaluation of run times.

5.4. Dataset Level Assessment

Our experimental evaluation indicates that Algorithm 1 (Sparse Time-Series Clustering), with sufficiently many (but still logarithmic in T) inducing points and for relatively small values of α in $(\alpha m/T)$-DTW, computes clusterings with ARI metrics quite similar to the ARI metrics of the baseline, computed by applying Algorithm 3 (i.e., standard k-means) to the original datasets. However, the asymptotic running time of Algorithm 1 is $\Omega(T^2/\log^2 T)$ times faster than the asymptotic running time of Algorithm 3.

Following the practices described in [22], in our experimental evaluation, we report the average ARI score over 10 runs of Algorithms 1 and 3 and also rank the performance (according to the average ARI in decreasing order) of Algorithms 1 and 3 with different parameter configurations in all the datasets of Table 1. A comprehensive summary of the average ARI and the average rank for each method (over all Table 1 UCR datasets for each parameter configuration) can be found in Table 3.

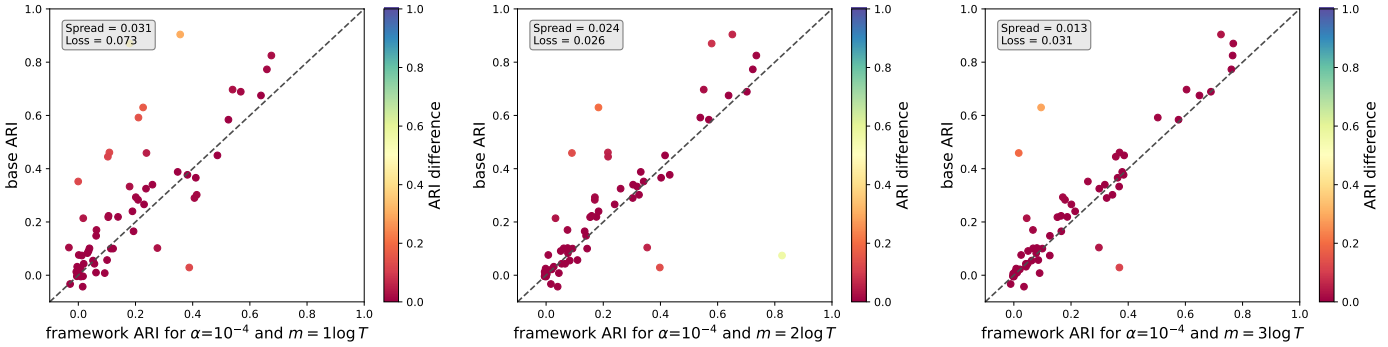
Table 3. Summarized ARI averages for different parameter configurations. The average relative order of the results achieved by each algorithm is reported in parentheses. ARI values for each dataset are reported in Table A1 (where $\alpha = 0$) and in Table A2 (where we report average ARI over all different values of $\alpha \in \{0, 10^{-4}, 10^{-3}, 10^{-2}\}$) in Appendix A.

	$(\alpha T/m)$-DTW					
Parameter α	**Baseline**	**$1 \cdot \log T$**	**$2 \cdot \log T$**	**$3 \cdot \log T$**	**$4 \cdot \log T$**	**$5 \cdot \log T$**
0	0.249 (2.524)	0.173 (4.270)	0.222 (3.683)	0.217 (3.492)	0.220 (3.762)	0.225 (3.270)
10^{-4}	0.249 (2.587)	0.175 (4.365)	0.222 (3.746)	0.217 (3.635)	0.220 (3.635)	0.226 (3.032)
10^{-3}	0.249 (2.746)	0.172 (4.508)	0.213 (3.841)	0.214 (3.317)	0.215 (3.444)	0.224 (3.143)
10^{-2}	0.249 (2.762)	0.162 (4.381)	0.196 (4.127)	0.210 (3.476)	0.215 (3.492)	0.228 (2.762)
best	0.249 (3.254)	0.193 (4.238)	0.238 (3.810)	0.238 (3.333)	0.243 (3.333)	0.252 (3.032)
mean	0.249 (2.556)	0.171 (4.413)	0.213 (3.857)	0.215 (3.556)	0.217 (3.587)	0.226 (3.032)

In each of the first four lines of Table 3, we compare the average ARI of the standard k-means (baseline) with the average ARI of our framework for a different number of inducing points while maintaining the parameter α in $(\alpha T/m)$-DTW constant. In the fifth (resp. the sixth) line, we take the best (resp. the mean) ARI over all values of α, for each different number of inducing points $\gamma \log_2 T$, for $\gamma \in \{1, 2, 3, 4, 5\}$.

In Table 3, we observe that average ARI values improve as the number of inducing points increases (for every fixed value of α, see also the spread diagrams in Figure 3, where we use $\alpha = 10^{-4}$). Moreover, for $m \leq 4 \log_2 T$ inducing points, average ARI values slightly deteriorate as the value of α increases (see also the spread diagrams in Figure 4, where we use $m = 2 \log_2 T$), while for $m = 5 \log_2 T$ inducing points, average ARI values marginally improve as α increases (see also the spread diagrams in Figure 5, where we use $m = 5 \log_2 T$). We note that as the number m of inducing points increases, $(\alpha T/m)$-DTW becomes less sensitive in an increase in α. Then, a combination of many inducing points and $\alpha \in [10^{-3}, 10^{-2}]$ produces quite satisfactory results (see also the spread diagrams in Figure 5c,d. For $m = 5 \log_2 T$ inducing points, keeping the best ARI over all values of α for each dataset gives clusterings of marginally better quality on average compared against the clusterings produced by the standard k-means algorithm (see the best ARI for $m = 5 \log_2 T$ in the best line of Table 3 and the spread diagram in Figure 5e). On the other hand, even for $m = 5 \log_2 T$ inducing points, any fixed value of α results in clusterings of slightly worse quality on average compared against the clusterings produced by the standard k-means algorithm (see the mean ARI for $m = 5 \log_2 T$ in the mean line of Table 3 and the spread diagram in Figure 5f).

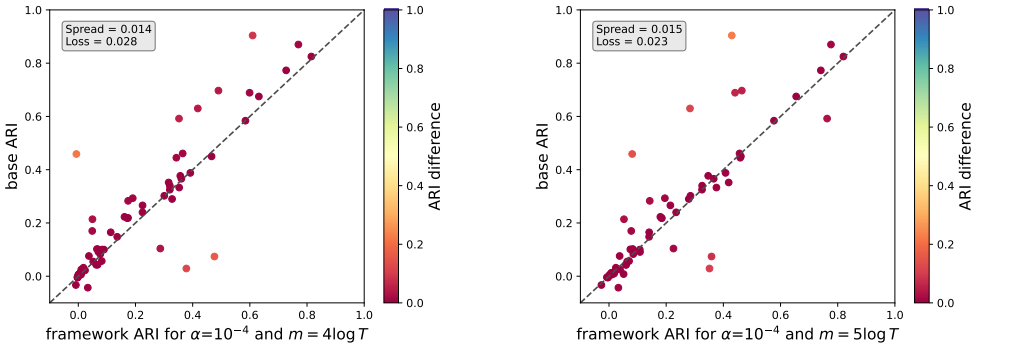
Even though it allows for conclusions that are informative and easy to grasp, averaging ARI results across different datasets (and possibly across different parameter configurations) is inadequate for a comprehensive evaluation of the proposed framework for Sparse Time-Series Clustering. A particularly poor or particularly good performance in specific datasets for certain parameter configurations may significantly affect the average ARI, potentially leading to misleading conclusions (notice also the standard deviations in Table A2). A characteristic example is the TwoPatterns dataset (see the corresponding rows in Tables A1 and A2), where the standard k-means achieves an average ARI of 0.870, while the average ARI of our framework for different numbers of inducing points (where average is taken across all values of α) ranges from 0.304 to 0.825 (see the corresponding row in Table A1 and the large standard deviations in the corresponding row of Table A2).



(**a**) ARI spread with $m = \log_2 T$ (loss = 0.073, spread = 0.031).

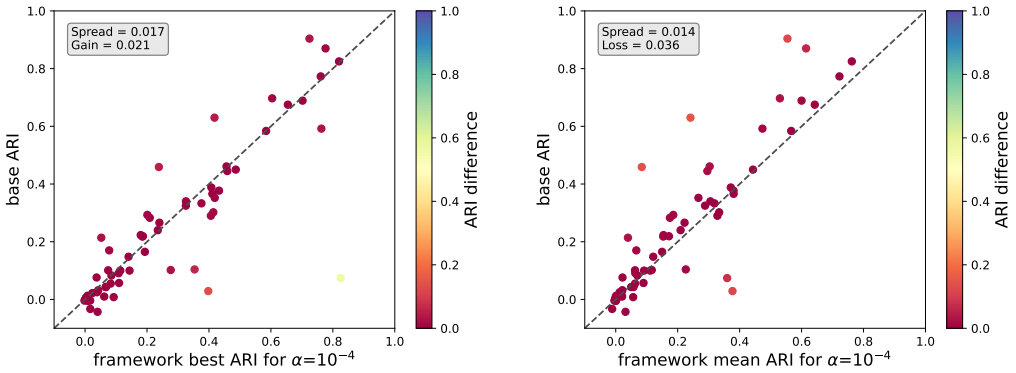
(**b**) ARI spread with $m = 2\log_2 T$ (loss = 0.026, spread = 0.024)

(**c**) ARI spread with $m = 3\log_2 T$ (loss = 0.031, spread = 0.013).



(**d**) ARI spread with $m = 4\log_2 T$ (loss = 0.028, spread = 0.014).

(**e**) ARI spread with $m = 5\log_2 T$ (loss = 0.023, spread = 0.015).



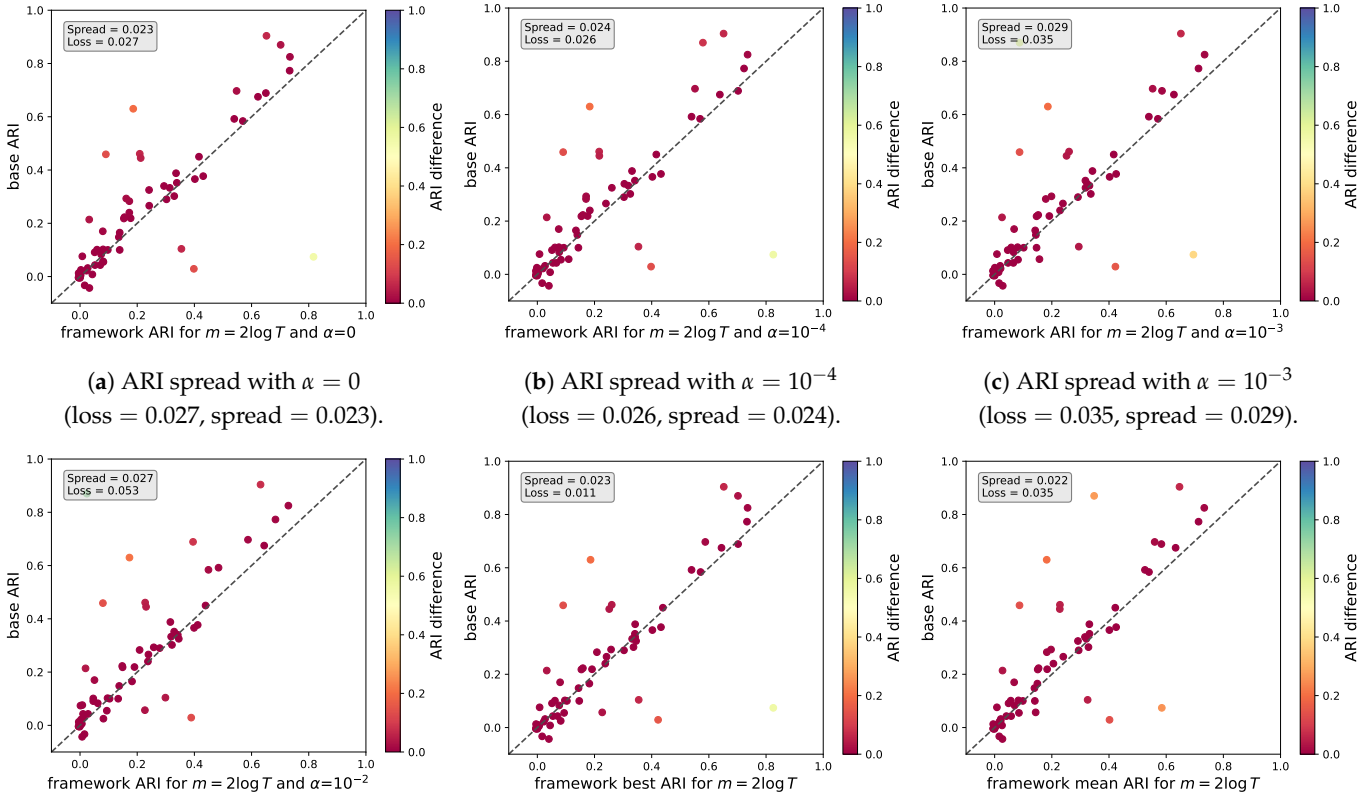
(**f**) Best ARI spread (gain = 0.021, spread = 0.017).

(**g**) Mean ARI spread (loss = 0.036, spread = 0.014).

Figure 3. Spread diagrams with comparative ARI results of Algorithm 1 against Algorithm 3 with $(10^{-4}T/m)$-DTW and $m = \gamma \log_2 T$ inducing points, for $\gamma \in \{1, 2, 3, 4, 5\}$. The spread and the average loss of the sparse framework are reported in the caption and in the upper left corner. We observe a slight improvement in clustering quality as the number of inducing points increases. We observe a small average gain in (**f**), where we keep the best ARI, and a small average loss in (**g**), where we take the mean ARI in each dataset (both across all values of $\gamma \in \{1, 2, 3, 4, 5\}$).

To provide a more detailed picture of the quality of clusterings produced by our framework for different datasets and how they compare against the clustering produced by the standard k-means, in Appendix A, we present in Table A1 the average ARI for each dataset for the baseline and our framework with $m = \gamma \log_2 T$ inducing points, for each value of $\gamma \in \{1, 2, 3, 4, 5\}$ and $\alpha = 0$, where average ARI is taken across all 10 runs for both the standard k-means and our framework. In Table A2, we present the same information

for all datasets along with the standard deviation computed across all 10 runs (for standard k-means) and all 10 runs and all different values of $\alpha \in \{0, 10^{-4}, 10^{-3}, 10^{-2}\}$. The information in Table A2 demonstrates the importance of careful (and possibly dependent on the parameters of the datasets) tuning. Nevertheless, the mean (across all different datasets and all values of α) loss of our framework for $5 \log_2 T$ inducing points is small, and keeping the best clustering for each dataset results in a small improvement in the average clustering quality of Algorithm 1 compared against the standard (and way more time demanding) k-means.



(**a**) ARI spread with $\alpha = 0$ (loss = 0.027, spread = 0.023).

(**b**) ARI spread with $\alpha = 10^{-4}$ (loss = 0.026, spread = 0.024).

(**c**) ARI spread with $\alpha = 10^{-3}$ (loss = 0.035, spread = 0.029).

(**d**) ARI spread with $\alpha = 10^{-2}$ (loss = 0.053, spread = 0.027).

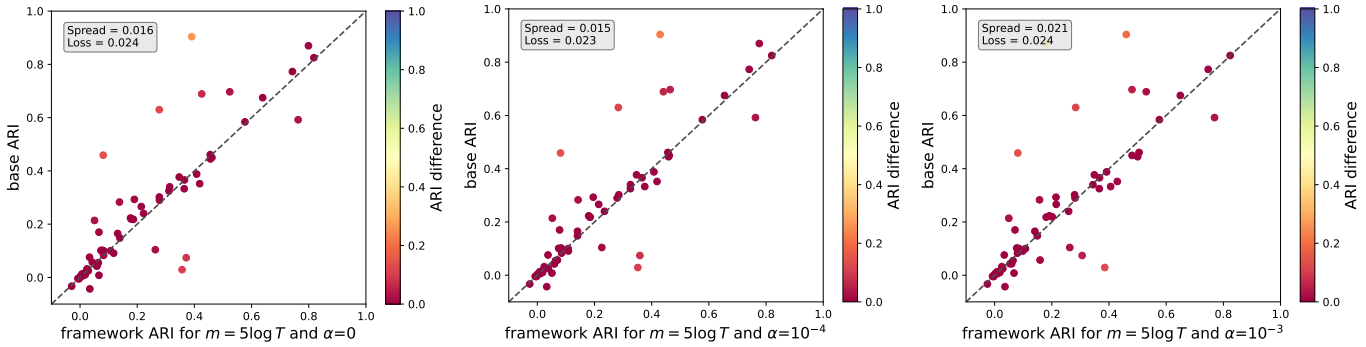
(**e**) Best ARI spread (loss = 0.011, spread = 0.023).

(**f**) Mean ARI spread (loss = 0.035, spread = 0.022).

Figure 4. Spread diagrams with comparative ARI results of Sparse Time-Series Clustering against standard k-means overall datasets for the $(\alpha T/m)$-DTW distance, with different values of $\alpha \in \{0, 10^{-4}, 10^{-3}, 10^{-2}\}$ and with $m = 2\log_2 T$ inducing points. The average spread and the average loss of the sparse framework are reported in the caption and on the upper left corner of each plot. We observe a slight deterioration in clustering quality as α increases and a small average loss in (**e**) and (**f**), where we keep the best ARI and the mean ARI in each dataset (both across all values of $\alpha \in \{0, 10^{-4}, 10^{-3}, 10^{-2}\}$).

5.5. Computational Efficiency

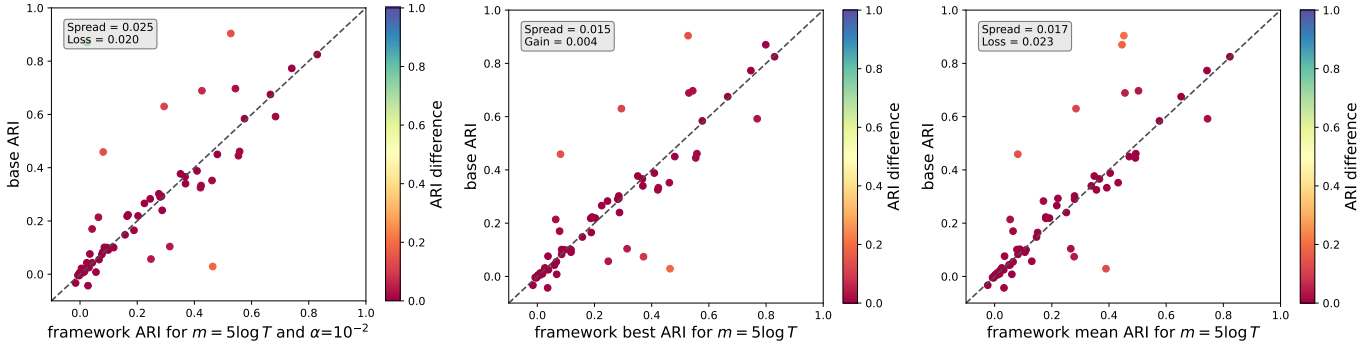
As noted in Section 4.3, an important contribution of our work is that the combined computational complexity of our Sparse Time-Series Clustering framework is $O(NT \log^2 T + INk \log^2 T)$, where I is the iterations of k-means, N is the number of time series in the dataset and T is their length, compared against a running time of $O(INkT^2)$ of the standard k-means algorithm applied to the original dataset. Next, we evaluate the running time and the CPU utilization of our framework in practice. .



(**a**) ARI spread with $\alpha = 0$ (loss = 0.024, spread = 0.016).

(**b**) ARI spread with $\alpha = 10^{-4}$ (loss = 0.023, spread = 0.015).

(**c**) ARI spread with $\alpha = 10^{-3}$ (loss = 0.024, spread = 0.021).



(**d**) ARI spread with $\alpha = 10^{-2}$ (loss = 0.025, spread = 0.020).

(**e**) Best ARI spread (gain = 0.004, spread = 0.015).

(**f**) Mean ARI spread (loss = 0.023, spread = 0.017).

Figure 5. Spread diagrams with comparative ARI results of Sparse Time-Series Clustering against standard k-means overall datasets for the $(\alpha T/m)$-DTW distance, with different values of $\alpha \in \{0, 10^{-4}, 10^{-3}, 10^{-2}\}$ and with $m = 5\log_2 T$ inducing points. The average spread and the average loss of the sparse framework are reported in the caption and on the upper left corner of each plot. In contrast to the case where $m = 2\log_2 T$, in Figure 4, with $m = 5\log_2 T$ inducing points, we observe a stable clustering quality, and even a slight improvement, as α increases. This behavior is attributed to the fact that larger m makes $(\alpha T/m)$-DTW less sensitive to an increase in α. We also observe a small average gain in (**e**), where we keep the best ARI, and a small average loss in (**f**), where we take the mean ARI in each dataset (both across all values of $\alpha \in \{0, 10^{-4}, 10^{-3}, 10^{-2}\}$).

Table 4 displays the CPU utilization times for our framework and standard k-means on all datasets. Specifically, we provide the following metrics: T_k is the CPU utilization time of the standard k-means algorithm applied to the original dataset; T_a is the CPU utilization time of the modeling phase, where Sparse Gaussian Process Regression is applied, in steps 4–6 of Algorithm 1; and T_b is the CPU utilization time of the k-means algorithm in step 7 of Algorithm 1, where Algorithm 3 is applied to the sparse time series produced by the Sparse Gaussian Process Regression. Moreover, $T_a + T_b$ is the total CPU utilization of our method, $(T_a + T_b)/T_k$ indicates the CPU utilization overhead due to the modeling process (which dominates the running time of our framework); T_k/T_b quantifies how much slower k-means become when applied to the original dataset compared against the same algorithm applied to the sparsified dataset; and T_a/N is the average running time of the modeling phase (Sparse Gaussian Process Regression) per time series in the particular dataset.

Table 4. CPU Utilization time of the standard k-means algorithm and the modeling and the clustering phase of Algorithm 1. T_k denotes the CPU utilization time of the standard k-means algorithm in the original dataset; T_a denotes the CPU utilization time of the modeling phase; and T_b denotes the CPU utilization time of the clustering phase. $T_a + T_b$ is the total CPU utilization of our method, $(T_a + T_b)/T_k$ indicates the CPU utilization overhead due to the modeling process; T_k/T_b quantifies how much slower k-means becomes when applied to the original dataset compared against the same algorithm applied to the sparsified dataset; and T_a/N is the average running time of the modeling phase per time series in the particular dataset.

Dataset	T_k	T_b	T_a	$\frac{T_k}{T_b}$	$\frac{(T_a+T_b)}{T_k}$	$\frac{T_a}{N}$
Adiac	809.606	117.040	5468.717	6.9	6.9	7.002
ArrowHead	449.036	5.795	1473.779	77.5	3.3	6.985
Beef	237.538	1.238	460.083	191.9	1.9	7.668
BeetleFly	91.529	0.609	289.232	150.3	3.2	7.231
BirdChicken	152.857	0.660	312.412	231.6	2.0	7.810
Car	350.987	2.881	968.850	121.8	2.8	8.074
CBF	127.093	45.560	5959.039	2.8	47.2	6.408
Coffee	100.512	0.798	383.234	126.0	3.8	6.843
Computers	1448.839	12.139	2334.388	119.4	1.6	4.669
CricketX	2457.813	62.977	5748.995	39.0	2.4	7.371
CricketY	1731.068	61.845	5792.806	28.0	3.4	7.427
CricketZ	1628.689	62.495	5524.100	26.1	3.4	7.082
DiatomSizeReduction	702.399	8.619	2606.539	81.5	3.7	8.095
DistalPhalanxOutlineCorrect	42.445	14.438	5711.787	2.9	134.9	6.520
DistalPhalanxOutlineAgeGroup	22.358	10.411	3348.379	2.1	150.2	6.212
DistalPhalanxTW	32.310	19.760	3462.503	1.6	107.8	6.424
Earthquakes	764.784	17.812	3897.637	42.9	5.1	8.455
ECG200	10.045	4.051	1430.853	2.5	142.8	7.154
ECGFiveDays	193.677	26.262	5910.736	7.4	30.7	6.686
FaceAll	681.338	267.684	13,986.357	2.5	20.9	6.216
FaceFour	174.810	2.035	583.206	85.9	3.3	5.207
FacesUCR	656.559	168.009	9845.755	3.9	15.3	4.376
FiftyWords	3719.422	110.433	4634.011	33.7	1.3	5.120
Fish	1401.001	14.399	3173.496	97.3	2.3	9.067
GunPoint	204.587	3.420	1606.681	59.8	7.9	8.033
Ham	498.690	5.834	2085.700	85.5	4.2	9.746
Herring	393.123	2.633	972.268	149.3	2.5	7.596
InsectWingbeatSound	8642.398	137.208	8987.362	63.0	1.1	4.085
ItalyPowerDemand	48.611	26.104	6500.469	1.9	134.3	5.931
LargeKitchenAppliances	3391.768	19.276	2969.682	176.0	0.9	3.960
Lightning2	280.387	3.525	1002.119	79.6	3.6	8.282
Lightning7	266.726	4.853	1057.223	55.0	4.0	7.393
Meat	140.371	2.340	1162.826	60.0	8.3	9.690
MedicalImages	193.023	82.434	7299.850	2.3	38.2	6.398
MiddlePhalanxOutlineAgeGroup	22.883	11.684	3700.447	2.0	162.2	6.680
MiddlePhalanxOutlineCorrect	40.408	15.421	5762.425	2.6	143.0	6.467
MiddlePhalanxTW	41.743	20.060	3480.027	2.1	83.8	6.293
MoteStrain	213.246	50.299	7216.290	4.2	34.1	5.673
OliveOil	63.692	0.953	534.073	66.8	8.4	8.901
OSULeaf	2005.386	21.865	3731.238	91.7	1.9	8.442
PhalangesOutlinesCorrect	151.922	47.915	14,683.742	3.2	97.0	5.524
Plane	12.636	4.129	1233.003	3.1	97.9	0.584
ProximalPhalanxOutlineCorrect	28.106	13.378	5230.995	2.1	186.6	24.910
ProximalPhalanxOutlineAgeGroup	21.364	12.143	3819.305	1.8	179.3	6.313
ProximalPhalanxTW	33.438	19.141	3490.010	1.7	104.9	3.917
RefrigerationDevices	2699.834	16.155	2965.134	167.1	1.1	4.901
ShapeletSim	291.462	5.064	1651.373	57.6	5.7	2.202
ShapesAll	9397.260	132.630	4991.169	70.9	0.5	24.956

Table 4. *Cont.*

Dataset	T_k	T_b	T_a	$\frac{T_k}{T_b}$	$\frac{(T_a+T_b)}{T_k}$	$\frac{T_a}{N}$
SmallKitchenAppliances	1683.222	18.953	3321.628	88.8	2.0	2.768
SonyAIBORobotSurface1	34.634	15.644	3655.760	2.2	106.0	4.874
SonyAIBORobotSurface2	71.301	25.905	6676.567	2.8	94.0	10.751
Strawberry	1183.540	23.964	7845.866	49.4	6.6	8.006
SwedishLeaf	224.676	67.156	5367.119	3.3	24.2	5.460
Symbols	4984.632	43.091	9755.276	115.7	2.0	8.671
SyntheticControl	26.540	16.777	3758.807	1.6	142.3	3.685
ToeSegmentation1	464.657	8.339	1931.603	55.7	4.2	3.219
ToeSegmentation2	263.977	4.073	1176.577	64.8	4.5	4.390
Trace	225.796	4.452	1505.905	50.7	6.7	9.072
TwoLeadECG	115.038	33.520	7091.992	3.4	61.9	35.460
TwoPatterns	767.798	313.861	28,274.694	2.4	37.2	24.333
Wine	51.230	1.493	831.680	34.3	16.3	0.166
WordSynonyms	3179.780	61.514	3889.850	51.7	1.2	35.044
Worms	1540.641	9.298	2799.553	165.7	1.8	3.093
mean	982.337	37.214	4401.955	53.8	40.0	8.095

We observe that k-means runs from 1.6 (in very small datasets) up to 230 times faster when applied to the sparsified dataset (compared against the k-means algorithm applied to the original instance, see also Figure 6). The speed-up is due to the improved running time for computing DTW from $\Theta(T^2)$ in the original data to $\Theta(\log^2 T)$ in the sparsified data. As expected, the speed-up becomes more apparent when it comes to datasets with time-series length T above a few hundred. On average, k-means runs more than 50 times faster when applied to the sparsified dataset than when applied to the original one.

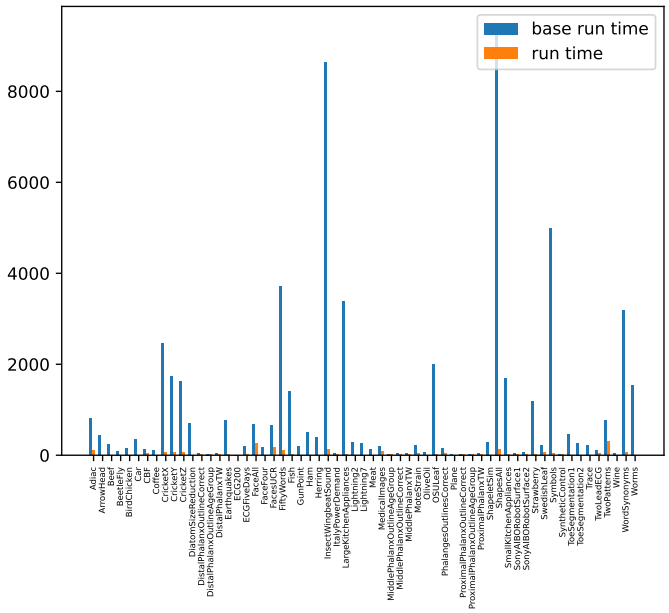


Figure 6. CPU utilization time comparison of the k-means algorithm applied to the original datasets (blue bars) and to the sparsified data sets (orange bars). k-means on the sparsified instance can run up to 230 times faster than k-means on the original instance.

On the other hand, the computational overhead of Sparse Gaussian Process Regression becomes quite high, compared against the running time of the standard k-means algorithm, when Algorithm 1 is applied to datasets of small to moderate size (and with time series of small to moderate length). However, for datasets with larger sizes and

longer time series, such as InsectWingbeatSound (with 2200 time series of length 256 each), RefrigerationDevices and LargeKitchenAppliances (both with 750 time series of length 720 each), the benefits of the significantly improved asymptotic computational complexity of our framework become apparent. In these datasets, the total $T_a + T_b$ running time of our framework is practically identical to the total running time of the standard k-means. For some larger datasets, such as ShapesAll (with 1200 time series of length 512 each), the total $T_a + T_b$ running time of our framework is half the total running time of the standard k-means. Moreover, there are datasets, such as ElectricDevices (with about 16,500 time series of length 96 each), where our framework is able to run successfully in our computational infrastructure and to produce clusterings with average ARI higher than that reported in [22], while it is impossible to successfully run the standard k-means algorithm due to the quadratic computational complexity of DTW and/or the large size of the dataset (see also Table 5 and Section 5.6).

Table 5. ARI results for the ElectricDevices dataset consisting of 16637 time series of length 96 each. The best ARI computed by the standard k-means in [22] is 0.19, while our framework achieves a best ARI of 0.21, which is 10% larger than that reported in [22].

Parameter a	$1 \cdot \log T$	$2 \cdot \log T$	$3 \cdot \log T$	$4 \cdot \log T$	$5 \cdot \log T$
0	0.18	**0.19**	0.18	0.17	0.17
$\mathbf{10^{-4}}$	**0.21**	0.18	0.18	**0.19**	**0.19**
$\mathbf{10^{-3}}$	0.14	**0.20**	0.18	0.18	0.18
$\mathbf{10^{-2}}$	0.04	0.12	**0.21**	0.18	**0.20**

In a nutshell, we observe that the total CPU utilization of our framework becomes comparable to that of applying k-means to the original data for datasets with NT^2 ranging from 10^8 to $3 \cdot 10^8$, while the benefits of our framework's significantly improved asymptotic computational complexity become apparent for datasets with NT^2 larger than $4 \cdot 10^8$. On the other hand, if we focus on k-means only, for datasets with T at most 100, applying k-means to sparsified datasets is about 10–20 times faster than its application to the original data. As T grows larger to 200–400, the speed-up factor of k-means increases to 60–80, and reaches values above 120–150 for a time-series length T above 600.

As an additional note regarding the high computational overhead due to Sparse Gaussian Process Regression, we should mention that (i) SGPR could run offline, independently of k-means (or any other shape-based time-series clustering algorithm) and only once per time series, with the resulting sparse time series stored for any future use; and (ii) that one could arrange for SGPR to run in parallel (and completely independently) for each different time series, which would result in a completion time about two orders of magnitude faster, without increasing the total CPU utilization.

5.6. Empirical Results: Time and Memory Considerations

As mentioned in Section 5.1, we excluded certain UCR univariate datasets from our experimental evaluation, because it was impossible to successfully run the standard k-means algorithm on the original dataset in our computational infrastructure)and [22] does not provide running time estimations for the UCR datasets). The application of k-means to those datasets was terminated either due to memory issues, because of the very large size $\Theta(NT)$ of the dataset, or due to running time exceeding two days without completing a single run of k-means.

Nevertheless, using our Sparse Time-Series Clustering framework, we managed to obtain results for those datasets successfully, demonstrating its usefulness. For example, Table 5 reports the ARI achieved by our framework for the ElectricDevices dataset, a very large dataset consisting of 16637 time series of length 96 each and $k = 7$ clusters (this is one of the datasets for which we could not run k-means with the original time series in our computational infrastructure). We note that the best ARI is obtained by k-means with the original data, and the standard DTW distance is 0.19, as reported in [22].

6. Discussion and Future Work

Our proposed framework, as we demonstrated in Section 5, has competitive results to the standard *k*-means algorithm for time-series clustering. In our comprehensive evaluation, we highlight the significant advantages of our framework in clustering quality, CPU utilization, and memory requirements, especially for larger datasets and for time series of moderate to large lengths.

The bottleneck of our method in terms of CPU utilization is the modeling phase. We underline that this step is completely independent for each time series; hence, it can be parallelized, reducing the total running time of our framework. Moreover, SGPR can be run once as an offline step, with its results stored for any future use. The modeling step allows us to significantly reduce the memory requirements for the clustering step since the time-series representation is logarithmic in the length of the original time series, making the clustering step feasible in huge datasets (with a large number of long time series) for the popular, but computationally demanding, DTW distance.

The extensive evaluation of our framework, including additional metrics, has been made accessible on GitHub, providing a comprehensive resource for researchers. This transparency ensures the reproducibility of results and facilitates further exploration and validation. The reported CPU utilization times comprise an important addition to the thus far assessment of time-series clustering methods.

Moving forward, there are several promising directions for future work. First, an in-depth exploration of the tuning process is warranted to establish a correlation between the nature of the dataset and the optimal parameter selection of the framework. This understanding could lead to refined configurations, enhancing the effectiveness of the proposed framework across diverse datasets and applications.

Additionally, the incorporation of different alignment distances in our framework presents an intriguing direction of research. For instance, Frechét distance [37] and Wasserstein distance [38], which can be used in time-series clustering, come with computational challenges. Therefore, our framework, with the reduction of the time-series length, may allow the efficient application of these distances in time-series clustering to longer univariate time-series.

Last but not least, the extension of our framework to handle multivariate time series is a natural direction for future work. The ability to effectively analyze and model complex, multi-dimensional time-series data expands the range of potential applications across diverse domains.

Author Contributions: Conceptualization, D.F.; Methodology, D.F., P.P., E.P. and M.X.; Software, E.P. and M.X.; Validation, E.P. and M.X.; Investigation, D.F., P.P., E.P. and M.X.; Data curation, M.X.; Writing—original draft, P.P., E.P. and M.X.; Writing—review & editing, D.F. and E.P.; Visualization, E.P.; Supervision, D.F.; Funding acquisition, D.F. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "First Call for H.F.R.I. Research Projects to support Faculty members and Researchers and the procurement of high-cost research equipment grant", project BALSAM, HFRI-FM17-1424. A significant part of this work was made while Michalis Xefteris was a student at the National Technical University of Athens.

Data Availability Statement: The source code, the datasets, and more results with additional parameter configurations and for clustering quality indicators other than ARI can be found in https://github.com/pseleni/ts_clustering (accessed on 25 January 2024).

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Tables with Detailed ARI and Results

Table A1. ARI of UCR datasets for Algorithm 3 (the baseline computed by standard k-means) and Algorithm 1 (Sparse Time-Series Clustering) with standard DTW (i.e., $\beta = 0$) and number of inducing points $m = \gamma \log_2 T$, for every $\gamma \in \{1, 2, 3, 4, 5\}$. The best ARI achieved is marked in bold. In parenthesis, we report the relative order of the corresponding ARI among the six ones reported, from 1 (best) to 6 (worst). In the last line, we report the average ARI (and the average relative order) for each column across all datasets.

Dataset	Baseline	$1 \cdot \log T$	$2 \cdot \log T$	$3 \cdot \log T$	$4 \cdot \log T$	$5 \cdot \log T$
Adiac	**0.266** (1)	0.229 (3)	0.242 (2)	0.201 (6)	0.227 (4)	0.214 (5)
ArrowHead	0.165 (2)	**0.195** (1)	0.139 (4)	0.163 (3)	0.111 (6)	0.132 (5)
Beef	0.100 (2)	**0.115** (1)	0.097 (3)	0.078 (6)	0.090 (4)	0.083 (5)
BeetleFly	0.091 (2)	0.036 (6)	0.051 (4)	0.048 (5)	0.079 (3)	**0.118** (1)
BirdChicken	0.010 (4)	**0.062** (1)	0.002 (6)	0.022 (2)	0.010 (5)	0.018 (3)
Car	0.102 (2)	**0.277** (1)	0.081 (4)	0.082 (3)	0.063 (6)	0.078 (5)
CBF	**0.689** (1)	0.511 (5)	0.650 (2)	0.641 (3)	0.584 (4)	0.426 (6)
Coffee	0.592 (2)	0.210 (6)	0.539 (3)	0.503 (4)	0.352 (5)	**0.763** (1)
Computers	0.043 (5)	0.023 (6)	0.052 (3)	0.052 (4)	**0.065** (1)	0.058 (2)
CricketX	**0.218** (1)	0.106 (6)	0.153 (5)	0.157 (4)	0.167 (3)	0.187 (2)
CricketY	**0.219** (1)	0.138 (6)	0.177 (4)	0.185 (2)	0.169 (5)	0.179 (3)
CricketZ	**0.223** (1)	0.109 (6)	0.156 (5)	0.167 (3)	0.162 (4)	0.177 (2)
DiatomSizeReduction	**0.904** (1)	0.355 (6)	0.652 (3)	0.729 (2)	0.610 (4)	0.391 (5)
DistalPhalanxOutlineCorrect	0.001 (2)	−0.003 (6)	**0.001** (1)	−0.001 (5)	−0.001 (4)	−0.001 (3)
DistalPhalanxOutlineAgeGroup	0.366 (3)	0.396 (2)	**0.402** (1)	0.363 (5)	0.360 (6)	0.365 (4)
DistalPhalanxTW	0.302 (4)	**0.423** (1)	0.330 (3)	0.343 (2)	0.300 (5)	0.277 (6)
Earthquakes	−0.043 (6)	0.027 (5)	0.033 (2)	0.027 (4)	0.030 (3)	**0.034** (1)
ECG200	0.100 (5)	0.117 (2)	**0.139** (1)	0.100 (4)	0.080 (6)	0.107 (3)
ECGFiveDays	0.008 (5)	**0.091** (1)	0.043 (4)	0.045 (3)	0.002 (6)	0.066 (2)
FaceAll	**0.461** (1)	0.072 (6)	0.209 (5)	0.367 (3)	0.358 (4)	0.455 (2)
FaceFour	0.352 (2)	0.001 (6)	0.339 (3)	0.277 (5)	0.316 (4)	**0.419** (1)
FacesUCR	0.445 (2)	0.070 (6)	0.212 (5)	0.353 (3)	0.342 (4)	**0.456** (1)
FiftyWords	**0.325** (1)	0.210 (6)	0.241 (5)	0.274 (4)	0.299 (3)	0.311 (2)
Fish	**0.283** (1)	0.204 (2)	0.172 (4)	0.179 (3)	0.169 (5)	0.138 (6)
GunPoint	−0.004 (3)	**0.026** (1)	−0.005 (6)	−0.003 (2)	−0.004 (4)	−0.005 (5)
Ham	0.032 (2)	−0.004 (6)	0.027 (3)	**0.045** (1)	0.023 (5)	0.024 (4)
Herring	**0.013** (1)	−0.007 (6)	−0.005 (5)	0.003 (4)	0.007 (3)	0.007 (2)
InsectWingbeatSound	0.057 (4)	0.079 (2)	**0.081** (1)	0.066 (3)	0.045 (5)	0.043 (6)
ItalyPowerDemand	**0.004** (1)	0.000 (6)	0.001 (3)	0.001 (4)	0.001 (5)	0.002 (2)
LargeKitchenAppliances	**0.170** (1)	0.065 (4)	0.080 (2)	0.065 (5)	0.051 (6)	0.066 (3)
Lightning2	0.025 (2)	0.008 (5)	0.003 (6)	0.008 (4)	0.017 (3)	**0.029** (1)
Lightning7	**0.293** (1)	0.197 (2)	0.162 (6)	0.169 (5)	0.190 (4)	0.190 (3)
Meat	**0.630** (1)	0.225 (4)	0.186 (5)	0.095 (6)	0.418 (2)	0.277 (3)
MedicalImages	**0.101** (1)	0.039 (6)	0.060 (5)	0.062 (4)	0.066 (3)	0.073 (2)
MiddlePhalanxOutlineAgeGroup	0.388 (4)	0.337 (5)	0.336 (6)	0.405 (2)	0.392 (3)	**0.408** (1)
MiddlePhalanxOutlineCorrect	−0.005 (6)	**0.025** (1)	0.000 (3)	0.000 (2)	−0.002 (4)	−0.003 (5)
MiddlePhalanxTW	0.290 (5)	**0.382** (1)	0.303 (4)	0.323 (3)	0.326 (2)	0.278 (6)
MoteStrain	0.029 (6)	0.386 (2)	**0.398** (1)	0.364 (4)	0.372 (3)	0.357 (5)
OliveOil	**0.459** (1)	0.238 (2)	0.090 (3)	0.017 (5)	−0.007 (6)	0.081 (4)
OSULeaf	**0.148** (1)	0.064 (6)	0.136 (3)	0.128 (5)	0.132 (4)	0.139 (2)
PhalangesOutlinesCorrect	0.006 (5)	**0.012** (1)	0.001 (6)	0.006 (4)	0.010 (3)	0.010 (2)
Plane	**0.825** (1)	0.665 (6)	0.734 (5)	0.770 (4)	0.815 (3)	0.818 (2)
ProximalPhalanxOutlineCorrect	0.055 (4)	0.051 (6)	**0.082** (1)	0.062 (3)	0.053 (5)	0.064 (2)
ProximalPhalanxOutlineAgeGroup	0.450 (4)	**0.486** (1)	0.416 (5)	0.385 (6)	0.465 (2)	0.461 (3)
ProximalPhalanxTW	0.377 (4)	0.380 (3)	**0.431** (1)	0.384 (2)	0.355 (5)	0.347 (6)
RefrigerationDevices	**0.076** (1)	0.002 (6)	0.008 (5)	0.026 (4)	0.040 (2)	0.033 (3)
ShapeletSim	**0.006** (1)	−0.003 (5)	−0.003 (6)	−0.001 (4)	0.002 (3)	0.004 (2)
ShapesAll	**0.340** (1)	0.247 (6)	0.294 (5)	0.315 (2)	0.313 (4)	0.314 (3)

Table A1. *Cont.*

Dataset	Baseline	$1 \cdot \log T$	$2 \cdot \log T$	$3 \cdot \log T$	$4 \cdot \log T$	$5 \cdot \log T$
SmallKitchenAppliances	**0.214** (1)	0.014 (6)	0.032 (5)	0.045 (4)	0.049 (3)	0.051 (2)
SonyAIBORobotSurface1	**0.697** (1)	0.535 (4)	0.548 (3)	0.599 (2)	0.487 (6)	0.524 (5)
SonyAIBORobotSurface2	0.104 (5)	−0.033 (6)	**0.355** (1)	0.295 (3)	0.300 (2)	0.263 (4)
Strawberry	−0.033 (6)	−0.029 (5)	**0.016** (1)	−0.011 (3)	−0.011 (2)	−0.029 (4)
SwedishLeaf	0.333 (4)	0.180 (6)	0.314 (5)	**0.367** (1)	0.356 (3)	0.364 (2)
Symbols	**0.675** (1)	0.642 (3)	0.622 (6)	0.658 (2)	0.639 (4)	0.639 (5)
SyntheticControl	**0.773** (1)	0.633 (6)	0.733 (4)	0.756 (2)	0.728 (5)	0.743 (3)
ToeSegmentation1	0.022 (4)	0.005 (6)	0.022 (3)	**0.027** (1)	0.027 (2)	0.021 (5)
ToeSegmentation2	0.043 (5)	0.063 (2)	**0.071** (1)	0.041 (6)	0.056 (4)	0.059 (3)
Trace	0.584 (2)	0.526 (6)	0.570 (5)	0.574 (4)	**0.585** (1)	0.577 (3)
TwoLeadECG	0.074 (5)	0.011 (6)	**0.817** (1)	0.138 (4)	0.519 (2)	0.371 (3)
TwoPatterns	**0.870** (1)	0.304 (6)	0.702 (5)	0.825 (2)	0.813 (3)	0.799 (4)
Wine	−0.004 (3)	−0.005 (5)	−0.005 (4)	−0.002 (2)	**−0.002** (1)	−0.007 (6)
WordSynonyms	**0.240** (1)	0.169 (6)	0.172 (5)	0.202 (4)	0.221 (3)	0.223 (2)
Worms	**0.083** (1)	0.031 (6)	0.074 (5)	0.079 (4)	0.080 (3)	0.082 (2)
	0.249 (2.524)	0.173 (4.270)	0.222 (3.683)	0.217 (3.492)	0.220 (3.762)	0.225 (3.270)

Table A2. Average ARI (and standard deviation) of UCR datasets for Algorithm 3 (the baseline computed by standard k-means) and Algorithm 1 (Sparse Time-Series Clustering) with number of inducing points $m = \gamma \log_2 T$, for $\gamma \in \{1, 2, 3, 4, 5\}$ and $(\alpha m/T)$-DTW. Averages and standard deviations are computed over ARI values of 10 different runs (for the baseline) and ARI values of 10 different runs for each different value of $\alpha\{0, 10^{-4}, 10^{-3}, 10^{-2}\}$ for the sparse framework. The best average ARI is reported in bold. In parenthesis, we report the relative order of the corresponding average ARI among the six ones reported, from 1 (best) to 6 (worst). In the last line, we report the average ARI (and the average relative order) for each column across all datasets.

Dataset	Baseline	$1 \cdot \log T$	$2 \cdot \log T$	$3 \cdot \log T$	$4 \cdot \log T$	$5 \cdot \log T$
Adiac	**0.266 ± 0.017** (1)	0.234 ± 0.005 (3)	0.240 ± 0.001 (2)	0.201 ± 0.001 (6)	0.227 ± 0.002 (4)	0.217 ± 0.005 (5)
ArrowHead	0.165 ± 0.058 (3)	**0.204 ± 0.011** (1)	0.149 ± 0.019 (5)	0.178 ± 0.026 (2)	0.135 ± 0.031 (6)	0.150 ± 0.022 (4)
Beef	0.100 ± 0.032 (2)	**0.114 ± 0.001** (1)	0.099 ± 0.004 (3)	0.086 ± 0.008 (6)	0.091 ± 0.006 (4)	0.086 ± 0.005 (5)
BeetleFly	0.091 ± 0.068 (2)	0.033 ± 0.005 (6)	0.049 ± 0.003 (5)	0.060 ± 0.014 (4)	0.072 ± 0.004 (3)	**0.106 ± 0.008** (1)
BirdChicken	0.010 ± 0.027 (3)	**0.058 ± 0.004** (1)	0.002 ± 0.000 (6)	0.010 ± 0.010 (4)	0.004 ± 0.006 (5)	0.018 ± 0.001 (2)
Car	0.102 ± 0.045 (2)	**0.286 ± 0.016** (1)	0.083 ± 0.008 (5)	0.092 ± 0.013 (3)	0.081 ± 0.024 (6)	0.088 ± 0.015 (4)
CBF	**0.689 ± 0.143** (1)	0.443 ± 0.107 (6)	0.583 ± 0.116 (2)	0.582 ± 0.105 (3)	0.543 ± 0.071 (4)	0.456 ± 0.043 (5)
Coffee	0.592 ± 0.243 (2)	0.217 ± 0.012 (6)	0.526 ± 0.024 (3)	0.498 ± 0.011 (4)	0.360 ± 0.008 (5)	**0.745 ± 0.035** (1)
Computers	0.043 ± 0.028 (3)	0.018 ± 0.005 (6)	0.041 ± 0.018 (4)	0.041 ± 0.018 (5)	**0.055 ± 0.022** (1)	0.050 ± 0.015 (2)
CricketX	**0.218 ± 0.030** (1)	0.108 ± 0.003 (6)	0.151 ± 0.003 (5)	0.152 ± 0.005 (4)	0.170 ± 0.005 (3)	0.179 ± 0.009 (2)
CricketY	**0.219 ± 0.028** (1)	0.136 ± 0.005 (6)	0.184 ± 0.007 (5)	0.199 ± 0.014 (2)	0.185 ± 0.014 (4)	0.192 ± 0.011 (3)
CricketZ	**0.223 ± 0.017** (1)	0.111 ± 0.004 (6)	0.154 ± 0.004 (5)	0.163 ± 0.007 (4)	0.164 ± 0.002 (3)	0.179 ± 0.008 (2)
DiatomSizeReduction	**0.904 ± 0.114** (1)	0.355 ± 0.001 (6)	0.647 ± 0.009 (3)	0.730 ± 0.009 (2)	0.619 ± 0.042 (4)	0.452 ± 0.050 (5)
DistalPhalanxOutlineCorrect	0.001 ± 0.001 (2)	−0.003 ± 0.001 (6)	**0.001 ± 0.000** (1)	−0.001 ± 0.000 (5)	−0.001 ± 0.000 (4)	−0.001 ± 0.000 (3)
DistalPhalanxOutlineAgeGroup	0.366 ± 0.138 (4)	**0.416 ± 0.013** (1)	0.401 ± 0.001 (2)	0.363 ± 0.001 (5)	0.360 ± 0.002 (6)	0.367 ± 0.001 (3)
DistalPhalanxTW	0.302 ± 0.030 (4)	**0.405 ± 0.025** (1)	0.328 ± 0.006 (3)	0.337 ± 0.007 (2)	0.296 ± 0.005 (5)	0.279 ± 0.003 (6)
Earthquakes	−0.043 ± 0.005 (6)	0.009 ± 0.012 (5)	0.027 ± 0.012 (3)	0.026 ± 0.010 (4)	**0.035 ± 0.008** (1)	0.033 ± 0.003 (2)
ECG200	0.100 ± 0.073 (5)	0.132 ± 0.020 (2)	**0.141 ± 0.005** (1)	0.106 ± 0.011 (4)	0.092 ± 0.011 (6)	0.110 ± 0.005 (3)
ECGFiveDays	0.008 ± 0.010 (5)	**0.084 ± 0.011** (1)	0.026 ± 0.019 (4)	0.074 ± 0.020 (2)	0.002 ± 0.000 (6)	0.060 ± 0.007 (3)
FaceAll	0.461 ± 0.040 (2)	0.099 ± 0.016 (6)	0.228 ± 0.020 (5)	0.387 ± 0.020 (3)	0.385 ± 0.028 (4)	**0.494 ± 0.042** (1)
FaceFour	0.352 ± 0.121 (2)	0.013 ± 0.019 (6)	0.332 ± 0.009 (4)	0.272 ± 0.009 (5)	0.335 ± 0.024 (3)	**0.432 ± 0.018** (1)
FacesUCR	0.445 ± 0.055 (2)	0.097 ± 0.015 (6)	0.228 ± 0.015 (5)	0.375 ± 0.023 (3)	0.368 ± 0.031 (4)	**0.492 ± 0.040** (1)
FiftyWords	0.325 ± 0.047 (4)	0.245 ± 0.025 (6)	0.291 ± 0.042 (5)	0.326 ± 0.045 (3)	0.346 ± 0.042 (2)	**0.356 ± 0.043** (1)
Fish	**0.283 ± 0.034** (1)	0.214 ± 0.009 (2)	0.182 ± 0.015 (5)	0.200 ± 0.030 (3)	0.189 ± 0.021 (4)	0.171 ± 0.044 (6)
GunPoint	−0.004 ± 0.003 (3)	**0.014 ± 0.011** (1)	−0.005 ± 0.000 (6)	−0.004 ± 0.001 (2)	−0.004 ± 0.001 (4)	−0.005 ± 0.000 (5)
Ham	0.032 ± 0.025 (2)	−0.004 ± 0.000 (6)	0.021 ± 0.006 (5)	**0.045 ± 0.001** (1)	0.028 ± 0.008 (3)	0.025 ± 0.001 (4)
Herring	**0.013 ± 0.015** (1)	−0.007 ± 0.001 (6)	−0.005 ± 0.001 (5)	0.004 ± 0.001 (4)	0.010 ± 0.004 (2)	0.008 ± 0.001 (3)
InsectWingbeatSound	0.057 ± 0.008 (6)	0.128 ± 0.043 (5)	**0.143 ± 0.055** (1)	0.139 ± 0.070 (2)	0.133 ± 0.077 (3)	0.130 ± 0.081 (4)
ItalyPowerDemand	0.004 ± 0.002 (2)	**0.008 ± 0.015** (1)	0.002 ± 0.001 (3)	0.001 ± 0.001 (5)	0.001 ± 0.000 (6)	0.002 ± 0.000 (4)
LargeKitchenAppliances	**0.170 ± 0.078** (1)	0.061 ± 0.004 (4)	0.068 ± 0.011 (2)	0.060 ± 0.009 (5)	0.047 ± 0.003 (6)	0.064 ± 0.013 (3)
Lightning2	0.025 ± 0.016 (2)	0.021 ± 0.027 (4)	0.021 ± 0.035 (3)	0.020 ± 0.019 (5)	0.012 ± 0.003 (6)	**0.032 ± 0.004** (1)
Lightning7	**0.293 ± 0.044** (1)	0.232 ± 0.038 (2)	0.197 ± 0.038 (6)	0.206 ± 0.039 (5)	0.209 ± 0.023 (4)	0.221 ± 0.038 (3)
Meat	**0.630 ± 0.183** (1)	0.222 ± 0.007 (4)	0.182 ± 0.005 (5)	0.098 ± 0.003 (6)	0.415 ± 0.007 (2)	0.285 ± 0.006 (3)
MedicalImages	**0.101 ± 0.018** (1)	0.031 ± 0.015 (6)	0.056 ± 0.006 (5)	0.065 ± 0.004 (4)	0.067 ± 0.001 (3)	0.078 ± 0.005 (2)
MiddlePhalanxOutlineAgeGroup	0.388 ± 0.077 (4)	0.364 ± 0.027 (5)	0.331 ± 0.010 (6)	0.393 ± 0.013 (2)	0.392 ± 0.001 (3)	**0.404 ± 0.007** (1)
MiddlePhalanxOutlineCorrect	−0.005 ± 0.000 (6)	**0.011 ± 0.008** (1)	−0.000 ± 0.000 (3)	−0.000 ± 0.001 (2)	−0.003 ± 0.000 (4)	−0.003 ± 0.000 (5)
MiddlePhalanxTW	0.290 ± 0.104 (5)	**0.396 ± 0.011** (1)	0.294 ± 0.010 (4)	0.326 ± 0.003 (2)	0.324 ± 0.005 (3)	0.280 ± 0.001 (6)

Table A2. *Cont.*

Dataset	Baseline	1 · log T	2 · log T	3 · log T	4 · log T	5 · log T
MoteStrain	0.029 ± 0.005 (6)	0.385 ± 0.009 (4)	**0.402 ± 0.012** (1)	0.383 ± 0.017 (5)	0.399 ± 0.025 (2)	0.389 ± 0.045 (3)
OliveOil	**0.459 ± 0.145** (1)	0.238 ± 0.000 (2)	0.087 ± 0.004 (3)	0.017 ± 0.000 (5)	−0.008 ± 0.001 (6)	0.081 ± 0.000 (4)
OSULeaf	**0.148 ± 0.023** (1)	0.067 ± 0.005 (6)	0.139 ± 0.003 (3)	0.126 ± 0.002 (5)	0.136 ± 0.004 (4)	0.147 ± 0.007 (2)
PhalangesOutlinesCorrect	0.006 ± 0.001 (3)	0.004 ± 0.005 (5)	0.004 ± 0.003 (6)	0.006 ± 0.000 (4)	0.010 ± 0.000 (2)	**0.010 ± 0.000** (1)
Plane	**0.825 ± 0.147** (1)	0.671 ± 0.008 (6)	0.733 ± 0.003 (5)	0.767 ± 0.006 (4)	0.808 ± 0.008 (3)	0.823 ± 0.004 (2)
ProximalPhalanxOutlineCorrect	0.055 ± 0.003 (4)	0.052 ± 0.000 (6)	**0.085 ± 0.005** (1)	0.063 ± 0.002 (3)	0.053 ± 0.000 (5)	0.065 ± 0.001 (2)
ProximalPhalanxOutlineAgeGroup	0.450 ± 0.108 (4)	**0.477 ± 0.010** (1)	0.422 ± 0.010 (5)	0.380 ± 0.009 (6)	0.463 ± 0.004 (3)	0.471 ± 0.010 (2)
ProximalPhalanxTW	0.377 ± 0.119 (3)	0.373 ± 0.010 (4)	**0.425 ± 0.008** (1)	0.379 ± 0.009 (2)	0.358 ± 0.003 (5)	0.349 ± 0.002 (6)
RefrigerationDevices	**0.076 ± 0.032** (1)	0.003 ± 0.002 (6)	0.008 ± 0.000 (5)	0.026 ± 0.001 (4)	0.039 ± 0.002 (2)	0.035 ± 0.002 (3)
ShapeletSim	**0.006 ± 0.011** (1)	−0.002 ± 0.001 (5)	−0.003 ± 0.000 (6)	−0.002 ± 0.001 (4)	−0.000 ± 0.002 (3)	0.002 ± 0.002 (2)
ShapesAll	**0.340 ± 0.027** (1)	0.265 ± 0.013 (6)	0.316 ± 0.018 (5)	0.334 ± 0.019 (4)	0.337 ± 0.021 (3)	0.338 ± 0.021 (2)
SmallKitchenAppliances	**0.214 ± 0.024** (1)	0.021 ± 0.007 (6)	0.028 ± 0.006 (5)	0.043 ± 0.002 (4)	0.052 ± 0.003 (3)	0.054 ± 0.006 (2)
SonyAIBORobotSurface1	**0.697 ± 0.051** (1)	0.423 ± 0.178 (6)	0.560 ± 0.016 (3)	0.599 ± 0.010 (2)	0.498 ± 0.016 (5)	0.503 ± 0.032 (4)
SonyAIBORobotSurface2	0.104 ± 0.081 (5)	−0.007 ± 0.043 (6)	**0.325 ± 0.029** (1)	0.321 ± 0.027 (2)	0.302 ± 0.011 (3)	0.266 ± 0.031 (4)
Strawberry	−0.033 ± 0.003 (6)	−0.017 ± 0.016 (4)	**0.016 ± 0.001** (1)	−0.012 ± 0.001 (3)	−0.009 ± 0.002 (2)	−0.024 ± 0.005 (5)
SwedishLeaf	0.333 ± 0.041 (4)	0.171 ± 0.012 (6)	0.321 ± 0.007 (5)	0.375 ± 0.008 (3)	0.376 ± 0.022 (2)	**0.392 ± 0.023** (1)
Symbols	**0.675 ± 0.103** (1)	0.641 ± 0.015 (5)	0.633 ± 0.009 (6)	0.667 ± 0.015 (2)	0.646 ± 0.026 (4)	0.652 ± 0.010 (3)
SyntheticControl	**0.773 ± 0.126** (1)	0.594 ± 0.074 (6)	0.713 ± 0.019 (5)	0.751 ± 0.013 (2)	0.730 ± 0.004 (4)	0.743 ± 0.002 (3)
ToeSegmentation1	0.022 ± 0.018 (2)	0.004 ± 0.002 (6)	0.017 ± 0.008 (5)	**0.025 ± 0.016** (1)	0.021 ± 0.012 (3)	0.018 ± 0.008 (4)
ToeSegmentation2	0.043 ± 0.049 (5)	0.047 ± 0.018 (4)	**0.058 ± 0.017** (1)	0.042 ± 0.011 (6)	0.050 ± 0.016 (3)	0.054 ± 0.007 (2)
Trace	**0.584 ± 0.116** (1)	0.489 ± 0.053 (6)	0.540 ± 0.052 (5)	0.561 ± 0.014 (4)	0.584 ± 0.001 (2)	0.576 ± 0.001 (3)
TwoLeadECG	0.074 ± 0.014 (5)	0.009 ± 0.005 (6)	**0.585 ± 0.341** (1)	0.098 ± 0.055 (4)	0.342 ± 0.196 (2)	0.278 ± 0.119 (3)
TwoPatterns	**0.870 ± 0.032** (1)	0.131 ± 0.119 (6)	0.348 ± 0.296 (5)	0.447 ± 0.352 (3)	0.448 ± 0.347 (2)	0.445 ± 0.347 (4)
Wine	−0.004 ± 0.004 (3)	−0.005 ± 0.000 (5)	−0.005 ± 0.000 (4)	−0.003 ± 0.001 (2)	**−0.002 ± 0.000** (1)	−0.007 ± 0.000 (6)
WordSynonyms	0.240 ± 0.016 (3)	0.190 ± 0.016 (6)	0.206 ± 0.028 (5)	0.236 ± 0.030 (4)	0.247 ± 0.028 (2)	**0.251 ± 0.025** (1)
Worms	**0.083 ± 0.020** (1)	0.028 ± 0.003 (6)	0.070 ± 0.006 (5)	0.081 ± 0.002 (3)	0.080 ± 0.004 (4)	0.082 ± 0.002 (2)
	0.249 (2.556)	0.171 (4.413)	0.213 (3.857)	0.215 (3.556)	0.217 (3.587)	0.226 (3.032)

References

1. Fu, T.C. A Review on Time-Series Data Mining. *Eng. Appl. Artif. Intell.* **2011**, *24*, 164–181.
2. Aghabozorgi, S.; Shirkhorshidi, A.S.; Wah, T.Y. Time-series clustering—A decade review. *Inf. Syst.* **2015**, *53*, 16–38. [CrossRef]
3. Hung, J.L.; Wang, M.C.; Wang, S.; Abdelrasoul, M.; Li, Y.; He, W. Identifying at-risk students for early interventions—A time-series clustering approach. *IEEE Trans. Emerg. Top. Comput.* **2015**, *5*, 45–55. [CrossRef]
4. Bandara, K.; Bergmeir, C.; Smyl, S. Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach. *Expert Syst. Appl.* **2020**, *140*, 112896. [CrossRef]
5. Kotsakos, D.; Trajcevski, G.; Gunopulos, D.; Aggarwal, C.C. Time-Series Data Clustering. In *Data Clustering: Algorithms and Applications*; Data Mining and Knowledge Discovery Series; CRC Press: Boca Raton, FL, USA; Taylor and Francis Group: Abingdon, UK, 2014; Volume 15, pp. 357–380.
6. Warren Liao, T. Clustering of time series data—A survey. *Pattern Recognit.* **2005**, *38*, 1857–1874. [CrossRef]
7. Gunopulos, D.; Das, G. Time series similarity measures and time series indexing. In Proceedings of the SIGMOD Conference, Santa Barbara, CA, USA, 21–24 May 2001; p. 624.
8. Kate, R.J. Using dynamic time warping distances as features for improved time-series classification. *Data Min. Knowl. Discov.* **2016**, *30*, 283–312. [CrossRef]
9. Rakthanmanon, T.; Campana, B.; Mueen, A.; Batista, G.; Westover, B.; Zhu, Q.; Zakaria, J.; Keogh, E. Searching and mining trillions of time series subsequences under dynamic time warping. In Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), Beijing, China, 12–16 August 2012; pp. 262–270.
10. Tan, C.W.; Webb, G.I.; Petitjean, F. Indexing and classifying gigabytes of time series under time warping. In Proceedings of the SIAM International Conference on Data Mining, Houston, TX, USA, 27–29 April 2017; pp. 1–10.
11. Andoni, A.; Nosatzki, N.S. Edit Distance in Near-Linear Time: It's a Constant Factor. In Proceedings of the 61st IEEE Annual Symposium on Foundations of Computer Science (FOCS 2020), Durham, NC, USA, 16–19 November 2020; pp. 990–1001.
12. Keogh, E.; Chakrabarti, K.; Pazzani, M.; Mehrotra, S. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. In Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, Santa Barbara, CA, USA, 21–24 May 2001; SIGMOD '01, pp. 151–162. [CrossRef]
13. Iorio, C.; Frasso, G.; D'Ambrosio, A.; Siciliano, R. Parsimonious time series clustering using P-splines. *Expert Syst. Appl.* **2016**, *52*, 26–38. [CrossRef]
14. Rasmussen, C.E.; Williams, C.K.I. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*; The MIT Press: Cambridge, MA, USA, 2005.
15. Leibfried, F.; Dutordoir, V.; John, S.; Durrande, N. A tutorial on sparse Gaussian processes and variational inference. *arXiv* **2020**, arXiv:2012.13962.
16. Titsias, M. Variational Learning of Inducing Variables in Sparse Gaussian Processes. In Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics, Clearwater Beach, FL, USA, 16–18 April 2009; van Dyk, D., Welling, M., Eds.; Proceedings of Machine Learning Research; Hilton Clearwater Beach Resort: Clearwater Beach, FL, USA, 2009; Volume 5, pp. 567–574.

17. Quiñonero-Candela, J.; Ramussen, C.; Williams, C. Approximation methods for Gaussian process regression. In *Large-Scale Kernel Machines*; MIT Press: Cambridge, MA, USA, 2007; pp. 203–223.
18. Micchelli, C.A.; Xu, Y.; Zhang, H. Universal Kernels. *J. Mach. Learn. Res.* **2006**, *7*, 2651–2667.
19. Petitjean, F.; Ketterlin, A.; Gançarski, P. A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognit.* **2011**, *44*, 678–693. [CrossRef]
20. Dau, H.A.; Keogh, E.; Kamgar, K.; Yeh, C.C.M.; Zhu, Y.; Gharghabi, S.; Ratanamahatana, C.A.; Yanping; Hu, B.; Begum, N.; et al. The UCR Time Series Classification Archive. 2018. Available online: https://www.cs.ucr.edu/~eamonn/time_series_data_2018/ (accessed on 25 January 2024).
21. Faloutsos, C.; Ranganathan, M.; Manolopoulos, Y. Fast subsequence matching in time-series databases. *ACM Sigmod Rec.* **1994**, *23*, 419–429. [CrossRef]
22. Javed, A.; Lee, B.S.; Rizzo, D.M. A benchmark study on time series clustering. *Mach. Learn. Appl.* **2020**, *1*, 100001. [CrossRef]
23. Paparrizos, J.; Gravano, L. Fast and accurate time-series clustering. *ACM Trans. Database Syst. (TODS)* **2017**, *42*, 1–49. [CrossRef]
24. Rand, W.M. Objective criteria for the evaluation of clustering methods. *J. Am. Stat. Assoc.* **1971**, *66*, 846–850. [CrossRef]
25. Morey, L.C.; Agresti, A. The measurement of classification agreement: An adjustment to the Rand statistic for chance agreement. *Educ. Psychol. Meas.* **1984**, *44*, 33–37. [CrossRef]
26. Hubert, L.; Arabie, P. Comparing Partitions. *J. Classif.* **1985**, *2*, 193–218. [CrossRef]
27. Vinh, N.X.; Epps, J.; Bailey, J. Information theoretic measures for clusterings comparison: Is a correction for chance necessary? In Proceedings of the 26th Annual International Conference on Machine Learning, Montreal, QC, Canada, 14–18 June 2009; pp. 1073–1080.
28. Chatterjee, S.; Simonoff, J.S. *Handbook of Regression Analysis*; John Wiley & Sons: Hoboken, NJ, USA, 2013.
29. Wang, J. An intuitive tutorial to Gaussian processes regression. *Comput. Sci. Eng.* **2023**, 1–8. [CrossRef]
30. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
31. Liu, D.C.; Nocedal, J. On the limited memory BFGS method for large scale optimization. *Math. Program. B* **1989**, *45*, 503–528. [CrossRef]
32. Snelson, E.; Ghahramani, Z. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems 18*; MIT Press: Cambridge, MA, USA, 2006; pp. 1257–1264.
33. Csató, L.; Opper, M. Sparse online Gaussian processes. *Neural Comput.* **2002**, *14*, 641–668. [CrossRef]
34. McIntire, M.; Ratner, D.; Ermon, S. Sparse Gaussian Processes for Bayesian Optimization. In Proceedings of the UAI, New York, NY, USA, 25–29 June 2016.
35. Gardner, J.R.; Pleiss, G.; Bindel, D.; Weinberger, K.Q.; Wilson, A.G. GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 3–8 December 2018.
36. Tavenard, R.; Faouzi, J.; Vandewiele, G.; Divo, F.; Androz, G.; Holtz, C.; Payne, M.; Yurchak, R.; Rußwurm, M.; Kolar, K.; et al. Tslearn, A Machine Learning Toolkit for Time Series Data. *J. Mach. Learn. Res.* **2020**, *21*, 1–6.
37. Driemel, A.; Krivošija, A.; Sohler, C. Clustering time-series under the Fréchet distance. In Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Arlington, VA, USA, 10–12 January 2016; pp. 766–785.
38. Muskulus, M.; Verduyn-Lunel, S. Wasserstein distances in the analysis of time-series and dynamical systems. *Phys. D Nonlinear Phenom.* **2011**, *240*, 45–58. [CrossRef]