



# Time series clustering with random convolutional kernels

Marco-Blanco Jorge<sup>1</sup> · Cuevas Rubén<sup>1,2</sup>

Received: 14 July 2023 / Accepted: 11 March 2024  
© The Author(s) 2024

## Abstract

Time series data, spanning applications ranging from climatology to finance to healthcare, presents significant challenges in data mining due to its size and complexity. One open issue lies in time series clustering, which is crucial for processing large volumes of unlabeled time series data and unlocking valuable insights. Traditional and modern analysis methods, however, often struggle with these complexities. To address these limitations, we introduce R-Clustering, a novel method that utilizes convolutional architectures with randomly selected parameters. Through extensive evaluations, R-Clustering demonstrates superior performance over existing methods in terms of clustering accuracy, computational efficiency and scalability. Empirical results obtained using the UCR archive demonstrate the effectiveness of our approach across diverse time series datasets. The findings highlight the significance of R-Clustering in various domains and applications, contributing to the advancement of time series data mining.

**Keywords** Time series · Clustering · Machine learning · Feature extraction

## 1 Introduction

Time is a natural concept that allows us to arrange events in a sorted way from the past to the future in an evenly distributed manner (such as days, seasons of the year, or years). Data points are usually taken at successive equally spaced points in time. Based on this way of describing events, the concept of time series as a sequence of discrete data points evenly distributed arises naturally. From this perspective,

---

Responsible editor: Eamonn Keogh.

---

✉ Marco-Blanco Jorge  
jorge.marco@uc3m.es  
Cuevas Rubén  
rcuevas@it.uc3m.es

<sup>1</sup> Universidad Carlos III de Madrid, Getafe 28903, Madrid, Spain

<sup>2</sup> UC3M-Santander Big Data Institute, Getafe 28903, Madrid, Spain

time series can describe a wide range of natural and social phenomena that evolve in time. A few examples are climate and weather trends, seismic measures, stock prices, sales data, biomedical measurements, body movements, or Internet traffic (Aghabozorgi et al. 2015). The study of time series involves identifying its statistical properties, including trends, seasonality, and autocorrelation, and using this information to build models for prediction and classification.

According to Yang and Wu (2006), who reviewed the work of the most active researchers in data mining and machine learning for their opinions, the problem of sequential and time series data is one of the ten most challenging in data mining. Its most relevant applications are prediction and classification (supervised or unsupervised). As for time series data availability, the significant advances in data storage, collection, and processing produced during the last decades have made available a large amount of data which is growing exponentially. Unfortunately, most of this data remains unlabelled, making it useless for some critical applications such as supervised learning.

With the ever-increasing growth rate of available data, we need scalable techniques to process it. Clustering serves as a solution for extracting valuable insights from unlabelled data, allowing a vast quantity of this data to be processed efficiently. Clustering can be defined as the task of grouping a set of objects into different subsets, or clusters, where the objects in each subset share certain common characteristics. Since it does not require human supervision or hand-labeling, clustering is classed as an unsupervised learning technique. This makes clustering particularly suited to keeping pace with the influx of available data.

Although time series are usually considered a collection of data points sorted chronologically, they can also be regarded as a single object (Kumar and Nagabhushan 2006) and, therefore, subject to clustering. In scenarios involving significant volumes of time series data, clustering is especially helpful in discovering patterns. In the case of rare patterns, it facilitates anomaly and novelty detection, while for frequent patterns, it aids in prediction and recommendation tasks (Aghabozorgi et al. 2015). Examples of such applications include detecting web traffic anomalies in Informatics (Lakhina et al. 2005) and gene classification in Biology (McDowell et al. 2018). Time series clustering can also serve as a pre-processing technique for other algorithms such as rule discovery, indexing, or classification (Aghabozorgi et al. 2015).

Despite the crucial role of time series clustering across diverse fields, existing approaches often struggle with the complexity of high-dimensional, noisy real-world time series data. Furthermore, while automated feature extraction methods have shown success, they require more parameters, data, and longer training periods, as discussed in Section 2. There is, therefore, a need for more efficient, accurate, and scalable methods for time series clustering.

In this paper, we make several contributions to the field of time series clustering. In Section 3, we introduce R-Clustering, a novel time series clustering algorithm that uses convolutional architectures with static, randomly selected kernel parameters. This approach addresses the challenge of scalability and resource-intensive training, prevalent in current methods. Subsequently, in Section 4 we provide a comprehensive evaluation of R-Clustering, benchmarking its performance against

state-of-the-art methods through the use of the UCR archive (Dau et al. 2019; Keogh and Foliás 2002) and detailed statistical analyses. We contrast R-Clustering against eight other reference clustering algorithms across diverse time series datasets. Remarkably, R-Clustering outperforms the other algorithms in 33 datasets, with the second-best performing algorithm leading in only 13. It achieves the highest mean rank and is also the fastest algorithm across the datasets evaluated. We further demonstrate its scalability for larger datasets in terms of the number of time series and time series length. As an auxiliary contribution, we also present the results of a method aimed to enhance clustering algorithms that rely on Euclidean distance. The results indicate improved clustering quality for the tested algorithms, yet R-Clustering still prevails despite such advancements. These findings highlight the superior performance and scalability of R-Clustering, showing its potential for use in large-scale applications.

## 2 Related work

In this section, we will first review the most relevant methods for time series clustering. Then, our focus will shift to feature extraction methods applied to 2D and 1D data: this discussion will cover the general applications of these methods and make a distinction between those methods that use learnable kernels and those that use static kernels. To conclude, we will explore the potential application of static kernel methods to time series clustering.

### 2.1 Different approaches to time series clustering

Different approaches to the clustering of time series, according to Aghabozorgi et al. (2015), can be classified into three groups: model-based, shape-based, and feature-based. In the model-based methods, the time series are adjusted to a parametric model; since the parameters do not have a time series structure, a universal clustering algorithm is applied. Some popular parametric models are autoregressive-moving-average (ARMA) models or the Hidden Markov Model. However, these techniques might not accurately model high-dimensional and noisy real-world time series (Långkvist et al. 2014). Existing methods that may address the noise issue, such as wavelet analysis or filtering, often introduce lag in the processed data, reducing the prediction accuracy or the ability to learn significant features from the data (Yang and Wu 2006).

In the shape-based approach, the clustering algorithm is applied directly to the time series data with no previous transformation. Unlike the model-based approach, these methods employ a clustering method equipped with a similarity measure appropriate for time series. For instance, (Paparrizos and Gravano 2015) introduce the k-shape algorithm with a shape-based approach. In this work, the authors suggest that feature-based strategies may be less optimal due to their domain dependence, requiring the modification of the algorithms for different datasets. Others, like the Unsupervised Salient Subsequence Learning model (USSL), utilize pseudo-labels

and spectral analysis to discover shapelets (salient subsequences) without requiring labeled data (Zhang et al. 2018). A different approach in shape-based clustering involves utilizing a universal clustering algorithm, such as the widely used K-means algorithm (MacQueen 1967), along with a suitable distance measure that considers the unique characteristics of time series data, such as dynamic time warping (DTW), introduced by Berndt and Clifford (1994). DTW is known for its effectiveness in time series clustering. Its computational demand, however, can vary significantly based on the chosen implementation and warping window, making it generally more intensive than the Euclidean distance. The latter, often employed in the K-means algorithm (Likas et al. 2003), has a complexity of  $\mathcal{O}(n)$  (Jain et al. 1999), indicating a more straightforward computational requirement.

Feature-based methods first extract relevant time series features and later apply a conventional clustering algorithm (Aghabozorgi et al. 2015). As we will see in the next section, feature-based algorithms have been proven quite successful for image clustering and classification. According to (Bengio et al., 2013, p. 1798), the performance of machine learning algorithms relies strongly on the correct choice of feature representation.

## 2.2 Feature extraction

The incorporation of expert knowledge has been shown to improve machine learning performance, as demonstrated by previous research such as Bengio et al. (2013) and Sebastiani (2002). Traditionally, this has been achieved through the manual design of feature extractors based on the specific task. Examples of such strategy include Lowe's algorithm (Lowe 1999) for image feature extraction using hand-designed static filters to induce specific invariances in the transformed data. Also, it is well known that the discrete Gaussian kernel can be used to perform a wide range of filtering operations, such as low-pass filtering, high-pass filtering, and band-pass filtering (Kailath 1980). In the context of digital signal processing, FIR (Finite Impulse Response) filters operate by applying a weight set, or "coefficients", to a selection of input data to calculate an output point, essentially a discrete convolution operation (Proakis and Manolakis 1996). It is important to remark, however, that this process requires manual calibration of the filter to selectively allow certain frequencies through while suppressing others. These techniques, while effective, present scalability limitations due to their dependency on human supervision for the design process. As data volumes continue to grow exponentially, there is an increasing need for automated feature extraction techniques that can efficiently handle large datasets without the extensive requirement for expert knowledge.

In some domains, such as the 2D shape of pixels in images or time series having a 1D structure, it is possible to learn better features using only a basic understanding of the input as an alternative to incorporating expert knowledge. In images and time series data, patterns are likely to reproduce at different positions, and neighbours are likely to have strong dependencies. The existence of such local correlations is the basis for applying a local feature extractor all over the input and transforming it into a feature map of a similar structure. A convolution operation usually performs

this task with a convolutional kernel (Bengio et al., 2013, p. 1820). The use of local feature extractors has several benefits in machine learning. First, it allows for the efficient extraction of relevant features from large datasets without requiring expert knowledge. Second, it can improve the accuracy of machine learning models by capturing local correlations in the input data. Finally, it can reduce the dimensionality of the input data, making it easier to process and analyze. The academic community has achieved substantial advances during the last decade applying this technique to the problems of image classification (Krizhevsky et al. 2012), time series forecasting (Greff et al. 2016) or time series classification (Ismail Fawaz et al. 2019) among others. In the field of time series clustering, (Ma et al. 2019) have developed a convolutional model with deep autoencoders.

From another perspective, spectral clustering (Shi and Malik 2000; Von Luxburg 2007) offers significant advances in unsupervised feature selection. Li et al. (2012) perform spectral clustering and feature selection at the same time while imposing a negative constraint in the objective function boosting performance. Shi et al. (2014) addresses the challenge of selecting features that are not only relevant but also robust to variations and noise within the data. Additionally, (Li and Tang 2015) introduces measures to explicitly manage feature redundancy. Within the field of time series clustering, (Zhang et al. 2018) leverage spectral analysis and pseudo-labels to discover discriminative shapelets, improving clustering accuracy.

### 2.2.1 Feature extraction with learnable kernels

In most convolution algorithms, kernel weights are typically learnt during the training process, as described in Bengio et al. (2013). Over the last few years, the academic community has been actively exploring the applications of such convolutional architectures for tasks such as image classification and segmentation. This is illustrated by works like Ciresan et al. (2011); Pereira et al. (2016), and Krizhevsky et al. (2012). These models use convolutional layers in neural networks to transform image data, refining and compressing the original input into a compact, high-level feature representation. The architecture then applies a linear classification algorithm on this condensed feature representation as the final step.

Some authors have applied convolutional architectures to the problem of image clustering, as described in Caronet et al. (2018) and Xie et al. (2016). In both models, a network transforms input data into an enhanced feature representation, followed by a clustering algorithm. This dual optimization approach adjusts both the network weights and the clustering parameters simultaneously. The first model predicts input labels based on the clustering results, and then the error in these predictions is backpropagated through the network to adjust the network parameters. The second model utilizes an autoencoder with a middle bottleneck. The network parameters are first optimized using the reconstruction loss. Then, the decoder section of the architecture is discarded and the output features from the bottleneck form the new data representation. This condensed data representation is then processed by a clustering algorithm.

Convolutional neural networks (CNNs) have also become increasingly popular in the field of time series analysis due to their ability to capture local patterns and

dependencies within the data. For example, (Wang et al. 2017) proposed a model that uses a combination of 1D and 2D CNNs to extract both temporal and spatial features from multivariate time series data. Another model, proposed by Zhao et al. (2017), uses a dilated convolutional neural network to capture long-term dependencies in time series data. More recently, Ismail Fawaz et al. (2019) used deep convolutional neural networks for time series classification achieving state-of-the-art results. Despite the success of CNN-based models in time series analysis, there are still challenges that need to be addressed. One challenge is the selection of appropriate hyperparameters, such as the number of filters and the filter size, which can greatly affect the performance of the model.

A key challenge in feature extraction lies in identifying features that not only provide strong individual discriminatory power but also minimize redundancy. The Catch22 feature set Lubba et al. (2019) addresses this challenge through a three-step process. It begins with a statistical prefiltration to identify features that individually possess discriminatory power across. Next, the remaining features are evaluated for their classification performance across a diverse range of datasets. Finally, redundancy is minimized. With a different focus, (Ma et al. 2019) recently proposed a novel clustering algorithm for time series data that builds upon previous image clustering techniques. The model leverages an autoencoder to obtain a reconstruction loss, which measures the difference between the original time series and its reconstructed version. Simultaneously, the model employs a clustering layer to predict the labels of the time series, which is used to compute a prediction loss. By combining both losses, the model jointly optimizes the parameters of the network and the clustering parameters. This approach allows for more accurate clustering of time series data, as it considers both the reconstruction error and the predicted labels.

Deep learning faces considerable challenges due to the complexity of its models, which typically involve multiple layers and a significant number of parameters (Goodfellow et al. 2016; Bengio et al. 2013). This complexity results in practical issues, including the requirement for substantial computational resources and vast amounts of training data. There is also the potential for overfitting and the necessity for fine-tuning to optimize the model's performance.

When deep learning is applied to time series analysis, further complications arise. Time series data exhibit unique characteristics like temporal dependencies and seasonal patterns, requiring specialized treatment as indicated by Wang et al. (2017) and Ismail Fawaz et al. (2019). In addition to the challenges associated with deep learning models, time series data also pose unique challenges due to their often high-dimensional and highly variable nature (Långkvist et al. 2014). This can make it difficult to select appropriate hyperparameters and optimize the model's performance.

### 2.2.2 Feature extraction with static random kernels

Convolutional models with static parameters offer a distinct approach to feature extraction. Instead of learning the weights of the convolutional filters during the training process, these models use fixed or static parameters, resulting in faster computation times and simpler architectures (Bengio et al. 2013). The works of Huang (2014) and Jarrett et al. (2009) support this approach, demonstrating that convolutional models with

weights that are randomly selected, or random kernels, can successfully extract relevant features from image data. Additionally, Saxe et al. (2011) argue that choosing the right network design can sometimes be more important than whether the weights of the network are learned or random. They showed that convolutional models with random kernels are frequency selective and translation invariant, which are highly desirable properties when dealing with time series data.

Dempster et al. (2020) provide evidence that convolutional architectures with random kernels are effective for time series analysis. The authors of the paper demonstrate that their proposed method, called ROCKET (Random Convolutional Kernel Transform), achieves state-of-the-art accuracy on several time series classification tasks while requiring significantly less computation than existing methods. The authors further improved the efficiency of their method by introducing MiniRocket (Dempster et al. 2021), an algorithm that runs up to 75 times faster than ROCKET on larger datasets, while maintaining comparable accuracy

Motivated by the success of random kernels applied to the problem of feature extraction of time series and image data, we propose a simple and fast architecture for time series clustering using random static kernels. To the best of our knowledge, this is the first attempt to use convolutional architectures with random weights for time series clustering. This approach eliminates the need for an input-reconstructing decoder or a classifier for parameter adjustments, commonly found in previous time series clustering works. Instead, the method applies the convolution operation with random filters over the time series input data to obtain an enhanced feature representation, which is then fed into a K-means clustering algorithm. By eliminating the need for a reconstruction loss or classification loss, our method is more efficient and easier to implement than existing methods for time series clustering. We have conducted several experiments to test the effectiveness and scalability of the algorithm, and our results show that it outperforms current state-of-the-art methods. As such, our research provides a substantial contribution to the field of time series clustering, offering a promising new avenue for further advancements.

### 3 Methodology

This section explains the proposed clustering algorithm and the evaluation methods, including a statistical analysis. We use the UCR archive (Dau et al. 2019) as our data source in this study, a vital tool for time series researchers, with over one thousand papers employing it. Our study is based on the 2018 version, which consists of 128 univariate time series datasets of different types: devices, ECGs, motion tracking, sensors, simulated data, and spectrographs. From these, we exclude 11 datasets with time series of variable lengths.

#### 3.1 Clustering algorithm

We introduce *R-Clustering*, a new algorithm for time series clustering, which is composed of three stages of data processing elements connected in series. The first

element is a feature extractor, comprised of a single layer of static convolutional kernels with random values. This extractor transforms the input time series into an enhanced data representation. The second processing element employs Principal Component Analysis (PCA) (Pearson 1901; Jolliffe 2002) for dimensionality reduction, selecting a combination of features that account for most of the variance in the input data. The last element is a K-means algorithm that utilizes Euclidean distance, thereby providing the algorithm with its clustering capabilities. Next, we describe each of these elements in detail.

### 3.1.1 Feature extraction

For the feature extractor, we propose a modified version of the one used in Mini-rocket (Dempster et al., 2021, p. 251), which is based on a previous algorithm called ROCKET. In particular, we employ randomly selected values for the bias and adjust the configuration of the hyperparameters to better suit the distinct challenge of clustering. We have conducted an optimization process of the hyperparameters (kernel length and number of kernels in this case). To avoid overfitting the UCR archive, we have chosen a development set of 41 datasets, the same group of datasets used in the original algorithm (ROCKET).

Our feature extractor is composed of 500 kernels, each of length 9, optimized for clustering quality as the result of a hyperparameter search detailed in Section 1. The kernel weights are restricted to either 2 or -1. As Dempster et al. (2021) demonstrated, constraining the weight values does not significantly compromise performance, but it substantially enhances efficiency.

Dilations represent another significant parameter in the model. Dilation in 1D kernels refers to the expansion of the receptive field of a convolutional kernel in one dimension. This expansion is achieved by inserting gaps between the kernel elements, which increases the distance between the elements and allows the kernel to capture information from a larger area. This technique expands the receptive field of the kernel, allowing them to identify patterns at various scales and frequencies. Our model follows the configuration proposed by Dempster et al. (2021), which employs varying dilations adapted to the specific time series being processed, to ensure recognition of most potential periodicities and scales in the data. The number of dilations is a fixed function of the input length and padding.

Another relevant configuration parameter is the selection of the bias values, which is performed as follows: firstly, each kernel is convolved with a time series selected at random from the dataset. Then, the algorithm draws the bias values randomly from the quantiles of the result of the convolution. This process ensures that the scale of the bias values aligns with that of the output.

After the convolution of the input series with these kernels under the mentioned configuration, the proportion of positive values (PPV) of the resulting series is calculated. The transformed data consists of 500 features (the same as the number of kernels) with values between 0 and 1.

A thorough examination of the original feature extractor stage reveals there are artificial autocorrelations, not from the time series data, produced by the implementation of the algorithm. This behaviour could affect the performance of the



clustering stage of R-Clustering, or a future algorithm using this feature extractor because it would likely detect these unnatural patterns. Also, these patterns could mask legitimate features of time series, obtaining misleading results. We have identified the origin of this issue in the selection of the bias values. For a time series  $X$  convolved with a kernel  $W$  PPV is computed as the proportion of positive values of  $(W * X - \text{bias})$  where  $*$  denotes convolution. To determine the bias values, our original algorithm selects a training instance  $X$  and calculates its convolution with the specified dilation  $d$  and kernel:  $W_d$ . The quantiles of the resulting convolution output are used as the bias values. In the implementation of the original algorithm, we identified how the sorting of bias values produced artificial autocorrelations. To rectify this, we have randomly permuted them, effectively removing the artificial autocorrelations. (see Section 4.1 for a detailed description of the output of the feature extractor after and before the modification).

### 3.1.2 Dimensionality reduction with principal component analysis

The “curse of dimensionality” can potentially impact the performance of the K-means clustering in the third stage of our algorithm, particularly given our high-dimensional context with 500 features, as detailed in the previous subsection. In high-dimensional space, the distance between data points becomes less meaningful, and the clustering algorithm may struggle to identify meaningful clusters (Beyer et al. 1999). This is because the distance between any two points tends to become more uniform, leading to the loss of meaningful distance metrics. As the number of dimensions increases, the volume of the space increases exponentially, and the data becomes more sparse, making it difficult to identify meaningful clusters (Aggarwal et al. 2001). For these reasons, it is convenient to reduce the number of features to improve the performance of K-means clustering in the context of a high-dimensional space. In our particular case, due to the random nature of the kernel weights used in the convolutions with the input data, we expect that many components of the transformed data may not be significant. Hence, implementing a dimensionality reduction method can be beneficial in multiple ways.

We propose using Principal Component Analysis (PCA), a dimensionality reduction technique that can identify the crucial dimensions or combinations of dimensions that account for the majority of data variability. Given that our problem is unsupervised, PCA is especially suitable: it focuses on the inherent statistical patterns within the data, independently of any evaluation algorithm. As to why not using PCA directly to the time series data and skipping the convolution transformation, PCA does not take into account the sequential ordering of data, therefore it may not fully capture the underlying structure of the time series.

One challenge associated with implementing PCA is determining the optimal number of principal components to retain, which will define the final number of features. Common techniques for this include the elbow method and the Automatic Choice of Dimensionality for PCA (Minka 2000). The elbow method involves visualizing the explained variance as a function of the number of components, with the elbow in the plot suggesting the optimal number. However, because this technique relies on visual interpretation, it may not be suitable for our automated algorithm.

The second method, based on Bayesian PCA, employs a probabilistic model to estimate the optimal number of dimensions. This method, while powerful, might not always be applicable, particularly in the context of high-dimensional data like time series, where it may be challenging to satisfy the assumption of having more samples than features.

We therefore opt to determine the number of components by analyzing the explained variance of the data introduced by each additional dimension until the increase in variance becomes insignificant. We consider increments of 1% to be insignificant. As demonstrated by experiments from Beyer et al. (1999), the number of selected dimensions typically lies between 10 and 20. Beyond this range, the effect of the curse of dimensionality can lead to the instability of algorithms based on Euclidean distance.

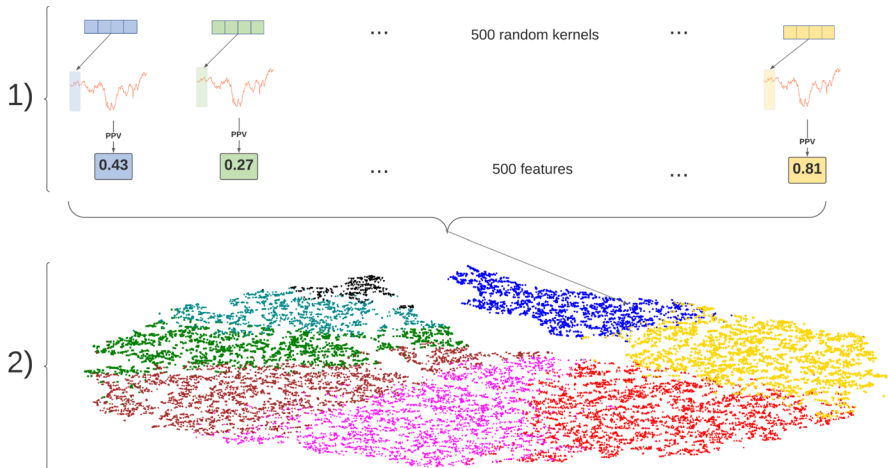
In summary, we incorporate an additional stage into our algorithm that employs Principal Component Analysis (PCA) to reduce the dimensionality of the features prior to the implementation of the K-means algorithm.

### 3.1.3 K-means with Euclidean distance

The findings in Section 4.1, demonstrating the absence of artificial autocorrelations in the output of the first stage, along with the dimensionality reduction via Principal Component Analysis (PCA) of the second stage, suggest that our algorithm's transformation considerably reduces the time series properties of the features following the first two stages. This reduction simplifies the problem, making it more amenable to traditional raw data algorithms which are typically less complex and less demanding in terms of computational resources compared to algorithms designed specifically for time series data, such as those using Dynamic Time Warping (DTW) or shape-based distances. Consequently, in the third stage of R-Clustering, we adopt a well-established clustering technique: the K-means algorithm with Euclidean distance. This combination is widely recognized and has been extensively tested within the scientific community for clustering problems (Jain et al. 1999; Likas et al. 2003). K-means partitions data into a number  $K$  (set in advance) of clusters by iteratively assigning each data point to the *nearest* mean center (centroid) of a cluster. After each new assignment, the centroids are recalculated. When the training process finishes, the resulting centroids can be used to classify new observations. To evaluate the *nearest* centroid, a distance metric must be defined. Using the Euclidean distance will result in a more efficient algorithm since it has a time complexity of  $\mathcal{O}(n)$ . In contrast, using DTW as a distance metric would result in a time complexity  $\mathcal{O}(n^2)$ . Figure 1 provides a schema of R-Clustering algorithm and its stages.

## 3.2 Evaluation method

To the authors' knowledge, the benchmark presented (Javed et al. 2020) is the only comparison of time series clustering methods using the widely adopted UCR archive. Their benchmark evaluates eight popular methods across partitioned, density-based, and hierarchical clustering, utilizing Euclidean, Dynamic Time Warping,



**Fig. 1** The figure illustrates the various steps involved in R-Clustering algorithm: 1)Initially, the input time series is convolved with 500 random kernels. Following this, the Positive Predictive Value (PPV) operation is applied to each of the convolution results, generating 500 features with values spanning between 0 and 1. 2) The next phase involves applying Principal Component Analysis (PCA) for dimensionality reduction. This procedure results in a more manageable set of features, reducing the original 500 to between 10 and 20 (For illustrative purposes, we employ t-SNE on this PCA-reduced data to create a 2D visualization). Finally, the processed and dimensionality-reduced data are clustered using the K-means algorithm

and shape-based distance measures. To evaluate the performance of R-Clustering, we adopt the dataset and methodology defined in this benchmark. Their study employs 112 datasets without pre-processing, ensuring uniform-length time series and a known number of clusters (excluding those with less than two classes). The use of established clustering methods and dataset-level assessment with the ARI metric provides a robust framework to isolate the impact of each method. We adhere to these evaluation procedures, ensuring a fair and direct comparison to the benchmark study. The number of clusters for each dataset is known in advance since the UCR archive provides labels, and this number is used as an input for the clustering algorithms in the benchmark and R-Clustering algorithm. In case the number of clusters was not known, different methods exist to estimate them, such as the elbow method (Thorndike 1953), but evaluating these methods is not part of the benchmark's paper or this paper.

In addition to these comparisons, we also employed the technique of optimizing the warping window width  $w$  in Dynamic Time Warping (DTW) for clustering time series data, as proposed by Dau et al. Dau et al. (2018). This method uses pseudo user annotations to automatically determine the optimal  $w$  value. The algorithm starts by randomly sampling objects from the dataset and generating warped copies of these instances. These warped instances are reintroduced into the dataset with a 'must-link' constraint, mimicking the scenario where they should naturally fall into the same cluster as their original counterparts. This pseudo-annotated dataset serves as the basis for identifying the optimal  $w$  value in DTW-based clustering algorithms. Empirical validation in the originating study showed the effectiveness of

this approach in enhancing clustering performance. Our experiments confirm these findings, thereby substantiating the value of this technique for automating the selection of  $w$  in time series clustering tasks. Building on this, we establish an alternative benchmark featuring an improved baseline. For this, we select top-performing algorithms and apply the  $w$ -optimization technique to those amenable to. Consistent with the original study, we experimented with warping windows ranging from 1 (Euclidean) to 20.

Several metrics are available for the evaluation of a clustering process, such as Rand Index (RI) (Rand 1971), Adjusted Mutual Information (Vinh et al. 2009) or Adjusted Rand Index (ARI) (Hubert and Arabie 1985). Among these, ARI is particularly advantageous because its output is independent of the number of clusters while not adjusted metrics consistently output higher values for a higher number of clusters (Javed et al. 2020). It is essentially an enhancement of the RI, adjusted to account for randomness. Additionally, Steinley (2004) explicitly recommends ARI as a superior metric for evaluating clustering performance. Based on these reasons and for comparability with the benchmark, we use the Adjusted Rand Index (ARI) to evaluate R-Clustering. This choice also ensures compatibility with the benchmark study, facilitating meaningful comparisons of our results. As the problem of finding the optimal partition of  $n$  data points into  $k$  clusters is an NP-hard problem with a non-convex loss function, we run the algorithm multiple times with different randomly initialized centroids to avoid local minima and enhance performance. Specifically, in line with the procedure adopted in the benchmark study, the algorithm is executed ten times, each time with differently randomly initialized centroids. The initialization is randomly chosen for every algorithm evaluated and for each of the ten runs.

In comparing the performance of various algorithms, we adhere to the methods used in the benchmark study. Specifically, we calculate the following: the number of instances where an algorithm achieves the highest Adjusted Rand Index (ARI) score, denoted as the ‘number of wins’; and the mean rank of all algorithms. Results in Subsections 4.3 and 4.4 indicate that R-Clustering outperforms the other algorithms, even those improved with the  $w$ -optimization technique, across all these measures, demonstrating its efficacy. We also conducted statistical tests to determine the significance of the results and considered any limitations or assumptions of the methods.

The problem of comparing multiple algorithms over multiple datasets in the context of machine learning has been treated by several authors (Demšar 2006; Garcia and Herrera 2008; Benavoli et al. 2016). Following their recommendations, we first compare the ranks of the algorithms as suggested by Demšar (2006) and use the Friedman test to decide whether there are significant differences among them. If the test rejects the null hypothesis (“there are no differences”), we try to establish which algorithms are responsible for these differences. As Garcia and Herrera (2008) indicate, upon a rejection of the null hypothesis by the Friedman test, we should proceed with another test to find out which algorithms produce the differences, using pairwise comparisons. Following the recommendations of the authors of the UCR archive, (Dau et al. 2019; Keogh and Folias 2002, we choose the Wilcoxon sign-test for the pairwise comparisons between R-Clustering algorithm and

the rest. As outlined in Subsection 4.3, we initially conduct comparisons between all the algorithms in the original benchmark and R-Clustering, employing the latter as a control classifier to identify any significant differences. The results indicate that R-Clustering's superior performance compared to the other algorithms is statistically significant.

Additionally, to enhance the insights provided by the benchmark study and following the suggestions from Garcia and Herrera (2008), we carry out a new experiment that involves pairwise comparisons among all possible combinations within the set comprising of the benchmark algorithms and R-Clustering.

### 3.3 Implementation and reproducibility

We use the Python 3.6 software package on a Windows OS with 16GB RAM, and Intel(R) Core(TM) i7-2600 CPU 3.40GHz processor for implementing the algorithms. In our study, we use a variety of reputable libraries that have been widely used and tested, therefore ensuring reliability. These include:

- *aeon*, a Python library for time series tasks such as forecasting, classification, and clustering. It is used for extracting data from the UCR archive.
- *sktime* (Löning et al. 2019), a standard Python library used for evaluating the Agglomerative algorithm.
- The code from Paparrizos and Gravano (2015), which we utilize to evaluate the computation time of the K-shape algorithm.
- *scikit-learn* (Pedregosa et al. 2011), employed to execute the K-means stage of R-Clustering and evaluate the adjusted rand index.
- Functions provided in Ismail Fawaz et al. (2019), used for calculating statistical results.

We make our code publicly available<sup>1</sup> and base our results on a public dataset. This provides transparency and guarantees full reproducibility and replicability of the paper following the best-recommended practices in the academic community.

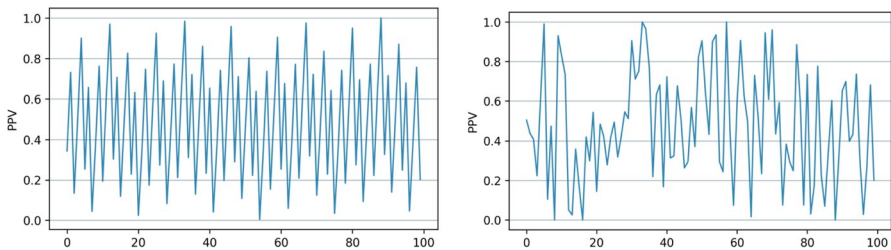
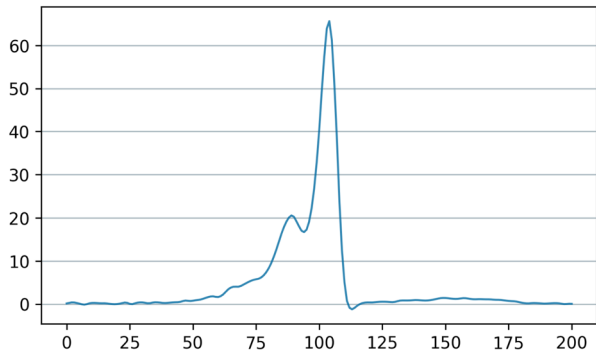
## 4 Results

This section begins with an investigation of the outputs generated during the feature extraction stage, as explicated in Section 3.1.1. We continue with a search for optimal hyperparameters by examining various algorithm configurations. Following this, we present the results of R-Clustering algorithm evaluated against the algorithms in the benchmark study (Javed et al. 2020), including a scalability study and a full statistical comparison, first using R-Clustering as a control classifier and subsequently comparing each algorithm against the others. We conclude the section by

---

<sup>1</sup> <https://github.com/jorgemarcoes/R-Clustering>

**Fig. 2** Sample time series from the Fungi dataset included in the UCR archive. The dataset contains high-resolution melt curves of the rDNA internal transcribed spacer (ITS) region of 51 strains of fungal species. The figure shows the negative first derivative ( $-dF/dt$ ) of the normalized melt curve of the ITS region of one of such species



**Fig. 3** Transformed sample time series from Fungi dataset (Fig. 2) with the original feature extractor from Minirocket (left) and the modified feature extractor (right)

comparing R-Clustering against an enhanced baseline of top-performing algorithms, enhanced with the “optimized w” method when applicable.

#### 4.1 Analysis of the Feature Extraction Stage Output

Figure 2 shows a sample time series from the UCR archive, while Fig. 3 (left) illustrates its transformation through the original feature extractor, highlighting evident autocorrelations. To analyze these autocorrelation properties, we employ the Ljung-Box test, whose null hypothesis is that no autocorrelations exist at a specified lag. As expected from observing Fig. 3 (left), the test rejects the null hypothesis for every lag at 95% confidence level, therefore we assume the presence of autocorrelations. To find out whether these periodic properties originate in the time series data, we introduce noisy data in the feature extractor, repeat the Ljung-Box test, and still observe autocorrelations at every lag. This observation leads us to conclude that the original feature extractor introduces artificial autocorrelations, which are not inherent to the input time series. After modifying the algorithm as explained in Section 3.1.1 and introducing the same noise, we rerun the Ljung-Box test. In contrast to the previous findings, the test does not reject the null hypothesis at any lag. Thus, we assume the absence of autocorrelations, indicating that the modified feature extractor works as intended, that is producing

noisy data from noisy inputs. For a sample transformation of a time series with the updated feature extractor (see Fig. 3, right).

## 4.2 Search for optimal hyperparameters

We explore several configurations of the algorithm and the effect on clustering quality. We focus on the main hyperparameters, specifically the kernel length and the number of kernels. For comparison, we set the number of kernels to range from 100 to 20,000 and kernel lengths to range from 7 to 13 and test various combinations across these entire ranges. We evaluated 20 combinations of hyperparameters on the development set. Among these, the combination of 500 kernels of length 9 yielded the highest number of wins (6) and the best mean rank (8.12), as shown in table 1. Based on these results, we selected the configuration of 500 kernels of length 9. Not only did this configuration obtain top positions in two of three categories, but its use of 500 kernels also made it faster than competitors utilizing 1000 or more kernels.

## 4.3 Analysis of R-clustering in the context of an existing benchmark

In this subsection, we evaluate the performance of R-Clustering on validation datasets against algorithms from the existing clustering benchmark (Javed et al. 2020), employing the same procedures used in the benchmark. We conduct a statistical analysis in line with the recommendations from the creators of the UCR archive and conclude with the results of the scalability study.

### 4.3.1 Performance of R-clustering algorithm

We compare the performance of R-Clustering to the other algorithms of the benchmarks over the validation datasets under several perspectives using the ARI metric. We count the number of wins considering that ties do not sum and calculate the mean rank. R-Clustering obtains the highest number of wins (33) followed by Agglomerative (13) (see Table 2) and the best mean rank (3.47) followed by K-means-DTW (4.47) (see Table 3)

In addition, we perform a statistical comparison of R-Clustering with the rest of the algorithms, in line with the recommendations provided by Dau et al. (2019) and Demšar (2006). These recommendations suggest comparing the rank of the classifiers on each dataset. Initially, we perform the Friedman test (Friedman 1937) at a 95% confidence level, which rejects the null hypothesis of no significant difference among all the algorithms. According to Benavoli et al. (2016), upon rejecting the null hypothesis, it becomes necessary to identify the significant differences among the algorithms. To accomplish this, we conduct a pairwise comparison of R-Clustering versus the other classifiers using the Wilcoxon signed-rank test at a 95% confidence level. This test includes the

**Table 1** Results for different hyperparameters configurations across the development set

| Algorithm | Mean Rank  | Winning Count |
|-----------|------------|---------------|
| 500-9     | <b>8.5</b> | <b>6</b>      |
| 100-13    | 13.3       | 4             |
| 100-9     | 11.7       | 4             |
| 5000-9    | 9.6        | 3             |
| 100-7     | 14.7       | 2             |
| 10000-9   | 9.1        | 2             |
| 100-11    | 13.0       | 2             |
| 1000-11   | 8.5        | 2             |
| 10000-13  | 11.3       | 1             |
| 500-13    | 11.2       | 1             |
| 500-7     | 10.8       | 1             |
| 1000-9    | 9.3        | 1             |
| 1000-13   | 10.4       | 1             |
| 10000-7   | 9.4        | 1             |
| 1000-7    | 9.9        | 1             |
| 5000-13   | 11.2       | 0             |
| 5000-7    | 10.0       | 0             |
| 500-11    | 8.9        | 0             |
| 10000-11  | 9.8        | 0             |
| 5000-11   | 9.4        | 0             |

The first term represents the number of kernels and the second is the kernels' length. For instance, 500-9' stands for a configuration with 500 kernels of length 9

**Table 2** Number of wins for each algorithm over the validation datasets in terms of best ARI

| Algorithm                 | Winning count |
|---------------------------|---------------|
| R-Clustering              | 33            |
| Agglomerative (Euclidean) | 13            |
| K-Shape                   | 10            |
| Density Peaks (DTW)       | 5             |
| K-means (DTW)             | 4             |
| K-means (Euclidean)       | 3             |
| C-means (Euclidean)       | 2             |
| K-medoids (Euclidean)     | 1             |
| Density Peaks (Euclidean) | 1             |

Holm correction for the confidence level, which adjusts for family-wise error (the chance of observing at least one false positive in multiple comparisons). Table 4 presents the p-values from the Wilcoxon signed-rank test comparing R-Clustering with each of the other algorithms, together with the adjusted alpha



**Table 3** Mean rank for R-Clustering and the rest of the algorithms considered in the benchmark

| Algorithm                 | Mean rank |
|---------------------------|-----------|
| R-Clustering              | 3.47      |
| Agglomerative (Euclidean) | 4.47      |
| K-means (Euclidean)       | 4.49      |
| K-means (DTW)             | 4.60      |
| K-Shape                   | 4.84      |
| C-means (Euclidean)       | 5.20      |
| K-medoids (Euclidean)     | 5.51      |
| Density Peaks (Euclidean) | 5.89      |
| Density Peaks (DTW)       | 6.53      |

values. The statistical rank analysis can be summarized as follows: R-Clustering emerges as the best-performing algorithm with an average rank of 3.47 as presented in Table 3. The pairwise comparisons using R-Clustering as a control

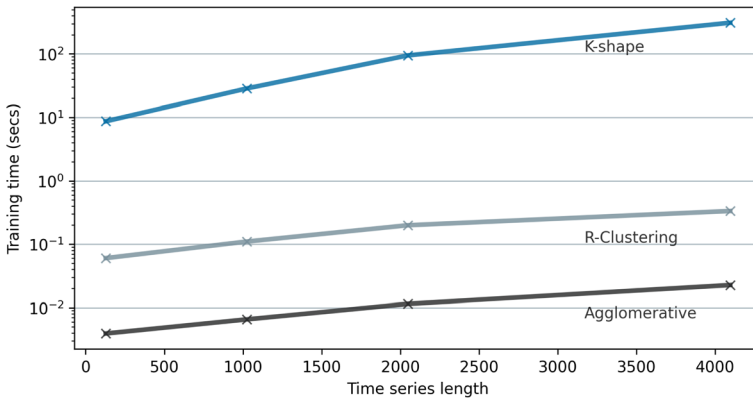
**Table 4** Results of the Wilcoxon signed-rank test between R-Clustering and the rest of the algorithms in consideration

| Algorithm                 | p-value  | alpha w/<br>Holm correction |
|---------------------------|----------|-----------------------------|
| Density Peaks (DTW)       | 0.000001 | 0.006                       |
| Density Peaks (Euclidean) | 0.000010 | 0.007                       |
| K-Shape                   | 0.000012 | 0.008                       |
| K-medoids (Euclidean)     | 0.000048 | 0.010                       |
| C-means (Euclidean)       | 0.000633 | 0.012                       |
| K-means (Euclidean)       | 0.001235 | 0.017                       |
| K-means (DTW)             | 0.004060 | 0.025                       |
| Agglomerative (Euclidean) | 0.045667 | 0.050                       |

The left column indicates the algorithm R-Clustering is compared to, the second column indicates the p-value of the Wilcoxon signed-rank test, and the third column the alpha value with the Holm correction at a 95% confidence level

**Table 5** Results for R-Clustering, R-Clustering without PCA, and the rest of the algorithms considered in the benchmark

| Algorithm                 | Mean Rank | Winning Count |
|---------------------------|-----------|---------------|
| R-Clustering              | 4.00      | 21            |
| R-Clustering W/O PCA      | 4.07      | 17            |
| Agglomerative (Euclidean) | 5.09      | 8             |
| K-means (Euclidean)       | 5.13      | 3             |
| K-means (DTW)             | 5.26      | 3             |
| K-Shape                   | 5.43      | 9             |
| C-means (Euclidean)       | 5.88      | 2             |
| K-medoids (Euclidean)     | 6.20      | 1             |
| Density Peaks (Euclidean) | 6.66      | 1             |
| Density Peaks (DTW)       | 7.30      | 5             |



**Fig. 4** Performance on the DucksAndGeese dataset. The dataset size is fixed at 100 time series, with time series lengths varying. The graph depicts how changes in time series length impact the efficiency of the three algorithms

classifier indicate that R-Clustering presents significant differences in terms of mean rank with all other algorithms.

To conclude this subsection, we incorporate the performance results of R-Clustering algorithm without the PCA stage (See Table 5). This step is taken to validate and understand the contribution made by the PCA stage to the overall performance of the algorithm. In addition to being faster, R-Clustering outperforms R-Clustering without the PCA stage. However, it is worth noting that R-Clustering without PCA still achieves significant results and secures the second position across all three measured magnitudes.

#### 4.3.2 Computation time and scalability

In this subsection, we compare the computation time and scalability of R-Clustering algorithm with the Agglomerative algorithm and K-Shape, which are the second- and third-best performers based on winning counts. These two algorithms represent diverse approaches to time clustering, with Agglomerative employing a hierarchical strategy, and K-Shape utilizing a shape-based approach. The total computational time across all 117 datasets is 7 minutes for R-Clustering (43 minutes for R-Clustering without PCA), 4 hours 32 minutes for K-Shape, and 8 minutes for the Agglomerative algorithm. Nevertheless, it is important to highlight that the time complexity of the Agglomerative algorithm is  $\mathcal{O}(n^2)$  (De Hoon et al. 2004). This characteristic might pose computational challenges for larger datasets and may limit proper scalability, as the subsequent experiment will show.

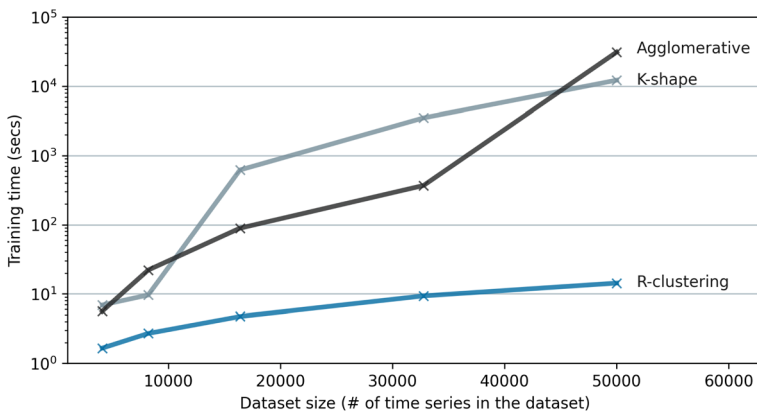
In the scalability study, we use two recent datasets not included in the benchmark: DucksAndGeese and InsectSound. The DucksAndGeese dataset consists of 100 time series across 5 classes, making it the longest dataset in the archive with a length of 236,784 points. The InsectSound dataset comprises 50,000 time series, each with a length of 600 points and spread across 5 classes. The results are depicted

in Figs. 4, 5, which demonstrate the scalability of R-Clustering in terms of both time series length and size. R-Clustering scales linearly with respect to two parameters: the length of the time series and the number (or size) of time series in the dataset. For smaller dataset sizes, Agglomerative algorithm performs the fastest, even for long time series. However, R-Clustering outperforms the other algorithms when dealing with moderate to large datasets.

Despite the fact that the training stage of the Agglomerative algorithm is faster for certain data sizes, it exhibits drawbacks in some applications. R-Clustering, like other algorithms based on K-means, can classify new data points easily using the centroids calculated during the training process. This is accomplished by assigning the new instance to the class represented by the nearest centroid. In contrast, the Agglomerative algorithm does not generate any parameter that can be applied to new instances. Consequently, when using Agglomerative to classify new data, the entire training process must be repeated, incorporating both the training data and the new observation.

### 4.3.3 Statistical analysis of the benchmark

To strengthen the results of the cited benchmark, in accordance with the recommendations provided by Garcia and Herrera (2008), we repeat the pairwise comparisons among each of the algorithms in the benchmark, not only with the newly presented method as a control classifier. The results are displayed in Table 7 in the appendix, which indicates which pairs of algorithms exhibit a significant difference in performance regarding mean rank at a 95% confidence level. It is important to note that the threshold for alpha value is not fixed at 0.05, but it is adjusted according to the Holm correction to manage the family-wise error (Demšar 2006). In this comparison, we notice that R-Clustering doesn't exhibit significant differences with certain



**Fig. 5** Comparative performance of the algorithms on the InsectSound dataset. The time series length is fixed at 600 points, while the size of the dataset varies. The graph illustrates the impact of changes in dataset size on the efficiency of the three algorithms

algorithms as it did in the earlier comparison where it was the control classifier. The reason for this difference is the increased number of pairwise tests being conducted, which, in turn, diminishes the overall statistical power of the experiment.

While the authors of the UCR archive recommend the use of a critical differences diagram for illustrating these types of comparisons (Dau et al., 2019, p. 1296,1299), in our case, such a diagram would not provide much insight due to the large number of resultant groups. As a more informative alternative, we present the results in the aforementioned Table 7, clearly displaying the significant differences between each pair of algorithms.

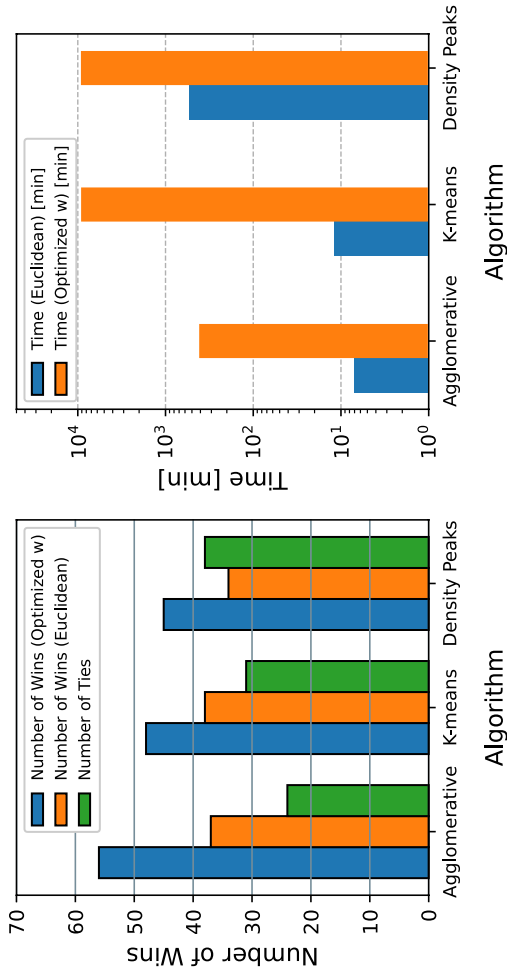
#### 4.4 Analysis of R-clustering against an improved baseline

We present the results of an enhanced benchmark that focuses on high-performing algorithms and integrates Dynamic Time Warping (DTW) optimization where suitable. We adjust the warping window width ( $w$ ) as described in Section 3.2, for algorithms that can replace the Euclidean distance with DTW. In this context, Euclidean distance is a specialized case of DTW when  $w$  is equal to 1. The algorithms selected for this DTW optimization are Agglomerative, K-means, and Density Peaks, as they are naturally amenable to DTW. The enhanced benchmark also features R-Clustering and K-Shape. However, the DTW optimization is not relevant for R-Clustering because it lacks a time series structure in its K-means stage and for K-Shape due to its inherent design.

##### 4.4.1 Performance gains and losses using an optimized warping window

We start by comparing the clustering quality and the computation time of the three mentioned algorithms: Agglomerative, K-means, and Density peaks, considering two different window settings: “Window 1” or Euclidean and “optimized w” (see Fig. 6) Our analysis utilizes 117 time series datasets from the UCR archive, excluding 11 with variable lengths from the 128 univariate dataset version. An essential observation is the trade-off between clustering quality and computational time. While the “optimized w” version generally achieves higher ARI scores across all algorithms, the computational time is significantly higher compared to “Window 1” or Euclidean version. For instance, the K-means algorithm takes more than 150 hours with “Optimized w”, compared to just 12 minutes with “Window 1”. This raises important considerations for real-world applications where time efficiency may be a crucial factor.

The p-values obtained through statistical testing provide additional insights. For Agglomerative, the p-value of 0.010 indicates that the difference in ARI scores between the two window settings is statistically significant. This suggests that the higher computational time for “optimized w” is justified by a significant improvement in clustering quality. However, for K-means and Density Peaks, the p-values were above the 0.05 significance level, indicating that there is not enough evidence to reject the null hypothesis that both versions of the algorithm perform equally. Hence, the extra computational time may not be justifiable in these cases.



**Fig. 6** Comparative evaluation of three clustering algorithms: Agglomerative, K-means, and Density Peaks, using two different configurations: “optimized w” and Euclidean window of size 1. The left figure displays the number of wins for each algorithm under the two settings, based on Adjusted Rand Index (ARI) scores. The right figure represents the computational time required by each algorithm for both window configurations, in a logarithmic scale

**Table 6** Clustering quality and computation time for the algorithms included in the improved benchmark

| Classifier Name             | Winning Count | Mean Rank | Computation Time (min) |
|-----------------------------|---------------|-----------|------------------------|
| R-Clustering                | 31            | 2.52      | 7                      |
| Agglomerative (optimized w) | 19            | 2.73      | 413                    |
| K-Shape                     | 15            | 3.06      | 272                    |
| Density Peaks (optimized w) | 11            | 3.67      | 9211                   |
| K-means (optimized w)       | 3             | 3.02      | 9171                   |

The Agglomerative algorithm showed a clear advantage for “optimized w” both in terms of ARI. It may be particularly well-suited for scenarios where high performance is a priority, and computational time is not a severe constraint. On the other hand, while K-means and Density peaks algorithms did show more wins with “optimized w”, the large computational time and lack of statistical evidence call for caution. The choice between “optimized w” and “Window 1” may depend on the specific requirements of the application.

In conclusion, although the “optimized w” setting typically results in higher ARI scores, the trade-off comes in the form of significantly increased computational time. Together with the lack of statistical support in some cases, this suggests a careful evaluation to choose the right version of the algorithm.

#### 4.4.2 Results with the improved benchmark

For the improved benchmark, we select top-performing algorithms: R-Clustering, Agglomerative, K-means, K-Shape, and Density Peaks. The “optimized w” method is applied to those algorithms susceptible to it, namely: Agglomerative, K-means, and Density Peaks. We evaluate clustering quality using the Adjusted Rand Index (ARI) across validation time series datasets from the UCR archive and assess the computation time across both the validation and development datasets. The results are summarized in Table 6.

As summarized in the table, the clustering quality reveals R-Clustering to be the superior performer, with the highest number of wins (31) and the lowest mean rank (2.52) compared to all other algorithms. The optimized version of Agglomerative also performed well with 19 wins and a mean rank of 2.73. It’s worth noting that it was the only “optimized w” algorithm to show statistical significance in the earlier comparisons. Following closely is K-Shape algorithm with 15 wins and a mean rank of 3.06. To assess the statistical significance of these results, a Friedman test was conducted under the null hypothesis of there being no differences in performance in terms of mean rank. The hypothesis could not be rejected at the 95% confidence level, indicating that more experiments could be needed to get stronger statistical conclusions. Regarding computation time, there is a marked difference between R-Clustering and the rest of the algorithms. R-Clustering emerges as the fastest, completing in 7 minutes, without compromising on clustering quality. The

time data places K-Shape algorithm in a similar category as the “optimized w” version of Agglomerative but makes it notably faster than both K-means and Density Peaks using “optimized w”. In conclusion, R-Clustering stands out for its balance of speed and performance, indicating its potential as an effective algorithm in time series clustering tasks.

## 5 Conclusions and future work

We have presented R-Clustering, a clustering algorithm that incorporates static random convolutional kernels in conjunction with Principal Component Analysis (PCA). This algorithm transforms time series into a new data representation by first utilizing kernels to extract relevant features and then applying PCA for dimensionality reduction. The resultant embedding serves as an input for the K-means algorithm with Euclidean distance. We evaluated this new algorithm by adhering to the procedures of a recent clustering benchmark study that utilizes the UCR archive - the largest public archive of labeled time series datasets - and state-of-the-art clustering algorithms. Notably, R-Clustering obtains first place among all the evaluated algorithms across all datasets, using the same evaluation methods deployed in the reference study. The results also show that R-Clustering maintains its high performance even against stronger baselines, demonstrating its versatility and strength. Furthermore, we demonstrate the scalability of R-Clustering concerning both time series length and dataset size, with it becoming the fastest algorithm for large datasets. To provide more robustness to these results, and in alignment with recent recommendations from the machine learning academic community, we complemented the cited benchmark results with a pairwise statistical comparison of the included algorithms. This statistical analysis, coupled with the fact that the code used for generating the results in this paper is publicly available, should facilitate the testing of future time series clustering algorithms.

The effectiveness of random kernels in improving the clustering quality of time series has been demonstrated through our experimental results. This finding opens up several future research directions, such as adapting R-Clustering for multivariate series, extending its use to other types of data like image clustering, and investigating the relationship between the number of kernels and performance. We anticipate that such a study could suggest an optimal number of kernels depending on the time series length.

The excellent performance of the convolution operation with static random kernels has been demonstrated through experimental results. We also encourage the academic community to engage in a more detailed analysis of the theoretical aspects of random kernel transformations. Progress in this direction could enhance our understanding of the process.

In conclusion, the R-Clustering algorithm has shown promising results in clustering time series data. Its scalability and superior performance make it a valuable tool for various applications. Future research and analysis in this field will contribute to the advancement of time series clustering algorithms and our understanding of random kernel transformations.

## Appendix A Results of the pairwise-comparisons

**Table 7** Pairwise comparisons of the algorithms according to the Wilcoxon signed-rank test with a 95% confidence level and Holm correction applied to the alpha values

| Algorithm 1               | Algorithm 2               | p value   | Sign. difference |
|---------------------------|---------------------------|-----------|------------------|
| Density Peaks (Euclidean) | R-clustering              | 2.939e-08 | True             |
| K-means (Euclidean)       | K-medoids (Euclidean)     | 7.516e-08 | True             |
| Density Peaks (DTW)       | R-clustering              | 3.651e-07 | True             |
| Agglomerative (Euclidean) | K-medoids (Euclidean)     | 1.183e-06 | True             |
| K-medoids (Euclidean)     | R-clustering              | 6.988e-06 | True             |
| Density Peaks (DTW)       | K-means (DTW)             | 9.209e-06 | True             |
| Agglomerative (Euclidean) | Density Peaks (DTW)       | 1.863e-05 | True             |
| Density Peaks (Euclidean) | K-means (Euclidean)       | 4.440e-05 | True             |
| Density Peaks (DTW)       | K-means (Euclidean)       | 7.363e-05 | True             |
| C-means (Euclidean)       | R-clustering              | 9.099e-05 | True             |
| Agglomerative (Euclidean) | Density Peaks (Euclidean) | 1.541e-04 | True             |
| K-means (Euclidean)       | R-clustering              | 1.910e-04 | True             |
| Density Peaks (Euclidean) | K-means (DTW)             | 2.032e-04 | True             |
| Density Peaks (DTW)       | K-medoids (Euclidean)     | 5.362e-04 | True             |
| Agglomerative (Euclidean) | K-means (Euclidean)       | 6.785e-04 | True             |
| K-means (DTW)             | K-medoids (Euclidean)     | 6.857e-04 | True             |
| K-means (DTW)             | R-clustering              | 1.207e-03 | True             |
| Agglomerative (Euclidean) | C-means (Euclidean)       | 1.264e-03 | True             |
| Density Peaks (Euclidean) | K-shape                   | 1.682e-03 | True             |
| C-means (Euclidean)       | K-means (Euclidean)       | 2.149e-03 | True             |
| C-means (Euclidean)       | K-means (DTW)             | 2.311e-03 | True             |
| K-shape                   | R-clustering              | 4.440e-03 | False            |
| Density Peaks (DTW)       | K-shape                   | 4.906e-03 | False            |
| Agglomerative (Euclidean) | R-clustering              | 1.117e-02 | False            |
| Density Peaks (Euclidean) | K-medoids (Euclidean)     | 2.044e-02 | False            |
| C-means (Euclidean)       | Density Peaks (DTW)       | 5.935e-02 | False            |
| Density Peaks (DTW)       | Density Peaks (Euclidean) | 1.089e-01 | False            |
| K-medoids (Euclidean)     | K-shape                   | 1.367e-01 | False            |
| C-means (Euclidean)       | K-shape                   | 1.850e-01 | False            |
| C-means (Euclidean)       | Density Peaks (Euclidean) | 2.050e-01 | False            |
| K-means (DTW)             | K-means (Euclidean)       | 2.188e-01 | False            |
| K-means (DTW)             | K-shape                   | 2.249e-01 | False            |
| Agglomerative (Euclidean) | K-shape                   | 3.193e-01 | False            |
| Agglomerative (Euclidean) | K-means (DTW)             | 7.882e-01 | False            |
| C-means (Euclidean)       | K-medoids (Euclidean)     | 9.768e-01 | False            |
| K-means (Euclidean)       | K-shape                   | 9.918e-01 | False            |



**Acknowledgements** We sincerely appreciate the creators and maintainers of the UCR archive (Dau et al. 2019; Keogh and Foliás 2002). Their work collecting, cleaning and curating the 128 datasets proved essential for benchmarking R-Clustering. Additionally, we are grateful to the reviewers and the editor for their feedback, which significantly improved the quality of this work.

**Funding** Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. The research conducted to obtain the results presented in this paper, as well as the elaboration of the paper itself, have received funding from the following grants: Ministerio de Asuntos Económicos y Transformación Digital and the European Union-NextGenerationEU through the project PRITIA-CLOUD; Ministerio de Ciencia e Innovación/AEI and the European Union-NextGenerationEU through the project Towards an Auditable Internet (AUDINT) under the grant number TED2021-132076B-I00. Funding for APC: Universidad Carlos III de Madrid (Agreement CRUE-Madroño 2024).

## Declarations

**Conflicts of interest** The authors declare that they maintain no conflicts of interest and have no associations with or participation in any organization or entity that possesses any financial stake or non-financial interest in the topics discussed within this manuscript.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

## References

- Aggarwal CC, Hinneburg A, Keim DA (2001) On the surprising behavior of distance metrics in high dimensional space. In: Database theory-ICDT 2001: 8th international conference London, UK, January 4–6, 2001 Proceedings 8, Springer, pp 420–434
- Aghabozorgi S, Shirkhorshidi AS, Wah TY (2015) Time-series clustering—a decade review. *Inform Syst* 53:16–38
- Benavoli A, Corani G, Mangili F (2016) Should we really use post-hoc tests based on mean-ranks? *The J Mach Learn Res* 17(1):152–161
- Bengio Y, Courville A, Vincent P (2013) Representation learning: A review and new perspectives. *IEEE Trans Pattern Anal Mach Int* 35(8):1798–1828
- Berndt DJ, Clifford J (1994) Using dynamic time warping to find patterns in time series. *KDD workshop*. Seattle, WA, USA, pp 359–370
- Beyer K, Goldstein J, Ramakrishnan R, et al (1999) When is “nearest neighbor” meaningful? In: Database Theory—ICDT’99: 7th International conference Jerusalem, Israel, January 10–12, 1999 Proceedings 7, Springer, pp 217–235
- Caron M, Bojanowski P, Joulin A, et al. (2018) Deep clustering for unsupervised learning of visual features. In: Proceedings of the European conference on computer vision (ECCV), pp 132–149
- Ciresan DC, Meier U, Masci J, et al. (2011) Flexible, high performance convolutional neural networks for image classification. In: Twenty-second international joint conference on artificial intelligence
- Dau HA, Silva DF, Petitjean F et al (2018) Optimizing dynamic time warping’s window width for time series data mining applications. *Data Mining Knowl Discovery* 32:1074–1120
- Dau HA, Bagnall A, Kamgar K et al (2019) The ucr time series archive. *IEEE/CAA J Auto Sinica* 6(6):1293–1305
- De Hoon MJ, Imoto S, Nolan J et al (2004) Open source clustering software. *Bioinformatics* 20(9):1453–1454

- Dempster A, Petitjean F, Webb GI (2020) Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining Knowl Discovery* 34(5):1454–1495
- Dempster A, Schmidt DF, Webb GI (2021) Minirocket: A very fast (almost) deterministic transform for time series classification. In: *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pp 248–257
- Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. *The J Mach Learn Res* 7:1–30
- Friedman M (1937) The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *J Amer Stat Assoc* 32(200):675–701
- García S, Herrera F (2008) An extension on " statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of machine learning research* 9(12)
- Goodfellow I, Bengio Y, Courville A (2016) *Deep learning*. MIT press
- Greff K, Srivastava RK, Koutník J et al (2016) Lstm: A search space odyssey. *IEEE Trans Neural Netw Learn Syst* 28(10):2222–2232
- Huang GB (2014) An insight into extreme learning machines: random neurons, random features and kernels. *Cognitive Computation* 6(3):376–390
- Hubert L, Arabie P (1985) Comparing partitions. *Journal of Classification* 2(1):193–218
- Ismail Fawaz H, Forestier G, Weber J et al (2019) Deep learning for time series classification: a review. *Data Mining Knowl Discovery* 33(4):917–963
- Jain AK, Murty MN, Flynn PJ (1999) Data clustering: a review. *ACM Comput Surv (CSUR)* 31(3):264–323
- Jarrett K, Kavukcuoglu K, Ranzato M, et al. (2009) What is the best multi-stage architecture for object recognition? In: *2009 IEEE 12th international conference on computer vision, IEEE*, pp 2146–2153
- Javed A, Lee BS, Rizzo DM (2020) A benchmark study on time series clustering. *Mach Learn Appl* 1:100001
- Jolliffe IT (2002) *Principal component analysis for special types of data*. Springer
- Kailath T (1980) *Linear systems*, vol 156. Prentice-Hall Englewood Cliffs, NJ
- Keogh E, Folias T (2002) The ucr time series data mining archive
- Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems* 25
- Kumar RP, Nagabhushan P (2006) Time series as a point-a novel approach for time series cluster visualization. In: *DMIN, Citeseer*, pp 24–29
- Lakhina A, Crovella M, Diot C (2005) Mining anomalies using traffic feature distributions. *ACM SIGCOMM Comput Commun Rev* 35(4):217–228
- Långkvist M, Karlsson L, Loufi A (2014) A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recogn Lett* 42:11–24
- Li Z, Tang J (2015) Unsupervised feature selection via nonnegative spectral analysis and redundancy control. *IEEE Trans Image Process* 24(12):5343–5355
- Li Z, Yang Y, Liu J, et al. (2012) Unsupervised feature selection using nonnegative spectral analysis. In: *Proceedings of the AAAI conference on artificial intelligence*, pp 1026–1032
- Likas A, Vlassis N, Verbeek JJ (2003) The global k-means clustering algorithm. *Pattern Recogn* 36(2):451–461
- Löning M, Bagnall A, Ganesh S, et al. (2019) sktime: A unified interface for machine learning with time series. *arXiv preprint arXiv:1909.07872*
- Lowe DG (1999) Object recognition from local scale-invariant features. In: *Proceedings of the seventh IEEE international conference on computer vision, Ieee*, pp 1150–1157
- Lubba CH, Sethi SS, Knaute P et al (2019) Catch22: canonical time-series characteristics: Selected through highly comparative time-series analysis. *Data Mining Knowl Discovery* 33(6):1821–1852
- Ma Q, Zheng J, Li S, et al. (2019) Learning representations for time series clustering. *Advances in Neural Information Processing Systems* 32
- MacQueen J (1967) Classification and analysis of multivariate observations. In: *5th Berkeley Symp. math. statist. probability*, pp 281–297
- McDowell IC, Manandhar D, Vockley CM et al (2018) Clustering gene expression time series data using an infinite gaussian process mixture model. *PLoS Computat Biology* 14(1):e1005896
- Minka T (2000) Automatic choice of dimensionality for pca. *Advances in Neural Information Processing Systems* 13
- Paparrizos J, Gravano L (2015) k-shape: Efficient and accurate clustering of time series. In: *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pp 1855–1870

- Pearson K (1901) Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine J Sci* 2(11):559–572
- Pedregosa F, Varoquaux G, Gramfort A et al (2011) Scikit-learn: Machine learning in Python. *J Mach Learn Res* 12:2825–2830
- Pereira S, Pinto A, Alves V et al (2016) Brain tumor segmentation using convolutional neural networks in mri images. *IEEE Trans Med Imaging* 35(5):1240–1251
- Proakis JG, Manolakis DG (1996) *Digital signal processing: principles, algorithms, and applications. Digital signal processing: principles*
- Rand WM (1971) Objective criteria for the evaluation of clustering methods. *J Amer Stat Assoc* 66(336):846–850
- Saxe AM, Koh PW, Chen Z, et al (2011) On random weights and unsupervised feature learning. In: *Proceedings of the 28th international conference on international conference on machine learning*, pp 1089–1096
- Sebastiani F (2002) Machine learning in automated text categorization. *ACM Comput Surv (CSUR)* 34(1):1–47
- Shi J, Malik J (2000) Normalized cuts and image segmentation. *IEEE Trans Pattern Anal Mach Intell* 22(8):888–905
- Shi L, Du L, Shen YD (2014) Robust spectral learning for unsupervised feature selection. In: *2014 IEEE International conference on data mining, IEEE*, pp 977–982
- Steinley D (2004) Properties of the hubert-arable adjusted rand index. *Psychological Methods* 9(3):386
- Thorndike RL (1953) Who belongs in the family. In: *Psychometrika, Citeseer*
- Vinh NX, Epps J, Bailey J (2009) Information theoretic measures for clusterings comparison: is a correction for chance necessary? In: *Proceedings of the 26th annual international conference on machine learning*, pp 1073–1080
- Von Luxburg U (2007) A tutorial on spectral clustering. *Statistics and Computing* 17(4):395–416
- Wang Z, Yan W, Oates T (2017) Time series classification from scratch with deep neural networks: A strong baseline. In: *2017 International joint conference on neural networks (IJCNN), IEEE*, pp 1578–1585
- Xie J, Girshick R, Farhadi A (2016) Unsupervised deep embedding for clustering analysis. In: *International conference on machine learning, PMLR*, pp 478–487
- Yang Q, Wu X (2006) 10 challenging problems in data mining research. *Intern J Inform Technol Decision Making* 5(04):597–604
- Zhang Q, Wu J, Zhang P et al (2018) Salient subsequence learning for time series clustering. *IEEE Trans Pattern Anal Mach Intell* 41(9):2193–2207
- Zhao B, Lu H, Chen S et al (2017) Convolutional neural networks for time series classification. *J Syst Eng Electron* 28(1):162–169

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.