

Feature-driven Time Series Clustering

Donato Tiano
donato.tiano@univ-lyon1.fr
Lyon 1 University
Lyon, France

Angela Bonifati
angela.bonifati@univ-lyon1.fr
Lyon 1 University
Lyon, France

Raymond Ng
rng@cs.ubc.ca
University of British Columbia
Vancouver, Canada

ABSTRACT

The problem of clustering time series has several applications in real-life contexts, especially in data science and data analytics pipelines. Existing time series clustering algorithms are ineffective for feature-rich real-world time series since they only compute the similarity of time series based on raw data or use a fixed set of features. In this paper, we develop a feature-based semi-supervised clustering framework addressing the above issues for variable-length and heterogeneous time series. Specifically, we rely on a graph encoding of the time series that is obtained by considering a high number of significant extracted features. We then employ community detection and leverage a co-occurrence matrix in order to group together all the best clustering results. Our extensive experimental assessment shows the scalability and robustness of our approach along with its superiority against state of the art clustering algorithms on both real-world healthcare data and UCR benchmark data.

1 INTRODUCTION

The goal of clustering is to organize unlabeled data objects into homogeneous groups while minimizing intra-cluster dissimilarity and maximizing inter-cluster dissimilarity [9]. In this paper, we present FeatTS, a Semi-Supervised Clustering method that leverages features extracted from the raw time series to create clusters that reflect, as much as possible, the original time series. The FeatTS algorithm leverages the concepts of Constrained Clustering, more specifically Clustering by Seeding. The most prominent approach in this category is Seeded kMeans [3], which relies on a small amount of labels of the original dataset in order to create two kinds of links, i.e. Must Link and Cannot Link. Must links are connections between two data points that represent a “constraint of belonging”. This means that the data points (or time series at large) should be clustered together. Cannot links do the opposite thus leading to separate data points. Leveraging these two kinds of links, Seeded kMeans allows to discover clusters that respect them.

Our approach differs from existing methods in the literature since it employs the features of the time series, whereas existing methods focus on the similarity of the time series themselves [20]. The novelty of FeatTS consists in automatically selecting the most appropriate statistical features based on the dataset provided as input, the latter characteristic being relevant for data science and data analytics pipelines. In fact, not all the features have the same quality and choosing a subset of high-quality features for each dataset is beneficial for the clustering step. Moreover, the features of time series are interpretable by humans, thus leading to a more transparent and human-centric clustering process. To the best of our knowledge, FeatTS is the

first feature-based semi-supervised clustering framework with these characteristics.

In designing our approach, we were inspired by the peculiarities of real-life time series, in particular those found in the time series of patients suffering from end-stage kidney diseases. These time series describe the change over time of the Glomerular Filtration Rate (GFR) signal, estimating how much blood passes through the glomeruli each minute. How GFR changes is crucial for the patient’s survival, since rapidly descending values of GFR over time indicate a dangerous condition that might lead to kidney failure and even death. A more stable evolution of GFR over time is less worrisome for the concerned patient, thus guiding an appropriate treatment. The key question to tackle is how to select the subset of features that help medical doctors to discriminate the patients based on the label while performing clustering.

Once the features are selected, FeatTS computes the global relationships between the time series based on their statistical features. We use graph networks to obtain such an encoding. Indeed, FeatTS converts each time series into nodes and creates weighted edges. Each edge represents the distance between the connected nodes of the edge, i.e. the difference between the values of two different time series using the selected feature. FeatTS prunes the graphs based on a threshold and applies a Community Detection algorithm in order to obtain the global relationship among time series. As a final step, FeatTS will holistically merge the results of the communities into a Co-Occurrence matrix, on which clustering (K-Medoid) is applied. Figure 1 depicts at a high level the various steps of our proposed algorithm, while a detailed explanation is provided in the rest of the paper.

As already observed, in the supervised feature selection step, FeatTS allows to select the most appropriate statistical features based on the labels of the time series. The selected features are then used to cluster the entire dataset, including the time series that may have unknown labels. Using Figure 2(a) as a running example on GFR data, our algorithm allows us to obtain the resulted clusters for the four time series reported in Figure 2 (d) even with missing labels for TS_2 and TS_4 .

An observant reader may question how this clustering task is different from classification, in which features are selected to build classifiers separating the time series based on the class labels. The key advantage of the clustering task we perform here is that the number of clusters to be formed can be arbitrarily different from the number of classes. For instance, in our kidney failure example, medical researchers may want more clusters to be formed than the two classes of kidney failures or not. Notice that the same separation cannot be achieved with classification [2] as a classifier cannot “sub-divide” the “kidney failure” label. We summarize our main contributions as follows:

- (1) We introduce a novel semi-supervised clustering method leveraging the most discriminating features extracted from the time series. We treat the time series similar to each other as communities and we encode the different communities into a co-occurrence matrix, allowing to obtain a unified similarity value for the time series.

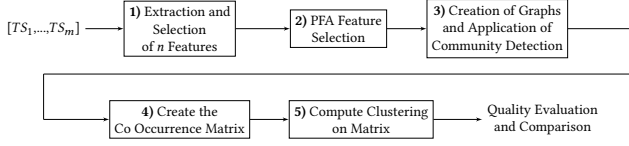


Figure 1: The algorithmic pipeline of FeatTS.

- (2) Contrarily to previous work, our method allows to treat at par all the features of a given dataset instead of pre-selecting a fixed number of them for all datasets or just leveraging the similarity of raw data.
- (3) Our method achieves more high-quality results compared with the latest baselines (among which Seeded kMeans [3] and k-Shape [17]) on the literature datasets. In addition, it obtains excellent results in terms of scalability.

The paper is organized as follows. Section 2 discusses the related work. In Sections 3, we describe in detail the steps of our clustering framework. In Section 4, we describe our experimental setup on real-life and on benchmarking data. Section 5 presents the various results of our experimental assessment. Finally, Section 6 concludes our work and discusses future directions.

2 RELATED WORK

Semi-supervised learning is a combination of supervised and unsupervised learning. It uses a small amount of labeled data and a large amount of unlabeled data in order to train a model. As such, it avoids the problem of finding a large amount of labeled data. A subcategory of semi-supervised clustering is Constrained Clustering, that enables the creation of Must Link and Cannot Link, as already explained in Section 1. There exist various methods to create these constraints; among which active learning [19] by relying on the user to provide such constraints. Another method would exploit the labelled data provided on input, as done by Seeded KMeans[3]. The latter uses the labels provided as input to find the constraints among the data and the centroids and then applies the kMeans algorithm to find the clusters.

In the literature, approaches similar to ours exist that perform time series clustering. However, other feature-based approaches [15, 21] only consider a predefined set of features that are limited with real-life time series exhibiting a richer number of features.

Among the unsupervised clustering algorithms [1], kShape[17] can be considered as the state of the art algorithm. It computes the most representative time series in a given cluster and inserts each new time series into one of these clusters based on distance. The problem with this approach is that it can often lead to assigning spurious time series to clusters. As shown in the literature [7], the usage of raw time series can spur high noise levels.

3 A FULL-FLEDGED PIPELINE

The pipeline of FeatTS as illustrated in Figure 1 is a combination of steps that contribute to the quality of the clustering results. We describe these steps in detail in the following.

3.1 Feature Extraction and Selection

The first step of the pipeline is the *feature extraction* step from the time series corresponding to step 1) in Figure 1. We consider feature extraction methods available in the literature and in readily predefined libraries [5]. Formally, given a vector of features $[f_1, f_2, \dots, f_n]$ extracted, we construct a table containing the value of each feature, thus having as columns the features and

having the time series $[TS_1, TS_2, \dots, TS_m]$ as rows. As a simple example, in Figure 2(a), we show an instance of the table for 4 time series and 6 features. Moreover, each time series is displayed with its corresponding label.

The features shown in Figure 2(a) represent only a tiny subset of the set of features that can be extrapolated from the time series. Indeed, the tsfresh[5] library allows us to extract a significantly higher number of features. Therefore, *feature selection* becomes pivotal in our setting, since not all the features have the same relevance for the subsequent clustering steps. In particular, we compute the relevance of the extracted features by solely using the feature values corresponding to the class label of the time series (e.g. in Figure 2(a) the class ‘Kidney Failure’ or ‘No Kidney Failure’).

The Benjamini-Yekutieli is a supervised procedure [4] that allows us to identify the relevance of the features, based on the label associated with the time series. It computes the p-value of each feature provided as input based on their relevance. The p-value is an important metric that allows us to quantify the significance of each feature. The output of Benjamini-Yekutieli procedure is a list of features ranked by their p-values. Among these features, usually only a subset of them have an acceptable relevance. From our empirical study, it has been evinced that the top-20 features in order of relevance are sufficient to obtain high-quality clustering.

Usually, one of the main problems when computing the p-value is that the redundancy of the obtained features. It is desirable to find a duplicate-free combination of the features that is still quality preserving and small in number. Therefore, once we select the 20 features from the list produced by Benjamini-Yekutieli, we need an algorithm that allows us to find a minimum number of features that is representative of the other features not included in the analysis.

To do so, we apply a technique called Principal Feature Analysis (PFA) [12]. PFA is a variation of Principal Component Analysis (PCA). The key difference is that PFA preserves the original values of the features and thus the distance between them. Thus, we can leverage the concept of explained variance, representing the ratio between the variance of one single feature and the sum of variances of all individual features. We fix a value t of the explained variance, which is in our experiments equal to 0.9. That is, out of the 20 features selected in the Benjamini-Yekutieli procedure, we choose the minimum number of features for which the sum of their variance covers the 90% of the information produced by the rest of the features. This value is the best result produced empirically with various values of the threshold t . In our example, among all the features presented in Figure 2(a), we have selected only *quantile*, *trend_stderr* and *trend_rvalue*, as shown in Figure 2(b).

3.2 Graph Rendering and Community Detection

We convert the time series and their relationships into edge-weighted graphs. The encoding of time series into edge-weighted graphs allows us to represent our clustering problem in another dimension space without loss of information. This operation is crucial in order to be able to capture the global relationships among the raw time series samples.

Suppose we have a feature F_i (as selected by PFA in the previous step) and a set of n time series $\{TS_1, \dots, TS_n\}$. Let TS_i be a node v_i in the set of vertices V of a graph G . Let E be the set of

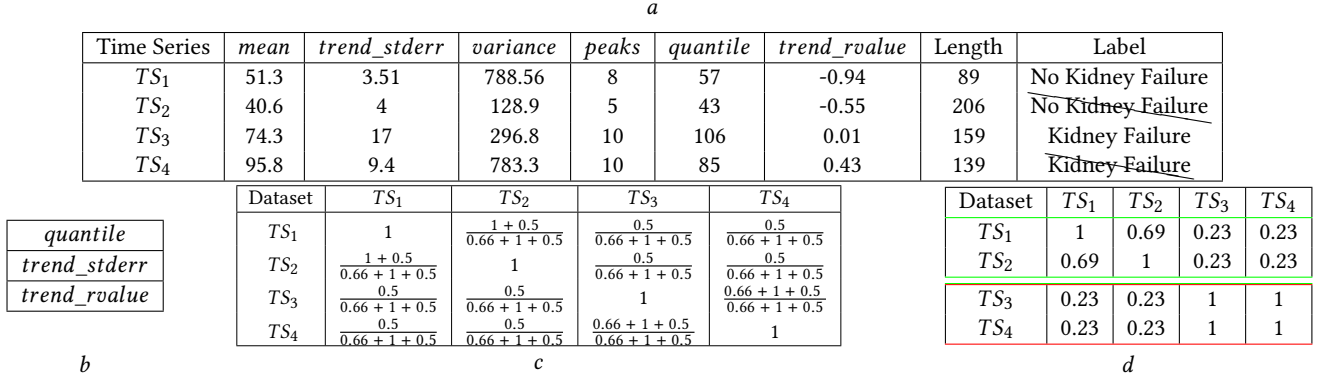
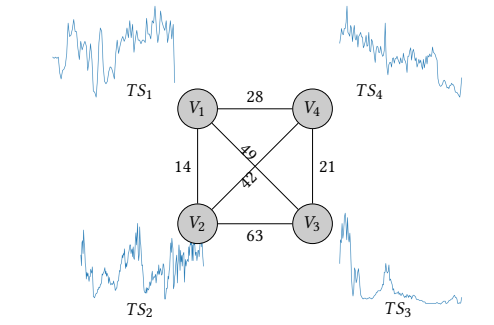
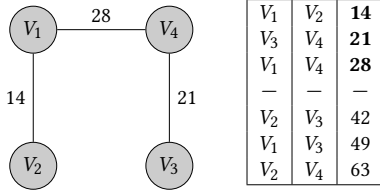


Figure 2: A running example on real-world healthcare data.



(a) Edge-weighted graph with distances as weights.



(b) The graph for a single feature after application of the threshold

Figure 3: Encoding from time series to graph.

edges of graph G , where each edge e_i connects two nodes in G representing two distinct time series. Let $w : E \rightarrow \mathbb{R}$ be an edge-based weight function. Each edge e_i is thus assigned a weight $w(e_i)$ representing the distance between the connected nodes of the edge, i.e. the difference between the values of two different time series using the feature F_i . In order to capture similarity, we only retain in G the edges whose weight is less than a given threshold distance th .

Example 3.1. As an example, let us consider the four time series as in Figure 2(a), each of which has the values of quantile; we will compute all the distances between these values. Figure 3a shows the graph encoding of these time series where the weights on the edges represent the distance between the time series, based on quantile.

One immediate question is the choice of the threshold th . Given n nodes in G corresponding to the n time series, there are $\frac{N*(N-1)}{2}$ distances between all pairs of nodes. To capture similarity, we use a simple heuristic of a percentage x that represents the proportion of the smallest distances to be kept. The threshold th is thus selected based on this x percentage.

Example 3.2. For instance, for the graph in Figure 3a, the array in Figure 3b contains the distances between the vertices in ascending

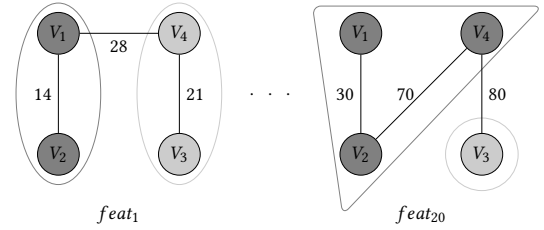


Figure 4: Application of Community Detection algorithm for each feature.

order. Suppose that the user specifies as percentage 50% of the vector. This implies that the distance boundary will be 28 and the distances higher than 28 will be discarded (i.e. the corresponding edges in the graph will be ignored). Once we have chosen the boundary distance, we can create the corresponding graph as depicted in Figure 3b.

Notice that a higher threshold would consider lower significance edges and thus weaker similarities between the times series. On the other hand a lower threshold may cut important edges. In our empirical evaluation, we used a threshold determined by a user-specified percentage of 80%, which works well in practical scenarios as we will see in the remainder of the paper. The chosen threshold will be used for all features selected by PFA and thus for all graphs created.

Note that each graph is created based on one selected feature from PFA in the previous step. Thus, if PFA selects k features, there will be k graphs, each corresponding to one notion of similarity between a pair of time series. The intention is to combine these different notions of similarity in time series clustering, by leveraging the structures of connectivity in the various graphs.

To this purpose, we apply a community detection (CD) algorithm in order to search for groups of densely connected vertices forming communities. Among the different tested algorithms, we have opted for the Greedy Modularity Algorithm [16] in the NetworkX library [8]. This algorithm turns out to strike a balance between speed and robustness and does not require any additional input parameter other than the graphs.

In Figure 4, we show an example of clustering obtained by applying this algorithm to a family of graphs. We can notice that the clustering varies from one graph to another graph. A natural question is how we can unify the different clusters in order to obtain understandable results.

3.3 Creation of the Co-Occurrence Matrix

The underlying intuition is that if two time series are similar, they will be similar for the majority of their discriminating features.

We employ a *co-occurrence matrix* [14] to put this in practice. The matrix consists of recording for each pair of time series how many times they are grouped within the same community. Intuitively, the more times they are placed within the same community, the more similar the time series are.

Co-Occurrence Matrices without weights. Assuming we have M time series and L features, we know that, once applied the CD algorithm on the L graphs, we will obtain the following result:

$$\begin{aligned} Feature_1 &= \{(TS_1, TS_3, \dots, TS_s), \dots, (TS_2, TS_4, \dots, TS_p)\} \\ Feature_2 &= \{(TS_2, TS_3, \dots, TS_t), \dots, (TS_1, TS_4, \dots, TS_i)\} \\ &\vdots \\ Feature_n &= \{(TS_2, TS_1, \dots, TS_m), \dots, (TS_3, TS_4, \dots, TS_q)\} \end{aligned}$$

where for each feature $Feature_i$ selected by the PFA, we obtain different communities (TS_1, \dots, TS_i) composed by the time series. We can now create a matrix in which the rows and columns contain all the time series of the dataset. Each cell x_{ij} in the matrix corresponds to the similarity between time series TS_i (in row i of the matrix) and TS_j (in column j of the matrix).

Next we convert the counts in the Co-Occurrence matrix into a similarity metric for the eventual clustering. We consider the number of times that a pair of time series is present in all possible communities where at least one of the two time series belongs. That is, given the time series TS_i, TS_j , the communities C and the set of all the time series M , the similarity between TS_i and TS_j will be as follows.

$$\forall TS_i, TS_j \in M, \forall c \in C \quad x_{ij} = \frac{|\{c \in C \mid TS_i \in c \ \& \ TS_j \in c\}|}{|\{c \in C \mid TS_i \in c\}|} \quad (1)$$

That is, the number of times that the two time series TS_i and TS_j fall within the same community divided by the number of times that TS_i is found within any community.

Notice that (1) is completely symmetrical. Indeed, the communities found for each feature are considered as hard clustered, i.e. a time series TS_i cannot be part of two communities of the same feature and must necessarily belong to one community. Thus, if TS_i and TS_j are within the same community of a specific feature, neither of them can be part of other communities of the same feature. Therefore, the value x_{ij} , given by the number of times TS_i and TS_j are in the same community, will be equal to x_{ji} , because TS_j and TS_i must also be in the same community.

Co-Occurrence Matrices with Weights. The application of the CD algorithm and its processing with co-occurrence matrices without weights might incur the problem of community fragmentation. More precisely, the CD algorithm might lead to the formation of a high number of communities each of which contains a few time series. This is due to the fact that some features are often not discriminatory enough for that dataset.

To overcome this problem, we assign an approximate weight to each feature, based on the number of communities that the CD algorithm derives from the graph. Again, to correctly determine the weights, we require the input of the user on the expected number of clusters.

Let w_i be a weighting function defined on each feature F_i as follows:

$$\begin{cases} w_i = \frac{C}{O_i}, & \text{if } O_i > C \\ w_i = \frac{O_i}{C}, & \text{if } C > O_i \\ w_i = 1, & \text{otherwise} \end{cases} \quad (2)$$

| Dataset | TS_1 | TS_2 | TS_3 | TS_4 |
|---------|--------|--------|--------|--------|
| TS_1 | 0 | 0.64 | 1.36 | 1.36 |
| TS_2 | 0.64 | 0 | 1.36 | 1.36 |
| TS_3 | 1.36 | 1.36 | 0 | 0 |
| TS_4 | 1.36 | 1.36 | 0 | 0 |

Table 1: Co-Occurrence Matrix with weights.

where C is the number of clusters expected by the user and O_i is the number of communities extracted by means of the CD algorithm. Hence, the weights will be higher if the number of obtained communities O is equal or sufficiently close to C and lower otherwise.

The weights will be propagated to the similarity matrix, which will now reflect the importance of each feature from a user view-point. Not surprisingly, instead of simply counting the number of times that the time series TS_i and TS_j co-occur in the same community, we now sum their weights and divide by the sum of the weights of all time series, as also shown in the following.

Example 3.3. As shown in Figure 2(b), the PFA feature selection has chosen three features, namely [trend_stderr, quantile, trend_rvalue]. After applying the CD algorithm, we obtained the following communities (per feature) among the 4 Time Series in Figure 2(a):

$$\begin{aligned} \text{quantile} &= \{(TS_1, TS_2), (TS_3, TS_4)\} \\ \text{trend_stderr} &= \{(TS_1), (TS_2), (TS_3, TS_4)\} \\ \text{trend_rvalue} &= \{(TS_1, TS_2, TS_3, TS_4)\} \end{aligned}$$

Assume now that the user specified an expected number of clusters equal to 2. Trend_stderr and trend_rvalue do not satisfy the number of clusters expected by the user. Therefore, trend_stderr will have a weight of $\frac{2}{3}$ (0.66), while trend_rvalue will have a weight of $\frac{1}{2}$ (0.5), and quantile we will have 1 as weight. We report the intermediate computation of the co-occurrence matrix with weights for this example in the Table in Figure 2(c) and the final result in the Table in Figure 2(d).

3.4 Clustering the Co-Occurrence Matrix

The co-occurrence matrix obtained in the previous step allows us to quantify the similarity between two time series. In order to be prepared for the creation of the time series clusters, we need one more intermediate step, i.e. to compute the distances between the rows of the Co-Occurrence Matrix. We employ a standard Euclidean distance to perform the row comparison.

As an example, applying the Euclidean distance between the rows of the table in Figure 2(d), we obtain Table 1. For instance, the value of the cell $C_{3,4}$ of the Table 1 is 0 because the row 3 and 4 of the Table in Figure 2(d) are equal. As a consequence, the time series 3 and 4 are always together in each cluster discovered by the CD algorithm.

Finally, we apply the standard K-Medoid algorithm [10] on the distances computed above. K-Medoid allows us to extract the time series that have the smallest distance among them.

To complete our running example, applying the K-Medoid algorithm to Table 1 requiring 2 clusters as the input parameter, we obtain two clusters $Cl_1 = \{TS_1, TS_2\}$ and $Cl_2 = \{TS_3, TS_4\}$, as shown in the Table in Figure 2(d).

The time complexity of the FeatTS is summarized in Lemma 3.4. The proof of the Lemma is available in the online repository¹.

LEMMA 3.4. Let D a dataset composed by m time series and let L the number of features chosen by PFA among the N features

¹<https://github.com/DonaTPProject/FeatTS>

extracted and k the number of requested clusters. A dataset D is evaluated by FeatTS in time $O(L(m^2) + m^2 + k(m - k)^2 + n \cdot t_f)$ and in space $O(n + L(m + E) + m^2)$.

4 EXPERIMENTAL SETUP AND A CLINICAL CASE STUDY

We use real-life time series courtesy of the Personalized Medicine Department at the European Hospital George Pompidou in Paris. These time series contain signals from patients suffering from kidney diseases. As a background, the human kidney has a lot of functions including maintenance of acid-base balance. Proper function of the kidney requires that it receives and adequately filters blood. This is performed at the microscopic level by many hundreds of thousands of filtration units called renal corpuscles, each of which is composed of a glomerulus. A global assessment of renal function is often ascertained by estimating the rate of filtration, called the glomerular filtration rate (GFR). GFR estimates how much blood passes through the glomeruli each minute. Kidney failure occurs when GFR is under $90\text{mL}/\text{min}/1.73\text{m}^2$, whereas when it drops to $15\text{mL}/\text{min}/1.73\text{m}^2$, it means that the patient needs dialysis or a transplant. Thus, it is very important to understand when a patient needs medical treatment before the GFR reaches its lowest possible value. Moreover, since dialysis is an invasive operation, it is important to understand if a sudden drop in the GFR occurs. In this case, medical doctors might recommend urgent surgery or to resort to dialysis depending on the GFR values over time.

We ran experiments on two variants of this dataset. The first variant named *Kidney3Yr* contains 222 patients (one time series per patient) and spans 1 to 3 years with a variable length between 90 and 230 data points in the time series. The second variant called *Kidney5Yr* is composed of 278 patients spanning 5 years with time series having roughly 100 data points. In both cases, we ran our experiments using only the 20% of the labeled time series in order to compute the set of features necessary to run the clustering algorithm and to emulate the real-world scenario where not all the labels of the data points are available. The features thus being ordered based on their relevance have been employed to cluster the entire unlabeled dataset into those with GFR signals concerning high-risk patients and those containing GFR for patients with lower risk.

UCR datasets. We also used 64 benchmark datasets belonging to the UCR collection[6], including both real-life and synthetic datasets. The entire list of datasets used for benchmark is available online². For consistency, we also used only 20% of the labeled time series during feature extraction during the clustering step in all experiments.

Implementation and reproducibility. Our code base is available online² with more details about the reproducibility.

5 EXPERIMENTAL RESULTS

For each dataset, we consider 20 as the upper bound of the number of features we consider in the analysis. A higher number of features are supported by our method but not indispensable to obtain better accuracy. Moreover, a higher number of features deteriorates the performance. Furthermore, we chose 80% as the threshold value of the percentage of features selected by the user. We used the AMI [18] metric, which is a well-established

| Dataset | FeatTS | kShape | SeededKMeans |
|------------------|-------------|-------------|--------------|
| Adiac | 0,31 | 0,39 | 0,52 |
| MoteStrain | 0,48 | 0,01 | 0,02 |
| TwoLeadECG | 0,88 | 0,10 | 0,07 |
| ECG200 | 0,34 | 0,11 | 0,06 |
| Computers | 0,09 | 0,06 | 0,01 |
| Coffee | 1 | 0,35 | 0,88 |
| GunPoint | 0,52 | 0 | 0 |
| Arrowhead | 0,29 | 0,26 | 0,27 |
| ItalyPowerDemand | 0,54 | 0,39 | 0 |
| Meat | 0,4 | 0,64 | 0,75 |
| OliveOil | 0,27 | 0,52 | 0,53 |
| Trace | 0,74 | 0,52 | 0,69 |
| Wine | 0,12 | 0 | 0,01 |
| Worms | 0,16 | 0,06 | 0,12 |
| ShapesAll | 0,08 | 0,62 | 0,45 |

Table 2: Results showing the values of AMI for UCR datasets

measurement of the quality of clustering. We adopt the same metric for our comparisons as well.

We consider two main baselines, the first being the state-of-the-art algorithm for time series clustering, i.e. KShape[17], and the second being the state-of-the-art algorithm for Semi Supervised time series clustering i.e. Seeded kMeans [3], sharing the same category of our approach (as detailed in Section 2). KShape[17] could not be used on the real-world GFR time series as it cannot process variable-length time series. Hence, we limit the comparison with KShape to the UCR datasets.

Other baselines of semi-supervised clustering algorithms such as SSSL[20] (Self Labeling Algorithm) and SUCCESS[13] (Cluster then Label Algorithm) could not be used in our study due to the lack of available source code.

All experiments have been executed on a Server running Linux with 64GB of RAM, Intel Xeon CPU Skylake, IBRS @ 2.6GHz.

5.1 UCR Dataset

The results in Table 2 are an excerpt of the entire results obtained by our algorithm and its baselines for various UCR datasets. It can be observed that FeatTS obtains the best results among all.

Indeed, out of 64 datasets used for the comparison, FeatTS performed better on 37 datasets. In addition, kShape performed well only on 15 datasets (out of 64) and Seeded kMeans outperformed the others in only 12 datasets (out of 64).

5.2 Kidney Case Study

As shown in Table 3, the results obtained by FeatTS are significantly more accurate than Seeded kMeans on the clinical case study. For the patients under medical supervision for 5 years, as shown in Table 3, we have obtained results following a similar trend.

| Dataset | FeatTS | SeededKMeans |
|-----------|-------------|--------------|
| Kidney3Yr | 0.56 | 0.44 |
| Kidney5Yr | 0.58 | 0.48 |

Table 3: Results on Kidney 3Yr and 5Yr Datasets

5.3 Scalability

We have assessed the scalability of our method by increasing both the number and length of time series in a dataset. In this experiment, we have used synthetic time series generated with GRATIS [11]. This tool allows a controlled generation of time series by using diverse characteristics as spectral entropy, trend, seasonality, stability, etc.

In our case, in the synthetic generation we have opted for spectral entropy and trend as the underlying characteristics since they reflect the real-life time series we have used in the rest of our experimental assessment. The spectral entropy allows to

²<https://github.com/DonaTPProject/FeatTS>

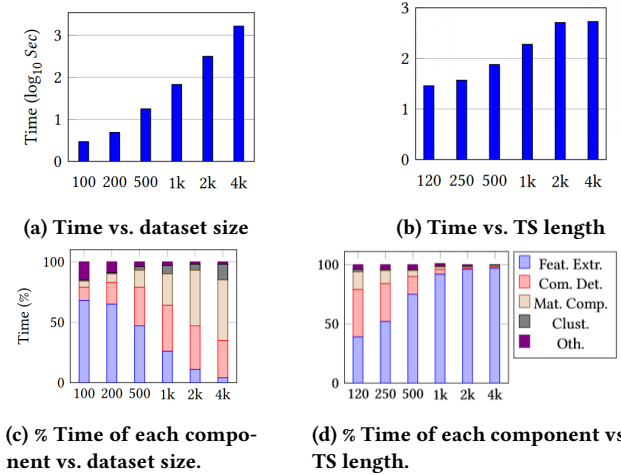


Figure 5: Scalability Results.

measure the “forecastability” of a time series. It has a range of values between 0 to 1 and a low value of entropy indicates a high signal-to-noise ratio, while large values occur when a time series is difficult to forecast. For this characteristic, we have fixed a value of spectral entropy equal to 0.6. Conversely, the trend allows to represent the occurrence of low-frequency variations in the time series and as such it is opposite to seasonality. It has a range of values between 0 to 1 and we have chosen a value of 0.9 for this experiment.

In the first experiment, we increase the number of time series for each tested dataset while the length of the time series is fixed and equal to 60. Figure 5a shows the results obtained on datasets consisting of 100, 200, 500, 1000, 2000, 4000 time series, respectively. The results show the scalability of the method in terms of time performance, while an important increase can be observed when shifting to more than 2000 time series. The times in Figure 5a are in logarithmic scale for clarity of exposition.

In order to better understand the results, we have studied the percentage of time due to each component of our pipeline as shown in Figure 5c. Upon increasing the size of the dataset, the component that is computationally more demanding is the creation of the co-occurrence matrix. Obviously, since the Co-Occurrence Matrix depends on the number of time series, the time required for its creation increases as the size increases.

In the second experiment, we have increased the length of the time series while fixing to 500 the number of time series belonging to each dataset. Figure 5b shows the results obtained increasing the length of the time series between 120 and 4000. The figure shows the scalability of the approach for time series under 2000 and a sudden increase of the time beyond this value. Also in this case, the times in Figure 5b are in logarithmic scale for better clarity. The time breakdown in Figure 5d shows that the more expensive step of the pipeline for this experiment is the feature extraction step.

6 CONCLUSION AND FUTURE WORK

Our work on clustering of time series shows that there is no one-size-fits-all solution regarding the set of features to use. In fact, we leveraged the features drawn from the data itself rather than taking a predefined set of features for all the datasets. Our flexible graph encoding allows us to process the most significant features in parallel and the further steps of our method allow us to combine the results.

This work could be improved by rendering the entire pipeline unsupervised instead of the current semi-supervised approach. This requires non-trivial extensions in order to be able to cluster the time series without loss of performance. Another improvement would be to dynamically choose the threshold for graph creation based on the processed features. Finally, the weights of the community detection algorithm could be combined with relevance degrees of the features.

7 ACKNOWLEDGEMENTS

Research funded by ANR (grant nr. 18-CE23-0002 QualiHealth). We thank B. Rance and A. Rogier for kindly providing us with the GFR data.

REFERENCES

- [1] Saeed Aghabozorgi, Ali Seyed Shirkhorshidi, and Teh Ying Wah. 2015. Time-series clustering—a decade review. *Information Systems* 53 (2015), 16–38.
- [2] Anthony J. Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn J. Keogh. 2017. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery* (2017).
- [3] Sugato Basu, Arindam Banerjee, and Raymond Mooney. 2002. Semi-supervised clustering by seeding. In *In Proceedings of ICML*. Citeseer.
- [4] Yoav Benjamini and Daniel Yekutieli. 2001. The control of the false discovery rate in multiple testing under dependency. *Annals of Statistics* (2001).
- [5] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W. Kempa-Liehr. 2018. Time Series Feature Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package). *Neurocomputing* 307 (2018).
- [6] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. 2018. The UCR Time Series Archive. *arXiv:cs.LG/1810.07758*
- [7] Levent Ertöz, Michael Steinbach, and Vipin Kumar. 2003. Finding clusters of different sizes, shapes, and densities in noisy, high dimensional data. In *Proceedings of the 2003 SIAM international conference on data mining*.
- [8] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring Network Structure, Dynamics, and Function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, Gaël Varoquaux, Travis Vaught, and Jarrod Millman (Eds.). Pasadena, CA USA, 11 – 15.
- [9] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd Edition. Springer. <https://doi.org/10.1007/978-0-387-84858-7>
- [10] Anil K. Jain and Richard C. Dubes. 1988. *Algorithms for Clustering Data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.
- [11] Yanfei Kang, Rob J. Hyndman, and Feng Li. 2020. GRATIS: GeneRAting Time Series with diverse and controllable characteristics. *Statistical Analysis and Data Mining: The ASA Data Science Journal* (2020).
- [12] Yijuan Lu, Ira Cohen, Xiang Sean Zhou, and Qi Tian. 2007. Feature selection using principal feature analysis. In *Proceedings of the 15th ACM international conference on Multimedia*. 301–304.
- [13] Kristóf Marussy and Krisztian Buza. 2013. Success: a new approach for semi-supervised classification of time-series. In *International Conference on Artificial Intelligence and Soft Computing*.
- [14] Loris Nanni, Sheryl Brahnam, Stefano Ghidoni, Emanuele Menegatti, and Tonya Barrier. 2013. Different approaches for extracting information from the co-occurrence matrix. *PloS one* 8, 12 (2013), e83554.
- [15] Alex Nanopoulos, Rob Alcock, and Yannis Manolopoulos. 2001. Feature-based classification of time-series data. *International Journal of Computer Research* (2001).
- [16] Mark Newman. 2010. *Networks: An Introduction*. Oxford University Press, Oxford, 784 pages.
- [17] John Paparrizos and Luis Gravano. 2016. K-Shape: Efficient and Accurate Clustering of Time Series. *SIGMOD Record*. (2016).
- [18] Simone Romano, Nguyen Xuan Vinh, James Bailey, and Karin Verspoor. 2016. Adjusting for chance clustering comparison measures. *The Journal of Machine Learning Research* 17, 1 (2016), 4635–4666.
- [19] Toon Van Craenendonck, Wannes Meert, Sebastijan Dumančić, and Hendrik Blockeel. 2018. Cobras ts: A new approach to semi-supervised clustering of time series. In *International Conference on Discovery Science*. Springer, 179–193.
- [20] Haishuai Wang, Qin Zhang, Jia Wu, Shirui Pan, and Yixin Chen. 2019. Time series feature learning with labeled and unlabeled data. *Pattern Recognition* (2019).
- [21] Xiaozhe Wang, Kate Smith, and Rob Hyndman. 2006. Characteristic-based clustering for time series data. *Data mining and knowledge Discovery* 13, 3 (2006), 335–364.