Time series clustering

D. Barbe, A. Debant, X. Shang
Department of Computer Science
École normale supérieure de Rennes
35170, Bruz

***Abstract*—Due to its specificity, time series clustering remains an open problem. Questions like measuring the difference between two time series and computing means are indeed crucial if one wants to be insensitive to time warping. In this paper, we present a survey of main clustering techniques and discuss about the possibilities to adapt them to time series. Then we will run experiments on each of these techniques and try to compare their performances.**

I. Introduction

The clustering problem is the following: separate a set of data into homogeneous groups. Such data can be pictures, user profiles, voice samples, etc. There are many goals. Clustering data can identify tendencies. These groups can serve as examples to classify new informations. In general, the goal is to extract informations from huge sets of data that cannot be examined by hand.

For data being represented as n-dimensional points, a lot of clustering techniques exist. They all have their advantages and disadvantages. Some can guess the number of clusters, others cannot. Some have a high computational complexity. Some behave poorly on some instances, whereas others do not.

Time series are a class of data that bring specific problems. Representing them as n-dimensional point is not the best way to take into account the temporal information. Therefore, one can search how to adapt the existing clustering techniques to time series. The notion of similarity and dissimilarity is crucial, yet is hard to define in this case. We give an example to illustrate this issue. The data is a set of voice samples pronouncing one word. The goal is to have one cluster for each word. How should the distance between two samples be defined to achieve this? Samples can vary in amplitude. But time-specific variations also occur: pitch, time offset or pronunciation length. This example highlights the need of specific tools for studying time series.

In this paper, we start by describing some standard clustering techniques. Then we discuss time series specific problems and *ad hoc* clustering algorithms. Finally, we detail experiments we ran to compare the different methods.

II. Clustering

Three methods of clustering will be presented in this section, which are *k-means Clustering*, *Kernel k-means Clustering* and *Kernel Density Estimation Clustering*. These methods are very popular and have been shown effective in many domains. Furthermore, they are simple to implement. We describe them three along with the pros and cons for each.

A. k-means Clustering

k-means clustering is one of the simplest methods of clustering, and one one the must intuitive. The algorithm takes a set of n points of dimension d, $X = (\mathbf{x_1}, ..., \mathbf{x_n})$, and a fixed number of clusters k. It returns a partition of the points into k clusters.

In other word, the basic idea here is to group all the points into k sets (clusters) $S = \{S_1, S_2, \cdots, S_k\}$ with respect to their distance to the centers of the clusters so as to minimize the sum of the squared distances within each cluster. Thus, its objective is to find the value:

$$\arg\min_{S} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \mathbf{c_i}\|^2$$

where $\mathbf{c_i}$ is the center of S_i.

It proceeds iteratively, starting by randomly choosing k centers and assigning each point to its nearest center, thus forming k clusters. Then, until convergence, the new barycenter of each cluster is computed and the assignment process is repeated.

A pseudo-code is given in Algorithm I.

B. Kernel k-means Clustering

The k-means algorithm divides a space into k clusters on the basis of the $\mathbb{L}^2$-distance. This performs well for data which can be separated linearly by a hyperplane. Otherwise, this distance can be computed in a *feature*

```
procedure K-MEANS(X, k)
    n ← size(X)
    # Initial assignment of points
    assign ← initialize(n, k)
    while not(convergence()) do
        # Computation of barycenters
        for i ∈ {1..k} do
            C[i] ← computeCenter(X, assign)
        end for
        # New assignment of points
        for i ∈ {1..n} do
            assign[i] = nearestCenter(X[i], C)
        end for
    end while
end procedure
```

Algorithm I: k-means algorithm.

space, i.e. a transformed space where the linear separability assumption holds. To this end, on the basis of kernel theory [1], we rely on a kernel function which represents the inner product in the feature space:

$$K(\mathbf{x_i}, \mathbf{x_j}) = \Phi(\mathbf{x_i}) \cdot \Phi(\mathbf{x_j})$$

where Φ is a mapping from the input space to the feature space. The Euclidean distance $D(x_i, x_j)$ in the feature space can be computed, without knowing explicitly Φ, using the so called *distance kernel trick* [4]:

$$\begin{aligned} D(\mathbf{x_i}, \mathbf{x_j})^2 = & \|\Phi(\mathbf{x_i}) - \Phi(\mathbf{x_j})\|^2 \\ = & (\Phi(\mathbf{x_i}) - \Phi(\mathbf{x_j})) \cdot (\Phi(\mathbf{x_i}) - \Phi(\mathbf{x_j})) \\ = & \Phi(\mathbf{x_i}) \cdot \Phi(\mathbf{x_i}) + \Phi(\mathbf{x_j}) \cdot \Phi(\mathbf{x_j}) \\ & -2\Phi(\mathbf{x_i}) \cdot \Phi(\mathbf{x_j}) \\ = & K(\mathbf{x_i}, \mathbf{x_i}) + K(\mathbf{x_j}, \mathbf{x_j}) - 2K(\mathbf{x_i}, \mathbf{x_j}) \end{aligned}$$

Several different kernels exist, and in our experiments, a Gaussian kernel is used [2]:

$$K(\mathbf{x_i}, \mathbf{x_j}) = exp\left(-\frac{\|\mathbf{x_i} - \mathbf{x_j}\|^2}{2\sigma^2}\right)$$

in which the hyperparameter σ has to be fixed.

Using a kernel function makes the computation of barycenters in the feature space impossible. However, in practice, we can always compute the distance from one point to the center of a cluster using the distance trick.

Let $\mathbf{c_i}$ be a centroid in feature space. We can write it as a combination of data points in feature space:

$$\mathbf{c_j} = \sum_{k=1}^{n} \alpha_{jk} \Phi(\mathbf{x_k})$$

where α_{jk} equals one if $\mathbf{x_k}$ belongs to the cluster associated with $\mathbf{c_i}$.

Then the distance from $\mathbf{x_i}$, which is a data point in feature space, to the center of a cluster can be computed as:

$$\begin{aligned} \|\Phi(\mathbf{x_i}) - \mathbf{c_j}\|^2 = & \|\Phi(\mathbf{x_i}) - \textstyle\sum_{k=1}^{n} \alpha_{jk} \Phi(\mathbf{x_k})\|^2 \\ = & K(\mathbf{x_i}, \mathbf{x_i}) - 2 \textstyle\sum_k \alpha_{jk} K(\mathbf{x_i}, \mathbf{x_k}) \\ & + \textstyle\sum_h \sum_l \alpha_{jh} \alpha_{jl} K(\mathbf{x_h}, \mathbf{x_l}) \end{aligned}$$

C. Kernel Density Estimation Clustering

Both approaches presented above have a particularity: the number k of clusters is a parameter of the algorithm. Consequently, it presumes that we already have some knowledge on the datasets to be analyzed.

Remark: Some methods exist to estimate the number of clusters automatically for the k-means clustering, such as the Silhouette method described in [3].

The Kernel Density Estimation Clustering (KDEC) is a clustering method without this requirement.

KDEC is an iterative algorithm. It starts with a single cluster of one point. Then, for each iteration, the distribution of probability corresponding to the clusters is computed and if a point is close enough to the distribution it is added to the current cluster. If none is found, then an unassigned point is chosen and a new cluster is built.

The notion of "close enough" is defined by a parameter p, which denotes the probability threshold.

The probability of being similar to the actual distribution is computed by representing each point in clusters by a Gaussian centered on it and of fixed bandwidth σ that is a parameter of the method. The pseudo-code of KDEC is shown in Algorithm II.

```
procedure KDEC(X, p, σ)
    # Initial single cluster with one point
    clusters ← [choosePoint(X)]
    # Compute the closest point to the distribution
    while (onePointNotAssigned()) do
        computeDistribution(clusters, σ)
        x ← closestPointToDistribution()
        if (probability(x) > p) then
            add x in currentCluster(clusters)
        else
            makeNewCluster(x, clusters)
        end if
    end while
end procedure
```

Algorithm II: KDEC algorithm.

Remark: It is not necessary to represent each point by a Gaussian. Any kernel can be used.

Regarding this method, we can see that the final number of clusters is not required to make clusters but an other kind of knowledge is. Parameters p and σ have

a strong influence on the resulting clusters. Finding good values for them may be a tricky issue.

D. Robust KDE

In practice, the result of KDEC may be greatly influenced by outliers. We thus try to use a variation of this approach which is called robust KDE (RKDE) [8] in order to exhibit robustness to contamination of the training sample.

RKDE achieves its robustness by combining a traditional KDE with ideas from classical M-estimation. Indeed, KDE is sensitive to the presence of outliers as it is a solution of a least squares problem. To reduce this effect, we can use M-estimation proposed by [9] to find a robust sample mean of the $\Phi(\mathbf{x_i})$'s. The idea is to add a robust loss function. Well known examples of such functions are Huber's and Hampel's ρ.

In our implementation, we choose to employ the Huber's ρ which is defined as:

$$\rho(x) = \begin{cases} x^2/2 & \text{if } 0 \leq x \leq a \\ ax - a^2/2 & \text{if } a < x \end{cases}$$

Unlike the quadratic loss, these loss functions' derivative ϕ is bounded. For the Huber's ρ:

$$\phi(x) = \begin{cases} x & \text{if } 0 \leq x \leq a \\ a & \text{if } a < x \end{cases}$$

where a is a parameter.

III. Time series

We would like now to focus on the clustering of time series, which is actually the main goal of this paper. Thus in this section, we will introduce some time series clustering approaches inspired from the classic clustering methods presented in the last section.

Definition: A time series is a function consisting of successive numerical measurements collected over a time interval. In practice, it is not continuous, thus it is a sequence of discrete data points: $X = (\mathbf{x_1}, ..., \mathbf{x_n})$.

A. Difficulties with Time Series

Applying directly the k-means algorithm to time series produces poor results. Indeed, it is designed to work with points in a finite dimension space, not sequences of such points. There are two obstacles. First, a suitable distance between must be defined for time series. Then, given this distance, a notion of average must be given too.

A simple metric to compare functions would be the $\mathbb{L}^2$-norm:

$$d(f_1, f_2) = \|f_1 - f_2\|_2 = \int_{t_0}^{t_1} |f_1(x) - f_2(x)|^2 dx$$

In reality, the functions manipulated are not continuous, and the formula must be adapted for discrete domains. Considering series of length n made of d-dimensional points (*i.e* living in $(\mathbb{R}^d)^n$):

$$d(f_1, f_2) = \|f_1 - f_2\|_2 = \sum_{x=0}^{n-1} \|\mathbf{f_1}(\mathbf{x}) - \mathbf{f_2}(\mathbf{x})\|^2$$

However, this distance is not suitable for time series. Indeed, looking at Fig. 1, we can see that in the $\mathbb{L}^2$-norm context, the red curve is nearer to the zero-function than the blue one while it is only a translation of the first. That is to say, two times series can be measured as totally different, while in fact they are very close to each other.

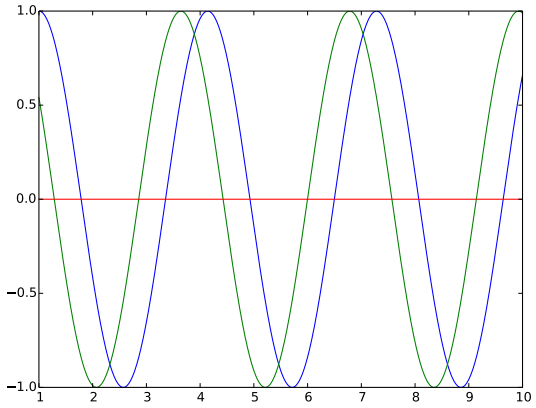

Fig. 1: Translation of time series.

Because of this issue, another distance must be used. The most common similarity measure used for time series is *Dynamic Time Warping* (DTW).

B. Dynamic Time Warping

DTW is a sequence alignment algorithm which aims to minimize the difference between two numerical sequences (e.g. time series) through a time warping process.

Suppose we have two numerical sequences $X = (\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_n})$ and $Y = (\mathbf{y_1}, \mathbf{y_2}, \cdots, \mathbf{y_m})$. A $m \times n$ matrix M will be used in which the (i, j) element will represent the (Euclidean) distance between $\mathbf{x_i}$ and $\mathbf{y_j}$.

A warping path or an alignment π is a pair of increasing integral vectors (π_1, π_2) of length p where $max(n, m) \leq p \leq n + m - 1$. It is a path through the matrix M which continuously connects elements $M(1, 1)$ and $M(n, m)$.

An acceptable alignment has to advance one step at a time, either to the right, either to the top, either to the

top right. These constraints can be described in a formal way. Indeed, we should have $1 = \pi_1(1) \leq \pi_1(2) \leq \cdots \leq \pi_1(p) = n$ and $1 = \pi_2(1) \leq \pi_2(2) \leq \cdots \leq \pi_2(p) = m$. In addition, for all indices $1 \leq i \leq p-1$:

$$(\pi_1(i+1)-\pi_1(i), \pi_2(i+1)-\pi_2(i)) \in \{(0,1),(1,0),(1,1)\}$$

To find the best match between X and Y, one needs to find the alignment that minimizes the total distance. We denote $\mathfrak{A}(n,m)$ as the set of all alignments between X and Y, then the DTW distance between these two time series is defined as:

$$DTW(X,Y) = \min_{\pi \in \mathfrak{A}(n,m)} D_{X,Y}(\pi)$$

where $D_{X,Y}(\pi)$ is defined as:

$$D_{X,Y}(\pi) = \sum_{i=1}^{|\pi|} \phi(\mathbf{x}_{\pi_1(i)}, \mathbf{y}_{\pi_2(i)})$$

Here ϕ is a local divergence that measures the discrepancy between any two points $\mathbf{x_i}$ and $\mathbf{y_j}$ from X and Y. As we have already mentioned above, we use simply the squared Euclidean distance to define this local divergence, $\phi(\mathbf{x},\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|^2$.

To compute all possible routes and then take the minimum is a naive method to implement the algorithm, but may take too long to achieve. We can, instead, compute recursively the cost of the optimal alignment by defining a cumulative distance on the subsequences $X_i = (\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_i})$ and $Y_j = (\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_j})$. The current cumulative distance will be the sum of the distance of the current cell and the minimum of the cumulative distances of its adjacent elements [5]:

$$d_{cum}(i,j) = d(\mathbf{x_i}, \mathbf{y_j}) + min\{d_{cum}(i-1, j-1), d_{cum}(i, j-1), d_{cum}(i-1, j)\}$$

However, a direct implementation of this recursive definition leads to an exponential time complexity. Thus, a dynamic programming approach is implemented by storing the partial results in a matrix, which leads to a time complexity of $O(|X| \times |Y|)$. The algorithm is shown in Algorithm III.

Remark: DTW measures a distance-like quantity between two sequences, however it is **not** a distance properly speaking, indeed, it does not guarantee the triangle inequality to hold.

Based on this measure, k-means can be applied more efficiently to numerical sequences, and in particular to time series. However, the k-means algorithm also requires to compute the averaging of a set of sequences, and this task is not obvious using only DTW. Several methods exist to deal with this problem. Our experiments used the *DTW Barycenter Averaging* (DBA) [10].

```
1: procedure DTW(X, Y)
2:     # Create the distance matrix
3:     dtwMatrix ← makeNewMatrix(m, n)
4:     for i ∈ {1..m} do
5:         dtwMatrix[i, 0] ← infinity
6:     end for
7:     for i ∈ {1..n} do
8:         dtwMatrix[0, j] ← infinity
9:     end for
10:    # Update the matrix using a dynamic approach
11:    for i ∈ {1..m} do
12:        for i ∈ {1..n} do
13:            dtwMatrix[i, j] ←
14:            distance(X[i], Y[j])+
15:              minimum(dtwMatrix[i − 1, j],
16:              dtwMatrix[i, j − 1],
17:              dtwMatrix[i − 1, j − 1])
18:        end for
19:    end for
20:    return dtwMatrix
21: end procedure
```

Algorithm III: Dynamic Time Warping.

C. Averaging of Time Series

The DTW Barycenter Averaging (DBA) method aims to minimize the sum of squared DTW distances which consists of single distances between each coordinate of the average sequence and coordinates of sequences associated to it. The main idea is to compute each coordinate of the average as the barycenter of its associated coordinates.

Suppose that we have a set of sequences $\{X_1, X_2, \cdots, X_s\}$ whose average sequence is to be determined. First, we initialize an average sequence. DBA then iteratively refines this average sequence by computing the DTW between the current average sequence and each single sequence.

Indeed, let $C = \{\mathbf{c_1}, \mathbf{c_2}, \cdots, \mathbf{c_T}\}$ be the current average sequence, and let $C' = \{\mathbf{c'_1}, \mathbf{c'_2}, \cdots, \mathbf{c'_T}\}$ be the update of C, and we want to find coordinates of C'. We consider the function *assoc* which allows us to compute all coordinates associated to each coordinate of the current average sequence. For exemple,

$$assoc(\mathbf{c_i})$$

denotes the set of elements from X_l that are associated to $\mathbf{c_i}$ through DTW alignement. Then, the i^{th} coordinate of the new average sequence can be computed as:

$$\mathbf{c'_i} = barycenter(assoc(\mathbf{c_i}))$$

Remark: The initial average sequence has to be chosen carefully in order to accelerate the convergence. We will not provide the details here, more detailed analysis of this heuristic can be found in [10].

IV. Experiments

We used 3 different datasets (*Lighting2*, *synthetic_control* and *Trace*) from a standard database of time series called UCR (University of California, Riverside) Time Series Classification Archive [11]. We tried to run 7 clustering algorithms on each of these datasets.

The first clustering algorithm is the k-means using DTW as a distance and DBA as an averaging. The combination DTW/DBA is a standard, and will be used to determine the efficiency of the other methods. The other methods are the kernel k-means and the RKDE, each using three different kernels, described next.

A. Time series specific kernels

In this subsection, we will use the same notations as in the section III-B.

1) Gaussian Kernel: The DTW distance can be turned into a kernel, using the following formula:

$$\forall X, Y \; K(X,Y) = \exp\left(\frac{-DTW(X,Y)^2}{\sigma}\right)$$

As we have already mentioned in the previous section, DTW is not a distance as it does not satisfy the triangular inequality. Thus, the formula cited right above does not define a function respecting the properties of a kernel as it is not positive definite. As a result, DTW does not quantify dissimilarity in a meaningful way. Yet, running experiments with this function may be worthwhile, as DTW is still widely used to find the closest match(es) in a database given a time series of interest.

2) Global Alignment Kernel (GAK): We just explained that the DTW distance can not be used to define a kernel immediately. Yet, time-series specific kernels based on DTW exist. Global Alignment (GA) kernels [12], which are positive definite, seem to do a better job of quantifying all similarities coherently, because they take into account *all* possible alignments.

More precisely, the GA kernel is defined as the exponentiate *soft-minimum* of all alignment distances:

$$K_{GA}(X,Y) = \sum_{\pi \in \mathfrak{A}(n,m)} \exp^{-D_{X,Y}(\pi)}$$

Remark: The equation above can be rewritten using a local similarity function $k = e^{-\phi}$:

$$K_{GA}(X,Y) = \sum_{\pi \in \mathfrak{A}(n,m)} \prod_{i=1}^{|\pi|} k(\mathbf{x}_{\pi_1(i)}, \mathbf{y}_{\pi_2(i)})$$

The similarity described by this kernel incorporates the whole spectrum of costs $\{D_{X,Y}(\pi), \pi \in \mathfrak{A}(n,m)\}$. Thus, it provides a richer statistic than considering only the minimum of that set.

However, the computational effort required to compute K_{GA} is in $O(mn)$, similar to the DTW distance.

3) Triangular Global Alignment (TGA): Triangular Global Alignment (TGA) kernels are a variation of the GA kernel. They consider a smaller subset of alignments. They are faster to compute and positive definite, and can be seen as trade-off between the full GA kernel (accurate, versatile but slow) and a Gaussian kernel (fast but limited).

Indeed, exact DTW distances are expensive to compute for time series of dimension d since the complexity is in $O(dmn)$ at each evaluation. We can speed up this algorithm by only considering a small subset of all alignments [15]. We thus reformulate the cost function $D_{X,Y}(\pi)$ by weighting the divergence ϕ:

$$D^{\gamma}_{X,Y}(\pi) = \sum_{i=1}^{|\pi|} \gamma_{\mathbf{x}_{\pi_1(i)}, \mathbf{y}_{\pi_2(i)}} \phi(\mathbf{x}_{\pi_1(i)}, \mathbf{y}_{\pi_2(i)})$$

For instance, γ_k can be defined so as to ensure that only alignments that are close to the diagonal are taken into account [16]:

$$\gamma_{i,j} = \begin{cases} 1 & \text{if } |i-j| < T \\ \infty & \text{if } |i-j| \geq T \end{cases}$$

where T is a parameter to be determined called *Triangular*.

B. Implementations

We implemented the classic methods of clustering and DTW in python 2.7.6. We also created an interface which is generic enough so that we can adapt easily new kernels and methods to it. And for every new training dataset, we just have to compute the matrix of distances of this dataset, then pass it through our interface.

For GA kernels we used a python wrapper called TGA_python_wrapper [14] in which two parameters *Kernel bandwidth* and *Triangular* are required. In practice, when *Triangular* is set to 0, the routine returns the original GA kernel. When it is bigger than 1, the routine only takes into account alignments for which $-T < \pi_1(i) - \pi_2(i) < T$ for all indices of the alignment.

C. Parameter Tuning

The *Kernel bandwidth* σ can be set as a multiple of a simple estimate of the median distance of different points observed in different time-series of the training set, scaled by the square root of the median length of time-series in the training set, as hinted by the creator of these kernels.

For RKDE, we have to decide the *Cut-off probability* p. For the k-means algorithm, the number of clusters was known. We used this information to infer a good *Cut-off probability*. We used a dichotomic search to have the RKDE algorithm return the right number of clusters. We assumed the function considered (the number of clusters, given a random seed and p) was regular enough to perform this: when p is 0, there is only 1 cluster; when p is 1, there is 1 cluster per point; when in between, we assumed small variations on p entail small variations on output. These hypothesis turned reasonable for our experiments. Should a dichotomic search had failed, we would have ignored the corresponding clustering technique, as we do not have a systematic method to compare clustering with different number of groups.

Finally, for TGA, we have to decide the *Triangular* parameter T. This can be set to a reasonable multiple of the median length, e.g 0.2 or 0.5.

D. Evaluation of the Results

We choose the Cohen's kappa to evaluate our clustering.

Cohen's kappa coefficient is a statistic which measures inter-rater agreement between two raters who each classify several items into several mutually exclusive categories, thus seems to be an appropriate way to evaluate our results. It is generally thought to be a more robust measure than simple percent agreement calculation, since κ takes into account the agreement occurring by chance.

The κ is calculated as follow:

$$\kappa = \frac{p_0 - p_e}{1 - p_e}$$

where p_0 is the relative observed agreement among raters, and p_e is the hypothetical probability of chance agreement, using the observed data to calculate the probabilities of each observer randomly saying each category. If the raters are in complete agreement then $\kappa = 1$. If there is no agreement among the raters other than what would be expected by chance then $\kappa \leq 0$.

E. Results

In this subsection, we will first show some figures to illustrate our clustering. Then we will present the Cohen's kappa of each clustering algorithm and try to interpret them.

Fig. 2 shows the original classification of the dataset called *synthetic_control*. Each subplot of this figure represents one cluster of the dataset.

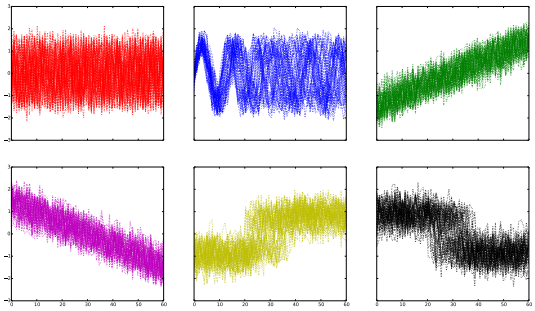

Fig. 2: Original.

And from Fig. 3 to Fig. 5 are the results of clustering obtained with the k-means using DTW/DBA, the k-means using GA kernels and the RKDE using GA kernels ran on this dataset. Visibly, the DTW/DBA has a better performance than other methods, and this can be confirmed by the Cohen's kappa coefficient showed next.

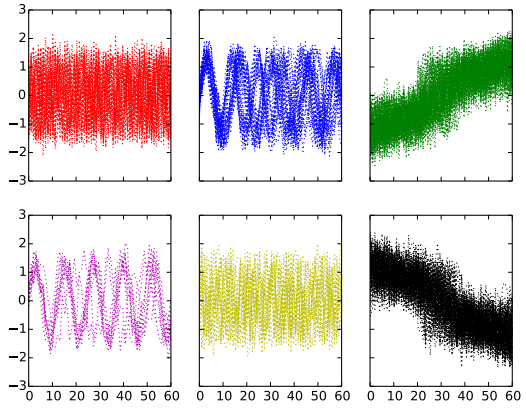

Fig. 3: DTW/DBA.

The Fig. 6 shows the Cohen's kappa of each clustering algorithm ran on the datasets *Lighting2*, *Trace* and *synthetic_control*.

For all the datasets we used, the k-means method using both DTW and DBA yields the best clustering. We will try to explain this result.

Having a proper averaging method may be the reason it achieved a better clustering than the others k-means.

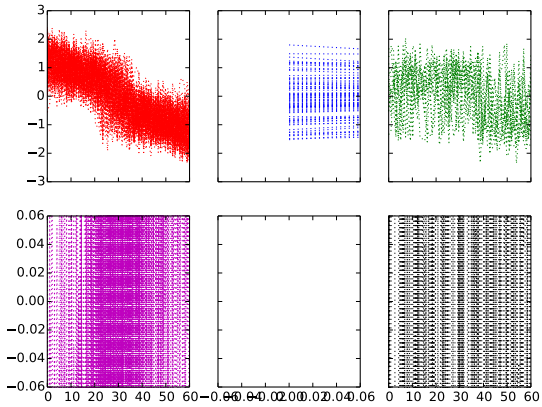

Fig. 4: k-means GA.

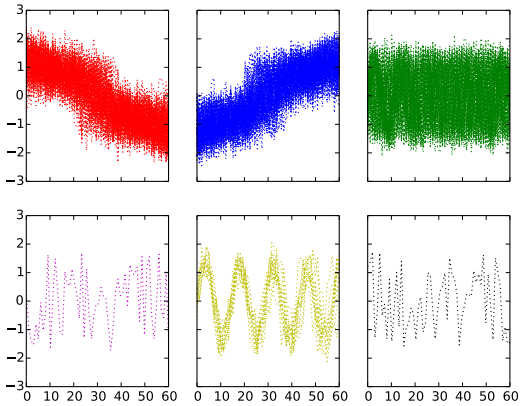

Fig. 5: RKDE GA.

As for the kernel used, they are based on DTW, so they must be adapted to the same kind of series.

The RKDE method has the advantage of guessing the right number of clusters, but it is of no use here. It can also theoritically work with any shape of clusters, but this didn't help too. Its results are similar to the k-means method using the same kernels, and no situation where it shines have been found.

The DTW/DBA seems to surpass similar methods. Yet, its success is still limited, as it is not adapted to all cases: nothing achieved good results for the *Lighting2* dataset, and other results are far from being perfect. Finding a new approach to comparing time series may be the next step toward fine clustering of time series, rather then optimizing the DTW/DBA method.

	Lighting2	Trace	synthetic control
RKDE (tga)	0.0294	0.399	0.428
RKDE (gak)	0.0656	0.385	0.436
RKDE (gau)	0.0526	0.399	0.216
k-means (tga)	0.0192	0.355	0.480
k-means (gak)	0.0667	0.426	0.440
k-means (gau)	0.0217	0.418	0.488
DTW/DBA	0.114	0.682	0.496

Fig. 6: Cohen's kappa

V. Conclusion

In this paper, we showed some classic methods and kernel methods of clustering and their applications to time series clustering. We ran several experiments based on these methods and tried to compare them using Cohen's kappa coefficient as the criteria for quality of our clustering. The DTW/DBA seems to be more convincing than other methods but still not very satisfying.

In the future work, there are two ways we may try to improve the results. The first one is try to develop some new kernels (instead of classic kernels like Gaussian kernels). Another possible way is try to apply some optimal transport methods to time series clustering as we can consider time series as "discrete" distributions in some aspects.

Acknowledgement

We are heartily thankful to our supervisor, Thomas Corpetti and Romain Tavenard, whose encouragement, guidance and support from the initial to the final level enabled us to develop an understanding of the subject.

And we offer our regards and blessings to all of those who supported us in any respect during the completion of the project.

References

[1] Bishop, C. M. (2006). Pattern recognition and machine learning. Springer.

[2] Filippone, M., Camastra, F., Masulli, F., & Rovetta, S. (2008). *A survey of kernel and spectral methods for clustering*. Pattern recognition, 41(1), 176-190.

[3] Rousseeuw, P. J. (1987). *Silhouettes: a graphical aid to the interpretation and validation of cluster analysis*. Journal of computational and applied mathematics, 20, 53-65.

[4] Müller, K. R., Mika, S., Rätsch, G., Tsuda, K., & Schölkopf, B. (2001). *An introduction to kernel-based learning algorithms*. Neural Networks, IEEE Transactions on, 12(2), 181-201.

[5] Liao, T. W. (2005). *Clustering of time series data—a survey*. Pattern recognition, 38(11), 1857-1874.

[6] Sheather, Simon J., & Michael C. Jones. *A reliable data-based bandwidth selection method for kernel density estimation*. Journal of the Royal Statistical Society. Series B (Methodological) (1991): 683-690.

[7] Turlach, B. A. *Bandwidth selection in kernel density estimation: A review*. Université catholique de Louvain, 1993.

[8] Kim, J. and Scott, C. D. (2012). *Robust kernel density estimation*. The Journal of Machine Learning Research, 13(1), 2529-2565.

[9] Huber, Peter J. (1964). *Robust estimation of a location parameter*. The Annals of Mathematical Statistics 35.1 (1964): 73-101.

[10] Petitjean, F., Ketterlin, A., & Gançarski, P. (2011). *A global averaging method for dynamic time warping, with applications to clustering*. Pattern recognition, 44(3), 678-693.

[11] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen and Gustavo Batista (2015). *The UCR Time Series Classification Archive*. www.cs.ucr.edu/~eamonn/time_series_data/

[12] Cuturi, M., Vert, J. P., Birkenes, O. and Matsui, T. (2007, April). *A kernel for time series based on global alignments*. In Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on (Vol. 2, pp. II-413). IEEE.

[13] Cuturi, M. (2011). *Fast global alignment kernels*. In Proceedings of the 28th international conference on machine learning (ICML-11) (pp. 929-936).

[14] Adrien Gaidon http://www.iip.ist.i.kyoto-u.ac.jp/member/cuturi/Code/TGA_python_wrapper_v1.0.tar.gz

[15] Rabiner, L. and Juang, B. H. (1993). *Fundamentals of speech recognition*.

[16] Sakoe, H. and Chiba, S. (1978). *Dynamic programming algorithm optimization for spoken word recognition*. Acoustics, Speech and Signal Processing, IEEE Transactions on, 26(1), 43-49.