Efficient similarity-based data clustering by optimal object to cluster reallocation

**Mathias Rossignol · Mathieu Lagrange ·
Arshia Cont**

Abstract We present an iterative flat hard clustering algorithm designed to operate on arbitrary similarity matrices, with the only constraint that these matrices be symmetrical. Although functionally very close to kernel k-means, our proposal performs a maximization of average intra-class similarity, instead of a squared distance minimization, in order to remain closer to the semantics of similarities. We show that this approach permits the relaxing of some conditions on usable affinity matrices like semi-positiveness, as well as opening possibilities for computational optimization required for large datasets. Systematic evaluation on a variety of data sets shows that compared with kernel k-means and the spectral clustering methods, the proposed approach gives equivalent or better performance, while running much faster. Most notably, it significantly reduces memory access, which makes it a good choice for large data collections. Material enabling the reproducibility of the results is made available online.

Keywords clustering, similarity, non-metric data representation, time series clustering

Mathias Rossignol
IRCAM
1, place Igor Stravinski
Paris, France
E-mail: mathias.rossignol@gmail.com

Mathieu Lagrange
LS2N
Nantes, France
E-mail: lagrange.mathieu@cnrs.fr

Arshia Cont
IRCAM
1, place Igor Stravinski
Paris, France E-mail: arshia.cont@ircam.fr

1 Introduction

Clustering collections of objects into classes that bring together similar ones is probably the most common and intuitive tool used both by human cognition and artificial data analysis in an attempt to make that data organized, understandable, manageable. When the studied objects lend themselves to this kind of analysis, it is a powerful way to expose underlying organizations and approximate the data in such a way that the relationships between its members can be statistically understood and modeled. Given a description of objects, we first attempt to quantify which ones are "similar" from a given point of view, then group those n objects into C clusters, so that the similarity between objects within the same cluster is maximized. Finding the actual best possible partition of objects into clusters is, however, an NP-complete problem, intractable for useful datasets sizes. Many approaches have been proposed to yield an approximate solution: analytic, iterative, flat or hierarchical, agglomerative or divisive, soft or hard clustering algorithms, *etc.*, each with their strengths and weaknesses (Jain 2010), performing better on some classes of problems than others (Steinbach et al. 2000; Thalamuthu et al. 2006).

Iterative divisive hard clustering algorithms, usually perform well to identify high-level organization in large data collections in reasonable running time. For that reason, they are a sensible choice in many data mining situations, and constitute our focus in this paper. If the data lies in a vector space, *i.e.* an object can be described by a m-dimensional feature vector without significant loss of information, the seminal *k-means* algorithm (MacQueen 1967) is probably the most efficient approach, since the explicit computation of the cluster centroids ensure both computational efficiency and scalability. This algorithm is based on the centroid model, and minimizes the intra cluster Euclidean distance. As shown by Banerjee et al. (2005), any kind of Bregman divergence, such as the KL-divergence (Dhillon et al. 2003) or the Itakura-Saito divergence (Linde et al. 1980), may also be considered to develop such efficient clustering algorithms.

However, for many types of data, the projection of a representational problem into an vector space cannot be done without significant loss of descriptive efficiency. To reduce this loss, specifically tailored measures of similarity are considered. As a result, the input data for clustering is no longer a $n \times m$ matrix storing the m-dimensional vectors describing the objects, but a (usually symmetric) square matrix S of size $n \times n$ which numerically encodes some sort of relationship between the objects. In this case, one has to resort to clustering algorithms based on connectivity models, since the cluster centroids cannot be explicitly computed.

Early attempts to solve this issue considered the k-medoids problem, where the goal is to find the k objects that maximize the average similarity with the

other objects of their respective clusters, or *medoids*. The Partition Around Medoids (PAM) algorithm (Kaufman and Rousseeuw 1990) solves the k-medoids problem but with a complexity of $O(k(n-k)^2)i$, n being the number of objects and i number of iterations. Due to the high complexity and the low convergence rate, this algorithm cannot be applied to decent size datasets. In order to scale the approach, the Clustering LARge Applications (CLARA) algorithm (Kaufman and Rousseeuw 1990) draws a sample of objects before running the PAM algorithm. This sampling operation is repeated several times and the most satisfying set of medoids is retained. In contrast, CLARANS (Ng and Han 1994) preserves the whole set of objects but cuts complexity by drawing a sample of neighbors in each search for the medoids.

Another classical approach to the issue is to run a variant of k-means that considers average intra-cluster similarity as the guiding criterion for object-to-class reallocation (Duda et al. 2001, Chapter 10.7). This straightforward technique performs well in many cases, but the complexity of computing anew intra-cluster similarities at each iteration makes it impractical for large datasets.

Following work on kernel projection (Vapnik 1995), that is, the fact that a nonlinear data transformation into some high dimensional feature space increases the probability of the linear separability of patterns within the transformed space, Girolami (2002) introduced a kernel version of the K-means algorithm, whose input is a kernel matrix $\mathcal{K}$ that must be a Gram matrix, *i.e.* semi definite positive. Dhillon et al. (2007) linked a weighted version of the kernel K-means objective to the popular spectral clustering Von Luxburg (2007), introducing an efficient way of solving the normalized cut objective Shi and Malik (2000).

The kernel k-means algorithm proves to be equally useful when considering arbitrary similarity problems if special care is taken to ensure definite positiveness of the input matrix (Roth et al. 2003). This follows original algorithmic considerations where vector space data is projected into high dimensional spaces using a carefully chosen kernel function.

Despite such improvements, kernel k-means cannot be easily applied to large scale datasets without special treatments because of high algorithmic and memory access costs. Chitta et al. (2011) considered sampling of the input data, Zhang and Rudnicky (2002) considered block storing of the input matrix, and a pre-clustering approach Bradley et al. (1998) is considered by Kulis et al. (2008) with a coarsening and refining phases as respectively a pre- and post-treatment of the actual clustering phase.

We show in this paper that by using a semantically equivalent variant of the average intra-cluster similarity presented in (Duda et al. 2001, Chapter 10.7), it becomes possible to perform a computationally efficient greedy optimization (Duda et al. 2001, Chapter 10.8) with guaranteed convergence for arbitrary similarity measures. Although both of those techniques are well known, combining them is non trivial, and lead to a clustering algorithm, which we call k-averages with the following properties:

– input data can be arbitrary symmetric similarity matrices,
– it has fast and guaranteed convergence, with a number of object to clusters reallocations experimentally found to be roughly equal to the number of objects,
– it provides good scalability thanks to a reduced need for memory access, and
– on a collection of synthetic and natural test data, its results are equivalent to those of kernel k-means, and obtained in a fraction of its computing time, particularly when paged memory is required.

To summarize, the main contribution of the paper is to present a clustering algorithm:

– that can handle arbitrary affinity matrices, *i.e.* semi-positiveness is not a mandatory requirement for guaranteed convergence
– thanks to a carefully designed strategy to update the membership of objects to cluster, the algorithm is fast and memory efficient.

The remaining of the paper is organized as follows: Section 2 presents the kernel k-means objective function and the basic algorithm that minimizes this function, and Section 3 introduces the concepts behind the k-averages algorithm, followed by a detailed algorithmic description in Section 4. The complexity of the two algorithms in terms of arithmetic operations and memory access is then studied in Section 6. The above presented properties of the proposed k-averages algorithm are then validated on synthetic controlled data in Section 7 and on 43 datasets of time series issued from various sources in Section 8.

2 Kernel k-means

Since its introduction by Girolami (2002), kernel k-means has been an algorithm of choice for flat data clustering with known number of clusters (Kulis et al. 2008; Roth et al. 2003). It makes use of a mathematical technique known as the "kernel trick" to extend the classical k-means clustering algorithm (MacQueen 1967) to criteria beyond simple euclidean distance proximity. Since it constitutes the closest point of comparison with our own work, we dedicate this section to its detailed presentation.

In the case of kernel k-means, the kernel trick allows us to consider that the k-means algorithm is operating in an unspecified, possibly very high-dimensional Euclidean space; but instead of specifying the properties of that space and the coordinates of objects, the equations governing the algorithm are modified so that everything can be computed knowing only the scalar products between points. The symmetrical matrix containing those scalar products is known as a kernel, noted $\mathcal{K}$.

2.1 Kernel k-means objective function

In this section and the following, we shall adopt the following convention: N is the number of objects to cluster and C the number of clusters; N_c is the number of objects in cluster c, and μ_c is the centroid of that cluster. z_{cn} is the membership function, whose value is 1 if object o_n is in class c, 0 otherwise. In the folowing equations, μ_c and o_n are assumed to be column vectors of equal dimension.

Starting from the objective function minimized by the k-means algorithm, expressing the sum of squared distances of points to the centroids of their respective clusters:

$$S = \sum_{c=1}^{C} \sum_{n=1}^{N} z_{cn} \left(o_n - \mu_c \right)^{\top} \left(o_n - \mu_c \right)$$

And using the definition of centroids as:

$$\mu_c = \frac{1}{N_c} \sum_{n=1}^{N} z_{cn} o_n$$

S can be developed and rewritten in a way that does not explicitly refer to the centroid positions, since those cannot be computed:

$$S = \sum_{c=1}^{C} \sum_{n=1}^{N} z_{cn} Y_{cn}$$

where

$$
\begin{aligned}
Y_{cn} &= \left(o_n - \mu_c \right)^{\top} \left(o_n - \mu_c \right) \\
&= o_n o_n - 2 o_n^{\top} \mu_c + \mu_c^{\top} \mu_c \\
&= o_n^{\top} o_n - 2 o_n^{\top} \frac{1}{N_c} \sum_{i=1}^{N} z_{ci} o_i + \left(\frac{1}{N_c} \sum_{i=1}^{N} z_{ci} o_i \right)^{\top} \left(\frac{1}{N_c} \sum_{i=1}^{N} z_{ci} o_i \right) \\
&= o_n^{\top} o_n - \frac{2}{N_c} \sum_{i=1}^{N} z_{ci} o_n^{\top} o_i + \frac{1}{N_c^2} \sum_{i=1}^{N} \sum_{j=1}^{N} z_{ki} z_{kj} o_i^{\top} o_j \\
&= \mathcal{K}_{nn} - \frac{2}{N_c} \sum_{i=1}^{N} z_{ci} \mathcal{K}_{ni} + \frac{1}{N_c^2} \sum_{i=1}^{N} \sum_{j=1}^{N} z_{ki} z_{kj} \mathcal{K}_{ij}
\end{aligned}
\tag{1}
$$

Since the sum of K_{nn} over all points remains constant, and the sum of squared centroid norms (third, quadratic, term of Equation 1) is mostly bounded by the general geometry of the cloud of objects, we can see that minimizing this value implies maximizing the sum of the central terms, which are the average scalar products of points with other points belonging to the same class. Therefore, given a matrix gathering similarities between objects, if that matrix possesses the necessary properties to be considered as a kernel

(positive semidefinitness), then the kernel k-means algorithm can be applied to it in order to create clusters that locally maximize the average intra-cluster similarity.

2.2 Algorithm

Finding the configuration that globally minimizes S (eq. 2.1) is an NP-complete problem. However, several approaches allow finding an acceptable approximation. We shall only focus here on the fastest and most popular, an iterative assignment / update procedure commonly referred to as the "k-means algorithm" (MacQueen 1967), or as a discrete version of Lloyd's algorithm, detailed in Algorithm 1.

Data: number of objects N, number of classes C, kernel matrix $\mathcal{K}$
Result: label vector L defining a partition of the objects into C classes

1 **Initialization:** fill L with random values in $[1..C]$;

2 **while** L *is modified* **do**
3   **for** $n \leftarrow 1$ *to* N **do**
4   **for** $c \leftarrow 1$ *to* C **do**
5   Compute Y_{cn} following eq. 1 (note: $z_{cn} = (L_n == c)\,?\,1\,:\,0$)
6   **end**
7         $L_n = \operatorname{argmin}_c(Y_{cn})$;
8   **end**
9 **end**

Algorithm 1: Lloyd's algorithm applied to minimizing the kernel k-means objective.

The version given here is the most direct algorithmic translation of the mathematical foundations developed above, and as we shall see in section 6, it can easily become more efficient. Before that, we introduce our proposed k-averages algorithm.

3 Foundations of the k-averages algorithm

In our proposal, we adopt an alternative objective function which, unlike kernel k-means, does not rely on a geometric interpretation but an explicit account of the similarity matrix. The goal is to maximize the average intra-cluster similarity between points, a commonly used metric to evaluate clustering quality, and one whose computation is direct—linear in time.

Due to its simplicity, however, the objective function cannot be simply "plugged into" the standard kernel k-means algorithm: it lacks the geometric requisites to ensure convergence. We must therefore propose a specifically tailored algorithmic framework to exploit it: first, we show here that it is possible to easily compute the impact on the global objective function of moving

a single point from one class to another; this allows us to develop a greedy optimization algorithm taking advantage of that formula.

3.1 Conventions and objective function

In addition to the notations presented above, we index here the set of elements belonging to a given cluster c_k as $c_k = \{o_{k1}, \ldots, o_{kN_k}\}$. For simplicity, we omit the first index and note $c = \{o_1, \ldots, o_{N_c}\}$ when considering a single class.

The similarity between objects shall be written $s(o_i, o_j)$. We extend the notation s to the *similarity of an object to a class* defined as the average similarity of an object with all objects of the class. $s(o, c)$ accepts two definitions, depending on whether or not o is a member of c:

If $o \notin c$,

$$s(o, c) = \frac{1}{N_c} \sum_{i=1}^{n_c} s(o, o_i) \tag{2}$$

If $o \in c$, then necessarily $\exists i \mid o = o_i$

$$s(o, c) = s(o_i, c) = \frac{1}{N_c - 1} \sum_{j=1\ldots n_c, j \neq i} s(o_i, o_j) \tag{3}$$

Let us call the "quality" of a class the average intra-class object-to-object similarity, and write it $\mathcal{Q}$:

$$\mathcal{Q}(c) = \frac{1}{N_c} \sum_{i=1}^{n_c} s(o_i, c)$$

$$\mathcal{Q}(c) = \frac{1}{N_c} \sum_{i=1}^{n_c} s(o_i, c) \tag{4}$$

In our framework, we do not explicitly refer to class centroids, preferring to directly consider averages of similarity values between individuals within clusters. Using the notations above, we define our objective function as the average class quality, normalized with class sizes:

$$O_2 = \frac{1}{N} \sum_{i=1}^{C} N_i \mathcal{Q}(c_i)$$

$$\mathcal{O} = \frac{1}{N} \sum_{i=1}^{C} N_i \mathcal{Q}(c_i) \tag{5}$$

Since, informally, our goal is to bring together objects that share high similarity, a first idea would be to simply repeatedly move each object to the class with whose members it has the highest average similarity. This is what we call the "naive k-averages" algorithm.

3.2 Naive k-averages algorithm

Algorithm 2 presents a method that simply moves each object to the class
with which it has the highest average similarity, until convergence is reached.
The algorithm is straightforward and simple; however, experiments show that
while it can often produce interesting results, it also sometimes cannot reach
convergence because the decision to move an object to a different cluster is
taken without considering the impact of the move on the quality of the source
cluster.

Data: number of objects N, number of classes C, similarity matrix $\mathcal{S}$
Result: label vector L defining a partition of the objects into C classes

1 **Initialization:** Fill L with random values in $[1..C]$;
2 Compute initial object-class similarities S following eq. 3 or eq. 2;
3 **while** *L is modified* **do**
4   **for** $i \leftarrow 1$ *to* N **do**
5   previousClass $\leftarrow L_i$;
6   nextClass $\leftarrow \operatorname{argmin}_k S(i,k)$ **if** *nextClass* $\neq$ *previousClass* **then**
7 $L_i \leftarrow$ nextClass;
8   **for** $j \leftarrow 1$ *to* N **do**
9   Update $S(j, nextClass)$ and $S(j, previousClass)$
10   **end**
11   **end**
12   **end**
13 **end**

Algorithm 2: The naive k-averages algorithm.

To ensure convergence, we need to compute the impact on the objective
function of moving one object from one class to another. Using such formula-
tion and performing only reallocation that have a positive impact, the conver-
gence of such an iterative algorithm is guaranteed.

3.3 Impact of object reallocation on class quality

Considering a class c, let us develop the expression of $\mathcal{Q}(c)$ into a more useful
form. Since all objects are in c, we use the formula in (eq. 3) to get:

$$\mathcal{Q}(c) = \frac{1}{N_c} \sum_{i=1}^{N_c} \frac{1}{N_c - 1} \sum_{\substack{j=1\dots N_c \\ j \neq i}} s(o_i, o_j)$$

$$= \frac{1}{N_c(N_c - 1)} \sum_{i=1}^{N_c} \sum_{\substack{j=1\dots N_c \\ j \neq i}} s(o_i, o_j)$$

Using the assumption that the similarity matrix is symmetrical, we can reach:

$$\mathcal{Q}(c) = \frac{2}{N_c(N_c - 1)} \sum_{i=2}^{N_c} \sum_{j=1}^{i-1} s(o_i, o_j) \tag{6}$$

For future use and given the importance of the above transformation, we define:

$$\Sigma(c) = \sum_{i=2}^{N_c} \sum_{j=1}^{i-1} s(o_i, o_j)$$

Thus:

$$\mathcal{Q}(c) = \frac{2}{N_c(N_c - 1)} \Sigma(c) \qquad \text{and} \qquad \Sigma(c) = \frac{N_c(N_c - 1)\mathcal{Q}(c)}{2}$$

3.3.1 Removing an object from a class

Assuming that $o \in c$, necessarily $\exists i \mid o = o_i$. Since the numbering of objects is arbitrary, we can first simplify the following equation by considering that $o = o_{N_c}$, in order to reach a formula that is independent from that numbering.

$$\mathcal{Q}(c \smallsetminus o_{N_c}) = \frac{2}{(N_c - 1)(N_c - 2)} \sum_{i=2}^{N_c-1} \sum_{j=1}^{i-1} s(o_i, o_j)$$

$$= \frac{2}{(N_c - 1)(N_c - 2)} \left[\Sigma(c) - \sum_{j=1}^{N_c-1} s(o_{N_c}, o_j) \right]$$

$$= \frac{2}{(N_c - 1)(N_c - 2)} \left[\Sigma(c) - (N_c - 1)s(o_{N_c}, c) \right]$$

$$= \frac{2 N_c(N_c - 1)\mathcal{Q}(c)}{2(N_c - 1)(N_c - 2)} - \frac{2(N_c - 1)s(o_{N_c}, c)}{(N_c - 1)(N_c - 2)}$$

$$= \frac{N_c \mathcal{Q}(c) - 2s(o_{N_c}, c)}{N_c - 2}$$

The quality of a class after removal of an object is thus:

$$\mathcal{Q}(c \smallsetminus o) = \frac{N_c \mathcal{Q}(c) - 2s(o, c)}{N_c - 2} \tag{7}$$

And the change in quality from its previous value:

$$\mathcal{Q}(c \smallsetminus o) - \mathcal{Q}(c) = \frac{N_c \mathcal{Q}(c) - (N_c - 2)\mathcal{Q}(c) - 2s(o, c)}{N_c - 2}$$

$$= \frac{2(\mathcal{Q}(c) - s(o, c))}{N_c - 2} \tag{8}$$

3.3.2 Adding an object to a class

Assuming that $o \notin c$, we can similarly to what has been done previously (numbering is arbitrary) consider for the sake of simplicity that o becomes o_{N_c+1} in the modified class c. Following a path similar to above, we get:

$$
\begin{aligned}
\mathcal{Q}(c \cup o_{N_c+1}) &= \frac{2}{N_c(N_c+1)} \sum_{i=2}^{N_c+1} \sum_{j=1}^{i-1} s\left(o_i, o_j\right) \\
&= \frac{2}{N_c(N_c+1)} \left[\Sigma(c) + N_c s\left(o_{N_c+1}, c\right) \right] \\
&= \frac{(N_c-1)\mathcal{Q}(c) + 2s\left(o_{N_c+1}, c\right)}{N_c+1}
\end{aligned}
$$

The quality of a class c after adding an object o is thus:

$$
\mathcal{Q}\left(c \cup o\right) = \frac{(N_c-1)\mathcal{Q}(c) + 2s\left(o, c\right)}{N_c+1} \tag{9}
$$

And the change in quality from its previous value:

$$
\mathcal{Q}\left(c \cup o\right) - \mathcal{Q}\left(c\right) = \frac{2\left(s\left(o, c\right) - \mathcal{Q}(c)\right)}{N_c+1} \tag{10}
$$

3.4 Impact of object reallocation on the global objective function

When moving an object o from class c_s ("source"), to whom it belongs, to a distinct class c_t ("target"), $(N_s - 1)$ objects are affected by the variation in (8), and N_t are affected by that in (10); in addition, one object o moves from a class whose quality is $\mathcal{Q}(c_s)$ to one whose quality is $\mathcal{Q}\left(c_t \cup o\right)$, as expressed by ~~equation~~eq. 9, which leads to an impact of moving object o from class c_s to class c_t wich can be computed as follows:

$$
\begin{aligned}
\delta_o(c_s, c_t) =& \frac{2N_t\left(s\left(o, c_t\right) - \mathcal{Q}(c_t)\right)}{N_t+1} + \\
& \frac{2(N_s-1)\left(\mathcal{Q}(c_s) - s\left(o, c_s\right)\right)}{N_s-2} + \\
& \frac{(N_t-1)\mathcal{Q}(c_t) + 2s\left(o, c_t\right)}{N_t+1} - \mathcal{Q}(c_s)
\end{aligned} \tag{11}
$$

As can be seen, computing this impact is a fixed-cost operation. We can therefore use the formula as the basis for an efficient iterative algorithm.

Data: number of objects N, number of classes C, similarity matrix $\mathcal{S}$
Result: label vector L defining a partition of the objects into C classes

1 Initialization: Fill L with random values in $[1..C]$; Compute initial object-class similarities S following eq. 3 or eq. 2; Compute initial class qualities $\mathcal{Q}$ following eq. 6;

2 while L *is modified* **do**
3   **for** $i \leftarrow 1$ *to* N **do**
4   previousClass $\leftarrow L_i$; nextClass $\leftarrow \text{argmax}_k \, \delta_i(\text{previousClass}, k)$ (following the definition of δ in eq. 11); **if** $nextClass \neq previousClass$ **then**
5 $L_i \leftarrow$ nextClass; Update $\mathcal{Q}_{\text{previousClass}}$ following eq. 7; Update $\mathcal{Q}_{\text{nextClass}}$ following eq. 9; **for** $j \leftarrow 1$ *to* N **do**
6   Update $S(j, nextClass)$ and $S(j, previousClass)$
7   following eq. 12;
8   **end**
9   **end**
10   **end**
11 end

Algorithm 3: The K-averages algorithm.

4 K-averages algorithm

Our approach does not allow us to benefit, like kernel k-means, from the convergence guarantee brought by the geometric foundation of k-means. In consequence, we cannot apply a "batch" approach where at each iteration all elements are moved to their new class, and all distances (or similarities) are computed at once. To guarantee convergence, we must update the class properties for the two modified classes (source and destination), as well as recompute the average class-object similarities for them for each considered object, after finding its ideal new class. This is the principle of the "greedy" k-means algorithm (Duda et al. 2001, Chapter10.8), but whereas for k-means that approach increases complexity (and even more so for kernel k-means), in this case it leads to a much improved computational performance.

Indeed, at a first glance, dynamically updating objectives as a result of object reallocation might seem to have negative performance impact. However, our simple non-quadratic updates make such dynamic changes easily tractable. New class qualities are thus given by ~~Equations~~eqs. 7 and 9, and new object-class similarities can be computed by:

$$s(i, c_s(t+1)) = \frac{N_s(t).s(i, c_s(t)) + s(i, n)}{N_s(t) + 1}$$
$$s(i, c_t(t+1)) = \frac{N_t(t).s(i, c_s(t)) - s(i, n)}{N_t(t) - 1} \tag{12}$$

where i is any object index, n is the recently reallocated object, c_s the "source" class that object i was removed from, and c_t the "target" class that object n was added to.

The full description of k-averages is given in Algorithm 3.

Initialization: Fill L with random values in [1..C] Compute initial object-class similarities S following eq 3 or eq 2 Compute initial class qualities

5 Convergence

The kernel k-means algorithm ensures convergence if the similarity matrix is semi-definite positive. The k-averages algorithm relaxes this constraint by only requiring symmetricity of the similarity matrix to ensure convergence.

An algorithm is guaranteed to converge if its successive iterations can be tied to a strictly monotonous and bounded quantity. For the k-averages algorithm, this quantity is the objective function itself, as we now show.

Thanks to the rewriting of the class quality function done in eq. 6, which only requires the similarity matrix to be symmetrical, we can directly define the allocation decision criterion δ (eq. 11) to be the change in the global objective function $\mathcal{O}$ (eq. 5) implied by reallocating an object to a new class. It follows that, as long as reallocations are only performed when $\delta > 0$, $\mathcal{O}$ is strictly increasing throughout the execution of the algorithm.

Moreover, $\mathcal{O}$, defined as a weighted average of the average intra-class similarities for the produced clusters, can be proven to never exceed the maximal similarity between two objects.

Indeed, the average similarity of an object to other members of its class, expressed as $s(o, c)$ in eq. 2, is an average of similarities, and therefore lower than or equal to their maximum value. Similarly, the quality $\mathcal{Q}$ following eq 6 L is modified$i \leftarrow 1$ to NpreviousClass $\leftarrow L_i$ nextClass $\leftarrow \text{argmin}_k \delta_i(\text{previousClass}, k)$ (following the definition of δ in 11) nextClass $\neq$ previousClass$L_i \leftarrow$ nextClass Update $\mathcal{Q}_{\text{previousClass}}$ following eq 7 Update $\mathcal{Q}_{\text{nextClass}}$ following eq 9 $j \leftarrow 1$ to NUpdate $S(j, nextClass)$ and $S(j, previousClass)$ following eq 12 The K-averages algorithm. of a class, defined as the average of object to class similarities (eq. 4), is inferior to the maximum value of $s(o, c)$, and therefore to the maximum similarity between objects. Finally, $\mathcal{O}$, a weighted average of $\mathcal{Q}$ values, is, once again, inferior to their maximum value.

The objective function $\mathcal{O}$ is thus upper-bounded, and increases at each iteration of the outer loop of Algorithm 3; which guarantees its convergence.

6 Complexity analysis

In this section, we study the complexity of the two approaches presented above, first from the point of view of raw complexity, second by focusing on memory access.

6.1 Computational complexity

6.1.1 Kernel k-means

As can be seen in Algorithm 1, the operation on line 5 is the most costly part of the algorithm: for each object n and class c, at each iteration, it is necessary to compute Y_{cn} from ~~Equation~~eq. 1—an $O(N^2)$ operation in itself, per object. The impossibility of simply computing the distances to a known centroid as done in the k-means algorithm leads to a much higher complexity for the kernel k-means algorithm, globally $O(N^3)$ per iteration, independent of how many objects are moved for that iteration.

It is however possible to improve the performance of kernel k-means by noting than in ~~Equation~~eq. 1, the third term of the equation, which has the highest complexity, is only dependent on class definitions and not on the considered object. We can therefore rewrite ~~Equation~~eq. 1 as:

$$Y_{cn} = \mathcal{K}_{nn} - \frac{2}{N_c} \sum_{i=1}^{N} z_{ci} \mathcal{K}_{ni} + M_c \tag{13}$$

where

$$M_c = \frac{1}{N_c^2} \sum_{i=1}^{N} \sum_{j=1}^{N} z_{ki} z_{kj} \mathcal{K}_{ij} \tag{14}$$

Algorithm 1 thus becomes Algorithm 4, where the values of M_c are computed once at the beginning of each loop (line 4) then reused on line 8, thus reducing the overall complexity to $O(n^2)$ per iteration. This optimized version of kernel k-means is the one we shall consider for performance comparison in the remainder of this article.

Data: number of objects N, number of classes C, kernel matrix $\mathcal{K}$
Result: label vector L defining a partition of the objects into C classes

1 **Initialization:** fill L with random values in $[1..C]$;
2 **while** *L is modified* **do**
3   **for** $c \leftarrow 1$ *to* C **do**
4   Compute M_c following eq. 14
5   **end**
6   **for** $n \leftarrow 1$ *to* N **do**
7   **for** $c \leftarrow 1$ *to* C **do**
8   Compute Y_{cn} following eq. 13 (note: $z_{cn} = (L_n == c)\,?\,1\,:\,0$)
9   **end**
10 $L_n = \mathrm{argmin}_c(Y_{cn})$;
11   **end**
12 **end**

Algorithm 4: Lloyd's algorithm applied to minimizing the kernel k-means objective, optimized version.
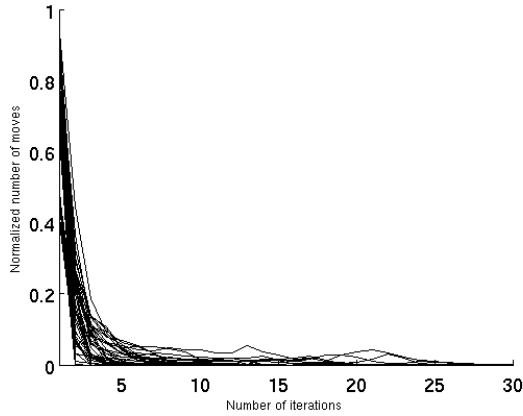
Fig. 1 Number of moved objects per iteration when clustering a variety of datasets with the k-averages algorithm, normalized by the total number of objects to cluster. The datasets used to create this figure are the real-life time series data that we employ for experimental validation evaluated under the Dynamic Time Warping (DTW) similarity measure, *cf.* Section 8.

6.1.2 K-averages

For the k-averages method presented as Algorithm 3, the complexity of each iteration is

– $O(NC)$ corresponding to the best class search at line 4
– $O(NM)$ corresponding to the object-to-class similarity update at line 6, where M is the number of objects moved at a given iteration.

In the worst case scenario, $M = N$, and the complexity for one iteration of the algorithm remains the same as for the optimized kernel k-means algorithm, $O(N^2)$. In practice, however, as can be seen on Figure 1, the number of objects moving from one class to another decreases sharply after the first iteration, meaning that the complexity of one iteration becomes quickly much lower than $O(N^2)$. Thus, while the first iteration of k-averages has a similar complexity with kernel k-means, the overall cost of a typical run of the algorithm (from 10 to 50 iterations) is much lower.

To go further in this analysis, we display on Figure 2 the total number of object reallocation over a full run of the k-averages algorithm for several datatsets. The datasets used to create this figure are the real-life time series data that we employ for experimental validation evaluated under the Dynamic Time Warping (DTW) similarity measure, *cf.* Section 8. As can be seen, the correlation is roughly linear with the number of objects to cluster. In fact, the number of reallocations is roughly equal to the number of objects to cluster, which allows us to reach for k-averages a (statistical) total complexity of $O(N^2)$, instead of $O(N^2)$ per iteration.
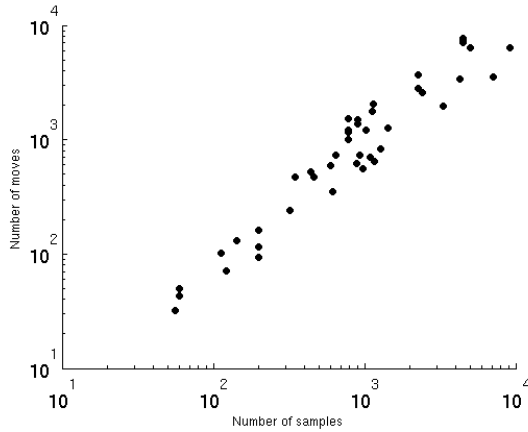
Fig. 2 Total number of object reallocations over a run of the k-averages algorithm, plotted against the number of objects to be clustered. The datasets used to create this figure are the real-life time series data that we employ for experimental validation, *cf.* Section 8.

6.2 Memory access

The lowered computational costs is also accompanied by a decrease in memory access: as can be seen from Equation 12, in order to compute the new object-to-class similarities after moving an object n, only line n of the similarity matrix needs to be read. For the remaining of the algorithm, only the (much smaller) object-to-class similarity matrix is used. By contrast, in the case of kernel k-means, the computation of M_c values at each iteration require that the whole similarity matrix be read, which can be a serious performance bottleneck in the case of large object collections.

Moreover, the similarity update function of k-averages, by reading one line of the matrix at a time, presents good data locality properties, which make it play well with standard memory paging strategies.

To illustrate and confirm the theoretical complexity computed here, the next section proposes some performance figures measured on controlled datasets.

7 Validation

In order to reliably compare the clustering quality and execution speed between the two approaches, we have written plain C implementations of Algorithms 4 and 3, with minimal operational overhead: reading the similarity matrix from a binary file where all matrix values are stored sequentially in standard reading order, line by line, and writing out the result of the clustering as a label text file. Both implementations use reasonably efficient code, but without advanced optimizations or parallel processing.

The figures presented in this section were obtained on synthetic datasets, created in order to give precise control on the features of the analyzed data: for

n points split between C classes, C centroids are generated at random in two dimensional space, and point coordinates are generated following a Gaussian distribution around class centroids. In addition to the numbers of objects and classes, the variance of Gaussian distributions are adjusted to modulate how clearly separable clusters are. Similarities are computed as inverse Euclidean distances between points.

7.1 Reproducibility

In order to ease reproducibility of the results, the data is taken from a public repository of several benchmark datasets used in academia Chen et al. (2015) and the code of the proposed method as well as the experimental code used for generated the figures is publicly available[1].

7.2 Clustering performance

Several metrics are available to evaluate the performance of a clustering algorithm. The one closest to the actual target application is the raw accuracy, that is the average number of items labeled correctly after an alignment phase of the estimated labeling with the reference (Kuhn 1955).

Another metric of choice is the Normalized Mutual Information (NMI) criterion. Based on information theoretic principles, it measures the amount of statistical information shared by the random variables representing the predicted cluster distribution and the reference class distribution of the data points. If P is the random variable denoting the cluster assignments of the points, and C is the random variable denoting the underlying class labels on the points then the NMI measure is defined as:

$$\mathbf{NMI} = \frac{2I(C; K)}{H(C) + H(K)}$$

where $I(X; Y) = H(X)H(X|Y)$ is the mutual information between the random variables X and Y, $H(X)$ is the Shannon entropy of X, and $H(X|Y)$ is the conditional entropy of X given Y. Thanks to the normalization, the metric stays between 0 and 1, 1 indicating a perfect match, and can be used to compare clustering with different numbers of clusters. Interestingly, random prediction gives an NMI close to 0, whereas the accuracy of a random prediction on a balanced bi-class problem is as high as 50 %.

In this paper, for simplicity sake, only the NMI is considered for validations. However, we found that most statements hereafter in terms of performance ranking of the different algorithms still hold while considering the accuracy metric as reference.

Figure 3 presents the quality of clusterings obtained using kernel k-means and k-averages on two series of datasets: one featuring 5 classes, the other

---

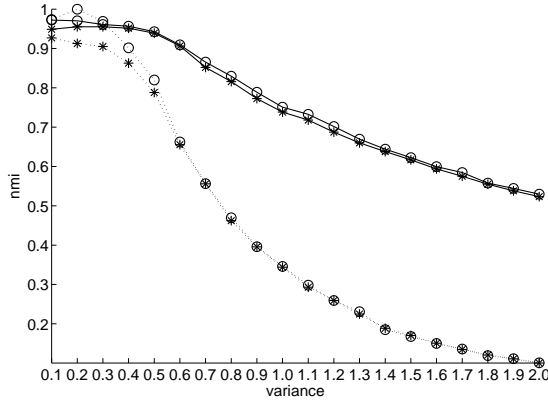[1] Available at: https://github.com/mathieulagrange/kaveragePaper

Fig. 3 NMI of kernel k-means (*) and k-averages (o) clustering relative to ground truth as a function of the "spread" of those classes for synthetic data sets of 5 and 40 classes, displayed in dashed and solid lines, respectively.

40 classes. On the x-axis is the variance of the Gaussian distribution used to generate the point cloud for each class: the higher that value, the more the classes are spread out and overlap each other, thus making the clustering harder.

The question of choosing the proper number of clusters for a given dataset without *a priori* is a well known and hard problem, and beyond the scope of this article. Therefore, for the purpose of evaluation, clustering is done by requesting a number of clusters equal to the actual number of classes in the dataset. In order to obtain stable and reliable figures, clustering is repeated 500 times with varying initial conditions, *i.e.* the initial assignment of points to clusters is randomly determined, and only the average performance is given. For fairness of comparison, each algorithm is run with the exact same initial assignments.

As can be seen on the figure, in the case of a 5-class problem, k-averages outperforms kernel k-means in the "easy" cases (low class spread), before converging to equivalent results. For the more complex 40-class datasets, k-averages consistently yields a better result than kernel k-means, especially for higher values of the variance. The lower values of NMI for 5-class experiments is in fact an artifact introduced by the normalization of NMI, and is not important here; we only focus, for each series of experiments, on the relative performances of kernel k-means and k-averages.

7.3 Time efficiency

Figure 4 shows the average time spent by kernel k-means and k-averages to cluster synthetic datasets or varying sizes. As previously, initial conditions on each run are identical for both algorithms. The reported run time is the one
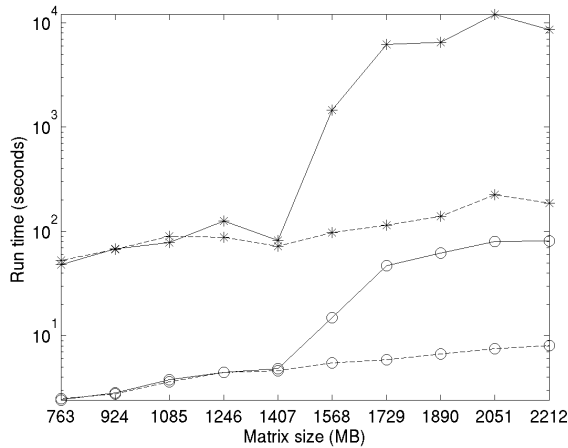
Fig. 4 Average computation time of the kernel k-means (*) and kaverages (o) algorithms on computers with 2 GB (solid line) and 32 GB (dashed line) of RAM, respectively. The "running time" axis follows a logarithmic scale.

measured on a 64 bits Intel® Core™ i7 runnning at 3.6 GHz with 32 Gb of RAM and standard Hard Disk Drive (HDD) operated by standard Linux distribution. For results with 2 GB of RAM, the same machine is used with a memory limitation specified to the kernel at boot time.

These figures confirm the theoretical complexity analysis presented in Section 6: k-averages runs at least 20 times faster on average than kernel k-means in ordinary conditions, when available memory is not an issue. When the matrix size exceeds what can be stored in RAM and the system has to resort to paged memory, as in the presented experiments when the matrix reaches about 1500MB, both algorithms suffer from a clear performance hit; however, kernel k-means is much more affected, and the difference becomes even more important: with a 2000MB similarity matrix on a memory-limited computer, k-averages runs about 100 times faster than kernel k-means.

Having established the interest of our proposed method relative to kernel k-means on synthetic object collections, we now proceed to a thorough evaluation on real data.

8 Experiments

In order to demonstrate the usefulness of k-averages when dealing with real data, we have chosen to focus on the clustering of time series as the evaluation task. Time series, even though represented as vectors and therefore suitable for any kinds of norm-based clustering, are best compared with elastic measures (Ding et al. 2008; Wang et al. 2013), partly due to their varying length. The Dynamic Time Warping (DTW) measure is an elastic measure widely used in many areas since its introduction for spoken word detection (Sakoe and

Table 1   Statistics of the datasets. The length of the times series is expressed in samples.

	min	average $\pm$ variance	max
number of classes	2	8 ± 9	50
number of time series	56	1626 ± 2023	9236
time series length	24	372 ± 400	1882

Chiba 1978) and has never been challenged for time series mining (Berndt and Clifford 1994; Rakthanmanon et al. 2013).

Effective clustering of time series using the DTW measure requires similarity based algorithms such as the k-averages algorithm. With some care, kernel based algorithm can also be considered provided that the resulting similarity matrix is converted into a kernel, *i.e.* the matrix is forced to be semi definite positive, *i.e.* to be a Gram matrix (Lanckriet et al. 2004) in order to guarantee convergence.

8.1 Datasets

To compare quality of clusterings obtained by the considered algorithms, we consider a large collection of 43 time series datasets made publicly available by many laboratories worldwide and compiled by Prof. Keogh. Thus, while all the experiments presented here are performed on time series (chosen for being a good example of a data type requiring similarity-based clustering, as opposed to a simple Euclidean approach), the great variety in the sources and semantics of said series (bio-informatics, linguistics, astronomy, gesture modeling, chemistry...) gives this validation a wide foundation. Statistics about the morphology of those datasets are summarized in Table 1. Individual details can be found in Appendix A.

8.2 Methods

Three algorithms are considered: the spectral clustering Von Luxburg (2007) approach as a high complexity reference, the kernel k-means algorithm implemented as described in Section 2 and the proposed k-averages algorithm. The spectral clustering algorithm tested here uses the normalization proposed by Jordan and Weiss Ng et al. (2002). This normalization is chosen over no normalization and the Shi and Malik one Shi and Malik (2000) as it is found to be the best performing in terms of average NMI over all the datasets. The implementation is done using the Matlab programming language. Even though a C implementation would probably be more efficient, we believe that the gain would be low as the main computational load is the diagonalization of the similarity matrix and the k-means clustering of the eigenvectors which are both efficient builtins Matlab functions. The kernel k-means is implemented

Table 2 NMI (in percents) of clusterings by kernel k-means and k-averages for bi-class datasets.

	spectral clustering	kernel k-means	k-averages
Coffee	3.4 ±0.0	6.9 ±4.1	**7.8 ±3.6**
ECG200	9.9 ±0.0	**14.8 ±1.0**	14.6 ±0.0
ECGFiveDays	0.1 ±0.0	**3.3 ±0.3**	2.5 ±0.0
Gun_Point	**14.8 ±1.3**	0.0 ±0.0	0.0 ±0.0
ItalyPowerDemand	**1.1**	0.9	0.9
Lighting2	4.1 ±0.3	**4.9 ±3.7**	4.3 ±4.2
MoteStrain	0.0 ±0.0	**48.9 ±0.6**	48.8 ±0.6
SonyAIBORobotSurface	3.4 ±0.0	45.0 ±20.7	**45.1 ±20.5**
SonyAIBORobotSurfaceII	2.5 ±0.0	**21.4 ±0.0**	**21.4 ±0.0**
TwoLeadECG	**1.4 ±0.0**	0.1 ±0.6	0.0 ±0.2
wafer	**0.0**	0.0	0.0
yoga	**0.5 ±0.0**	0.2 ±0.1	0.2 ±0.1

both in Matlab using the implementation provided by Mo Chen[2] and in the C programming language following Algorithm 1. The k-averages method is implemented in C following Algorithm 3.

8.3 Evaluation Protocol

For each dataset, since we perform clustering, and not supervised learning, the training and testing data are joined together. DTW similarities are computed using the implementation provided by Prof. Ellis [3] with default parameters.

As in our previous experiments with synthetic data, we choose here the normalized mutual information (NMI) as the measure of clustering quality; clustering is done by requesting a number of clusters equal to the actual number of classes in the dataset, and repeated 200 times with varying initial conditions, each algorithm being run with the exact same initial assignments. For the 200 clusterings thus produced, we compute the NMI between them and the ground truth clustering. Average and standard deviation statistics are then computed.

8.4 Clustering performance

For ease of readability and comparison, the presented results are split into 3 tables. Table 2 lists the results obtained on bi-class datasets, *i.e.* the datasets annotated in terms of presence or absence of a given property; Table 3 concerns the datasets with a small number of classes (from 3 to 7); and Table 4 focuses the datasets with a larger number of classes (from 8 to 50).

---

[2] Available at: https://fr.mathworks.com/matlabcentral/fileexchange/26182-kernel-kmeans

[3] Available at: http://www.ee.columbia.edu/~dpwe/resources/matlab/dtw

Table 3 NMI (in percents) of clusterings by kernel k-means and k-averages for datasets of 3 to 7 classes.

	spectral clustering	kernel k-means	k-averages
Beef	25.7 ±2.4	**35.5 ±2.8**	34.5 ±2.6
CBF	11.1 ±0.2	41.0 ±8.4	**41.0 ±7.0**
ChlorineConcentration	**3.9 ±0.0**	0.2 ±0.1	0.2 ±0.1
CinC_ECG_torso	**37.9 ±0.5**	24.2 ±1.5	24.6 ±1.6
DiatomSizeReduction	45.3 ±4.2	**79.8 ±6.3**	77.3 ±4.5
FaceFour	49.8 ±4.6	72.2 ±8.4	**74.9 ±6.3**
Haptics	3.0 ±0.5	**9.7 ±1.3**	9.4 ±1.2
InlineSkate	4.6 ±0.5	6.3 ±0.7	**6.4 ±0.7**
Lighting7	27.6 ±2.7	51.0 ±3.1	**51.3 ±1.5**
OSULeaf	22.9 ±0.8	22.9 ±2.2	**23.0 ±2.5**
OliveOil	10.6 ±2.2	**32.4 ±8.6**	30.6 ±7.8
StarLightCurves	54.3 ±0.0	**60.3 ±0.4**	60.3 ±0.0
Symbols	72.0 ±1.7	79.1 ±3.8	**79.5 ±1.6**
Trace	13.7 ±4.4	**54.5 ±5.1**	54.3 ±6.3
Two_Patterns	0.2 ±0.0	**10.1 ±10.0**	8.8 ±9.9
fish	18.5 ±1.5	34.9 ±1.4	**35.3 ±1.0**
synthetic_control	63.0 ±0.7	85.3 ±4.9	**89.5 ±0.8**

Table 4 NMI (in percents) of clusterings by kernel k-means and k-averages for datasets of 8 to 50 classes.

	spectral clustering	kernel k-means	k-averages
50words	46.9 ±1.0	69.6 ±0.8	**71.7 ±0.6**
Adiac	55.0 ±1.0	55.7 ±1.0	**58.9 ±0.6**
Cricket_X	19.2 ±1.1	26.1 ±1.8	**26.8 ±1.2**
Cricket_Y	22.8 ±1.3	33.0 ±1.5	**33.5 ±1.2**
Cricket_Z	19.2 ±1.3	25.5 ±2.2	**27.3 ±1.3**
FaceAll	36.4 ±1.2	**70.7 ±2.0**	68.2 ±2.3
FacesUCR	36.2 ±1.3	70.4 ±2.0	**71.5 ±1.6**
MALLAT	44.5 ±2.0	**90.3 ±4.6**	89.2 ±3.5
MedicalImages	19.2 ±1.3	29.7 ±1.7	**30.4 ±1.6**
SwedishLeaf	48.7 ±1.4	70.3 ±1.9	**70.8 ±1.4**
WordsSynonyms	31.9 ±1.0	50.9 ±1.2	**52.1 ±0.8**
uWaveGestureLibrary_X	24.4 ±0.6	46.0 ±1.2	**46.2 ±0.5**
uWaveGestureLibrary_Y	16.2 ±0.3	44.9 ±0.4	**45.0 ±0.2**
uWaveGestureLibrary_Z	23.2 ±0.6	**42.9 ±0.7**	42.6 ±0.5

For each experiment, the result of the best performing method is marked in bold. The Matlab and C implementations of the kernel k-means algorithm give exactly the same results in terms of NMI, thus only one column is used to display their performance.

A first observation is that the spectral clustering algorithm only performs favorably for 2 of the 43 dataset. Also, most of the bi-class problems (Table 2) do not seem to lend themselves well to this kind of approach: kernel k-means and k-averages produce quasi-identical results, poor in most cases. Concerning the medium numbers of classes (Table 3), k-averages performs best for 8 datasets out of 17. For the larger numbers of classes (Table 4), k-averages performs best for 11 datasets out of 14.

Table 5 Computation time (in seconds) for several implementations of the evaluated clustering algorithms for datasets of 2 classes.

	Matlab		C	
	spectral clustering	kk-means	kk-means	k-averages
Coffee	0.02	0.00	**0.00**	0.00
ECG200	0.03	0.00	0.00	**0.00**
ECGFiveDays	0.13	0.10	0.06	**0.01**
Gun_Point	0.02	0.00	0.00	**0.00**
ItalyPowerDemand	0.20	0.07	0.05	**0.01**
Lighting2	0.02	0.00	0.00	**0.00**
MoteStrain	0.33	0.11	0.08	**0.01**
SonyAIBORobotSurface	0.09	0.02	0.01	**0.00**
SonyAIBORobotSurfaceII	0.09	0.01	0.02	**0.01**
TwoLeadECG	0.13	0.16	0.10	**0.02**
wafer	19.62	0.72	1.04	**0.39**
yoga	3.55	0.80	0.66	**0.10**

Considering the standard deviation over the several runs of the algorithm with different initialization is helpful to study the sensitivity of the algorithm to its initialization and thus its tendency to be stuck into local minima. For most datasets, the standard deviation of the k-averages algorithm is smaller than the one of the kernel k-means one and thus seems experimentally more robust.

8.5 Efficiency

Computation time displayed in Tables 5, 6, and 7 is the average duration over 100 runs on a single core with no parallelization capabilities. For every datasets, the spectral clustering approach is the more time consuming due to the diagonalization of the matrix which is of $O(N^3)$. For the kernel k-means algorithm, the C implementation is most of the time more efficient than the Matlab implementation. The k-average algorithm is more efficient for 46 datasets out of the 47 by close to an order of magnitude for the larger datasets.

8.6 Overall results

To conclude on the performance of the evaluated algorithms on real datasets, Table 8 displays the NMI and computation time averaged over the 47 datasets. The k-averages method marginally improve the clustering accuracy compared to the kernel k-means approach by using less time to compute.

Table 6 Computation time (in seconds) for several implementations of the evaluated clustering algorithms for datasets of 3 to 7 classes.

	Matlab		C	
	spectral clustering	kk-means	kk-means	k-averages
Beef	0.02	0.00	0.00	**0.00**
CBF	0.17	0.08	0.05	**0.01**
ChlorineConcentration	7.15	2.74	2.09	**0.23**
CinC_ECG_torso	0.55	0.31	0.16	**0.04**
DiatomSizeReduction	0.04	0.01	0.00	**0.00**
FaceFour	0.02	0.00	0.00	**0.00**
Haptics	0.07	0.04	0.01	**0.01**
InlineSkate	0.12	0.08	0.03	**0.01**
Lighting7	0.04	0.01	0.00	**0.00**
OSULeaf	0.08	0.04	0.01	**0.00**
OliveOil	0.03	0.00	0.00	**0.00**
StarLightCurves	34.16	4.69	6.21	**0.91**
Symbols	0.26	0.15	0.06	**0.02**
Trace	0.02	0.01	0.00	**0.00**
Two_Patterns	11.99	3.99	3.71	**0.40**
fish	0.06	0.02	0.01	**0.00**
synthetic_control	0.09	0.03	0.01	**0.01**

Table 7 Computation time (in seconds) for several implementations of the evaluated clustering algorithms for datasets of 8 to 50 classes.

	Matlab		C	
	spectral clustering	kk-means	kk-means	k-averages
50words	0.89	0.19	0.05	**0.03**
Adiac	0.49	0.16	0.05	**0.03**
Cricket_X	0.24	0.10	0.05	**0.02**
Cricket_Y	0.22	0.09	0.05	**0.02**
Cricket_Z	0.22	0.09	0.04	**0.02**
FaceAll	2.08	0.78	0.40	**0.11**
FacesUCR	2.21	0.75	0.41	**0.12**
MALLAT	2.23	0.47	0.29	**0.12**
MedicalImages	0.42	0.24	0.12	**0.04**
SwedishLeaf	0.56	0.22	0.11	**0.04**
WordsSynonyms	0.44	0.16	0.08	**0.03**
uWaveGestureLibrary_X	8.60	5.15	3.82	**0.44**
uWaveGestureLibrary_Y	8.94	4.48	2.65	**0.45**
uWaveGestureLibrary_Z	8.53	4.00	2.89	**0.47**

Table 8 Performances averaged over the 47 datasets. The NMI is expressed in percents and the computation time in seconds.

	Matlab		C	
	spectral clustering	kk-means	kk-means	k-averages
nmi (%)	22.1	36.6	36.6	**36.8**
computation time	2.678	0.723	0.590	**0.096**

9 Conclusion

We have presented k-averages, an iterative flat clustering algorithm that operates on arbitrary similarity matrices by explicitly and directly aiming to optimize the average intra-class similarity. Having established the mathematical foundation of our proposal, including guaranteed convergence, we have thoroughly compared it with widely used standard methods: the kernel k-means and the spectral clustering techniques. We show that the k-averages algorithm converges much faster (20 times faster under ordinary conditions) and leads to equivalent or better clustering results for the task of clustering both synthetic data and realistic times series taken from a wide variety of sources, while also being more computationally efficient and more sparing in memory use.

10 Acknowledgments

References

A. Banerjee, S. Merugu, I.S. Dhillon, J. Ghosh, Clustering with bregman divergences. J. Mach. Learn. Res. **6**, 1705–1749 (2005)

D.J. Berndt, J. Clifford, Using Dynamic Time Warping to Find Patterns in Time Series., in *KDD Workshop*, ed. by U.M. Fayyad, R. Uthurusamy (AAAI Press, ???, 1994), pp. 359–370. ISBN 0-929280-73-3

P.S. Bradley, U.M. Fayyad, C. Reina, Scaling Clustering Algorithms to Large Databases, in *Knowledge Discovery and Data Mining*, 1998, pp. 9–15

Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, G. Batista, *The UCR Time Series Classification Archive*, 2015. www.cs.ucr.edu/~eamonn/time_series_data/

R. Chitta, R. Jin, T.C. Havens, A.K. Jain, Approximate Kernel K-means: Solution to Large Scale Kernel Clustering, in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '11 (ACM, New York, NY, USA, 2011), pp. 895–903. ISBN 978-1-4503-0813-7. doi:10.1145/2020408.2020558

I.S. Dhillon, Y. Guan, B. Kulis, Weighted graph cuts without eigenvectors a multilevel approach. IEEE Trans. Pattern Anal. Mach. Intell. **29**(11), 1944–1957 (2007). doi:10.1109/TPAMI.2007.1115

I.S. Dhillon, S. Mallela, R. Kumar, A divisive information theoretic feature clustering algorithm for text classification. J. Mach. Learn. Res. **3**, 1265–1287 (2003)

H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, E. Keogh, Querying and mining of time series data: Experimental comparison of representations and distance measures. Proc. VLDB Endowment **1**(2), 1542–1552 (2008). doi:10.14778/1454159.1454226

R.O. Duda, P.E. Hart, D.G. Stork, *Pattern Classification (2nd Ed)* (Wiley, ???, 2001)

M. Girolami, Mercer kernel-based clustering in feature space. IEEE Transactions on Neural Network **13**(3), 780–784 (2002). doi:10.1109/TNN.2002.1000150

A.K. Jain, Data clustering: 50 years beyond k-means. Pattern Recognition Letters **31**, 651–666 (2010)

L. Kaufman, P.J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis* (John Wiley and Sons, New York, 1990)

H.W. Kuhn, The Hungarian Method for the Assignment Problem. Naval Research Logistics Quarterly **2**(1–2), 83–97 (1955). doi:10.1002/nav.3800020109

B. Kulis, S. Basu, I. Dhillon, R. Mooney, Semi-supervised graph clustering: a kernel approach. Machine Learning **74**(1), 1–22 (2008). doi:10.1007/s10994-008-5084-4

G.R.G. Lanckriet, N. Cristianini, P. Bartlett, L.E. Ghaoui, M.I. Jordan, Learning the kernel matrix with semidefinite programming. J. Mach. Learn. Res. **5**, 27–72 (2004)

Y. Linde, A. Buzo, R.M. Gray, An algorithm for vector quantizer design. IEEE Transactions on Communications **28**, 84–95 (1980)

J.B. MacQueen, Some Methods for classification and Analysis of Multivariate Observations, in *Proc. of Berkeley Symposium on Mathematical Statistics and Probability*, 1967

A.Y. Ng, M.I. Jordan, Y. Weiss, On spectral clustering: Analysis and an algorithm, in *Advances in neural information processing systems*, 2002, pp. 849–856

R.T. Ng, J. Han, Efficient and Effective Clustering Methods for Spatial Data Mining, in *Proceedings of the 20th International Conference on Very Large Data Bases*. VLDB '94 (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994), pp. 144–155. ISBN 1-55860-153-8

T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, E. Keogh, Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping. ACM Trans. Knowl. Discov. Data **7**(3), 10–11031 (2013). doi:10.1145/2500489

V. Roth, J. Laub, M. Kawanabe, J.M. Buhmann, Optimal cluster preserving embedding of nonmetric proximity data. IEEE Trans. Pattern Anal. Mach. Intell. **25**(12), 1540–1551 (2003). doi:10.1109/TPAMI.2003.1251147

H. Sakoe, S. Chiba, Dynamic programming algorithm optimization for spoken word recognition. IEEE Transactions on Acoustics, Speech and Signal Processing **26**(1), 43–49 (1978). doi:10.1109/TASSP.1978.1163055

J. Shi, J. Malik, Normalized cuts and image segmentation. IEEE Transactions on pattern analysis and machine intelligence **22**(8), 888–905 (2000)

M. Steinbach, G. Karypis, V. Kumar, et al., A comparison of document clustering techniques, in *KDD workshop on text mining*, vol. 400, Boston, 2000, pp. 525–526. Boston

A. Thalamuthu, I. Mukhopadhyay, X. Zheng, G.C. Tseng, Evaluation and comparison of gene clustering methods in microarray analysis. Bioinformatics **22**, 2405–2412 (2006)

V.N. Vapnik, *The Nature of Statistical Learning Theory* (Springer, New York, NY, USA, 1995). ISBN 0-387-94559-8

U. Von Luxburg, A tutorial on spectral clustering. Statistics and computing **17**(4), 395–416 (2007)

X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, E. Keogh, Experimental comparison of representation methods and distance measures for time series data. Data Mining Knowledge Discovery **26**(2), 275–309 (2013). doi:10.1007/s10618-012-0250-5

R. Zhang, A.I. Rudnicky, A large scale clustering scheme for kernel K-Means, in *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, vol. 4, 2002, pp. 289–2924. doi:10.1109/ICPR.2002.1047453

A Description of the datasets

Table 9 Description of the time series datasets used for evaluation.

name	number of classes	number of samples
50words	50	905
Adiac	37	781
Beef	5	60
CBF	3	930
ChlorineConcentration	3	4307
CinC_ECG_torso	4	1420
Coffee	2	56
Cricket_X	12	780
Cricket_Y	12	780
Cricket_Z	12	780
DiatomSizeReduction	4	322
ECG200	2	200
ECGFiveDays	2	884
FaceAll	14	2250
FaceFour	4	112
FacesUCR	14	2250
Gun_Point	2	200
Haptics	5	463
InlineSkate	7	650
ItalyPowerDemand	2	1096
Lighting2	2	121
Lighting7	7	143
MALLAT	8	2400
MedicalImages	10	1141
MoteStrain	2	1272
OSULeaf	6	442
OliveOil	4	60
SonyAIBORobotSurface	2	621
SonyAIBORobotSurfaceII	2	980
StarLightCurves	3	9236
SwedishLeaf	15	1125
Symbols	6	1020
Trace	4	200
TwoLeadECG	2	1162
Two_Patterns	4	5000
WordsSynonyms	25	905
fish	7	350
synthetic_control	6	600
uWaveGestureLibrary_X	8	4478
uWaveGestureLibrary_Y	8	4478
uWaveGestureLibrary_Z	8	4478
wafer	2	7164
yoga	2	3300