



HAL
open science

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

Mathias Rossignol, Mathieu Lagrange, Arshia Cont

► **To cite this version:**

Mathias Rossignol, Mathieu Lagrange, Arshia Cont. Efficient similarity-based data clustering by optimal object to cluster reallocation. 2015. hal-01123756v1

HAL Id: hal-01123756

<https://hal.science/hal-01123756v1>

Preprint submitted on 5 Mar 2015 (v1), last revised 26 Jun 2018 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

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

Mathias Rossignol, Mathieu Lagrange, Arshia Cont *Member, IEEE*

Abstract—We present an iterative flat 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 an 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 allows relaxing the conditions on usable matrices, as well as opening better optimization possibilities. Systematic evaluation on a variety of data sets shows that the proposed approach outperforms or equals kernel k-means in a large majority of cases, while running much faster. Most notably, it significantly reduces memory access, which makes it a good choice for large data collections.

Index Terms—clustering, similarity, non-metric data representation, time series clustering

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 data 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 ([1]), performing better on some classes of problems than others ([2], [3]).

Iterative divisive hard clustering algorithms, such as the ubiquitous *k-means* ([4]), 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 ([4]) 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 [5], any kind of Bregman divergence, such as the KL-divergence ([6]) or the Itakura-Saito divergence ([7]), 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 explicitly be computed.

Earlier 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 ([8]) solves the *k-medoids* problem but with a high complexity ($O(k(n-k)^2)i$, n being the number of objects) and high number of iterations i , due to low convergence rate. In order to scale the approach, the Clustering LARge Applications (CLARA) algorithm ([8]) 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 ([9]) preserves the whole set of objects but cuts complexity by drawing a sample of neighbors in each search for the medoids.

Following work on kernel projection ([10]), 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, [11] 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. [12] linked a weighted version of the kernel *K-means* objective to the popular spectral clustering, introducing an efficient way of solving the normalized cut objective.

The kernel k-means algorithm proved to be equally useful when considering arbitrary similarity problems if special care is taken to ensure definite positiveness of the input matrix ([13]). 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 can hardly be applied to large scale datasets without special treatments because of high algorithmic and memory access costs. [14] considered sampling of the input data, [15] considered block storing of the input matrix, and a preclustering approach ([16], [17]) is considered by [18] with a coarsening and refining phases as respectively a pre- and post-treatment of the actual clustering phase.

The work we present in this paper results from an effort to directly maximize the average intra-class similarity, without resorting to a geometric interpretation of data. Following that idea allows us to propose a new hard 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 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 on average slightly better than those of kernel k-means, and obtained in a fraction of its computing time, particularly when paged memory is required.

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 5. The above presented properties of the proposed k-averages algorithm are then validated on synthetic controlled data in Section 6 and on 43 corpora of time series in Section 7.

2 KERNEL K-MEANS

Since its introduction by [11], kernel k-means has been an algorithm of choice for flat data clustering with known number of clusters ([18], [13]). It makes use of a mathematical technique known as the “kernel trick” to extend the classical k-means clustering algorithm ([4]) 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 consists in considering that the k-means algorithm is operating in an unspecified, possibly very high-dimensional Euclidian 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.

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:

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

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

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

Since the sum of \mathcal{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. Given a similarity matrix possessing the necessary properties to be considered as a kernel matrix (positive semidefinite), the kernel k-means algorithm can therefore be used to create clusters that somehow maximize the average intra-cluster similarity.

2.2 Algorithm

Finding the configuration that absolutely 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” ([4]), 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:
6          $z_{cn} = (L_n == c) ? 1 : 0$ )
7       end
8        $L_n = \text{argmin}_c(Y_{cn});$ 
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 5, 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; we then introduce an algorithm intended to take advantage of that formula.

3.1 Conventions and possible objective functions

In addition to the notations presented above, we index hereon the set of elements belonging to a given cluster c_k as $c_k = \{o_{k1}, \dots, o_{kN_k}\}$. For simplicity, we omit the first index and note $c = \{o_1, \dots, o_{N_c}\}$ 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 in c :

If $o \notin c$,

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

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

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

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

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

In our framework, we do not explicitly refer to class centroids, preferring to directly consider averages of similarity values between individuals within clusters. Indeed, we never refer to a geometrical interpretation of the data. However, it should be noted that since in k-means (and kernel k-means) the centroid of a class is defined as an average of all points in that class, \mathcal{Q} is strictly equivalent to the average point to centroid similarity.

Using the notations above, we define our objective function as the average class quality, normalized with class sizes:

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

Since informally, our goal is to bring together objects that share high similarity, a first idea would be to simply 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.

To ensure global convergence, we need to compute the impact on the global objective function of moving one object from one class to another. Using such formulation and performing only reallocation that have a positive impact, the convergence 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 (3) to get:

Data: number of objects N , number of classes C , similarity matrix 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  $S(j, \text{nextClass})$  and
            $S(j, \text{previousClass})$ 
10      end
11    end
12  end
13 end

```

Algorithm 2: The naive k-averages algorithm.

$$\begin{aligned} \mathcal{Q}(c) &= \frac{1}{N_c} \sum_{i=1}^{N_c} \frac{1}{N_c - 1} \sum_{\substack{j=1 \dots N_c \\ j \neq i}} s(o_i, o_j) \\ &= \frac{1}{N_c(N_c - 1)} \sum_{i=1}^{N_c} \sum_{\substack{j=1 \dots N_c \\ j \neq i}} s(o_i, o_j) \end{aligned} \quad (5)$$

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

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

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

$$\Sigma(c) = \sum_{i=2}^{N_c} \sum_{j=1}^{i-1} s(o_i, o_j) \quad (7)$$

Thus:

$$\mathcal{Q}(c) = \frac{2}{N_c(N_c - 1)} \Sigma(c) \quad (8)$$

and

$$\Sigma(c) = \frac{N_c(N_c - 1)\mathcal{Q}(c)}{2} \quad (9)$$

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.

$$\begin{aligned} \mathcal{Q}(c \setminus o_{N_c}) &= \frac{2}{(N_c - 1)(N_c - 2)} \sum_{i=2}^{N_c-1} \sum_{j=1}^{i-1} s(o_i, o_j) \\ &= \frac{2}{(N_c - 1)(N_c - 2)} \left[\Sigma(c) - \sum_{j=1}^{N_c-1} s(o_{N_c}, o_j) \right] \\ &= \frac{2}{(N_c - 1)(N_c - 2)} [\Sigma(c) - (N_c - 1)s(o_{N_c}, c)] \\ &= \frac{2N_c(N_c - 1)\mathcal{Q}(c)}{2(N_c - 1)(N_c - 2)} - \frac{2(N_c - 1)s(o_{N_c}, c)}{(N_c - 1)(N_c - 2)} \\ &= \frac{N_c\mathcal{Q}(c) - 2s(o_{N_c}, c)}{N_c - 2} \end{aligned} \quad (10)$$

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

$$\mathcal{Q}(c \setminus o) = \frac{N_c\mathcal{Q}(c) - 2s(o, c)}{N_c - 2} \quad (11)$$

And the change in quality from its previous value:

$$\begin{aligned} \mathcal{Q}(c \setminus o) - \mathcal{Q}(c) &= \frac{N_c\mathcal{Q}(c) - (N_c - 2)\mathcal{Q}(c) - 2s(o, c)}{N_c - 2} \\ &= \frac{2(\mathcal{Q}(c) - s(o, c))}{N_c - 2} \end{aligned} \quad (12)$$

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:

$$\begin{aligned} \mathcal{Q}(c \cup o_{N_c+1}) &= \frac{2}{N_c(N_c + 1)} \sum_{i=2}^{N_c+1} \sum_{j=1}^{i-1} s(o_i, o_j) \\ &= \frac{2}{N_c(N_c + 1)} [\Sigma(c) + N_c s(o_{N_c+1}, c)] \\ &= \frac{(N_c - 1)\mathcal{Q}(c) + 2s(o_{N_c+1}, c)}{N_c + 1} \end{aligned} \quad (13)$$

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

$$\mathcal{Q}(c \cup o) = \frac{(N_c - 1)\mathcal{Q}(c) + 2s(o, c)}{N_c + 1} \quad (14)$$

And the change in quality from its previous value:

$$\mathcal{Q}(c \cup o) - \mathcal{Q}(c) = \frac{2(s(o, c) - \mathcal{Q}(c))}{N_c + 1} \quad (15)$$

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 (12), and N_t are affected by that in (15); in addition, one object o moves from a class whose quality is $\mathcal{Q}(c_s)$ to one whose quality is $\mathcal{Q}(c_t \cup o)$, as expressed by equation 14:

$$\delta_o(c_s, c_t) = \frac{2N_t(s(o, c_t) - \mathcal{Q}(c_t))}{N_t + 1} + \frac{2(N_s - 1)(\mathcal{Q}(c_s) - s(o, c_s))}{N_s - 2} + \frac{(N_t - 1)\mathcal{Q}(c_t) + 2s(o, c_t)}{N_t + 1} - \mathcal{Q}(c_s) \quad (16)$$

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.

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. Therefore, for each considered object, after finding its ideal new class, 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.

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 makes such dynamic changes easily tractable. New class qualities are thus given by Equations 11 and 14, and new object-class similarities can be computed by:

$$s(i, c_s(t+1)) = \frac{N_s(t) \cdot s(i, c_s(t)) + s(i, n)}{N_s(t) + 1} \quad (17)$$

$$s(i, c_t(t+1)) = \frac{N_t(t) \cdot s(i, c_s(t)) - s(i, n)}{N_t(t) - 1}$$

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.

5 COMPLEXITY ANALYSIS

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

5.1 Computational complexity

5.1.1 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 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 1, the third term

Data: number of objects N , number of classes C , similarity matrix 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 Compute initial class qualities  $\mathcal{Q}$  following eq 6;
4 while  $L$  is modified do
5   for  $i \leftarrow 1$  to  $N$  do
6     previousClass  $\leftarrow L_i$ ;
7     nextClass  $\leftarrow \operatorname{argmin}_k \delta_i(\text{previousClass}, k)$ 
      (following the definition of  $\delta$  in 16);
8     if nextClass  $\neq$  previousClass then
9        $L_i \leftarrow$  nextClass;
10      Update  $\mathcal{Q}_{\text{previousClass}}$  following eq 11;
11      Update  $\mathcal{Q}_{\text{nextClass}}$  following eq 14;
12      for  $j \leftarrow 1$  to  $N$  do
13        Update  $S(j, \text{nextClass})$  and
           $S(j, \text{previousClass})$ 
          following eq 17;
14      end
15    end
16  end
17 end
18 end

```

Algorithm 3: The K-averages algorithm.

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 1 as:

$$Y_{cn} = \mathcal{K}_{nn} - \frac{2}{N_c} \sum_{i=1}^N z_{ci} \mathcal{K}_{ni} + M_c \quad (18)$$

where

$$M_c = \frac{1}{N_c^2} \sum_{i=1}^N \sum_{j=1}^N z_{ki} z_{kj} \mathcal{K}_{ij} \quad (19)$$

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.

5.1.2 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 7
- $O(NM)$ corresponding to the object-to-class similarity update at line 13, 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

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 19
5   end
6   for  $n \leftarrow 1$  to  $N$  do
7     for  $c \leftarrow 1$  to  $C$  do
8       Compute  $Y_{cn}$  following eq 18 (note:
9          $z_{cn} = (L_n == c) ? 1 : 0$ )
10      end
11       $L_n = \operatorname{argmin}_c(Y_{cn});$ 
12 end

```

Algorithm 4: Lloyd’s algorithm applied to minimizing the kernel k-means objective, optimized version.

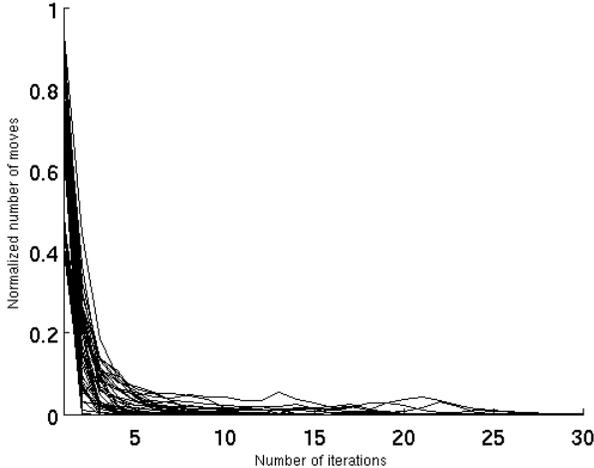


Fig. 1. Number of moved objects per iteration when clustering a variety of datasets with the k-averages algorithm, normalized by the total number of objects to cluster. The datasets used to create this figure are the real-life time series data that we employ for experimental validation, cf. Section 7.

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. As can be seen, the correlation is clearly 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.

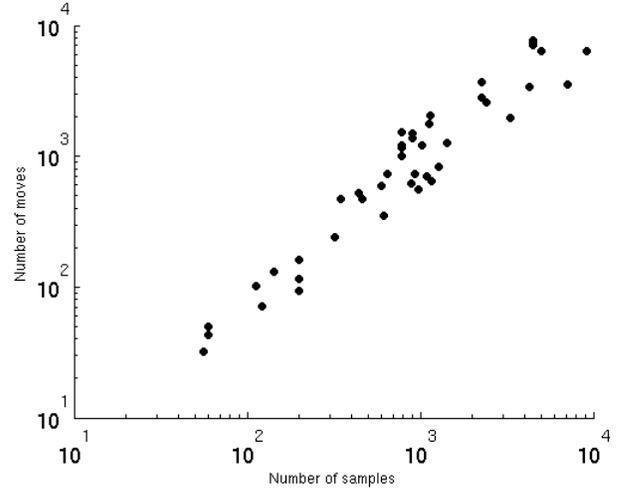


Fig. 2. Total number of object reallocations over a run of the k-averages algorithm, plotted against the number of objects to be clustered. The datasets used to create this figure are the real-life time series data that we employ for experimental validation, cf. Section 7.

5.2 Memory access

The lowered computational costs is also accompanied by a decrease in memory access: as can be seen from Equation 17, 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.

6 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 processing¹.

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

1. C and Matlab implementations can be found at <https://bitbucket.org/mlagrange/kaverages/downloads>

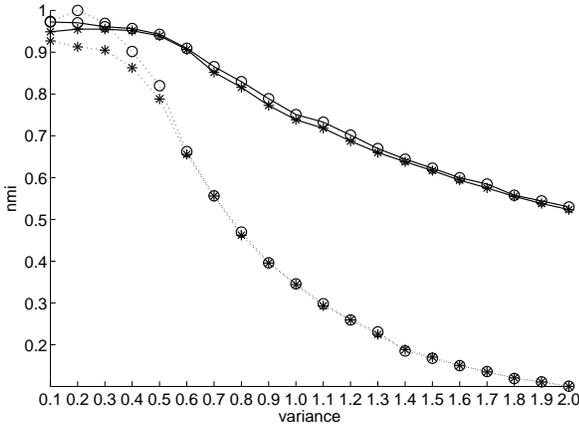


Fig. 3. NMI of kernel k-means (*) and k-averages (o) clustering relative to ground truth as a function of the “spread” of those classes for synthetic data sets of 5 and 40 classes, respectively displayed in dashed and solid lines.

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.

6.1 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 [19].

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:

$$\text{NMI} = \frac{2I(C; K)}{H(C) + H(K)} \quad (20)$$

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 and accessibility, only the NMI is considered for validations as in ([18]). However, it shall be noted 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 (dotted line) and k-averages (solid

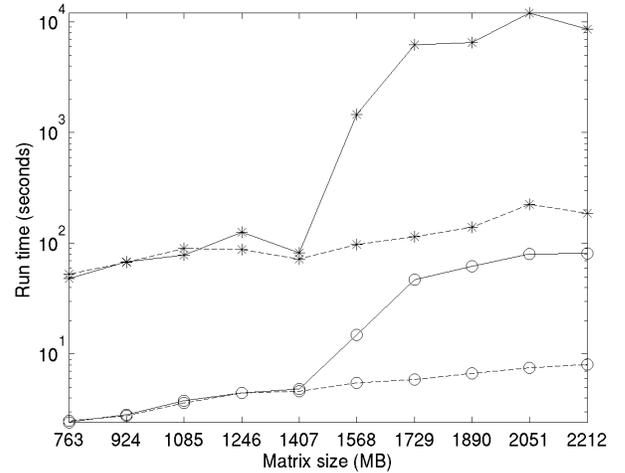


Fig. 4. Average computation time of the kernel k-means (*) and k-averages (o) algorithms on computers with 2 GB (solid line) and 32 GB (dashed line) of RAM, respectively. The “running time” axis follows a logarithmic scale.

line) on two series of datasets: one featuring 5 classes, the other 40 classes. On the x-axis is the variance of the Gaussian distribution used to generate 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, and is not significant here; we only focus, for each series of experiments, on the relative performances of kernel k-means and k-averages.

6.2 Time efficiency

Figure 4 shows the average time spent by kernel k-means and k-averages to cluster synthetic datasets of varying sizes. As previously, initial conditions on each run are identical for both algorithms. The reported run time is the one measured on a 64 bits Intel® Core™ i7 running 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.

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

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

These figures confirm the theoretical complexity analysis presented in Section 5: 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 significant: 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.

7 EXPERIMENTS

In order to demonstrate the usefulness of k-averages when dealing with real data, we have chosen to focus on the clustering of times 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 ([20], [21]), 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 [22] and has never been significantly challenged for time series mining ([23], [24]).

Effective clustering of time series using the DTW measure require similarity based algorithms such as the k-average 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 ([25]).

To showcase the relevance of the proposed approach for time-series similarity-based clustering using DTW, we use a simple k-means clustering as reference in result tables that follow.

7.1 Datasets

To compare quality of clusterings obtained by the considered algorithms, we consider a large collection of 43 time series datasets created by many laboratories all over the world and compiled by Prof. Keogh². Statistics about the morphology of those datasets are summarized in Table 1.

7.2 Evaluation Protocol

For each dataset, since we perform clustering, and not supervised learning, the training and testing data are joined.

2. Available at: www.cs.ucr.edu/~eamonn/time_series_data

id	classes	objects	k-means	k. k-means	k-averages
7	2	56	5 \pm 1	6\pm3*	5 \pm 1
12	2	200	12 \pm 2	14 \pm 2	15\pm1*
13	2	884	0 \pm 0	3\pm2*	3\pm2
17	2	200	0*	0*	0*
20	2	1096	0 \pm 0	1\pm0*	1 \pm 0
21	2	121	3\pm1*	1 \pm 1	1 \pm 0
25	2	1272	30 \pm 0	52 \pm 0	53\pm0*
28	2	621	51 \pm 25	78 \pm 0	78\pm0*
29	2	980	24*	21	21
34	2	1162	0 \pm 0	7\pm0*	7 \pm 0
42	2	7164	0*	0	0
43	2	3300	0 \pm 0	0\pm0	0\pm0*
significantly better:			3	3	3

TABLE 2
Average NMI in percents of clusterings by k-means, kernel k-means and k-averages for bi-class datasets over 500 runs of the algorithms.

If relevant, the DTW is computed using the implementation provided by Prof. Ellis³ with default parameters.

As in our previous experiments with synthetic data, we choose here the normalized mutual information as 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 500 times with varying initial conditions, each algorithm being run with the exact same initial assignments.

7.3 Results

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).

For each experiment, the result of the best performing method is marked by a star, and results equivalent to the best are highlighted in bold. Equivalence is tested with a paired t-test at 0.05 significance level. It should be noted that this decision is the result of a statistical analysis on the 500 runs performed; it can therefore happen, without contradiction, that a method has a higher average NMI than the other but is not significantly better, or, conversely, that it does perform significantly better despite an identical average NMI at the numerical precision of the tables.

A first obvious conclusion of the presented data is that, while k-means can occasionally outperform the more advanced methods, especially in the case of bi-class problems, its results are overall, as expected, clearly inferior to those of a similarity-based clustering using the DTW measure. We shall therefore focus on the difference between kernel k-means and k-averages.

Setting aside simple k-means, k-averages yields a significantly better clustering for 26 datasets out of 43, while kernel k-means takes the lead in 14 cases, three remaining ones being equivalent. Considering results more closely, one can see that k-averages is equivalent to kernel k-means for bi-class problems, slightly better for moderate numbers of classes,

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

id	classes	objects	k-means	k. k-means	k-averages
3	5	60	38±2*	35±3	35±2
4	3	930	36±1	51±4	51±3*
5	3	4307	0±0	0±0*	0±0
6	4	1420	24±3	43±5	44±4*
11	4	322	82±5	83±10	89±8*
15	4	112	44±4	66±8*	65±7
18	5	463	9±0	9±1*	9±0
19	7	650	5±1	5±1*	5±1
22	7	143	44±2	51±3	52±1*
26	6	442	22±2	22±3	23±3*
27	4	60	54±5*	46±8	50±7
30	3	9236	60±0	61±2*	60±0
32	6	1020	76±5	80±4*	80±1
33	4	200	52±2	57±6*	53±3
35	4	5000	2±0	15±11*	14±12
37	7	350	31±2	42±2	42±1*
38	6	600	79±3	84±4	88±3*
significantly better:			2	5	6

TABLE 3

Average NMI in percents of clusterings by k-means, kernel k-means and k-averages for datasets of 3 to 7 classes, over 500 runs of the algorithms.

id	classes	objects	k-means	k. k-means	k-averages
1	50	905	64±1	70±1	72±1*
2	37	781	58±1	63±1	66±1*
8	12	780	26±1	26±2	27±1*
9	12	780	33±2	33±2	33±1*
10	12	780	27±1	26±2	27±1*
14	14	2250	37±2	78±2*	75±2
16	14	2250	37±2	78±2	79±2*
23	8	2400	87±4	89±4	90±3*
24	10	1141	25±1	31±2	32±1*
31	15	1125	54±1	66±2	68±1*
36	25	905	43±1	51±1	52±1*
39	8	4478	44±1	46±1	46±1*
40	8	4478	44±0	45±0	45±0*
41	8	4478	42±1	43±0*	43±0
significantly better:			0	2	12

TABLE 4

Average NMI in percents of clusterings by k-means, kernel k-means and k-averages for datasets of 8 to 50 classes, over 500 runs of the algorithms.

and yields clearly better results (12 out of 14) for datasets featuring more than 8 classes. This is remarkably consistent with the observations made in Section 6 on synthetic data.

8 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 the widely used standard method, kernel k-means, showing that our algorithm converges much faster (20 times faster under ordinary conditions) and leading to better clustering results for the task of clustering time series, while also being more sparing in memory use.

A particularly interesting result is that k-averages seems to yield clearly better results than kernel k-means in the case of problems with many classes (8 and more); this is confirmed both on synthetic test data and on real-life data.

The linear updates and compensated memory access of the proposed k-average method also provides great potentials for Online Clustering methods, where time-series data arrive incrementally into the system, where classes and clusters should be most often reallocated. Explicit reallocation considerations proposed in this paper paves the way for robust Online Clustering methods that shall be further pursued.

REFERENCES

- [1] A. K. Jain, "Data clustering: 50 years beyond k-means," *Pattern Recognition Letters*, vol. 31, pp. 651–666, 2010.
- [2] 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.
- [3] A. Thalamuthu, I. Mukhopadhyay, X. Zheng, and G. C. Tseng, "Evaluation and comparison of gene clustering methods in microarray analysis," *Bioinformatics*, vol. 22, pp. 2405–2412, 2006.
- [4] J. B. MacQueen, "Some Methods for classification and Analysis of Multivariate Observations," in *Proc. of Berkeley Symposium on Mathematical Statistics and Probability*, 1967.
- [5] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh, "Clustering with bregman divergences," *J. Mach. Learn. Res.*, vol. 6, pp. 1705–1749, Dec. 2005.
- [6] I. S. Dhillon, S. Mallela, and R. Kumar, "A divisive information theoretic feature clustering algorithm for text classification," *J. Mach. Learn. Res.*, vol. 3, pp. 1265–1287, Mar. 2003.
- [7] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Transactions on Communications*, vol. 28, pp. 84–95, 1980.
- [8] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*. New York: John Wiley and Sons, 1990.
- [9] R. T. Ng and J. Han, "Efficient and effective clustering methods for spatial data mining," in *Proceedings of the 20th International Conference on Very Large Data Bases, ser. VLDB '94*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, pp. 144–155.
- [10] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York, NY, USA: Springer-Verlag New York, Inc., 1995.
- [11] M. Girolami, "Mercer kernel-based clustering in feature space," *IEEE Transactions on Neural Network*, vol. 13, no. 3, pp. 780–784, May 2002.
- [12] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors: a multilevel approach," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 11, pp. 1944–1957, Nov. 2007.
- [13] V. Roth, J. Laub, M. Kawanabe, and J. M. Buhmann, "Optimal cluster preserving embedding of nonmetric proximity data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 12, pp. 1540–1551, Dec. 2003.
- [14] R. Chitta, R. Jin, T. C. Havens, and 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, ser. KDD '11*. New York, NY, USA: ACM, 2011, pp. 895–903.
- [15] R. Zhang and A. Rudnick, "A large scale clustering scheme for kernel k-means," in *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, vol. 4, 2002, pp. 289–292 vol.4.
- [16] P. S. Bradley, U. M. Fayyad, and C. Reina, "Scaling clustering algorithms to large databases," in *Knowledge Discovery and Data Mining*, 1998, pp. 9–15.
- [17] V. Ganti, R. Ramakrishnan, J. Gehrke, A. L. Powell, and J. C. French, "Clustering large datasets in arbitrary metric spaces." in *ICDE*, M. Kitsuregawa, M. P. Papazoglou, and C. Pu, Eds. IEEE Computer Society, 1999, pp. 502–511.
- [18] B. Kulis, S. Basu, I. Dhillon, and R. Mooney, "Semi-supervised graph clustering: a kernel approach," *Machine Learning*, vol. 74, no. 1, pp. 1–22, Sep. 2008.
- [19] H. W. Kuhn, "The Hungarian Method for the Assignment Problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1–2, pp. 83–97, March 1955.
- [20] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh, "Querying and mining of time series data: Experimental comparison of representations and distance measures," *Proc. VLDB Endowment*, vol. 1, no. 2, pp. 1542–1552, Aug. 2008.

- [21] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. Keogh, "Experimental comparison of representation methods and distance measures for time series data," *Data Mining Knowledge Discovery*, vol. 26, no. 2, pp. 275–309, Mar. 2013.
- [22] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 26, no. 1, pp. 43–49, Feb 1978.
- [23] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series." in *KDD Workshop*, U. M. Fayyad and R. Uthurusamy, Eds. AAAI Press, 1994, pp. 359–370.
- [24] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh, "Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping," *ACM Trans. Knowl. Discov. Data*, vol. 7, no. 3, pp. 10:1–10:31, Sep. 2013.
- [25] G. R. G. Lanckriet, N. Cristianini, P. Bartlett, L. E. Ghaoui, and M. I. Jordan, "Learning the kernel matrix with semidefinite programming," *J. Mach. Learn. Res.*, vol. 5, pp. 27–72, Dec. 2004.