Noname manuscript No.
(will be inserted by the editor)

QuicK-means

Accelerating Inference for K-Means by Learning Fast Transforms

Luc Giffon · Valentin Emiya · Hachem Kadri · Liva Ralaivola

Received: date / Accepted: date

Abstract `K-means`—and the celebrated Lloyd's algorithm—is more than the clustering method it was originally designed to be. It has indeed proven pivotal to help increase the speed of many machine learning, data analysis techniques such as indexing, nearest-neighbor search and prediction, data compression and, lately, inference with kernel machines. Here, we introduce an efficient extension of `K-means`, dubbed `QuicK-means`, that rests on the idea of expressing the matrix of the K cluster centroids as a product of sparse matrices, a feat made possible by recent results devoted to find approximations of matrices as a product of sparse factors. Using such a decomposition squashes the complexity of the matrix-vector product between the factorized $K \times D$ centroid matrix $\mathbf{U}$ and any vector from $\mathcal{O}(KD)$ to $\mathcal{O}(A \log B \ + B)$, with $A = \min(K, D)$ and $B = \max(K, D)$, where D is the dimension of the data. This drastic computational saving has a direct impact in the assignment process of a point to a cluster. We propose to learn such a factorization during the Lloyd's training procedure. We show that resorting to a factorization step at each iteration does not impair the convergence of the optimization scheme, and demonstrate the benefits of our approach experimentally.

This work was funded in part by the French national research agency (grant number ANR16-CE23-0006 "Deep in France").

Luc Giffon
Aix Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France
E-mail: luc.giffon@lis-lab.fr

Valentin Emiya
Aix Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France
E-mail: valentin.emiya@lis-lab.fr

Hachem Kadri
Aix Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France
E-mail: hachem.kadri@lis-lab.fr

Liva Ralaivola
Criteo AI Lab, Criteo, France
E-mail: l.ralaivola@criteo.com

1 Introduction

Training vs inference in `K-means`. `K-means` is one of the most popular clustering algorithms (Hartigan and Wong, 1979; Jain, 2010) and is widely used in many applications, such as indexing, data compression, nearest-neighbor search, and local network community detection (Muja and Lowe, 2014; Van Laarhoven and Marchiori, 2016). When combined with the Nyström approximation, `K-means` also proves pivotal to increase the speed and the accuracy of learning with kernel machines (Si et al., 2016) or RBF networks (Que and Belkin, 2016). In all these applications, the K D-dimensional centroids $\mathbf{u}_1, \ldots, \mathbf{u}_K$ returned by `K-means` are stacked on top of each other to form a matrix $\mathbf{U} \in \mathbb{R}^{K \times D}$ which is employed as a linear operator, that we call the *centroid operator*. From now on, we will use the term *training* or *learning* to refer to the execution of the clustering algorithm, i.e., the process that extracts centroids from training data (typically via the Lloyd algorithm); and we will use *inference* to refer to the task of assigning a (test) point to the learned centroids, relying on the application of the centroid operator to a given data point.

Complexity of `K-means`. On the one hand, the training phase of `K-means` has a $\mathcal{O}(NKD)$ complexity per iteration when the number of data points to cluster is N. On the other hand, the inference phase requires the application of the centroid operator $\mathbf{U}$ to some vector, which entails a $\mathcal{O}(KD)$ complexity for every input vector. With the ever increasing amount of data, it is critical to have at hand cost-effective alternatives to the computationally expensive conventional `K-means`.

Cost-effective `K-means`. Some techniques have already been proposed to alleviate the computational burden of Lloyd's algorithm. A popular approach is to embed the input observations in a lower dimensional space where one gets a cheap clustering which is then plugged back in the higher dimensional space to obtain an approximation of the `K-means` centroids (Boutsidis et al., 2014; Liu et al., 2017; Shen et al., 2017). Another idea is to use the triangle inequality in order to remove redundant distance measures while preserving the exact solution of `K-means` (Elkan, 2003; Hamerly, 2010). Also, a sketch of the input dataset can be used as input in the so-called compressive `K-means` procedure (Keriven et al., 2017), which scales to very large databases. However, these techniques focus on accelerating the training phase of `K-means` and not the inference phase, so that the resulting centroid operator is generally a dense matrix with the same dimensions as the `K-means` centroid operator. The associated inference procedure shows the same $\mathcal{O}(KD)$ complexity per input vector. Being able to reduce the cost of the inference phase of `K-means` has received relatively less attention compared to the problem of containing the cost of the training phase, whereas it is of prominent interest not only from a sheer algorithmic point of view but also for real-world applications where the number of query observations may be unbounded. To our knowledge, Sculley (2010) proposed the only candidate method that tackles the problem of efficiency at inference time using sparsity. Their method consists in featuring each iteration of `K-means` with an ℓ_1 projection step of the centroids, thus producing a sparse centroid operator which can be used for fast inference. Here, we follow this same line of research by proposing an extension to `K-means` that learns the centroid operator as a *fast transform*.

Learning fast transforms. Fast transforms have recently received increased attention in machine learning, as they can be employed with a computational cost several orders of magnitude lower than the usual matrix-vector product. For instance, well-known Haar and Hadamard transforms have been used to speed up random features calculation (Le et al., 2013) and improve landmark-based approximations (Si et al., 2016). The question to learn data-dependent fast transforms has been addressed in several recent works (Ailon et al., 2020; Dao et al., 2019; Le Magoarou and Gribonval, 2016; Vahid et al., 2020) with the key idea to rely on the sparse factorization structure of fast linear transforms to reduce the computational burden. In (Ailon et al., 2020; Vahid et al., 2020), the recursive *butterfly* structure of the aforementioned fixed transforms (Li et al., 2015) is used to design deep architectures with low-complexity convolutional layers (Vahid et al., 2020) and fast dense layers for encoder-decoder networks (Ailon et al., 2020)—there, the sparse supports is fixed by the butterfly model, and only the values (not the locations) of the non-zero coefficients are learned at training time. The algorithm proposed in Dao et al. (2019) learns a factorization based on variants of the butterfly structure, as combinations of a small number of allowed permutation matrices, which makes it possible to adapt the sparse supports to the data within a limited number of combinations. Le Magoarou and Gribonval (2016) approached the problem differently and did not restrict their model to the butterfly structure. Instead, they proposed a procedure to approximate any matrix as a product of matrices with adaptive sparse patterns. This procedure is key to our contribution.

Contributions. Getting inspiration from Le Magoarou and Gribonval (2016), we investigate computationally-efficient variants of `K-means` by learning fast transforms from the data. After introducing the background elements in Section 2, we make the following contributions:

- we introduce `QK-means`, an extension of `K-means` that learns the centroid matrix in form of a fast transform which entails a reduction of the inference complexity from $\mathcal{O}(AB)$ to $\mathcal{O}(A \log B + B)$, where $A = \min(K, D)$ and $B = \max(K, D)$ (Section 3.1);
- we show that each update step of our algorithm reduces the overall objective, which establishes the convergence of `QK-means` (Section 3.2);
- we perform an empirical evaluation of `QK-means` performance on different datasets in the contexts of clustering, nearest neighbor search and kernel Nyström approximation (Section 4); here we show the improvements of `QK-means` in time and space complexity for the inference and provide detailed results on the quality of the obtained centroids in terms of clustering loss and of other relevant classification-based metrics.

2 Preliminaries

We briefly review the basics of `K-means` and give background on learning fast transforms. To assist the reading, the notations used in the paper are listed in Table 1.

Symbol	Meaning
$[\![M]\!]$	set of integers from 1 to M
$\|\cdot\|$	L_2-norm
$\|\cdot\|_F$	Frobenius norm
$\|\cdot\|_0$	L_0-norm
$\|\cdot\|_2$	spectral norm
$\mathbf{D_v}$	diagonal matrix with vector $\mathbf{v}$ on the diagonal
$[\mathbf{v}]_k$	k^{th} element of vector $\mathbf{v}$
$[\mathbf{W}]_i, j$	element at the i^{th} row and j^{th} column of matrix $\mathbf{W}$
D	data dimension
K	number of clusters
Q	number of sparse factors
$\mathbf{x}_1, \ldots, \mathbf{x}_N$	data points
$\mathbf{X} \in \mathbb{R}^{N \times D}$	data matrix
$\mathbf{t}$	cluster assignment vector
$\mathbf{u}_1, \ldots, \mathbf{u}_K$	`K-means` centroids
$\mathbf{U} \in \mathbb{R}^{K \times D}$	`K-means` centroid matrix
$\mathbf{v}_0, \ldots, \mathbf{v}_K$	`QK-means` centroids
$\mathbf{V} \in \mathbb{R}^{K \times D}$	`QK-means` centroid matrix
$\mathbf{S}_0, \ldots, \mathbf{S}_Q$	sparse matrices
$\mathcal{E}_1, \ldots, \mathcal{E}_Q$	sparsity constraint sets
$\delta_{\mathcal{E}}$	indicator functions for set $\mathcal{E}$

Table 1: Notation used in this paper.

2.1 `K-means`: Lloyd's algorithm and inference

The `K-means` problem aims to partition a set $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ of N vectors $\mathbf{x}_n \in \mathbb{R}^D$ into a predefined number K of clusters by minimizing the distance from each vector $\mathbf{x}_n$ to the centroid of its cluster—the optimal centroid $\mathbf{u}_k$ of cluster k being the mean vector of the points assigned to this cluster. The optimization problem of `K-means` is

$$\arg\min_{\mathbf{U},\mathbf{t}} \sum_{k \in [\![K]\!]} \sum_{n:t_n=k} \|\mathbf{x}_n - \mathbf{u}_k\|^2, \tag{1}$$

where $\mathbf{U} = \{\mathbf{u}_1, \ldots, \mathbf{u}_K\}$ is the set of centroids and $\mathbf{t} \in [\![K]\!]^N$ is the vector that assigns $\mathbf{x}_n$ to cluster k if $t_n = k$.

Lloyd's algorithm (a.k.a. `K-means` algorithm). The most popular procedure to approximately solve the `K-means` problem is Lloyd's algorithm, often simply referred to as `K-means` — as in here. It alternates between i) an assignment step that decides the current cluster to which each point $\mathbf{x}_n$ belongs and ii) a re-estimation step which adjusts the cluster centroids. After an initialization of the set $\mathbf{U}^{(0)}$ of K cluster centroids, the algorithm proceeds as follows: at iteration τ, the assignments

are updated as $\forall n \in [\![N]\!]$:

$$\begin{aligned} t_n^{(\tau)} &\leftarrow \arg\min_{k \in [\![K]\!]} \left\| \mathbf{x}_n - \mathbf{u}_k^{(\tau-1)} \right\|_2^2 \\ &= \arg\min_{k \in [\![K]\!]} \left\| \mathbf{u}_k^{(\tau-1)} \right\|_2^2 - 2 \left\langle \mathbf{u}_k^{(\tau-1)}, \mathbf{x}_n \right\rangle \\ &= \arg\min_{k \in [\![K]\!]} \left\| \mathbf{u}_k^{(\tau-1)} \right\|_2^2 - 2 \left[\mathbf{U}^{(\tau-1)} \mathbf{x}_n \right]_k , \end{aligned} \tag{2}$$

and the cluster centroids are updated as

$$\mathbf{u}_k^{(\tau)} \leftarrow \hat{\mathbf{x}}_k(\mathbf{t}^{(\tau)}) := \frac{1}{n_k^{(\tau)}} \sum_{n : t_n^{(\tau)} = k} \mathbf{x}_n, \quad \forall k \in [\![K]\!] , \tag{3}$$

where $n_k^{(\tau)}$ is the number of points in cluster k at time τ and $\hat{\mathbf{x}}_k(\mathbf{t})$ is the mean vector of the elements of cluster k according to assignment $\mathbf{t}$.

Complexity of `K-means`*.* The process of assigning an observation to a cluster is a particular instance of the inference phase of `K-means`. It involves the application of the centroid matrix to a vector hence its complexity is $\mathcal{O}(DK)$ as in Equation (2). During training iterations of Lloyd's algorithm, the bottleneck of the total time complexity stems from the the assignment step: at each iteration, it requires this procedure to be repeated over the whole dataset for a total cost of $\mathcal{O}(NDK)$ while that of the centroids update (3) is $\mathcal{O}(ND)$. Similarly at inference time, assigning N' observations to the learned clusters has a complexity in $\mathcal{O}(N'DK)$.

Hence, reducing the complexity of the assignment step is a major challenge for training and inference. At training time, it may be addressed by existing methods as explained in Section 1. However, those methods do not reduce the complexity of the inference phase. Our main contribution rests on the idea that (2) may be computed more efficiently if the matrix $\mathbf{U}$ of centroids is approximated by a fast-transform matrix, which is learned thanks to a dedicated procedure that we discuss in the next section.

2.2 Learning fast-transform structures

Linear operators structured as products of sparse matrices. The popularity of some linear operators from $\mathbb{R}^M$ to $\mathbb{R}^M$ (with $M < \infty$) like the discrete Fourier or Hadamard transforms comes from both the mathematical meaning associated with their use (e.g. signal decomposition for the discrete Fourier transform) and their ability to compute the mapping of some input $\mathbf{x} \in \mathbb{R}^M$ with efficiency, typically in $\mathcal{O}(M \log M)$ in lieu of $\mathcal{O}(M^2)$ operations. An interesting feature of the related fast algorithms is that the matrix $\mathbf{U} \in \mathbb{R}^{M \times M}$ of such linear operators can be written as the product $\mathbf{U} = \Pi_{q \in [\![Q]\!]} \mathbf{S}_q$ of $Q = \mathcal{O}(\log M)$ sparse matrices $\mathbf{S}_q$ with $\|\mathbf{S}_q\|_0 = \mathcal{O}(M)$ non-zero coefficients per factor (Le Magoarou and Gribonval, 2016; Morgenstern, 1975): for any vector $\mathbf{x} \in \mathbb{R}^M$, $\mathbf{U}\mathbf{x}$ can thus be computed as $\mathcal{O}(\log M)$ products $\mathbf{S}_1 (\mathbf{S}_2 (\cdots (\mathbf{S}_Q \mathbf{x})))$ between a sparse matrix and a vector, the cost of each product being $\mathcal{O}(M)$, amounting to a $\mathcal{O}(M \log M)$ time complexity.

Approximating any matrix by learning a fast transform. When the linear operator $\mathbf{U}$ is an arbitrary matrix, one may approximate it with such a sparse-product structure by learning the factors $\{\mathbf{S}_q\}_{q\in[\![Q]\!]}$ in order to benefit from a fast algorithm. Algorithmic strategies have been proposed by Le Magoarou and Gribonval (2016) to learn such a factorization. Based on the proximal alternating linearized minimization (`PALM`) algorithm (Bolte et al., 2014), the `PALM` for Multi-layer Sparse Approximation (`palm4MSA`) algorithm (Le Magoarou and Gribonval, 2016) aims at approximating a matrix $\mathbf{U} \in \mathbb{R}^{K\times D}$ as a product of sparse matrices by solving

$$\min_{\{\mathbf{S}_q\}_{q\in[\![Q]\!]}} \left\| \mathbf{U} - \prod_{q\in[\![Q]\!]} \mathbf{S}_q \right\|_F^2 + \sum_{q\in[\![Q]\!]} \delta_{\mathcal{E}_q}(\mathbf{S}_q), \tag{4}$$

where for each $q \in [\![Q]\!]$, $\delta_{\mathcal{E}_q}(\mathbf{S}_q) = 0$ if $\mathbf{S}_q \in \mathcal{E}_q$ and $\delta_{\mathcal{E}_q}(\mathbf{S}_q) = +\infty$ otherwise. $\mathcal{E}_q$ is a constraint set that typically imposes a sparsity structure on its elements, as well as a scaling constraint. Although this problem is non-convex and the computation of a global optimum cannot be ascertained, the `palm4MSA` algorithm converges to a limiting value (more details on `palm4MSA` can be found in Section A of the Appendix).

3 `QuicK-means`

We now introduce our main contribution, `QuicK-means`, shortened in the sequel as `QK-means`. We depict the algorithm, we show its convergence property and we provide an analysis of its computational complexity.

3.1 `QK-means`: encoding centroids as products of sparse matrices

`QK-means` is a variant of `K-means` in which the matrix of centroids $\mathbf{U}$ is approximated as a product $\prod_{q\in[\![Q]\!]} \mathbf{S}_q$ of sparse matrices $\mathbf{S}_q$. Doing so will allow us to cope with the computational bulk imposed by the product $\mathbf{Ux}$ that shows up in the inference phase of `K-means`.

Building upon the `K-means` optimization problem (1) and fast-operator approximation problem (4) the `QK-means` optimization problem writes:

$$\underset{\{\mathbf{S}_q\}_{q=1}^{Q},\mathbf{t}}{\arg\min}\, g\left(\{\mathbf{S}_q\}_{q=1}^{Q}, \mathbf{t}\right) := \sum_{k\in[\![K]\!]} \sum_{n:t_n=k} \|\mathbf{x}_n - \mathbf{v}_k\|^2 + \sum_{q\in[\![Q]\!]} \delta_{\mathcal{E}_q}(\mathbf{S}_q), \tag{5}$$

where

$$\mathbf{V} = \prod_{q\in[\![Q]\!]} \mathbf{S}_q. \tag{6}$$

This is a constrained version of `K-means` (1) in which centroids $\mathbf{v}_k$ are constrained to form a matrix $\mathbf{V}$ with a fast-operator structure. The indicator functions $\delta_{\mathcal{E}_q}$ impose the sparsity of matrices $\mathbf{S}_q$. Details on modeling choices such as the

Algorithm 1 `QK-means` algorithm and its time complexity. Here $A := \min(K, D)$ and $B := \max(K, D)$

Require: $\mathbf{X} \in \mathbb{R}^{N \times D}$, K, initialization $\left\{\mathbf{S}_q^{(0)} : \mathbf{S}_q^{(0)} \in \mathcal{E}_q\right\}_{q \in [\![Q]\!]}$

1: Set $\mathbf{V}^{(0)} : \mathbf{x} \mapsto \prod_{q \in [\![Q]\!]} \mathbf{S}_q^{(0)} \mathbf{x}$
2: **for** $\tau = 1, 2, \ldots$ until convergence **do**
3: $\quad \mathbf{t}^{(\tau)} := \arg\min_{\mathbf{t} \in [\![K]\!]^N} \sum_{n \in [\![N]\!]} \left\| \mathbf{x}_n - \mathbf{v}_{t_n}^{(\tau-1)} \right\|^2$ $\quad \mathcal{O}\left(N\left(A \log B \ + B\right) + AB\right)$
4: $\quad \forall k \in [\![K]\!], \mathbf{u}_k^{(\tau)} := \frac{1}{n_k} \sum_{n : t_n^{(\tau)} = k} \mathbf{x}_n$ with $n_k^{(\tau)} := |\{n : t_n^{(\tau)} = k\}|$ $\quad \mathcal{O}(ND)$
5: $\quad \mathbf{A}^{(\tau)} := \mathbf{D}_{\sqrt{\mathbf{n}^{(\tau)}}} \times \mathbf{U}^{(\tau)}$ $\quad \mathcal{O}(KD)$
6: $\quad \mathcal{E}_0 := \left\{\mathbf{D}_{\sqrt{\mathbf{n}}}\right\}$
7: $\quad \left\{\mathbf{S}_q^{(\tau)}\right\}_{q=0}^{Q} := \arg\min_{\{\mathbf{S}_q\}_{q=0}^{Q}} \left\| \mathbf{A}^{(\tau)} - \prod_{q=0}^{Q} \mathbf{S}_q \right\|_F^2 + \sum_{q=0}^{Q} \delta_{\mathcal{E}_q}(\mathbf{S}_q)$ $\quad \mathcal{O}\left(AB\left(\log^2 B\right)\right)$
8: $\quad$ Set $\mathbf{V}^{(\tau)} : \mathbf{x} \mapsto \prod_{q \in [\![Q]\!]} \mathbf{S}_q^{(\tau)} \mathbf{x}$ $\quad \mathcal{O}(1)$
9: **end for**

Ensure: assignement vector $\mathbf{t}$ and sparse matrices $\{\mathbf{S}_q : \mathbf{S}_q \in \mathcal{E}_q\}_{q \in [\![Q]\!]}$

dimension of the inner factors or the projection procedure for sparsity are given in Section 4.1.

Problem (5) can be solved using Algorithm 1, which proceeds in a similar way as Lloyd's algorithm by alternating an assignment step at line 3 and an update of the centroids at lines 4–8. The assignment step can be computed efficiently thanks to the structure of $\mathbf{V}$. In practice, the update of the centroids relies on the `palm4MSA` algorithm to learn a fast-operator $\mathbf{V}$ that approximates the true centroid matrix $\mathbf{U}$ weighted by the number of examples n_k assigned to each cluster k.

The keen reader might suggest a simpler strategy that consists in applying the `palm4MSA` procedure directly on the centroid operator learned by the standard `K-means` algorithm; this would also produce a centroid operator expressed as a fast-transform. However by doing so, one would only find an approximation of the `K-means` solution expressed as a product of sparse matrices and not necessarily solve the problem of Equation (5). In Section 3.2, we provide theoretical guarantees showing that the `QK-means` procedure converges to a limit of the objective function in Equation (5). This theoretical result is illustrated in our experiments in Section 4.3 in which we demonstrate that the objective value produced by `QK-means` is better than that of `palm4MSA` applied on top of `K-means`.

3.2 Convergence of `QK-means`

Similarly to `K-means`, `QK-means` converges locally as stated in the following proposition.

Proposition 1 *The iterates* $\left\{\mathbf{S}^{(\tau)}\right\}_{q \in [\![Q]\!]}$ *and* $\mathbf{t}^{(\tau)}$ *in Algorithm 1 are such that,* $\forall \tau \geq 1$

$$g\left(\left\{\mathbf{S}_q^{(\tau)}\right\}_{q=1}^{Q}, \mathbf{t}^{(\tau)}\right) \leq g\left(\left\{\mathbf{S}_q^{(\tau-1)}\right\}_{q=1}^{Q}, \mathbf{t}^{(\tau-1)}\right),$$

which implies the convergence of the procedure.

Proof. We show that each step of the algorithm does not increase the overall objective function.

Assignment step (Line 3). For a fixed $\mathbf{V}^{(\tau-1)}$, the optimization problem at Line 3 is separable for each example indexed by $n \in [\![N]\!]$ and the new indicator vector $\mathbf{t}^{(\tau)}$ is thus defined as:

$$t_n^{(\tau)} = \underset{k \in [\![K]\!]}{\arg\min} \left\| \mathbf{x}_n - \mathbf{v}_k^{(\tau-1)} \right\|_2^2. \tag{7}$$

This step minimizes the first term in (5) w.r.t. $\mathbf{t}$ while the second term is constant so we have

$$g\left(\left\{\mathbf{S}_q^{(\tau-1)}\right\}_{q=1}^{Q}, \mathbf{t}^{(\tau)}\right) \leq g\left(\left\{\mathbf{S}_q^{(\tau-1)}\right\}_{q=1}^{Q}, \mathbf{t}^{(\tau-1)}\right). \tag{8}$$

Centroids update step (Lines 4–8). We now consider a fixed assignment vector $\mathbf{t}^{(\tau)}$. We first note that for any cluster k with true centroid $\mathbf{u}_k^{(\tau)}$ and any vectors $\mathbf{v}_k^{(\tau)}$, the following holds:

$$\begin{aligned}
&\sum_{n:t_n^{(\tau)}=k} \left\| \mathbf{x}_n - \mathbf{v}_k^{(\tau)} \right\|^2 = \sum_{n:t_n=k} \left\| \mathbf{x}_n - \mathbf{u}_k^{(\tau)} + \mathbf{u}_k^{(\tau)} - \mathbf{v}_k^{(\tau)} \right\|^2 \\
&= \sum_{n:t_n^{(\tau)}=k} \left(\left\| \mathbf{x}_n - \mathbf{u}_k^{(\tau)} \right\|^2 + \left\| \mathbf{u}_k^{(\tau)} - \mathbf{v}_k^{(\tau)} \right\|^2 - 2 \left\langle \mathbf{x}_n - \mathbf{u}_k^{(\tau)}, \mathbf{u}_k^{(\tau)} - \mathbf{v}_k^{(\tau)} \right\rangle \right) \\
&= \sum_{n:t_n^{(\tau)}=k} \left\| \mathbf{x}_n - \mathbf{u}_k^{(\tau)} \right\|^2 + n_k \left\| \mathbf{u}_k^{(\tau)} - \mathbf{v}_k^{(\tau)} \right\|^2 - 2 \left\langle \underbrace{\sum_{n:t_n^{(\tau)}=k} \left(\mathbf{x}_n - \mathbf{u}_k^{(\tau)} \right)}_{=0}, \mathbf{u}_k^{(\tau)} - \mathbf{v}_k^{(\tau)} \right\rangle \\
&= \sum_{n:t_n^{(\tau)}=k} \left\| \mathbf{x}_n - \mathbf{u}_k^{(\tau)} \right\|^2 + \left\| \sqrt{n_k} \left(\mathbf{u}_k^{(\tau)} - \mathbf{v}_k^{(\tau)} \right) \right\|^2.
\end{aligned} \tag{9}$$

For $\mathbf{t}^{(\tau)}$ fixed, the new sparsely-factorized centroids $\left\{\mathbf{S}_q^{(\tau)}\right\}_{q=1}^{Q}$ are set as solutions of the problem $\arg\min_{\mathbf{S}_1,\ldots,\mathbf{S}_Q} g(\mathbf{S}_1, \ldots, \mathbf{S}_Q, \mathbf{t}^{(\tau)})$ which rewrites, thanks to (9) as

$$\begin{aligned}
\left\{\mathbf{S}_q^{(\tau)}\right\}_{q=1}^{Q} = \underset{\mathbf{S}_1,\ldots,\mathbf{S}_Q}{\arg\min} & \left\| \mathbf{D}_{\sqrt{\mathbf{n}^{(\tau)}}} (\mathbf{U}^{(\tau)} - \mathbf{V}) \right\|_F^2 + \sum_{k \in [\![K]\!]} c_k^{(\tau)} + \sum_{q \in [\![Q]\!]} \delta_q(\mathbf{S}_q) \\
& \text{s. t. } \mathbf{V} = \prod_{q \in [\![Q]\!]} \mathbf{S}_q,
\end{aligned} \tag{10}$$

where:

- $\mathbf{D}_{\sqrt{\mathbf{n}^{(\tau)}}}$ is the diagonal matrix with $\sqrt{\mathbf{n}^{(\tau)}}$ on the diagonal and $\forall k \in [\![K]\!]$, the k^{th} element of $\mathbf{n}^{(\tau)}$ is $n_k^{(\tau)} := \left| \left\{ n : t_n^{(\tau)} = k \right\} \right|$, *i.e.* the number of observation in cluster k at step τ. Hence $\mathbf{D}_{\sqrt{\mathbf{n}^{(\tau)}}} (\mathbf{U}^{(\tau)} - \mathbf{V})$ is the matrix with $\sqrt{n_k^{(\tau)}} \left(\mathbf{u}_k^{(\tau)} - \mathbf{v}_k \right)$ as row k;

- $\mathbf{U}^{(\tau)} \in \mathbb{R}^{K \times d}$ refers to the unconstrained centroid matrix obtained from the data matrix $\mathbf{X}$ and the indicator vector $\mathbf{t}^{(\tau)}$: $\mathbf{u}_k := \frac{1}{n_k} \sum_{n:t_n=k} \mathbf{x}_n$ (see Line 4);
- $c_k^{(\tau)} := \sum_{n:t_n^{(\tau)}=k} \left\| \mathbf{x}_n - \mathbf{u}_k^{(\tau)} \right\|$ is constant w.r.t. $\mathbf{S}_1, \ldots, \mathbf{S}_Q$.

We now introduce $\mathbf{A}^{(\tau)} := \mathbf{D}_{\sqrt{\mathbf{n}^{(\tau)}}} \mathbf{U}^{(\tau)}$ as the unconstrained centroid matrix re-weighted by the size of each cluster (see Line 5), so (10) can be simplified in:

$$\left\{ \mathbf{S}_q^{(\tau)} \right\}_{q=1}^{Q} = \underset{\mathbf{S}_1, \ldots, \mathbf{S}_Q}{\arg\min} \left\| \mathbf{A}^{(\tau)} - \prod_{q \in \{[\![Q]\!] \cup \{0\}\}} \mathbf{S}_q \right\|_F^2 + \sum_{q \in \{[\![Q]\!] \cup \{0\}\}} \delta_q(\mathbf{S}_q), \tag{11}$$

where the extra factor $\mathbf{S}_0$ is constrained to $\mathbf{D}_{\sqrt{\mathbf{n}^{(\tau)}}}$ by setting $\mathcal{E}_0$ to a singleton at Line 6.

Starting the minimization process at the previous estimates $\left\{ \mathbf{S}_q^{(\tau-1)} \right\}_{q \in [\![Q]\!]}$, we get

$$g(\mathbf{S}_1^{(\tau)}, \ldots, \mathbf{S}_Q^{(\tau)}, \mathbf{t}^{(\tau)}) \leq g(\mathbf{S}_1^{(\tau-1)}, \ldots, \mathbf{S}_Q^{(\tau-1)}, \mathbf{t}^{(\tau)}), \tag{12}$$

and plugging (12) back with (8), we finally have, for any τ:

$$g\left(\mathbf{S}_1^{(\tau)}, \ldots, \mathbf{S}_Q^{(\tau)}, \mathbf{t}^{(\tau)}\right) \leq g\left(\mathbf{S}_1^{(\tau-1)}, \ldots, \mathbf{S}_Q^{(\tau-1)}, \mathbf{t}^{(\tau)}\right) \leq g\left(\mathbf{S}_1^{(\tau-1)}, \ldots, \mathbf{S}_Q^{(\tau-1)}, \mathbf{t}^{(\tau-1)}\right),$$

which is a sufficient condition to assert that Algorithm 1 converges, *i.e.* the sequence of objective values is non-increasing and lower bounded, thus convergent. Note however that the preceding development proves that the objective function value converges to a limiting value but it doesn't guarantee that the centroids themselves converge to some fixed position. In fact, just like with the `K-means` algorithm, the centroids might oscillate forever while the objective value remains unchanged. □

3.3 Complexity analysis

To analyze the complexity, we set $A \doteq \min(K, D)$ and $B \doteq \max(K, D)$, and assume that the number of factors satisfies $Q = \mathcal{O}(\log B)$. The analysis is proposed under the following assumptions: the product between two dense matrices of shapes $N_1 \times N_2$ and $N_2 \times N_3$ can be computed in $\mathcal{O}(N_1 N_2 N_3)$ operations; the product between a sparse matrix with $\mathcal{O}(S)$ non-zero entries and a dense vector costs $\mathcal{O}(S)$ operations; the product between two sparse matrices of shapes $N_1 \times N_2$ and $N_2 \times N_3$, both having $\mathcal{O}(S)$ non-zero values costs $\mathcal{O}(S \min(N_1, N_3))$ and the number of non-zero entries in the resulting matrix is $\mathcal{O}(S^2)$.

Complexity of `K-means`. Recall that the complexity `K-means` is dominated by its cluster assignment step, requiring $\mathcal{O}(NKD) = \mathcal{O}(NAB)$ operations (see Eq. 2).

Complexity of `palm4MSA`. As `QK-means` extends `K-means` with a `palm4MSA` step, we start by providing complexity for this algorithm. The procedure consists in an alternate optimization of each sparse factor. At each iteration, the whole set of Q factors is updated with a cost in $\mathcal{O}\left(AB\left(\log^2 B\right)\right)$, as detailed in Section A of the Appendix. The bottleneck is the computation of the gradient, which benefits from fast computations with sparse matrices.

Dataset	Data dim. D	# classes	Train size N	Test size N'
`MNIST` (LeCun et al., 2010)	784	10	60 000	10 000
`Fashion-MNIST` (Xiao et al., 2017)	784	10	60 000	10 000
`Caltech256` (Griffin et al., 2007)	2 352	256	19 952	9 828
`Breast-cancer` (Dua and Graff, 2017)	30	2	469	100
`Coverage Type` (Dua and Graff, 2017)	54	7	576 012	5 000
`Coil20` (Nene et al., 1996)	1024	10	964	476
`Kddcup99` (Dua and Graff, 2017)	116	23	4 893 431	5 000

Table 2: Main features of the datasets. Discrete, unordered attributes for dataset Kddcup99 have been encoded as one-hot attributes. For the Caltech and Coil dataset, the images have been resized to 32×32 images.

Complexity of `QK-means` training. The complexity of each iteration of `QK-means` is $\mathcal{O}\left(N\left(A \log B \ + B\right) + AB \log^2 B\right)$. The complexities of the main steps are given in Algorithm 1.

The assignment step (line 3 and Eq. 2) benefits from the fast computation of $\mathbf{VX}$ in $\mathcal{O}\left(N\left(A \log B \ + B\right)\right)$ while the computation of the norms of the cluster centroids is in $\mathcal{O}\left(AB\right)$.

The computation of the centroids of each cluster at line 4 is the same as in `K-means` and takes $\mathcal{O}\left(ND\right)$ operations.

The update of the fast transform, in lines 5 to 8 is a computational overload compared to `K-means`. Its time complexity depends on the chosen procedure to solve the minimization problem (11); but in the case where the `palm4MSA` procedure is employed, itis dominated by the update of the sparse factors at line 7, in $\mathcal{O}\left(AB \log^2 B\right)$. Note that, if A and B have the same order of magnitude, the overall cost of `QK-means` is dominated by the cost of the assignment step as soon as the number of examples N is greater than $B \log^2 B$. One can see that the computational bottleneck of `K-means` is here reduced. Asymptotically, the `QK-means` *training phase* is computationally more efficient than `K-means`. In practice, this gain may only be observed when N, K and D are very large. We thus focus on the *inference phase* in the experiments.

Complexity of `QK-means` inference. Inference for `QK-means` consists in applying the resulting fast operator to a single observation. This is done by multiplying the test point with the resulting sparse factorization which costs $\mathcal{O}(A \log B + B)$ instead of $\mathcal{O}(KD)$ obtained during `K-means` inference. Note that this time complexity is directly linked to the space complexity of `QK-means` operator which is also of $\mathcal{O}(A \log B + B)$.

4 Experiments and applications

We perform a series of experiments to demonstrate the efficiency of our approach. We assess the benefits of our method from two perspectives. First, we evaluate the gain during inference in term of space – the number of non-zero values in our factorization – and computational cost – the number of floating point operations (FLOP) – induced by the use of the sparse factorization of the centroid matrix. Second, we validate the quality of the obtained centroids with respect to the `K-means` clustering loss (1) and other relevant classification-based metrics when some class information is

available. Altogether, these experiments show that `QK-means` generates a centroid operator that achieves a better *quality/complexity* ratio for inference than `K-means` and the available baseline.

The rest of this experimental section is organized as follows. Section 4.1 provides implementation details of our method; in Section 4.2, we study the influence of the hyperparameters involved by our method on the objective function (Equation 1); the quality of the estimated centroids is illustrated in Section 4.3; in Section 4.4, we demonstrate the savings in terms of space and computational complexity associated with the use `QK-means` centroids during inference compared to that of `K-means` and the ℓ_1-based sparse version of (Sculley, 2010). Finally, Section 4.5 and Section 4.6 show that `QK-means` learns a centroid matrix that, when used in inference for the purpose of nearest-neighbor search and efficient Nyström approximation, does not imply degraded performances with respect to `K-means`.

4.1 Experimental setting

Competing methods. We compare our method to two methods: the vanilla version of `K-means` and to the ℓ_1-based sparse method proposed by Sculley (2010). The vanilla version actually encompasses the core `K-means` algorithm and its accelerated variants proposed by Elkan (2003); Hamerly (2010) that make clever use of the triangle inequality during the training phase but which, ultimately, provide exactly the same centroids and clusters as the base `K-means`—in the sequel, we may use the notation `K-means`* to denote the vanilla `K-means` and the variants just mentioned. The method of Sculley (2010) differs from the reference algorithm as it projects the current centroids on a λ-radius ℓ_1-ball at each iteration, an operation that reduces the number of non-zero coefficients in the centroid operator; as for our method, the resulting time and space inference cost of the learned model is lowered. In the experiment section, the `K-means` algorithm with ℓ_1 projection is referred to as *`K-means` ℓ_1 Proj.* To our knowledge, there is no other method in the literature that is concerned with accelerating the inference time of `K-means` using sparsity. For the sake of completeness, we might mention other training acceleration strategies, in addition to those of Elkan (2003); Hamerly (2010): Boutsidis et al. (2014); Liu et al. (2017); Shen et al. (2017) proposed approaches that project the input data points in (low-dimensional) spaces where the training of `K-means` is faster, however, when retrieving the centroid in the initial space, there is nothing done by those approaches that would result in a speed up of the inference process. This is the reason why those methods, given the perspective that we take regarding inference acceleration, are not considered as competing methods; this explains why we do not provide any result tied to those approaches.

Implementation details. The code has been written in Python, including the `palm4MSA` algorithm. Fast operators $\mathbf{V}$ based on sparse matrices $\mathbf{S}_q$ are implemented with `csr_matrix` objects from the `scipy.linalg` package. While more efficient implementations may be beneficial for larger deployment, our implementation[1] is sufficient as a proof of concept for assessing the performance of the proposed approach as illustrated by the count of FLOPs in Section 4.4.

[1] Available online at `https://github.com/lucgiffon/qkmeans`.

Datasets. Our experiments are conducted on seven reference datasets to show the quality of the obtained centroids and the relative reduction in FLOP count offered by our method `QK-means` when the number of clusters and the dimensionality of the data grows. See Table 2 for details on the datasets. No particular pre-processing was performed for any of the methods.

Algorithm settings. Let $A \doteq \min(K, D)$ and $B \doteq \max(K, D)$. `QK-means` is used with $Q \doteq \log_2(B)$ sparse factors $\mathbf{S}_q$ and all factors are with shape $A \times A$ except the leftmost one, $\mathbf{S}_1$ with shape $K \times A$, and the rightmost one, $\mathbf{S}_Q$ with shape $A \times D$. The sparsity constraint of each factor $\mathbf{S}_q$ is set in $\mathcal{E}_q$ and is governed by a global parameter denoted as *sparsity level*, which indicates the desired number of non-zero coefficients in each row and in each column of $\mathbf{S}_q$; the impact of this parameter is discussed in Section 4.2. Since the projection onto this set of structured-sparsity constraints may be computationally expensive, this projection is relaxed in the implementation of `palm4MSA` and only guarantees that the number of non-zero coefficients in each row and each column is at least the sparsity level, as in Le Magoarou and Gribonval (2016). The actual number of non-zero coefficients in the sparse factors is measured at the end of the optimization process and reported in the results. Additional details about `palm4MSA` are given in Section A of the Appendix or can be found in `palm4MSA` original paper (Le Magoarou and Gribonval, 2016). The stopping criterion of `K-means` and `QK-means` consists in a maximum number of iterations set to 50, which was a sufficient number to reach convergence of the objective criterion in all datasets. The stopping criterion for `palm4MSA` consists of a tolerance set to 10^{-6} on the relative variation of the objective function and a maximum number of iterations; the influence of this maximum number of iterations on the objective loss is studied in Section 4.2. Each experiment have been replicated 5 times using different seed values for random generators. Competing techniques share the same seed values so that they are initialized with the same centroids. We evaluate two initialization strategies: the uniform sampling and the `K-means++` methods (Arthur and Vassilvitskii, 2006). As discussed in Section 4.2, our method also benefits from this latter initialization technique used for faster convergence and better clustering accuracy. The sparse factors used for the initialization of the `QK-means` algorithm were obtained by using once *Hierarchical*-`palm4MSA` on the initialized centroid matrix. *Hierarchical*-`palm4MSA` is an heuristic for `palm4MSA` proposed by Le Magoarou and Gribonval (2016). This heuristic is more expensive than `palm4MSA` but has been shown empirically to provide better approximation results. It is used in place of `palm4MSA` for the initialization but not inside `QK-means` because it hasn't shown much improvement of the performance while being time consuming. For the very first initialization of `palm4MSA` inside *Hierarchical*-`palm4MSA`, we use the same initialization as the one proposed by Le Magoarou and Gribonval (2016), that is: all factors are initialized to identity, except for the first one, which is fully initialized to zero.

The `K-means` ℓ_1 Proj. variant of Sculley (2010) uses two hyperparameters: λ as the radius of the ℓ_1 ball and ϵ as a tolerance parameter. We here set $\epsilon = 0.01$, as in the original paper. For a fair comparison, the value of λ has been adjusted for each dataset so that the obtained compression rate is the same as that of the `QK-means` method with a sparsity level of 3. In Table 4, the λ values for each dataset are 2550 for `MNIST`, 1485 for `Fashion-MNIST`, 600 for `Breast-cancer`, 972 for `Coverage type`, 5.6 for Coil20, 4.9 for `Kddcup99` and 868 for `Caltech`. In Figures 3 and 4,

concerning the Coverage Type dataset, the λ values for each cluster numbers are 4730 for 8 clusters, 3528 for 16 clusters, 2353 for 32 clusters, 1918 for 64 clusters, 1112 for 128 clusters and 968 for 256 clusters.

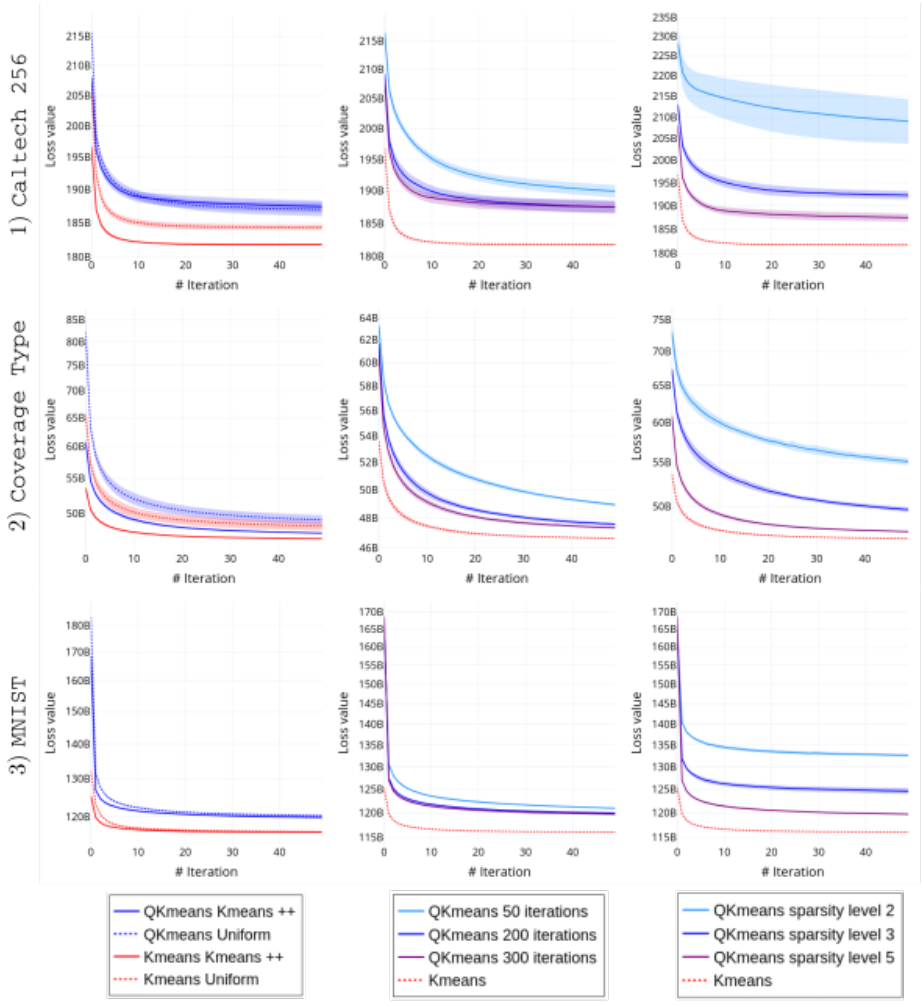
4.2 Influence of hyper-parameters

In this Section, we analyze the influence of the initialization procedure for the centroids in the `QK-means` algorithm, the number of iterations in the `palm4MSA` algorithm and the sparsity level in sparse factors.

Centroids initialization. Just like for the `K-means` algorithm, the `QK-means` solution is only guaranteed to converge to a local stationary point of the objective function. It means that the `QK-means` algorithm is also sensible to a good initialization. In practice, we show in the left-most column of Figure 1 that the `Kmeans++` initialization (Arthur and Vassilvitskii, 2006) provides the same benefit for the optimization in the `QK-means` algorithm as in the `K-means` algorithm, that is: a quicker convergence and lower objective function value. We hence use the `kmeans++` initialization scheme for the rest of the experiments.

palm4MSA *number of iterations.* The `palm4MSA` algorithm is an iterative algorithm which comes with convergence guarantees, provided that it is given enough iterations to reach the limiting value. To this end, one can choose a threshold value for the relative objective function difference between two successive iterations, as well as a maximum number of iterations to stop the optimization when it gets too long. In the middle column of Figure 1, we show the influence of the maximum number of iterations in `palm4MSA`. We see that beyond a given number of iterations, one doesn't get much improvement in the objective function value and convergence rate. Note also that, sometimes, the objective function of `QK-means` may undergo isolated «bumps» which we explain by the non-monotonous nature of the `palm4MSA` objective, possibly resulting in a worse result when the initialization is already close to the stationary point. This behavior was most often observed when the sparsity level and/or the number of iterations in `palm4MSA` were small. This is a minor phenomenon that is smoothed out when averaging results over replicates and we use 300 iterations for `palm4MSA` in the rest of the experiments, which empirically showed to generally prevent these «bumps».

Sparsity level. We call sparsity level the approximate number of non-zero values in each row and each column of the sparse matrices constituting the final centroid operator. We will later show that the choice of a lower sparsity level produces a higher compression rate for the centroids operator (Figure 3) but this happens at the expense of a higher value of the objective criterion and a slower convergence rate. Indeed, the right column in Figure 1 shows that on the three considered datasets, the lower the sparsity budget, the higher the final value of the objective function.



a) Influence of centroids initialization

b) Influence of the number of iterations in `palm4MSA`

c) Influence of the sparsity level in sparse factors

Fig. 1: Average and standard deviation (shaded area) of the loss value with respect to the iteration index in various settings of `QK-means`. Only three datasets are considered here but the same general patterns can be observed on all the datasets. When it is not the varying parameter, the initialization is `Kmeans++`, we use 300 `palm4MSA` iterations and the sparsity level is 5.

4.3 Clustering quality

An important question is the ability of the fast-structure model to fit arbitrary data. In order to assess the clustering quality of `QK-means`, we start by comparing the clustering objective criterion obtained with `QK-means` compared to i) `K-means` and its accelerated exact variants, i.e. `K-means`*, ii) `K-means` ℓ_1 Proj. and iii)

palm4MSA applied subsequently to the centroid matrix obtained by K-means (named K-means + palm4MSA). In Table 3, we first show that in all the considered datasets, the objective value (1) attained by QK-means is always lower than the ones achieved by K-means + palm4MSA and K-means ℓ_1 Proj. Note that, for a fair comparison, we used the *Hierarchical* version of palm4MSA in K-means + palm4MSA, just like in the initialization procedure of QK-means. This experiment illustrates the claim made in Section 3.1 that K-means + palm4MSA is a mere approximation of the K-means solution as a product of sparse factors whereas QK-means actually minimizes the objective of Equation (5) throughout the learning process. It also shows that the QK-means solution is closer by several orders of magnitude to the reference loss of K-means, in comparison to K-means ℓ_1 Proj.

Dataset	K-means*	K-means ℓ_1 Proj.	QK-means	K-means + palm4MSA
Caltech256 32	**1.818e11**	3.874e11	<u>1.875e11</u>	1.953e11
Breast Cancer	**5.579e5**	7.620e7	<u>5.645e5</u>	5.861e5
Mnist	**1.161e11**	1.749e11	<u>1.198e11</u>	1.209e11
Coil20 32	**1.332e4**	6.001e4	<u>1.661e4</u>	1.804e4
Kddcup99	**6.107e3**	3.503e5	<u>1.367e4</u>	3.552e4
Coverage Type	**4.663e10**	1.182e12	<u>4.734e10</u>	5.068e10
Fashion Mnist	**8.454e10**	2.456e11	<u>8.946e10</u>	9.224e10

Table 3: Objective loss achieved by K-means*, K-means ℓ_1 Proj., QK-means and palm4MSA applied on top of K-means. K-means* refers to the standard Lloyd algorithm and its accelerated variants that preserve the same result (see text). Results with palm4MSA were obtained with a sparsity level of 5. Best results are bold, second best are underlined.

Adjusted Mutual Information (Vinh et al., 2010) is a mutual information-based metric that can measure the difference between two clusterings, and it may be used to evaluate the soundness of a clustering when labels for the input data are available (and used as cluster indicators). The last three lines of Table 4 reports the AMI value between the actual labels of the datasets and the clusters given by the K-means, QK-means and K-means ℓ_1 Proj. algorithms. This metric (the higher, the better) shows that the clusters given by QK-means attain AMI values close to those of K-means and significantly better than those of K-means ℓ_1 Proj.

The approximation quality of the centroids can also be assessed visually, in a more subjective and interpretable way, in Figure 2, where the obtained centroids on the MNIST dataset are displayed as images. Although some degradations may be observed in some images, one can note that each image obtained with QK-means clearly represents a single visual item without noticeable interference with other items.

4.4 Compression performance

We here discuss the space and computational gain entailed by the use of QK-means centroid operator for inference in comparison to K-means and K-means ℓ_1 Proj.

Since running times are highly dependent on the implementation and the specifics of the computer used, we show in Table 4 the number of FLOPs required

(a) K-means centroids. (b) QK-means centroids.

Fig. 2: Visual representation of the K-means (left) and QK-means (right) centroid for the MNIST dataset for $K = 30$ clusters. The images were obtained with a sparsity level of 5.

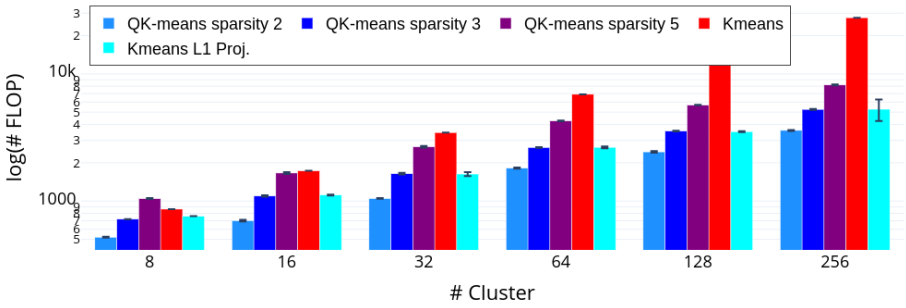


Fig. 3: Coverage Type: Number of FLOPs with respect to the number of clusters and the *sparsity level*, e.g. the minimum number of non-zero values in each row and each column. Note that this figure would be exactly the same for the count of non-zero values as its link to the number of FLOPs is linear.

for the QK-means, K-means and K-means ℓ_1 Proj. centroid operators to be applied to some vector along with the corresponding number of non-zero values in these operators.

We also show the compression rate between K-means and QK-means as well as the compression rate betwen K-means and K-means ℓ_1 Proj.. One can note that the compression rate is the same for the number of FLOPs and the number of non-zero values as the number of FLOPs is twice the number of non-zero values. Also, we remark here that the λ value for K-means ℓ_1 Proj. has been properly adjusted to get the same compression rate than with QK-means. These results give a clear illustration for the complexity advantage of the QK-means method: the greater the scale (K and/or D), the greater the compression rate, as with Caltech256 dataset where the compression rate reaches $\times 32.7$. It can be noticed that, for QK-means, the count of non-zero values and FLOPs may vary for two datasets with the same D and K such as with MNIST and Fashion-MNIST. This behavior is caused by our projection procedure which does not guarantee a fixed number of non-zero values in each projected factor (See Section A.1 of the Appendix).

Figure 3 shows how the number of clusters have a relatively-low impact on the number of FLOPs induced by the factorized operator compared to that of the standard dense matrix generated by K-means. As expected, we also see that increasing the *sparsity level* also increases the number of FLOPs. This figure shows

	Algorithm	MNIST D=784 K=64	F.-MNIST D=784 K=64	B.-cancer D=30 K=64	Covtype D=54 K=256	Coil20 D=1024 K=64	Kddcup99 D=116 K=256	Caltech D=2352 K=256
# FLOPS	K-means*	100 352	100 352	3 840	27 648	131 072	59 392	1 572 864
	K-means ℓ_1 Proj.	9 554	9 175	1 832	5 256	11 230	8 176	48 181
	QK-means	9 581	9 181	1 809	5 254	11 170	8 117	48 174
# non-zero values	K-means*	50 176	50 176	1 920	13 824	65 536	29 696	786 432
	K-means ℓ_1 Proj.	4 777	4 587	916	2 628	5 615	4 088	24 090
	QK-means	4 790	4 590	904	2 627	5 585	4 058	24 087
Compression rate	$\frac{\text{K-means}^*}{\text{QK-means}}$	×10.5	×10.9	×2.1	×5.3	×11.7	×7.3	×32.7
	$\frac{\text{K-means}^*}{\text{K-means } \ell_1 \text{ Proj.}}$	×10.5	×10.9	×2.1	×5.3	×11.7	×7.3	×32.7
1-NN Accuracy	K-means*	0.9507	0.8353	0.916	0.9669	0.9794	0.9992	**0.1065**
	K-means ℓ_1 Proj.	0.9506	0.8307	0.916	0.9526	0.9571	0.9992	0.1014
	QK-means	0.9514	0.8353	0.914	**0.9673**	0.9782	**0.9993**	0.1007
	Ball-tree	**0.9690**	**0.8497**	**0.9280**	*N/A*	**0.9895**	*N/A*	*N/A*
Nyström approximation error	K-means*	**0.0322**	**0.0191**	0.0001	0.0001	**0.0218**	**0.0003**	**0.0138**
	K-means ℓ_1 Proj.	0.0565	0.1122	0.0385	0.0362	0.1383	0.0726	0.1000
	QK-means	0.0427	0.0299	**1e−5**	**3e−5**	0.0343	0.0008	0.0259
	Uniform	0.0673	0.0443	0.005	0.0002	0.0541	0.0051	0.0194
	Un. F-Nys.	0.1702	0.2919	0.0449	0.0582	0.2501	0.1509	0.2191
	K. F-Nys.	0.1576	0.2623	0.0598	0.0381	0.2371	0.1275	0.2382
Nyström + SVM Accuracy	K-means*	**0.9235**	**0.8185**	0.914	0.6830	0.9702	**0.9989**	**0.1694**
	K-means ℓ_1 Proj.	0.8984	0.8104	0.914	0.667	0.9622	0.9987	0.1332
	QK-means	0.9200	0.8119	0.914	**0.6848**	**0.9798**	0.9982	0.1588
	Uniform	0.905	0.8142	**0.9340**	0.6818	0.9546	0.9972	0.1575
	Un. F-Nys.	0.7937	0.7341	0.932	0.5936	0.7399	0.9944	0.0954
	K. F-Nys.	0.7337	0.6872	0.93	0.6061	0.6756	0.9948	0.0751
AMI	K-means*	**0.5523**	**0.4883**	**0.1790**	**0.1040**	0.7093	0.5800	**0.0989**
	K-means ℓ_1 Proj.	0.4496	0.4151	0.1601	0.0548	0.6785	0.5931	0.0729
	QK-means	0.5337	0.4769	0.1772	0.1010	**0.7124**	**0.6637**	0.0841

Table 4: Results of numerical experiments concerning the inference phase of the clustering: FLOP and parameter count; average Nyström transformation error for a sample set of size 5000; 1-nearest neighbor and SVM classification accuracy on top of Nyström transformation on the test set. "Un. F-Nys" and "K. F-nys" stand for the Fast-Nyström algorithm (Si et al., 2016) with uniform and K-means* sampling schemes respectively. The QK-means results are obtained using sparse factors with a sparsity level of 3. Best results are in bold shape while second best results are underlined (when necessary). "Brute" and "KD-tree" version of 1-NN are omitted since they always perform worse than "Ball-tree". Only results with the largest K value are displayed. Some results in the "1-NN Accuracy" category are Not Available (N/A) when the corresponding test necessited more than 10 times the duration of the same test with K-means*. AMI stands for "Adjusted Mutual Information"; it is computed between the clustering partition and the real labels in the data.

results on the Coverage Type dataset but the same compression patterns can be observed for all datasets.

4.5 Nearest-neighbor search

We show that the QK-means centroid operator is comparable to the K-means operator and better than the K-means ℓ_1 Proj. one when used to speed up the nearest-neighbor search.

The nearest-neighbor (1-NN) search is a fundamental task that suffers from computational limitations when the dataset is large and fast strategies have been proposed, e.g., using KD-trees or ball trees. One may also use a clustering strategy to perform an approximate nearest-neighbor search: the query is first compared to K centroids computed beforehand by clustering the whole dataset, and the nearest neighbor search is then performed among a lower number of data points, within the related cluster. We compare the results of approximate 1-NN using `K-means`, `K-means` ℓ_1 Proj. and `QK-means` to several `scikit-learn` implementations (Pedregosa et al., 2011) of the nearest-neighbor search: brute-force search, KD-tree, Ball-tree. In our experiments, Ball-tree implementation always perform equal or better than brute-force search and KD-tree implementations and is faster. The results for the brute-force search, KD-tree and Ball-tree are not available for some dataset (N/A) because they lasted more than 10 times the `K-means` search version in the same setting. We note that this happens on the largest datasets, hence emphasizing the usefulness of using the centroid operator to fasten searches in such datasets. We see that the prediction performance for classification isn't impaired much when using the `QK-means` sparse factorization instead of the `K-means` centroid matrix for partitioning. When available, we also see that using the `QK-means`-partitionned 1-NN, algorithm doesn't impair much the performance compared to the vanilla 1-NN algorithms. Finally, this batch of experiments shows that our method outperforms the `K-means` ℓ_1 Proj. approach in terms of the classification accuracy of 1-NN.

4.6 Nyström approximation

The centroids given by `K-means` are good landmark points for an accurate and efficient Nyström approximation (Si et al., 2016). In this section, we show how we can take advantage of the fast-operator obtained as output of our `QK-means` algorithm in order to lighten the computation in the Nyström approximation. We start by giving background knowledge on the Nyström approximation then we present some recent work aiming at accelerating it using well known fast-transform methods. We finally stem on this work to present a novel approach based on our `QK-means` algorithm.

4.6.1 Background on the Nyström approximation

Standard kernel machines are often impossible to use in large-scale applications because of their high computational cost associated with the kernel matrix $\mathbf{K}$ which has $O(N^2)$ storage and $O(N^2D)$ computational complexity: $\forall i, j \in [\![N]\!], [\mathbf{K}]_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$. A well-known strategy to overcome this problem is to use the Nyström method which computes a low-rank approximation of the kernel matrix on the basis of some pre-selected landmark points (Williams and Seeger, 2001).

Given $K \ll N$ landmark points $\{\mathbf{u}_i\}_{i=1}^K$, the Nyström method gives the following approximation of the full kernel matrix:

$$\mathbf{K} \approx \tilde{\mathbf{K}} = \mathbf{C}\mathbf{W}^\dagger\mathbf{C}^T, \tag{13}$$

with $\mathbf{W} \in \mathbb{R}^{K\times K}$ containing all the kernel values between landmarks: $\forall i, j \in [\![K]\!]$ $[\mathbf{W}]_{i,j} = k(\mathbf{u}_i, \mathbf{u}_j)$; $\mathbf{W}^\dagger$ being the pseudo-inverse of $\mathbf{W}$ and $\mathbf{C} \in \mathbb{R}^{N\times K}$

containing the kernel values between landmark points and all data points: $\forall i \in [\![N]\!], \forall j \in [\![K]\!]\ [\mathbf{C}]_{i,j} = k(\mathbf{x}_i, \mathbf{u}_j)$.

4.6.2 Efficient Nyström approximation

A substantial amount of research has been conducted toward landmark point selection methods for improved approximation accuracy (Kumar et al., 2012; Musco and Musco, 2017), but much less has been done to improve computation speed. Si et al. (2016) proposed an algorithm to learn the matrix of landmark points with some structure constraint, so that its utilisation is fast, taking advantage of fast-transforms. This results in an efficient Nyström approximation that is faster to use both in the training and testing phases of some ulterior machine learning application.

Remarking that the main computational cost of the Nyström approximation comes from the evaluation of the kernel function between the train/test samples and the landmark points, Si et al. (2016) aim at accelerating this step. In particular, they focus on a family of kernel functions that has the form $k(\mathbf{x}_i, \mathbf{x}_j) = f(\mathbf{x}_i) f(\mathbf{x}_j) g(\mathbf{x}_i^T \mathbf{x}_j)$, where $f : \mathbb{R}^D \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$. Given a set of K landmark points $\mathbf{U} \in \mathbb{R}^{K \times D}$ and a sample $\mathbf{x}$, the computational time for computing the kernel between $\mathbf{x}$ and each row of $\mathbf{U}$ (necessary for the Nyström approximation) is bottlenecked by the computation of the product $\mathbf{U}\mathbf{x}$. They hence propose to write the $\mathbf{U}$ matrix as the concatenation of structured $S = K/D$ product of matrices such that

$$\mathbf{U} = \left[\mathbf{D}_{\mathbf{v}_1}\mathbf{H}^T, \cdots, \mathbf{D}_{\mathbf{v}_S}\mathbf{H}^T\right]^T, \tag{14}$$

where the $\mathbf{H}$ is a $D \times D$ matrix associated with a fast transform such as the *Haar* or *Hadamard* matrix, and the $\mathbf{D}_{\mathbf{v}_i}$ are some $D \times D$ diagonal matrices with $\mathbf{v}_i$ on the diagonal to be either chosen with a standard landmark selection method or learned using an algorithm they provide.

Depending on the chosen matrix $\mathbf{H}$, it is possible to improve the time complexity for the computation of $\mathbf{U}\mathbf{x}$ from $\mathcal{O}(KD)$ to $\mathcal{O}(K \log D)$ (*Fast Hadamard transform*) or $\mathcal{O}(K)$ (*Fast Haar Transform*).

4.6.3 `QK-means` in Nyström approximation

We propose to use the `QK-means` algorithm in order to learn directly the $\mathbf{U}$ matrix in the Nyström approximation so that the matrix-vector multiplication $\mathbf{U}\mathbf{x}$ is cheap to compute, but the structure of $\mathbf{U}$ is not constrained by some pre-defined transform matrix, which may results in better performance in practice. We use the RBF kernel ($k(\mathbf{x}, \mathbf{y}) = \exp(-\gamma||\mathbf{x} - \mathbf{y}||^2)$) because it is the most popular kernel function and Si et al. (2016) show that this kernel function is appropriate for the efficient Nyström technique.

As shown in section 4.6.4, our algorithm provides reasonably good Nyström approximation results compared to that of the standard `K-means` landmark points selection technique. Note that using the fast `QK-means` operator in place of the $\mathbf{U}$ matrix defined in Équation (14) brings the same complexity of $\mathcal{O}(K \log D)$ for computing the matrix-vector product $\mathbf{U}\mathbf{x}$.

4.6.4 Results

The results achieved in the Nyström approximation setting are summarized in Table 4. We see that our fast-operator keeps the relevant information for the Nyström approximation so that we obtain similar accuracy scores as with the standard `K-means` landmark points and better accuracy than with the `K-means` ℓ_1 Proj. method. For this evaluation, we consider the approximation error of the Nyström approximation based on different sampling schemes (`QK-means`, `K-means`, Uniform) w.r.t. the real kernel matrix. This relative error is computed from the Froebenius norm of the difference between the matrices as:

$$error = \frac{||\mathbf{K} - \tilde{\mathbf{K}}||_F}{||\mathbf{K}||_F}. \tag{15}$$

One can emphasize that the Uniform sampling scheme is known to give a worse Nyström approximation than with the K-means scheme and this behaviour is still observable with the `QK-means`. We also use the Nyström approximation based on `QK-means` as input for a linear SVM, which achieves as good performance as the one based on the `K-means` approach. Using this criterion, again, the `K-means` ℓ_1 Proj. approach shows worse performances.

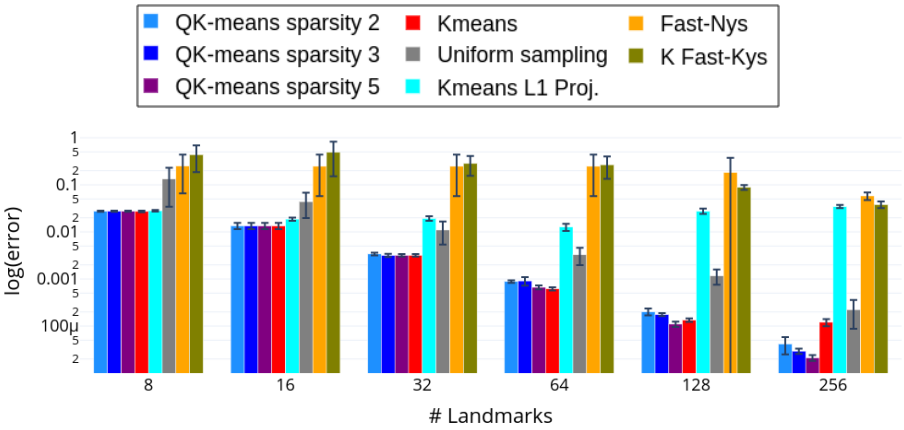
Finally, results in Table 4 and Figure 4 show that for a fixed number of landmark points, our technique significantly outperform the Fast-Nyström technique (Si et al., 2016) in term of approximation error and prediction accuracy. For this technique we used the *Hadamard transform* for $\mathbf{H}$ in equation (14) and the S seeds were taken from Uniform sampling or `K-means` sampling, respectively called «Un. F-Nys» and «K. F-Nys». These results illustrate our claim that it is beneficial in practice to learn a fast transform that fits the data instead of using a fixed fast transform algorithm with strong structural bias such as *the Hadamard transform*.

Figure 4 also shows that the *sparsity level* seems to have a rather limited impact on the Nyström approximation quality and performance accuracy whereas a growth in the number of cluster entails better results, as usual.

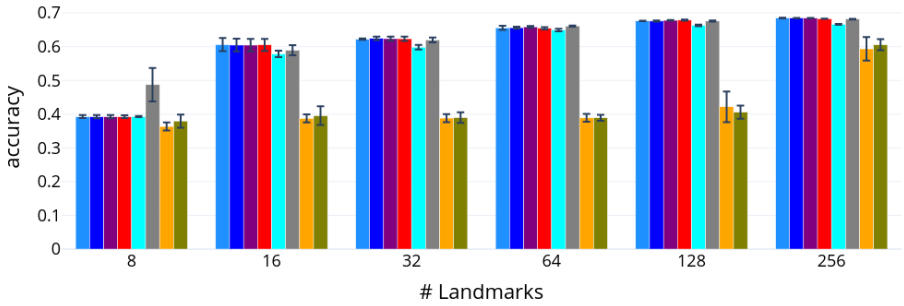
5 Conclusion

In this paper, we have proposed a variant of the `K-means` algorithm, named `QK-means`, designed to achieve a similar goal: clustering data points around K learned centroids. Our approach is based on the approximation of the matrix of centroids by an operator structured as a product of a small number of sparse matrices, resulting in a low time and space complexity when applied to data vectors. We have shown the convergence properties of the proposed algorithm and provided its complexity analysis.

An implementation prototype has been run in several core machine learning use cases including clustering, nearest-neighbor search and Nyström approximation. The experimental results illustrate the computational gain in high dimension at inference time as well as the good approximation qualities of the proposed model. The complexity analysis suggests that our `QK-means` procedure could also have computation benefits for the training phase when the number of observation N to be clustered is bigger than our considered datasets.



(a) Reconstruction error



(b) SVM accuracy

Fig. 4: Coverage Type: Nyström results for kernel matrix reconstruction error (Fig. 4a) and for SVM accuracy with input transformed by the Nyström approximation (Fig. 4b)

Beyond these modeling, algorithmic and experimental contributions to low-complexity high-dimensional machine learning, we have identified important questions that are still to be addressed: the expressiveness of the fast-structure model is still to be theoretically studied even though our experiments seems to show that arbitrary matrices may be well fitted by such models. We believe that learning fast-structure linear operators during the training procedure may be generalized to many core machine learning methods in order to speed them up and make them scale to larger dimensions.

References

Ailon N, Leibovich O, Nair V (2020) Sparse linear networks with a fixed butterfly structure: Theory and practice. arXiv preprint arXiv:200708864

Arthur D, Vassilvitskii S (2006) k-means++: The advantages of careful seeding. Tech. rep., Stanford

Bolte J, Sabach S, Teboulle M (2014) Proximal alternating linearized minimization or nonconvex and nonsmooth problems. Mathematical Programming 146(1-2):459–494

Boutsidis C, Zouzias A, Mahoney MW, Drineas P (2014) Randomized dimensionality reduction for k-means clustering. IEEE Transactions on Information Theory 61(2):1045–1062

Dao T, Gu A, Eichhorn M, Rudra A, Re C (2019) Learning fast algorithms for linear transforms using butterfly factorizations. In: International Conference on Machine Learning, pp 1517–1527

Dua D, Graff C (2017) UCI machine learning repository. URL `http://archive.ics.uci.edu/ml`

Elkan C (2003) Using the triangle inequality to accelerate k-means. In: Proceedings of the 20th international conference on machine learning (ICML-03), pp 147–153

Griffin G, Holub A, Perona P (2007) The caltech-256. Caltech Technical Report p 1

Hamerly G (2010) Making k-means even faster. In: Proceedings of the SIAM International Conference on Data Mining, SIAM, pp 130–140

Hartigan JA, Wong MA (1979) Algorithm as 136: A k-means clustering algorithm. Journal of the Royal Statistical Society Series C (Applied Statistics) 28(1):100–108

Jain AK (2010) Data clustering: 50 years beyond k-means. Pattern recognition letters 31(8):651–666

Keriven N, Tremblay N, Traonmilin Y, Gribonval R (2017) Compressive k-means. In: International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, pp 6369–6373

Kumar S, Mohri M, Talwalkar A (2012) Sampling methods for the nyström method. Journal of Machine Learning Research 13(Apr):981–1006

Le Q, Sarlós T, Smola A (2013) Fastfood—approximating kernel expansions in loglinear time. In: International Conference on Machine Learning

Le Magoarou L, Gribonval R (2016) Flexible multilayer sparse approximations of matrices and applications. IEEE Journal of Selected Topics in Signal Processing 10(4):688–700

LeCun Y, Cortes C, Burges C (2010) Mnist handwritten digit database. 2010. URL http://yann lecun com/exdb/mnist 7:23

Li Y, Yang H, Martin ER, Ho KL, Ying L (2015) Butterfly factorization. Multiscale Modeling & Simulation 13(2):714–732

Liu W, Shen X, Tsang I (2017) Sparse embedded k-means clustering. In: Advances in Neural Information Processing Systems, pp 3319–3327

Morgenstern J (1975) The linear complexity of computation. Journal of the ACM 22(2):184–194

Muja M, Lowe DG (2014) Scalable nearest neighbor algorithms for high dimensional data. IEEE transactions on pattern analysis and machine intelligence 36(11):2227–2240

Musco C, Musco C (2017) Recursive sampling for the nyström method. In: Advances in Neural Information Processing Systems, pp 3833–3845

Nene SA, Nayar SK, Murase H (1996) Columbia object image library (coil-20. Tech. rep.

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, et al. (2011) Scikit-learn: Machine learning

in python. Journal of Machine Learning Research 12(Oct):2825–2830
Que Q, Belkin M (2016) Back to the future: Radial basis function networks revisited. In: Artificial Intelligence and Statistics, pp 1375–1383
Sculley D (2010) Web-scale k-means clustering. In: Proceedings of the 19th international conference on World wide web, ACM, pp 1177–1178
Shen X, Liu W, Tsang I, Shen F, Sun QS (2017) Compressed k-means for large-scale clustering. In: Thirty-First AAAI Conference on Artificial Intelligence
Si S, Hsieh CJ, Dhillon I (2016) Computationally efficient nyström approximation using fast transforms. In: International Conference on Machine Learning, pp 2655–2663
Vahid KA, Prabhu A, Farhadi A, Rastegari M (2020) Butterfly transform: an efficient FFT based neural architecture design. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition
Van Laarhoven T, Marchiori E (2016) Local network community detection with continuous optimization of conductance and weighted kernel k-means. The Journal of Machine Learning Research 17(1):5148–5175
Vinh NX, Epps J, Bailey J (2010) Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. The Journal of Machine Learning Research 11:2837–2854
Williams CK, Seeger M (2001) Using the nyström method to speed up kernel machines. In: Advances in neural information processing systems, pp 682–688
Xiao H, Rasul K, Vollgraf R (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:170807747

Appendix

A palm4MSA algorithm

The palm4MSA algorithm Le Magoarou and Gribonval (2016) is given in Algorithm 2 together with the time complexity of each line, using $A = \min(K, D)$ and $B = \max(K, D)$. Even more general constraints can be used, the constraint sets $\mathcal{E}_q$ are typically defined as the intersection of the set of unit Frobenius-norm matrices and of a set of sparse matrices. The unit Frobenius norm is used together with the λ factor to avoid a scaling indeterminacy. Note that to simplify the model presentation, factor λ is used internally in palm4MSA and is integrated in factor $\mathbf{S}_1$ at the end of the algorithm (Line 14) so that $\mathbf{S}_1$ does not satisfy the unit Frobenius norm in $\mathcal{E}_1$ at the end of the algorithm. The sparse constraints we used, as in Le Magoarou and Gribonval (2016), consist of trying to have a given number of non-zero coefficients in each row and in each column. This number of non-zero coefficients is called sparsity level in this paper. In practice, the projection function at Line 9 keeps the largest non-zero coefficients in each row and in each column, which only guarantees the actual number of non-zero coefficients is at least equal to the sparsity level.

Algorithm 2 palm4MSA algorithm

Require: The matrix to factorize $\mathbf{U} \in \mathbb{R}^{K \times D}$, the desired number of factors Q, the constraint sets $\mathcal{E}_q$, $q \in [\![Q]\!]$ and a stopping criterion (e.g., here, a number of iterations I).

1: $\lambda \leftarrow \|S_1\|_F$ — $\mathcal{O}(B)$
2: $S_1 \leftarrow \frac{1}{\lambda} S_1$ — $\mathcal{O}(B)$
3: **for** $i \in [\![I]\!]$ while the stopping criterion is not met **do**
4: **for** $q = Q$ down to 1 **do**
5: $\mathbf{L}_q \leftarrow \prod_{l=1}^{q-1} \mathbf{S}_l^{(i)}$
6: $\mathbf{R}_q \leftarrow \prod_{l=q+1}^{Q} \mathbf{S}_l^{(i+1)}$
7: Choose $c > \lambda^2 ||\mathbf{R}_q||_2^2 ||\mathbf{L}_q||_2^2$ — $\mathcal{O}(A \log B + B)$
8: $\mathbf{D} \leftarrow \mathbf{S}_q^i - \frac{1}{c} \lambda \mathbf{L}_q^T \left(\lambda \mathbf{L}_q \mathbf{S}_q^i \mathbf{R}_q - \mathbf{U}\right) \mathbf{R}_q^T$ — $\mathcal{O}(AB \log B)$
9: $\mathbf{S}_q^{(i+1)} \leftarrow P_{\mathcal{E}_q}(\mathbf{D})$ — $\mathcal{O}\left(A^2 \log A\right)$ or $\mathcal{O}(AB \log B)$
10: **end for**
11: $\hat{\mathbf{U}} := \prod_{j=1}^{Q} \mathbf{S}_q^{(i+1)}$ — $\mathcal{O}\left(A^2 \log B + AB\right)$
12: $\lambda \leftarrow \frac{Trace(\mathbf{U}^T \hat{\mathbf{U}})}{Trace(\hat{\mathbf{U}}^T \hat{\mathbf{U}})}$ — $\mathcal{O}(AB)$
13: **end for**
14: $S_1 \leftarrow \lambda S_1$ — $\mathcal{O}(B)$

Ensure: $\{\mathbf{S}_q : \mathbf{S}_q \in \mathcal{E}_q\}_{q \in [\![Q]\!]}$ such that $\prod_{q \in [\![Q]\!]} \mathbf{S}_q \approx \mathbf{U}$

The complexity analysis is proposed under the following assumptions, which are satisfied in the mentioned applications and experiments: the number of factors is $Q = \mathcal{O}(\log B)$; all but one sparse factors are of shape $A \times A$ and have $\mathcal{O}(A)$ non-zero entries while one of them is of shape $A \times B$ or $B \times A$ with $\mathcal{O}(B)$ non-zero entries. In such conditions, the complexity of each line is:

Lines 1-2 Computing these normalization steps is linear in the number of non-zeros coefficients in $\mathbf{S}_1$.
Lines 5-6 Fast operators $\mathbf{L}$ and $\mathbf{R}$ are defined for subsequent use without computing explicitly the product.
Line 7 The spectral norm of $\mathbf{L}$ and $\mathbf{R}$ is obtained via a power method by iteratively applying each operator, benefiting from the fast transform.
Line 8 The cost of the gradient step is dominated by the product of sparse matrices.
Line 9 The projection onto a sparse-constraint set takes $\mathcal{O}\left(A^2 \log A\right)$ for all the $A \times A$ matrices and $\mathcal{O}(AB \log B)$ for the rectangular matrix at the leftmost or the rightmost position.
Line 11 The reconstructed matrix $\hat{\mathbf{U}}$ is computed using $\mathcal{O}(\log B)$ products between $A \times A$ sparse matrices, in $\mathcal{O}\left(A^2\right)$ operations each, and one product with a sparse matrix in $\mathcal{O}(AB)$.

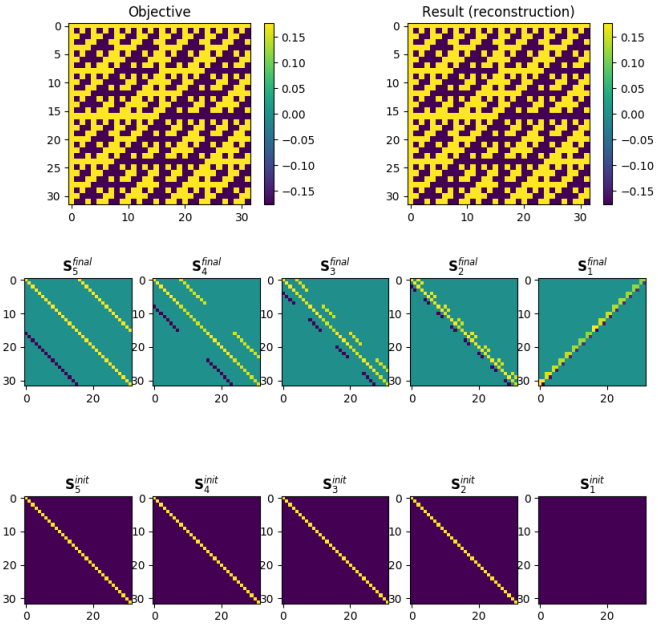


Fig. 5: Decomposition of hadamard matrix by sparse factors. Bottom line show the initialization of the factors while middle line shows their final form at the end of the algorithm. Figure inspired from Le Magoarou and Gribonval (2016).

Line 12 The numerator and denominator can be computed using a Hadamard product between the matrices followed by a sum over all the entries.

Line 14 Computing renormalization step is linear in the number of non-zeros coefficients in $\mathbf{S}_1$.

Hence, the overal time complexity of `palm4MSA` is in $\mathcal{O}\left(AB\log^2 B\right)$, due to Lines 8 and 9 repeated for each of the $\log B$ factors.

A.1 Projection function for `palm4MSA`

The function used for projecting the factors onto the set of possible solutions (Line 9 of Algorithm 2) is the same as the one used in Le Magoarou and Gribonval (2016) source code. This projection function relaxes the strong constraint of having exactly

$$\forall j \in Q,\ ||\mathbf{S}_j||_0 \leq \epsilon_j \tag{16}$$

to become the weaker

$$\begin{aligned} &\forall j \in Q,\ \epsilon_j \leq\ ||\mathbf{S}_j||_0 \leq 2\epsilon_j \\ &\forall i,\ ||\mathbf{S}_j[i]||_0 = \lceil \epsilon_j/A \rceil \text{ and } ||\mathbf{S}_j^T[i]||_0 = \lceil \epsilon_j/N \rceil \end{aligned} \tag{17}$$

,

with $\mathbf{S}_j[i]$ the i^th Line of $\mathbf{S}_j$ and $\mathbf{S}_j \in \mathbb{R}^{A\times A}$. In simple words, it ensures that each line and each column has *at least* $\lceil \epsilon_j/A \rceil$ non-zero values. This is easy to see as (i) each line of A must have $\lceil \epsilon_j/A \rceil$ non-zero values then the matrix has at least ϵ_j values and (ii) it can't have more than $2\epsilon_j$

even considering the line and columns of the matrix to not share the same non-zero values. We see in practice that there is actual collision between non-zero values of lines and columns. The procedure is summarized in Algorithm 3 where the function $get\_max\_by\_line(\mathbf{S}, c)$ returns $\mathbf{S}$ with only its c biggest values in each line and other values zeroed and $\neg mask(\mathbf{S})$ returns the negative of the mask of $\mathbf{S}$ *e.g.* the matrix $\mathbf{M}$ with $\mathbf{M}_{i,j} = 1$ where $\mathbf{S}_{i,j} = 0$ and $\mathbf{M}_{i,j} = 0$ otherwise.

Algorithm 3 Projection function for `palm4MSA`

Require: A factor $\mathbf{S} \in \mathbb{R}^{A \times A}$ to project so its $L0$ norm is $\epsilon \leq ||\mathbf{S}||_0 \leq 2\epsilon$
1: $c \leftarrow \lceil \epsilon / A \rceil$
2: $\mathbf{S}' \leftarrow get\_max\_by\_line(\mathbf{S}, c)$
3: $\mathbf{S}'' \leftarrow get\_max\_by\_line(\mathbf{S}^T, c)^T$
4: $\mathbf{M} \leftarrow \neg mask(\mathbf{S}'')$
5: $P_\epsilon(\mathbf{S}) \leftarrow \mathbf{S}' + \mathbf{S}'' \odot \mathbf{M}$
Ensure: $P_\epsilon(\mathbf{S})$ the projection of $\mathbf{S}$ onto the set of possible solutions