QuicK-means: Acceleration of K-means by learning a fast transform

Luc Giffon, Valentin Emiya, Liva Ralaivola, Hachem Kadri

QuicK-means: Acceleration of K-means by Learning a Fast Transform

Luc Giffon [1], **Valentin Emiya** [1], **Liva Ralaivola** [2], **Hachem Kadri**[1]
[1] Aix Marseille Université, Université de Toulon, CNRS, LIS, Marseille, France
[2] Criteo, France

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 and data analysis techniques such as indexing, nearest-neighbor search and prediction, data compression; its beneficial use has been shown to carry over to the acceleration of kernel machines (when using the Nyström method). Here, we propose a fast extension of `K-means`, dubbed `QuicK-means`, that rests on the idea of expressing the matrix of the K cluster centers 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 $K \times D$ center matrix $\mathbf{U}$ and any vector from $\mathcal{O}(KD)$ to $\mathcal{O}(A \log A + B)$, 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 precisely show that resorting to a factorization step at each iteration does not impair the convergence of the optimization scheme and that, depending on the context, it may entail a reduction of the training time. Finally, we provide discussions and numerical simulations that show the versatility of our computationally-efficient `QuicK-means` algorithm.

1 Introduction

`K-means` is one of the most popular clustering algorithms (Hartigan and Wong 1979; Jain 2010) and it can be used beyond clustering, for tasks such as indexing, data compression, nearest-neighbor search and prediction, and local network community detection (Muja and Lowe 2014; Van Laarhoven and Marchiori 2016). `K-means` is also a pivotal process to increase the speed and the accuracy of learning procedures, e.g., for kernel machines (Si, Hsieh, and Dhillon 2016) and RBF networks (Que and Belkin 2016), when combined with the Nyström approximation. The conventional `K-means` algorithm, i.e. Lloyd's algorithm, has a $\mathcal{O}(NKD)$ complexity per iteration when learning K clusters from N data points in dimension D. In addition, the larger the number of clusters, the more iterations are needed to converge (Arthur and Vassilvitskii 2006). As data dimensionality and sample size grow, it is critical to have at hand cost-effective alternatives to the computationally expensive conventional `K-means`. Known strategies to alleviate its computational issues rely on batch-, sparsity- and randomization-based methods (Sculley 2010; Boutsidis et al. 2014; Shen et al. 2017; Liu, Shen, and Tsang 2017).

Fast transforms have recently received increased attention in machine learning as they can speed up random projections (Le, Sarlós, and Smola 2013; Gittens and Mahoney 2016) and to improve landmark-based approximations (Si, Hsieh, and Dhillon 2016). These works focused on fixed fast transforms such as well-known Fourier and Hadamard transforms. A question is whether one can go beyond and learn fast transforms that fit the data. Recently, Le Magoarou and Gribonval (2016) introduced an approach aimed at reducing the complexity of applying linear operators in high dimension by approximately factorizing the corresponding matrix into few sparse factors. Indeed, the aforementioned fixed fast transforms can be factorized into few sparse matrices. In this paper, we take this idea further and investigate computationally-efficient variants of `K-means` by learning fast transforms from data. After introducing the background elements in Section 2, we make the following contributions:

- in Section 3.1, we introduce `QK-means`, a fast extension of `K-means` that rests on learning a fast transform that approximate the matrix of centers;
- in Section 3.2, we show that each update step in one iteration of our algorithm reduces the overall objective, which establishes the convergence of `QK-means`;
- in Section 3.3, we provide a complexity analysis of `QK-means`, showing that the computational gain has a direct impact in assigning a point to a cluster;
- in Section 4, an empirical evaluation of `QK-means` performance demonstrates its effectiveness on different datasets in the contexts of clustering, nearest neighbor search and kernel Nyström approximation.

2 Preliminaries

We briefly review the basics of `K-means` and give background on learning fast transforms. To assist the reading, we

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
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` centers
$\mathbf{U} \in \mathbb{R}^{K \times D}$	`K-means` center matrix
$\mathbf{v}_1, \ldots, \mathbf{v}_K$	`QK-means` centers
$\mathbf{V} \in \mathbb{R}^{K \times D}$	`QK-means` center matrix
$\mathbf{S}_1, \ldots, \mathbf{S}_Q$	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.

list the notations used in the paper in Table 1.

2.1 `K-means`

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 between each $\mathbf{x}_n$ to the center $\mathbf{u}_k \in \mathbb{R}^D$ of the cluster k it belongs to —the optimal center $\mathbf{u}_k$ is the mean vector of the points assigned to cluster k. 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 centers and $\mathbf{t} \in [\![K]\!]^N$ is the assignment vector that puts $\mathbf{x}_n$ in 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 referred to as the `K-means` algorithm —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 reestimation step which adjusts the cluster centers. After an initialization of the set $\mathbf{U}^{(0)}$ of K cluster centers, the algorithm proceeds as follows: at iteration τ, the assignments are updated as

$$t_n^{(\tau)} \leftarrow \arg\min_{k \in [\![K]\!]} \left\| \mathbf{x}_n - \mathbf{u}_k^{(\tau-1)} \right\|_2^2, \quad \forall n \in [\![N]\!], \tag{2}$$

and the reestimation of the cluster centers is done 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 Lloyd's algorithm. The cost of the assignment step (2) is $\mathcal{O}(NDK)$ while that of the centers update (3) is $\mathcal{O}(ND)$. Hence, the bottleneck of the total time complexity $\mathcal{O}(NDK)$ stems from the assignment step.

Contribution. Our main contribution rests on the idea that (2) may be computed more efficiently if the matrix $\mathbf{U}$ of centers is approximated by a fast-transform matrix, which is learned thanks to a dedicated procedure that we now discuss.

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 Fourier or Hadamard transforms comes from both their mathematical properties 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. The core 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 $\mathbf{S}_0(\mathbf{S}_1(\cdots(\mathbf{S}_{Q-1}\mathbf{x})))$ 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. Le Magoarou and Gribonval (2016) proposed algorithmic strategies to learn such a factorization. Based on the proximal alternating linearized minimization (`PALM`) algorithm (Bolte, Sabach, and Teboulle 2014), the `PALM` for Multi-layer Sparse Approximation (`palm4MSA`) algorithm 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 is able to find local minima with convergence guarantees. (In addition to the reference papers, details on `palm4MSA` are in the supplementary material.)

3 `QuicK-means`

We now introduce our main contribution, `QuicK-means`, shortened as `QK-means`, show its convergence property and analyze its computational complexity.

3.1 QK-means: Encoding Centers as Products of Sparse Matrices

QK-means is a variant of the K-means algorithm in which the matrix of centers $\mathbf{U}$ is approximated as a product $\prod_{\in[\![Q]\!]} \mathbf{S}_q$ of sparse matrices $\mathbf{S}_q$. Doing so will allow us to cope with the computational bulk imposed by the product $\mathbf{U}\mathbf{x}$ that shows up in the cluster assignment process —rewrite (2) by developing $\left\|\mathbf{x}_n - \mathbf{u}_k^{(\tau-1)}\right\|_2^2$ to see it.

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

$$\underset{\{\mathbf{S}_q\}_{q=1}^Q, \mathbf{t}}{\arg\min}\, g\left(\{\mathbf{S}_q\}_{q=1}^Q, \mathbf{t}\right), \tag{5}$$

where

$$g\left(\{\mathbf{S}_q\}_{q=1}^Q, \mathbf{t}\right) := \sum_{k\in[\![K]\!]} \sum_{n:t_n=k} \|\mathbf{x}_n - \mathbf{v}_k\|^2 + \sum_{q\in[\![Q]\!]} \delta_{\mathcal{E}_q}(\mathbf{S}_q) \tag{6}$$

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

This is a constrained version of the K-means optimization problem (1) in which centers $\mathbf{v}_k$ are constrained to form a matrix $\mathbf{V}$ with a fast-operator structure, the indicator functions $\delta_{\mathcal{E}_q}$ imposing the sparsity of matrices $\mathbf{S}_q$. More details on the modeling choices 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 centers at lines 4–8. The assignment step can be computed efficiently thanks to the structure of $\mathbf{V}$. The update of the centers relies on learning a fast-operator $\mathbf{V}$ that approximate the true center matrix $\mathbf{U}$ weighted by the number of examples n_k assigned to each cluster k.

3.2 Convergence of QK-means

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

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

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

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 (6) w.r.t. $\mathbf{t}$ while the second term is constant so we have

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

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

$$\sum_{n:t_n=k} \|\mathbf{x}_n - \mathbf{v}_k\|^2 = \sum_{n:t_n=k} \|\mathbf{x}_n - \mathbf{u}_k\|^2 + n_k \|\mathbf{u}_k - \mathbf{v}_k\|^2, \tag{8}$$

(See supplementary material Section D for step by step calculus)

For $\mathbf{t}$ fixed, the new sparsely-factorized centers are set as solutions of the problem $\arg\min_{\mathbf{S}_1,\ldots,\mathbf{S}_Q} g(\mathbf{S}_1,\ldots,\mathbf{S}_Q,\mathbf{t})$ which rewrites, thanks to (8) as

$$\begin{aligned}
&\underset{\mathbf{S}_1,\ldots,\mathbf{S}_Q}{\arg\min} \left\|\mathbf{D}_{\sqrt{\mathbf{n}}}(\mathbf{U}-\mathbf{V})\right\|_F^2 + \sum_{k\in[\![K]\!]} c_k + \sum_{q\in[\![Q]\!]} \delta_q(\mathbf{S}_q) \\
&\qquad \text{s. t. } \mathbf{V} = \prod_{q\in[\![Q]\!]} \mathbf{S}_q \\
&= \underset{\mathbf{S}_1,\ldots,\mathbf{S}_Q}{\arg\min} \left\|\mathbf{A} - \prod_{q\in\{[\![Q]\!]\cup\{0\}\}} \mathbf{S}_q\right\|_F^2 + \sum_{q\in\{[\![Q]\!]\cup\{0\}\}} \delta_q(\mathbf{S}_q)
\end{aligned} \tag{9}$$

where:

- $\sqrt{\mathbf{n}} \in \mathbb{R}^K$ is the pairwise square root of the vector indicating the number of observations $n_k := |\{n : t_n = k\}|$ in each cluster k;
- $\mathbf{U} \in \mathbb{R}^{K\times d}$ refers to the unconstrained center matrix obtained from the data matrix $\mathbf{X}$ and the indicator vector $\mathbf{t}$: $\mathbf{u}_k := \frac{1}{n_k}\sum_{n:t_n=k} \mathbf{x}_n$ (see Line 4);
- $\mathbf{D}_{\sqrt{\mathbf{n}}}(\mathbf{U}-\mathbf{V})$ is the matrix with $\sqrt{n_k}\,(\mathbf{u}_k - \mathbf{v}_k)$ as row k;
- $c_k := \sum_{t_n=k}^{n} \|\mathbf{x}_n - \mathbf{u}_k\|$ is constant w.r.t. $\mathbf{S}_1,\ldots,\mathbf{S}_Q$;
- $\mathbf{A} := \mathbf{D}_{\sqrt{\mathbf{n}}}\mathbf{U}$ is the unconstrained center matrix reweighted by the size of each cluster (see Line 5).

We note that problem (9) has exactly the form of (4) hence the palm4MSA algorithm is used in Line 7 to obtain an approximation of $\mathbf{A}$. The first factor $\mathbf{S}_0$ is set to $\mathbf{D}_{\sqrt{\mathbf{n}}}$ 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 that $g(\mathbf{S}_1^{(\tau)},\ldots,\mathbf{S}_Q^{(\tau)},\mathbf{t}^{(\tau)}) \leq g(\mathbf{S}_1^{(\tau-1)},\ldots,\mathbf{S}_Q^{(\tau-1)},\mathbf{t}^{(\tau)})$. We finally have, for any τ:

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

which is a sufficient condition to assert that Algorithm 1 converges (the sequence of objective values is nonincreasing and lower bounded, thus convergent). □

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: $\quad \mathbf{t}^{(\tau)} := \arg\min_{\mathbf{t} \in [\![K]\!]^N} \sum_{n \in [\![N]\!]} \left\| \mathbf{x}_n - \mathbf{v}_{t_n}^{(\tau-1)} \right\|^2$ $\quad \mathcal{O}\left(N\left(A \log A \ + B\right) + AB\right)$

4: $\quad \forall k \in [\![K]\!], \mathbf{u}_k := \frac{1}{n_k} \sum_{n : t_n^{(\tau)} = k} \mathbf{x}_n$ with $n_k := |\{n : t_n^{(\tau)} = k\}|$ $\quad \mathcal{O}(ND)$

5: $\quad \mathbf{A} := \mathbf{D}_{\sqrt{\mathbf{n}}} \times \mathbf{U}$ $\quad \mathcal{O}(KD)$

6: $\quad \mathcal{E}_0 := \left\{\mathbf{D}_{\sqrt{\mathbf{n}}}\right\}$

7: $\quad \left\{\mathbf{S}_q^{(\tau)}\right\}_{q=0}^{Q} := \arg\min_{\{\mathbf{S}_q\}_{q=0}^{Q}} \left\| \mathbf{A} - \prod_{q=0}^{Q} \mathbf{S}_q \right\|_F^2 + \sum_{q=0}^{Q} \delta_{\mathcal{E}_q}(\mathbf{S}_q)$ $\quad \mathcal{O}\left(AB\left(\log^2 A \ + \log B\right)\right)$

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 $\{\mathbf{S}_q : \mathbf{S}_q \in \mathcal{E}_q\}_{q \in [\![Q]\!]}$

3.3 Complexity analysis

Since the space complexity of the proposed QK-means algorithm is comparable to that of K-means, we only detail its time complexity. We set $A = \min(K, D)$ and $B = \max(K, D)$, and assume that the number of factors satisfies $Q = \mathcal{O}(\log A)$. 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 done $\mathcal{O}(N_1 N_2 N_3)$ operations; the product between a sparse matrix with $\mathcal{O}(S)$ non-zero entries and a dense vector can be done in $\mathcal{O}(S)$ 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 can be done in $\mathcal{O}(S \min(N_1, N_3))$ and the number of non-zero entries in the resulting matrix is $\mathcal{O}\left(S^2\right)$.

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

Complexity of algorithm palm4MSA. The procedure consists in an alternate optimization of each sparse factor. At each iteration, the whole set of Q factors is updated with at a cost in $\mathcal{O}\left(AB\left(\log^2 A \ + \log B\right)\right)$, as detailed in the supplemental material (Section A). The bottleneck is the computation of the gradient, which benefits from fast computations with sparse matrices.

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

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

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

The update of the fast transform, in lines 5 to 8 is a computational overload compared to K-means. Its time complexity is dominated by the update of the sparse factors at line 7, in $\mathcal{O}\left(AB \log^2 A\right)$. Note that the overall cost of QK-means is dominated by the cost of the assignement step as soon as the number of examples N is greater than $\log^3 A$. One can see that the computational bottleneck of K-means is here reduced, which shows the advantage of using QK-means when N, K and D are large.

4 Experiments and applications

4.1 Experimental setting

Implementation details. The code has been written in Python, including the palm4MSA algorithm. Running times are measured on a computer grid with 3.8GHz-CPUs (2.5GHz in Figure 3). 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 is sufficient as a proof of concept for assessing the performance of the proposed approach as illustrated by running times benchmarking in the Section C of supplementary material.

Datasets. We present results on real-world and toy datasets. Our experiments are conducted on synthetic and real-world data sets to show (i) — quantitatively and qualitatively — the good quality of the obtained centers when using our method on the MNIST (LeCun and Cortes 2010) and Fashion-Mnist (Pedregosa et al. 2011) ($D = 784$, $K = 10$) datasets and (ii) the speed up offered by our method QK-means when the number of clusters and the dimensionality of the data are sufficiently large on the blobs synthetic dataset from sklearn.dataset ($D = 2000$, $K = 1000$) and the Caltech256 (Griffin, Holub, and Perona 2007) dataset ($D = 2352$, $K = 256$) (see supplementary material Section B for compact statistics table).

Algorithm settings. QK-means is used with $Q := \log_2(A)$ sparse factors, where $A := \min(K, D)$. All factors $\mathbf{S}_q$ are with shape $A \times A$ except, depending on the shape of $\mathbf{A}$,

the leftmost one ($K \times A$) or the rightmost one ($A \times D$). 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 $\mathbf{S}_q$. 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 supplementary material Section A. The stopping criterion of `K-means` and `QK-means` consists of a tolerance set to 10^{-6} on the relative variation of the objective function and a maximum number of iterations set to 10. The same principle governs the stopping criterion of `palm4MSA` with a tolerance set to 10^{-6} and a maximum number of iterations set to 300. Each experiment have been replicated using different seed values for random initialisation. Competing techniques share the same seed values, hence share the same initialisation of centers as they are sampled uniformly from the dataset. For the `QK-means` experiments, the inital matrix of centroids is processed once by the *Hierarchical*-`palm4MSA` euristic proposed in (Le Magoarou and Gribonval 2016) but then the simple `palm4MSA` algorithm is used inside `QK-means` as it didn't impact much of the performance while saving a lot of time. (See supplementary material Section E for details).

4.2 Clustering

Approximation quality. One important question is the ability of the fast-structure model to fit arbitrary data. Indeed, no theoretical result about the expressivity of such models is currently available. In order to assess this approximation quality, the MNIST and Fashion-MNIST data have been clustered into $K = 30$ clusters by `K-means` and `QK-means` with several sparsity levels. Results are reported in Figure 1. In Figures 1a and 1b, one can observe that the objective function of `QK-means` is decreasing in a similar way as `K-means` over iterations. In particular, the use of the fast-structure model does not seem to increase the number of iteration necessary before convergence. At the end of the iterations, the value of objective function for `QK-means` is slightly above that of `K-means`. As expected, the sparser the model, the more degradation in the objective function. However, even very sparse models do not degrade the results significantly. The approximation quality can be assessed visually, in a more subjective and interpretable way, in Figures 1c to 5d where the obtained centers are displayed as images. Although some degradation 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.

Clustering assignation times. Higher dimensions are required to assess the computational benefits of the proposed approach, as shown here with the comparison between the `MNIST` dataset and the others: the `MNIST` dataset has low dimensionality and low cluster number then the vector assignation times are even worse due to the computational overload induced by our poor implementation. Results reported in Table 2 show that in this setting and with the current implementation, the computational advantage of `QK-means` is observed in high dimension, for $K = 256$ and $K = 512$ clusters. It is worth noticing that when K increases, the running times are not affected that much for `QK-means` while it significantly grows for `K-means`. These trends are directly related to the number of model parameters that are reported in the figure. It can be noticed that these numbers of parameters doesn't necessarily grow with the number of clusters because each sparse factor must have *at least* a given number of value for each line and each column of the sparse factor, in this case 2.

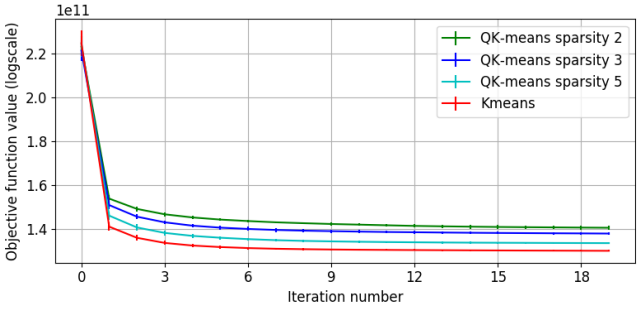
4.3 Nearest-neighbor search in a large dataset

The Nearest-neighbor search is a fundamental task that suffers from computational limitations when the dataset is large. 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 centers 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 this strategy using `K-means` and `QK-means` against the `scikit-learn` implementation (Pedregosa et al. 2011) of the nearest-neighbor search (brute force search, kd tree, ball tree). Inference time results and accuracy results are displayed in Table 2. The results for the Brute Force Search, KD Tree and Ball Tree are not available for some dataset because they were longer than 10 times the `K-means` search version in the same setting. The running times reported in the table show a drastic advantage of using a clustering-based approximate search and this advantage is even stronger with the clustering obtained by our `QK-means` method. We see that for the `Blobs` dataset, this speed-up comes at a cost as we can see a drop in classification performance but not for the other datasets. We believe our method is more sensible to the very noisy nature of the blobs dataset.

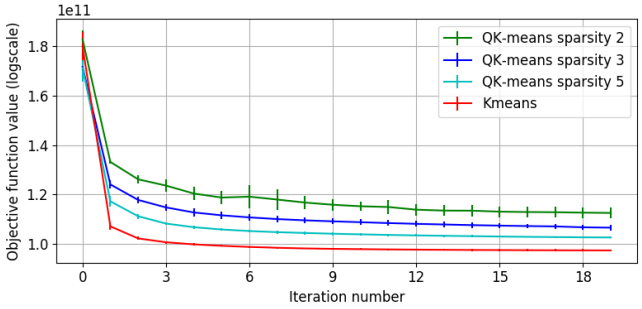
4.4 Nyström approximation

In this sub-section, we show how we can take advantage of the fast-operator obtained as output of our `QK-means` algorithm in order to speed-up 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 know fast-transform method. We finally stem on this work to present a novel approach based on our `QK-means` algorithm.

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 $O(n^2 d)$ 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

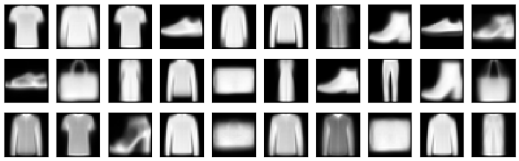


(a) MNIST, $K = 30$: objective function.



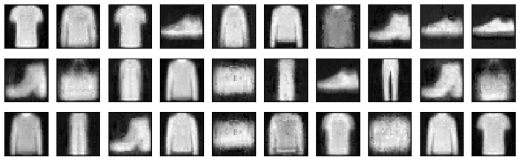
(b) Fashion-MNIST, $K = 30$: objective function.

(c) `K-means` centers.



(d) `K-means` centers.

(e) `QK-means` centers.



(f) `QK-means` centers.

Figure 1: Clustering results on MNIST (left) and Fashion-MNIST (right) for $K = 30$ clusters.

		Blobs K=128	Blobs K=256	Blobs K=512	Caltech K=128	Caltech K=256	Caltech K=512	MNIST K=10	MNIST K=16	MNIST K=30
Vector assignation time (ms)	K-means	**0.3**	3.1	5.7	**0.2**	**1.5**	4.2	**0.003**	**0.005**	**0.006**
	QK-means	1.2	**1.4**	**1.3**	1.3	**1.5**	**1.7**	0.5	0.6	0.8
Nyström Inference time (ms)	K-means	**0.11**	**0.17**	0.32	**0.09**	**0.14**	**0.21**	**0.06**	**0.06**	0.07
	QK-means	0.13	0.18	**0.22**	0.24	0.30	0.40	0.08	0.07	**0.06**
1NN Accuracy	K-means	**0.96**	**0.97**	**0.99**	**0.11**	**0.11**	**0.11**	0.96	0.96	0.96
	QK-means	0.61	0.49	0.37	0.10	0.10	0.09	0.96	0.96	0.95
	Ball-tree		timed-out			timed-out			**0.97**	
1NN Runtime (s)	K-means	17.2	15.8	9.5	74.4	50.0	35.3	**0.74**	**0.83**	**0.88**
	QK-means	**5.3**	**3.0**	**1.2**	**67.0**	**30.9**	**19.8**	0.73	0.79	0.86
	Ball-tree		timed-out			timed-out			553.0	
Nyström + SVM Accuracy	K-means	**0.98**	**1.0**	**1.0**	**0.16**	**0.17**	**0.18**	**0.74**	**0.83**	**0.88**
	QK-means	0.95	**1.0**	**1.0**	0.15	0.16	0.16	0.73	0.79	0.86
Number of parameters	K-means	256,000	512,000	1,024,000	301,056	602,112	1,204,224	7,840	12,544	25,088
	QK-means	**6,207**	**8,656**	**14,078**	**10,409**	**13,846**	**21,013**	**2,610**	**2,269**	**2,427**

Table 2: Results of numerical experiments: average Nyström transformation time for a sample set of size 5000; 1-nearest neighbor and SVM classification accuracy on top of Nyström transformation on the test set. The `QK-means` results are obtained with sparse factors with at least 2 values in each line/column. Every experiment results are averaged over 5 runs. Best results are bold while second best are underlined (when necessary). "Ball-tree" aggregates the results of the "brute" and "kd-tree" implementations of 1NN from `sklearn` as it has given the best results of those. The vector assignation time is measured for one vector while the other times are measured for a whole matrix at once (See supplementary material Section C for benchmark)

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.

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

$$\mathbf{K} \approx \tilde{\mathbf{K}} = \mathbf{C}\mathbf{W}^\dagger\mathbf{C}^T, \tag{10}$$

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: $\forall i \in [\![n]\!], \forall j \in [\![K]\!]\ \mathbf{C}_{i,j} = k(\mathbf{X}_i, \mathbf{U}_j)$.

Efficient Nyström approximation A substantial amount of research has been conducted toward landmark point selection methods for improved approximation accuracy (Kumar, Mohri, and Talwalkar 2012) (Musco and Musco 2017), but much less has been done to improve computation speed. In (Si, Hsieh, and Dhillon 2016), the authors propose 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 computation cost of the Nyström approximation comes from the computation of the kernel function between the train/test samples and the landmark points, (Si, Hsieh, and Dhillon 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 $g : \mathbb{R} \to \mathbb{R}$. They show that this family of functions contains some widely used kernels such as the Gaussian and the polynomial kernel. 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 $\mathbf{U} = \left[\mathbf{V}_1\mathbf{H}^T, \cdots, \mathbf{V}_s\mathbf{H}^T\right]^T$, where the $\mathbf{H}$ is a $d \times d$ matrix associated with a fast transform such as the *Haar* or *Hadamard* matrix, and the $\mathbf{V}_i$s are some $d \times d$ diagonal matrices to be either chosen with a standard landmark selection method or learned using an algorithm they provide.

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

`QK-means` in Nyström We propose to use our `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. We propose to take the objective $\mathbf{U}$ matrix as the `K-means` matrix of $\mathbf{X}$ since it has been shown to achieve good reconstruction accuracy in the Nyström method (Kumar, Mohri, and Talwalkar 2012).

As shown in the next sub-section, our algorithm allows to obtain an efficient Nyström approximation, while not reducing too much the quality of the `K-means` landmark points which are encoded as a factorization of sparse matrix.

Results The Table 2 summarizes the results achieved in the Nyström approximation setting. An histogram representation and an evaluation for other sparsity factors are available in the supplementary material Section F.

The "Nyström inference time" displays the average time for computing one line of the approximated matrix in Equation 10. For $K = 512$, when the landmark matrix is big enough, we clearly see the speed-up offered using the `QK-means` method on the `Blobs` dataset. Intringingly, this speed-up is not sensible for the `Caltech` dataset even though the vector assignation time was still faster.

From a more practical point of view, we show in Table 2 that the Nyström approximation based on `QK-means` can then be used in a linear SVM and achieve as good performance as the one based on the `K-means` approach.

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 centers – with a much lower computational complexity as the dimension of the data, the number of examples and the number of clusters get high. Our approach is based on the approximation of the matrix of centers 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.

Beyond these modeling, algorithmic and experimental contributions to low-complexity high-dimensional machine learning, we have identified several important questions that are still to be addressed. First, although learning the fast-structure operator has been nicely integrated in the training algorithm with an advantageous theoretical time and space complexity, exhibiting gains in actual running times has not been achieved yet for the `QK-means` learning procedure, compared to `K-means`. This may be obtained in even higher dimensions than in the proposed experimental settings, which may require a new version of `QK-means` using batches of data in order to process amounts of data that do not fit in memory. Second, the expressiveness of the fast-structure model is still to be theoretically studied, while our experiments seems to show that arbitrary matrices may be well fitted by such models. Third, 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

[Arthur and Vassilvitskii 2006] Arthur, D., and Vassilvitskii, S. 2006. How slow is the k-means method? In *Symposium on Computational Geometry*, volume 6, 1–10.

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

[Boutsidis et al. 2014] Boutsidis, C.; Zouzias, A.; Mahoney, M. W.; and Drineas, P. 2014. Randomized dimensionality reduction for k-means clustering. *IEEE Transactions on Information Theory* 61(2):1045–1062.

[Gittens and Mahoney 2016] Gittens, A., and Mahoney, M. W. 2016. Revisiting the nyström method for improved large-scale machine learning. *The Journal of Machine Learning Research* 17(1):3977–4041.

[Griffin, Holub, and Perona 2007] Griffin, G.; Holub, A.; and Perona, P. 2007. Caltech-256 object category dataset.

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

[Jain 2010] Jain, A. K. 2010. Data clustering: 50 years beyond k-means. *Pattern recognition letters* 31(8):651–666.

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

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

[Le, Sarlós, and Smola 2013] Le, Q.; Sarlós, T.; and Smola, A. 2013. Fastfood—approximating kernel expansions in loglinear time. In *International Conference on Machine Learning*.

[LeCun and Cortes 2010] LeCun, Y., and Cortes, C. 2010. MNIST handwritten digit database.

[Liu, Shen, and Tsang 2017] Liu, W.; Shen, X.; and Tsang, I. 2017. Sparse embedded k-means clustering. In *Advances in Neural Information Processing Systems*, 3319–3327.

[Morgenstern 1975] Morgenstern, J. 1975. The Linear Complexity of Computation. *Journal of the ACM* 22(2):184–194.

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

[Musco and Musco 2017] Musco, C., and Musco, C. 2017. Recursive sampling for the nystrom method. In *Advances in Neural Information Processing Systems*, 3833–3845.

[Pedregosa et al. 2011] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12:2825–2830.

[Que and Belkin 2016] Que, Q., and Belkin, M. 2016. Back to the future: Radial basis function networks revisited. In *AISTATS*, 1375–1383.

[Sculley 2010] Sculley, D. 2010. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, 1177–1178. ACM.

[Shen et al. 2017] Shen, X.; Liu, W.; Tsang, I.; Shen, F.; and Sun, Q.-S. 2017. Compressed k-means for large-scale clustering. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[Si, Hsieh, and Dhillon 2016] Si, S.; Hsieh, C.-J.; and Dhillon, I. 2016. Computationally efficient nyström approximation using fast transforms. In *International Conference on Machine Learning*, 2655–2663.

[Van Laarhoven and Marchiori 2016] Van Laarhoven, T., and 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.

Supplemental material

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 $c > \lambda^2 ||\mathbf{R}_q||_2^2 ||\mathbf{L}_q||_2^2$ — $\mathcal{O}(A \log A + B)$
8: $\mathbf{D} \leftarrow \mathbf{S}_q^i - \frac{1}{c} \lambda \mathbf{L}_q^T \left(\lambda \mathbf{L}_q \mathbf{S}_q^i \mathbf{R}_q - \mathbf{U}\right) \mathbf{R}_q^T$ — $\mathcal{O}(AB \log A)$
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: $\hat{\mathbf{U}} := \prod_{j=1}^{Q} \mathbf{S}_q^{(i+1)}$ — $\mathcal{O}\left(A^2 \log A + AB\right)$
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 $Q = \mathcal{O}(\log A)$; 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 $\mathcal{O}(\log A)$ products between $A \times A$ sparse matrices, in $\mathcal{O}\left(A^2\right)$ operations each, and one product with a sparse matrix in $\mathcal{O}(AB)$.

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 $\mathcal{O}\left(AB\left(\log^2 A + \log B\right)\right)$, due to Lines 8 and 9.

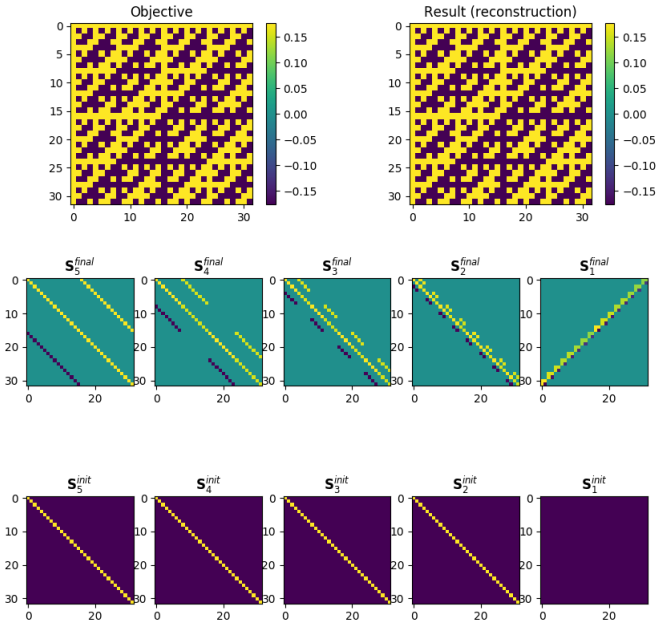


Figure 2: Decomposition of hadamard matrix by sparse factors. Bottom line show the initialization of the factors while middle line shows their final form at the end of the algorithm.

B Datasets

Table 3 provides some statistics about the used datasets.

Dataset	**Data dim.** D	**# classes**	**Training set size** N	**Test set size** N'
MNIST	784	10	60 000	10 000
Fashion-MNIST	784	10	60 000	10 000
Blobs (clusters std: 12)	2000	1000	29000	1000
Caltech256	2352	256	19952	9828

Table 3: Datasets statistics

C Sparse-factorization speed

While more efficient implementations may be beneficial for larger deployment, our implementation is sufficient as a proof of concept for assessing the performance of the proposed approach. In particular, the running times of fast operators of the form $\prod_{q\in[\![Q]\!]} \mathbf{S}_q$ have been measured when applying to random vectors, for several sparsity levels: as shown in Figure 3, they are significantly faster than dense operators – implemented as a `numpy.ndarray` matrix –, especially when the data size is larger than 10^3.

We have noted a difference of efficiency when applying our fast-transform operator to single vector or complete matrices (concatenated vectors) as shown in Figure 4. The fast-transform operator is more efficient compared to a numpy dense operator when applied to one single vector.

D `QK-means` proof

We propose in Equation 11 a step by step re-writting of Equation (8).

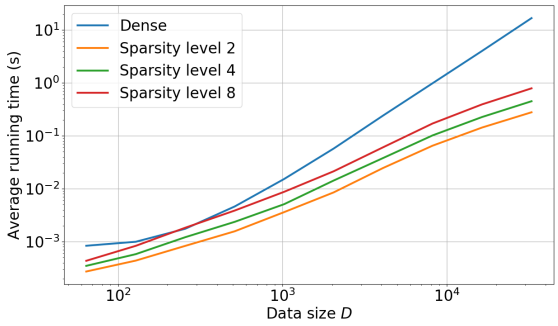


Figure 3: Running times, averaged over 30 runs, when applying dense or fast $D \times D$ operators to a set of 100 random vectors. The number of factors in fast operators equals $\log_2(D)$ and the sparsity level denotes the number of non-zero coefficients per row and per column in each factor.

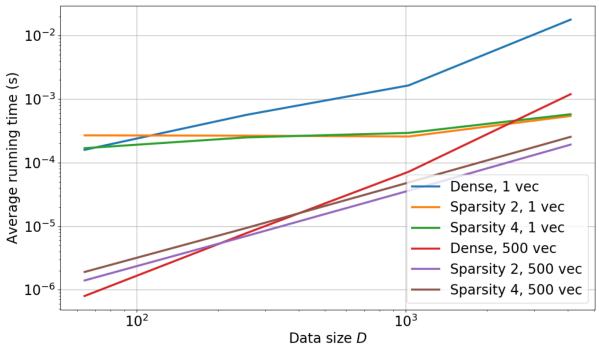
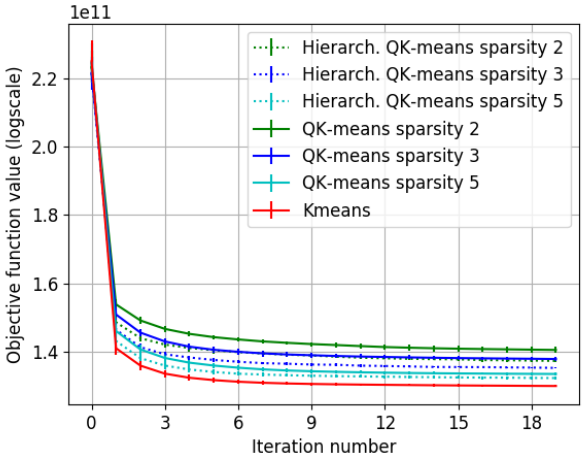


Figure 4: Running times,when applying dense or fast $D \times D$ operators to a set of 500 random vectors or a single vector. The number of factors in fast operators equals $\log_2(D)$ and the sparsity level denotes the number of non-zero coefficients per row and per column in each factor.

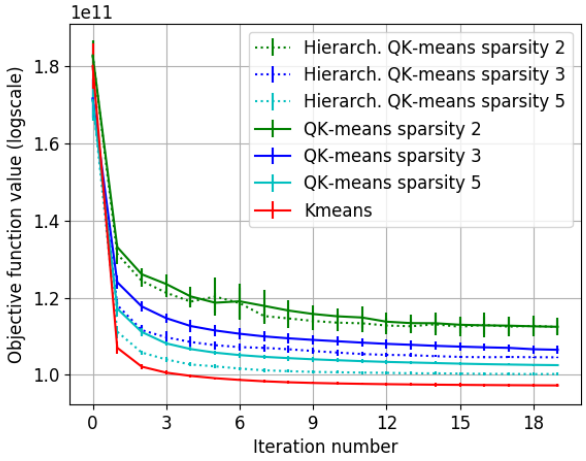
$$
\begin{aligned}
\sum_{n:t_n=k} \|\mathbf{x}_n - \mathbf{v}_k\|^2 &= \sum_{\substack{n \\ t_n=k}} \|\mathbf{x}_n - \mathbf{u}_k + \mathbf{u}_k - \mathbf{v}_k\|^2 \\
&= \sum_{\substack{n \\ t_n=k}} \left(\|\mathbf{x}_n - \mathbf{u}_k\|^2 + \|\mathbf{u}_k - \mathbf{v}_k\|^2 - 2\langle \mathbf{x}_n - \mathbf{u}_k, \mathbf{u}_k - \mathbf{v}_k \rangle \right) \\
&= \sum_{\substack{n \\ t_n=k}} \|\mathbf{x}_n - \mathbf{u}_k\|^2 + n_k \|\mathbf{u}_k - \mathbf{v}_k\|^2 - 2 \Big\langle \underbrace{\sum_{\substack{n \\ t_n=k}} (\mathbf{x}_n - \mathbf{u}_k)}_{=0}, \mathbf{u}_k - \mathbf{v}_k \Big\rangle \\
&= \sum_{\substack{n \\ t_n=k}} \|\mathbf{x}_n - \mathbf{u}_k\|^2 + \|\sqrt{n_k}\,(\mathbf{u}_k - \mathbf{v}_k)\|^2 \qquad (11) \\
&= \sum_{n:t_n=k} \|\mathbf{x}_n - \mathbf{u}_k\|^2 + n_k \|\mathbf{u}_k - \mathbf{v}_k\|^2, \qquad (12)
\end{aligned}
$$

E Hierarchical-Palm4MSA

The Hierarchical-`palm4MSA` algorithm has been used only for initialization of centroid sparse factors because it didn't improve that much the results when used on all iterations of `QK-means` while it was very time consuming. Figure 5 shows the evolution of the objective function and the final centroids images when using the hierarchical version in all iteration.

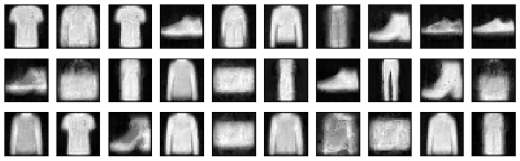


(a) MNIST, $K = 30$: objective function.



(b) Fashion-MNIST, $K = 30$: objective function.

(c) Hierarchical-`palm4MSA` `QK-means` centers.



(d) Hierarchical-`palm4MSA` `QK-means` centers.

Figure 5: Clustering results on MNIST (left) and Fashion-MNIST (right) for $K = 30$ clusters. Results marked as "hierarchical" refers to the one obtained with hierarchical-`palm4MSA` usage inside of the `QK-means` algorithm

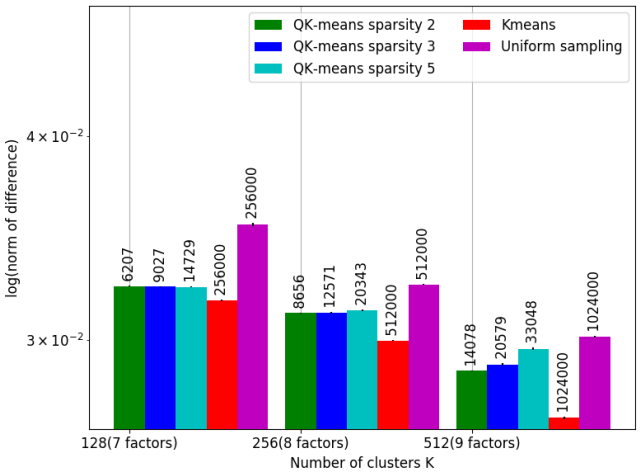
F Nyström approximation

Nyström approximation results: approximation accuracy (left) and running times (right). The uniform sampling based Nyström approximation running times are not displayed because they are the same as for the Nyström approximation based on `K-means` centers. Every experiment results are averaged over 5 runs. The vertical black lines are the standard deviation w.r.t. the runs. Numbers on top of each bar shows the number of parameter. Results are shown in an histogram form rather than table for clearer interpretation. The `QK-means` approach gives better reconstruction error than the Nyström method based on uniform sampling although it is slightly worse than the one obtained with the regular `K-means` centers. We see that that the difference in approximation error between `K-means` and `QK-means` is almost negligible when compared to the approximation error obtained with the uniform sampling scheme.

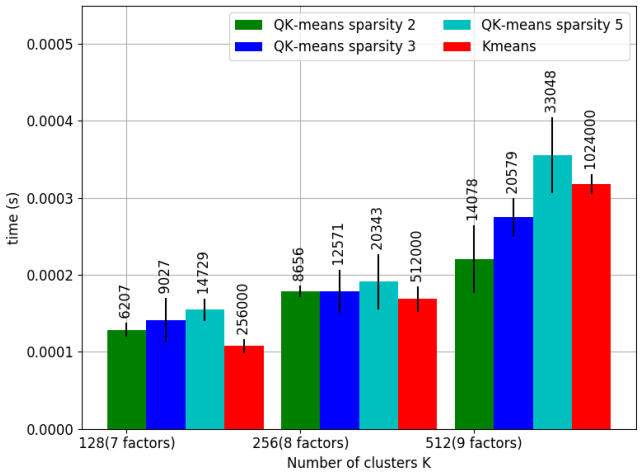
We show the approximation error of the Nyström approximation based on different sampling schemes w.r.t. the real kernel matrix. This error is computed by the Froebenius norm of the difference between the matrices and then normalized:

$$error = \frac{||\mathbf{K} - \tilde{\mathbf{K}}||_F}{||\mathbf{K}||_F}. \tag{13}$$

F.1 `Blobs`

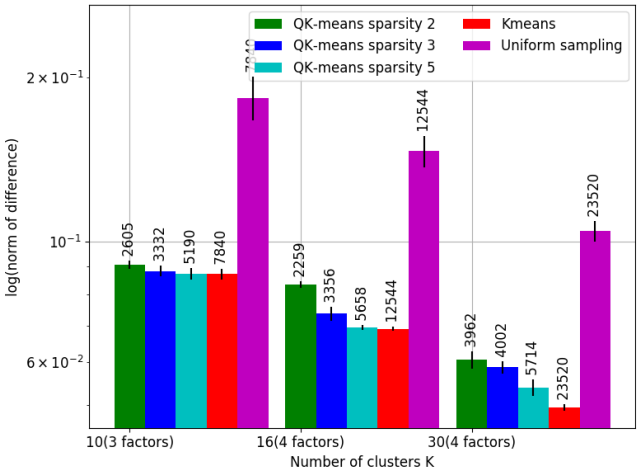


(a) `Blobs`: Nyström reconstruction error.

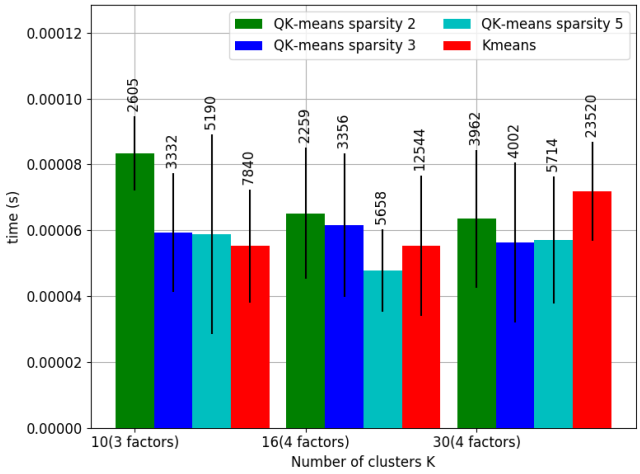


(b) `Blobs`: Nyström inference time.

F.2 `Mnist`

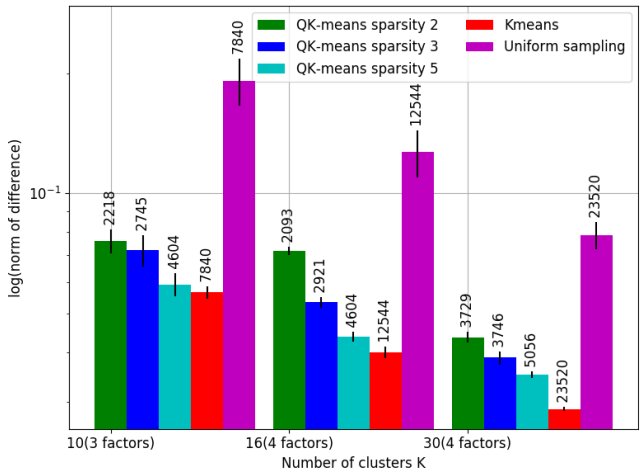


(a) `MNIST`: Nyström reconstruction error.

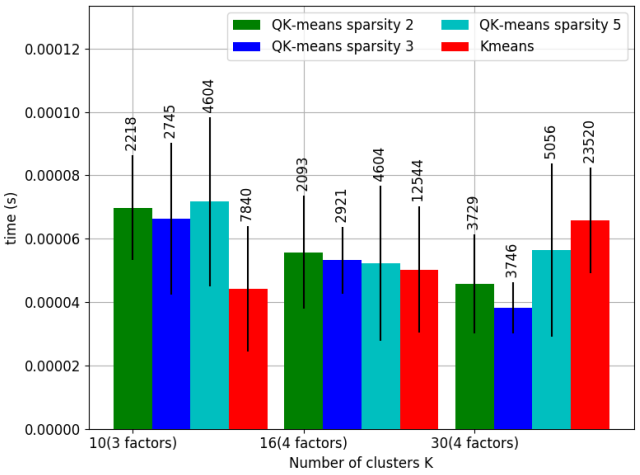


(b) `MNIST`: Nyström inference time.

F.3 `Fashion-MNIST`

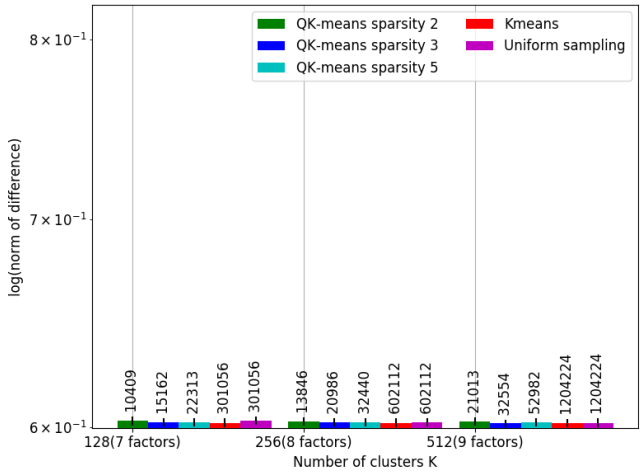


(a) `Fashion-MNIST`: Nyström reconstruction error.

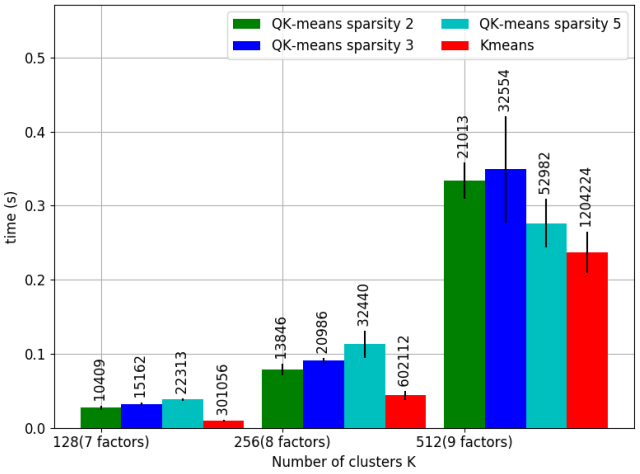


(b) `Fashion-MNIST`: Nyström inference time.

F.4 `Caltech256`



(a) `Caltech256`: Nyström reconstruction error.



(b) `Caltech256`: Nyström inference time.