Soft Computing (2020) 24:15317–15326
https://doi.org/10.1007/s00500-020-04865-0

METHODOLOGIES AND APPLICATION

Non-convex low-rank representation combined with rank-one matrix sum for subspace clustering

Xiaofang Liu[2] · Jun Wang[1] · Dansong Cheng[1] · Daming Shi[3] · Yongqiang Zhang[1]

Published online: 18 April 2020

Abstract
Exploring the multiple subspace structures of data such as low-rank representation is effective in subspace clustering. Non-convex low-rank representation (NLRR) via matrix factorization is one of the state-of-the-art techniques for subspace clustering. However, NLRR cannot scale to problems with large n (number of samples) as it requires either the inversion of an $n \times n$ matrix or solving an $n \times n$ linear system. To address this issue, we propose a novel approach, NLRR++, which reformulates NLRR as a sum of rank-one components, and apply a column-wise block coordinate descent to update each component iteratively. NLRR++ reduces the time complexity per iteration from $\mathcal{O}(n^3)$ to $\mathcal{O}(mnd)$ and the memory complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(mn)$, where m is the dimensionality and d is the target rank (usually $d \ll m \ll n$). Our experimental results on simulations and real datasets have shown the efficiency and effectiveness of NLRR++. We demonstrate that NLRR++ is not only much faster than NLRR, but also scalable to large datasets such as the ImageNet dataset with 120K samples.

Keywords Subspace clustering · Non-convex low-rank representation · Block coordinate descent · Rank-one matrix

1 Introduction

Many data mining and machine learning applications involve high-dimensional data, such as images, videos and documents (Wang et al. 2019; Zhang et al. 2019; Tang and Wei 2019; Ding et al. 2018; Fan et al. 2016; Du et al. 2017, 2018). In practice, such high-dimensional datasets can often be well approximated by multiple low-dimensional subspaces corresponding to multiple classes or categories (Fahmi et al. 2018; Wang et al. 2018; Amin et al. 2019; Tang et al. 2019). In the past decades, the subspace clustering as one of the most important machine learning technologies has been widely used in computer vision (Wang et al. 2016; Zhou et al. 2013) and machine learning (Bian et al. 2017; Du et al. 2017, 2016; Jia et al. 2017; Cheng et al. 2013). For instance, low-rank representation (LRR) (Liu et al. 2013) and sparse subspace clustering (SSC) (Elhamifar and Vidal 2013) aim to obtain a low-rank structure of multiple subspaces to fit high-dimensional data. However, both LRR and SSC require $\mathcal{O}(n^3)$ time complexity, so they are not able to scale to problems with large n (number of samples). Moreover, since LRR involves a nuclear norm regularization penalty of an n-by-n matrix, it requires computing singular value decomposition (SVD) (Wall et al. 2003) of an n-by-n matrix in every iteration.

In order to scale up to larger datasets, some subspace clustering methods have been proposed to mitigate the computation and memory cost. Non-convex low-rank representation (NLRR) (Shen and Li 2016) was proposed to alleviate the computational cost of LRR. The main idea is to take a non-convex reformulation of nuclear norm into account to avoid the repeated SVD computations in nuclear norm optimization solvers. However, it is not efficient to solve NLRR directly due to the use of an auxiliary variable, which leads to solving an $n \times n$ linear system at each step. To further address this computation issue, online low-rank subspace clustering (OLRSC) (Shen et al. 2016)

Communicated by V. Loia.

✉ Dansong Cheng
cdsinhit@hit.edu.cn

1 School of Computer Science and Technology, Harbin Institute Technology, 92 West Dazhi Street, Nan Gang District, Harbin, People's Republic of China

2 School of Electrical Engineering and Automation, Harbin Institute of Technology, 92 West Dazhi Street, Nan Gang District, Harbin, People's Republic of China

3 College of computer and software, Shenzhen University, 3688 Nanhai Ave, Nanshan District, Shenzhen, People's Republic of China

and online robust PCA (ORPCA) (Feng et al. 2013) were proposed. Despite that the time and memory cost per iteration are independent of n, OLRSC and ORPCA suffer from much slower convergence compared with batch NLRR algorithms.

Instead of using online updates, we propose a scalable algorithm NLRR in this paper, called NLRR++, which reduces the time complexity from $\mathcal{O}(n^3)$ to $\mathcal{O}(mnd)$ (usually $d \ll m \ll n$). NLRR++ is based on the main finding that by formulating $\mathbf{U}\mathbf{V}^{\mathrm{T}}$ in Eq. (3) (NLRR problem) as sum of rank-one matrices and updating the variables in a block coordinate descent manner, the NLRR problem can be solved without adding any auxiliary variable. However, the update for $\mathbf{u}_i$ (a column of $\mathbf{U}$) involves inverting an m-by-m matrix which leads to $\mathcal{O}(m^3)$ computational cost. By exploiting the structure of the linear system and pre-computing the SVD of the inverse part, NLRR++ is able to solve this linear system in $\mathcal{O}(mn)$ time. We show that NLRR++ converges to stationary points. Moreover, to address the issue of the high demand in memory and computational time for the final spectral clustering phase, we also propose an efficient clustering scheme that can further boost the performance of subspace clustering. Our experiments results on synthetic and real-world datasets show that the proposed algorithm is much faster than state-of-the-art space clustering approaches such as LRR (Liu et al. 2013), SSC (Elhamifar and Vidal 2013), NLRR (Shen and Li 2016), OLRSC (Shen et al. 2016) and ORPCA (Feng et al. 2013).

The spectral clustering has been used for subspace clustering (Von Luxburg 2007), which usually follows two steps. In the first step, a symmetric affinity matrix $\mathbf{W}$ is constructed. And in the second step, a weighted undirected graph is constructed, and the segmentation of the data is then found by clustering the eigenvectors of the graph Laplacian using some traditional clustering techniques such as K-means. One challenge is to build an affinity matrix to capture the relationship among data points. Most recent methods to construct affinity matrix target the sparse and low-rank representation of data points. The low-rank representation of the data matrix is then converted into symmetric and nonnegative affinity matrix, from which the segmentation is done by using spectral clustering. However, the spectral clustering-based methods cannot scale to large datasets due to the high cost of memory and computation time, as the affinity matrix $\mathbf{W} \in \mathbb{R}^{n \times n}$ is usually dense, demanding huge memory when n is large.

In this paper, we focus on LRR, which is guaranteed to have robust multiple subspace segmentation performance. LRR is to solve the following optimization problem:

$$\min_{\mathbf{X},\mathbf{E}} \frac{1}{2}\|\mathbf{Z} - \mathbf{A}\mathbf{X} - \mathbf{E}\|_F^2 + \beta\|\mathbf{X}\|_* + \lambda\|\mathbf{E}\|_{2,1}. \tag{1}$$

Here, $\mathbf{Z} = [\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n] \in \mathbb{R}^{m \times n}$ is the observation matrix with n samples and m features ($m \ll n$), and $\mathbf{A} \in \mathbb{R}^{m \times n}$ is a given dictionary matrix. Without any prior knowledge of dictionary, it is often assumed that $\mathbf{A} = \mathbf{Z}$. LRR aims to estimate two matrices: $\mathbf{E}$ is a sparse matrix capturing the corruptions of observations, and $\mathbf{X} \in \mathbb{R}^{n \times n}$ captures the low-rank structure of the observed data. Finally, the regularization parameters β, λ can be tuned to control the sparsity and low rankness of the solution.

If all the samples lie exactly in a small number of subspaces, under certain conditions the solution $\mathbf{X}$ of Eq. (1) will be a block-diagonal matrix up to permutation. The results can thus be used for subspace clustering. In practice, the representation matrix $\mathbf{X} = \mathbf{U}_x \boldsymbol{\Sigma}_x \mathbf{V}_x^{\mathrm{T}}$ can be used to construct an affinity matrix $\mathbf{W} = \mathbf{V}_x \mathbf{V}_x^{\mathrm{T}}$, which is then fed into spectral clustering to get the final clustering result.

Despite the desirable theoretical properties, LRR methods cannot scale to large datasets due to high memory and computational cost. To solve Eq. (1), we need to store $\mathbf{X}$ matrix, which needs ($\mathcal{O}(n^2)$) memory. Obviously, it is unfeasible to store for large-scale datasets with millions of samples. Furthermore, solving an optimization problem with nuclear norm penalty $\|\mathbf{X}\|_*$ involves singular values decomposition in every iteration, which is also computationally expensive (Recht et al. 2010).

Recently, Shen and Li (2016) mitigated the computation issue by introducing a non-convex reformulation of Eq. (1). Assuming the rank of $\mathbf{X}$ is at most d, Fazel et al. (2001) showed that:

$$\min_{\mathbf{U},\mathbf{V},\mathbf{X}=\mathbf{U}\mathbf{V}} \|\mathbf{X}\|_* = \frac{1}{2}(\|\mathbf{U}\|_F^2 + \|\mathbf{V}\|_F^2), \tag{2}$$

where $\mathbf{U} \in \mathbb{R}^{m \times d}, \mathbf{V} \in \mathbb{R}^{n \times d}$. Therefore, LRR in Eq. (1) has an equivalent but non-convex formulation (NLRR) as follows:

$$\min_{\mathbf{U},\mathbf{V},\mathbf{E}} \frac{1}{2}\|\mathbf{Z}-\mathbf{A}\mathbf{U}\mathbf{V}^{\mathrm{T}} - \mathbf{E}\|_F^2 + \frac{\beta}{2}(\|\mathbf{U}\|_F^2 + \|\mathbf{V}\|_F^2) + \lambda\|\mathbf{E}\|_{2,1}. \tag{3}$$

In order to solve this NLRR problem, a straightforward way is to update the three variables $\mathbf{U}, \mathbf{V}, \mathbf{E}$ alternately until convergence. However, Shen and Li (2016) pointed out that this alternating minimization scheme may be intractable. Assume $\mathbf{V}, \mathbf{E}$ are fixed, solving the subproblem with respect to $\mathbf{U}$ can be computationally intensive as the optimal $\mathbf{U}$ is:

$$\mathrm{vec}(\mathbf{U}) = (\mathbf{V}^{\mathrm{T}}\mathbf{V} \otimes (\mathbf{A}^{\mathrm{T}}\mathbf{A}) + \beta^{-1}\mathbf{I}_{mn})\mathrm{vec}(\mathbf{A}^{\mathrm{T}}(\mathbf{Z} - \mathbf{E})\mathbf{V}), \tag{4}$$

where $\otimes$ is a Kronecker product. In Eq. (4), $\mathbf{V}^{\mathrm{T}}\mathbf{V} \otimes (\mathbf{A}^{\mathrm{T}}\mathbf{A})$ is an mn-by-mn matrix, so the time complexity of solving (4) is $\mathcal{O}(m^3n^3)$ so that memory cost is $\mathcal{O}(m^2n^2)$ which is not practical even in small-scale applications. To solve the variable $\mathbf{U}$ efficiently, Shen and Li (2016) proposed to add an auxiliary variable $\mathbf{D} = \mathbf{AU}$ to Eq. (3), and then, updating $\mathbf{U}$ becomes an ordinary least square problem by differentiating it with respect to $\mathbf{U}$ as follows (details can be found in Ref. Shen and Li 2016):

$$\text{(NLRR-1)} \quad \mathbf{U} = (\mu\mathbf{A}\mathbf{A}^{\mathrm{T}} + \mathbf{I}_n)^{-1}\mathbf{A}^{\mathrm{T}}(\mathbf{W} + \mu\mathbf{D}). \tag{5}$$

As we can see from the above equation, introducing auxiliary variables can reduce the computational and memory cost of computing $\mathbf{U}$ to $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$, respectively.

2 Reformulated non-convex low-rank representation model

In this section, we introduce our proposed algorithm, NLRR++. By reformulating NLRR problem in Eq. (3), we show that a block coordinate descent algorithm can be used to solve NLRR efficiently without adding any auxiliary variable. In Sect. 2.1, we describe the one-variable update rule in detail. And in Sect. 2.2, we discuss how to group variables into blocks so that the block coordinate descent can be run efficiently, followed by describing several further improvements and theoretical guarantee in the rest of this section.

2.1 Solving strategy via block coordinate descent method

We reformulate the NLRR problem in Eq. (3) in order to mitigate the computational cost and memory requirement. Note that the solution $\mathbf{U}\mathbf{V}^{\mathrm{T}}$ can be represented as a sum of d outer products:

$$\mathbf{X} = \mathbf{U}\mathbf{V}^{\mathrm{T}} = \sum_{t=1}^{d} \mathbf{u}_t\mathbf{v}_t^{\mathrm{T}} \tag{6}$$

where $\mathbf{u}_t, \mathbf{v}_t, (t = 1, \ldots, d)$ are the column vectors of $\mathbf{U}, \mathbf{V}$ respectively. From the perspective of latent feature space, $\mathbf{u}_t$ and $\mathbf{v}_t$ correspond to the tth latent feature. Therefore, the objective function of Eq. (3) can be written as follows:

$$\begin{aligned} g(\{\mathbf{u}_t\}_{t=1}^{d}, \{\mathbf{v}_t\}_{t=1}^{d}, \mathbf{E}) &= \frac{1}{2}\|\mathbf{Z} - \mathbf{A}\sum_{t=1}^{d}\mathbf{u}_t\mathbf{v}_t^{\mathrm{T}} - \mathbf{E}\|_F^2 \\ &+ \frac{\beta}{2}\left(\sum_{t=1}^{d}\|\mathbf{u}_t\|_2^2 + \sum_{t=1}^{d}\|\mathbf{v}_t\|_2^2\right) + \lambda\|\mathbf{E}\|_{2,1}. \end{aligned} \tag{7}$$

In large-scale applications and related optimization problems, it is difficult to solve all variables at one time in each iteration because of the large data and variables. The block coordinate descent (BCD) approach is effective to solve this problem. It divides variables into smaller blocks and updates only one of them in each iteration, thus simplifying the solution of variables. The main idea of BCD is to update only one block coordinate variable at a time when the other variables are fixed. Hence, the key of applying BCD is to design the subproblem of updating single variable and the order of updating variables.

In the following, we propose a BCD algorithm for solving Eq. (7). In the outer loop, we alternately update $\mathbf{U}, \mathbf{V}$ when fixing $\mathbf{E}$ and update $\mathbf{E}$ when fixing $\mathbf{U}, \mathbf{V}$. In the inner loop, update variables $\mathbf{U}, \mathbf{V}$ in turn by BCD. First, we discuss how to update $\mathbf{U}, \mathbf{V}$ when fixing $\mathbf{E}$. In fact, Shen and Li (2016) discussed about this approach and found out it is impossible to update the whole $\mathbf{U}$ or $\mathbf{V}$ efficiently. But here, we show that by splitting these two matrices into columns, each column actually has a closed form solution that can be computed efficiently. Minimizing Eq. (7) with respect to $\mathbf{u}_t, \mathbf{v}_t (t = 1, \ldots, d)$, it can then be written as:

$$\begin{aligned} g(\{\mathbf{u}_t\}_{t=1}^{d}, \{\mathbf{v}_t\}_{t=1}^{d}) = \beta\sum_{t=1}^{d}\|\mathbf{u}_t\|_2^2 + \beta\sum_{t=1}^{d}\|\mathbf{v}_t\|_2^2 \\ + \|\mathbf{Z} - \mathbf{A}\sum_{t=1}^{d}\mathbf{u}_t\mathbf{v}_t^{\mathrm{T}} - \mathbf{E}\|_F^2. \end{aligned} \tag{8}$$

In the following, we show that the subproblem in Eq. (8) can be solved efficiently by a column-wise coordinate descent, where at each step, we update the whole column of $\mathbf{u}_t$ or $\mathbf{v}_t$ together.

– Update $\mathbf{u}_t (t = 1, \ldots, d)$: When $\mathbf{E}, \mathbf{V}, \mathbf{u}_j (j \neq t)$ are fixed, differentiating the objective function in Eq. (8) with respect to $\mathbf{u}_t$ and arranging the other terms yield

$$\mathbf{u}_t = (c_t\mathbf{A}^{\mathrm{T}}\mathbf{A} + \beta\mathbf{I}_n)^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{R}_t\mathbf{v}_t \tag{9}$$

where $\mathbf{R}_t = \mathbf{Z} - \mathbf{A}\sum_{j\neq t}\mathbf{u}_j\mathbf{v}_j^{\mathrm{T}} - \mathbf{E}$ is the tth residual matrix and $c_t = \|\mathbf{v}_t\|_2^2$. However, computing $\mathbf{Q}\mathbf{v} = (c_t\mathbf{A}^{\mathrm{T}}\mathbf{A} + \beta\mathbf{I}_n)^{-1}\mathbf{v}$ (where $\mathbf{v} = \mathbf{A}^{\mathrm{T}}\mathbf{R}_t\mathbf{v}_t$ is a vector and $\mathbf{Q} \in \mathbb{R}^{n\times n}$) requires inverting a n-by-n matrix or solving an n-dimensional linear systems, which takes $O(n^3)$ time.

To resolve this problem, we need to exploit the structure of the $\mathbf{Q}$ matrix. Note that $\mathbf{Q} = c_t(\mathbf{A}^{\mathrm{T}}\mathbf{A} + \frac{\beta}{c_t}\mathbf{I}_n)^{-1}$, so the first part remains unchanged throughout the whole optimization procedure, and only the second term changed by a constant c_t. Therefore, by caching some information about $\mathbf{A}^{\mathrm{T}}\mathbf{A}$, we can compute the inverse efficiently. In order to do this, we first use the Woodbury matrix identity[1] to transform the dimensions of the inverse operation from n-by-n to m-by-m (m ≪n), i.e.,

$$\mathbf{Q} = \frac{1}{\beta}\mathbf{I}_n - \frac{1}{\beta^2}\mathbf{A}^{\mathrm{T}}\left(\frac{c_t}{\beta}\mathbf{A}\mathbf{A}^{\mathrm{T}} + \mathbf{I}_m\right)^{-1}\mathbf{A}.$$

Then, Eq. (9) can be written as follows:

$$\mathbf{u}_t = \frac{1}{\beta}\mathbf{A}^{\mathrm{T}}\mathbf{R}_t\mathbf{v}_t - \frac{1}{\beta^2}\mathbf{A}^{\mathrm{T}}\mathbf{Q}_m\mathbf{A}\mathbf{A}^{\mathrm{T}}\mathbf{R}_t\mathbf{v}_t \tag{10}$$

where $\mathbf{Q}_m = (\frac{c_t}{\beta}\mathbf{A}\mathbf{A}^{\mathrm{T}} + \mathbf{I}_m)^{-1}$. To update $\mathbf{u}_t$, we have to compute the matrix inverse $(\frac{c_t}{\beta}\mathbf{A}\mathbf{A}^{\mathrm{T}}+\mathbf{I}_m)^{-1}$ in every iteration which requires the computational complexity of $\mathcal{O}(m^3)$.

Now, we take the structure of $\mathbf{Q}_m$ into account and make adjustments to $\mathbf{Q}_m$. Outside the outer loop of the algorithm, we pre-compute the SVD of $\mathbf{A}\mathbf{A}^{\mathrm{T}} = \mathbf{U}_*\boldsymbol{\Sigma}_*\mathbf{V}_*^{\mathrm{T}}$ in the beginning of the algorithm, and then, $\mathbf{Q}_m$ can be written as

$$\mathbf{Q}_m = \mathbf{U}_*\mathrm{diag}\left(\frac{1}{1+\frac{c_t}{\beta}\mathrm{diag}(\boldsymbol{\Sigma}_*)}\right)\mathbf{V}_*^{\mathrm{T}}. \tag{11}$$

It can be seen from the above that in each iteration, only c_t is changeable, but U_*,Σ_*, V_*^{T} are unchanged. Therefore, calculating $\mathbf{u}_t$ only includes the matrix vector product whose computational complexity is $\mathcal{O}(mn)$.

- Update $\mathbf{v}_t(t = 1, \dots, d)$: When $\mathbf{E}, \mathbf{V}, \mathbf{v}_j(j \neq t)$ are fixed, taking the gradient of Eq. (8) with respect to $\mathbf{v}_t$ and setting it to zero, we can get the update rule for $\mathbf{v}_t$:

$$\mathbf{v}_t^{\mathrm{T}} = \frac{1}{d_t+\beta}(\mathbf{A}\mathbf{u}_t)^{\mathrm{T}}\mathbf{R}_t, (d_t = \mathbf{u}_t^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{u}_t). \tag{12}$$

- We follow Ref. Hale et al. (2008) to find the local minimizer of $\mathbf{E}$ by the following soft-thresholding operator when $\mathbf{u}_t, \mathbf{v}_t$ are fixed:

$$[\mathbf{E}]_{:,\mathbf{i}} = S_\lambda([\mathbf{R}]_{:,\mathbf{i}}), \tag{13}$$

where $[E]_{:,i}$ is ith column vector of matrix E, $\mathbf{R} = \mathbf{R}_t - \mathbf{A}\mathbf{u}_t\mathbf{v}_t^{\mathrm{T}}$, and the soft-thresholding operator is defined by

$$S_\lambda([\mathbf{R}]_{:,\mathbf{i}}) = \max\{\|[\mathbf{R}]_{:,\mathbf{i}}\|_2 - \lambda, 0\}\frac{[R]_{:,i}}{\|[R]_{:,i}\|_2}\ (i = 1, \dots, n).$$

2.2 Strategy for updating $\mathbf{u_t}$, $\mathbf{v_t}$

To update the columns of $\mathbf{U}$, $\mathbf{V}$, there are multiple choices of the update sequence. In this subsection, we investigate two main inner updating sequences.

[1] $(\mathbf{A}+\mathbf{UCV})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{C}^{-1}+\mathbf{V}\mathbf{A}^{-1}\mathbf{U})^{-1}\mathbf{V}\mathbf{A}^{-1}$.

The first choice is to update all the columns of $\mathbf{U}$ and then all the columns of $\mathbf{V}$:

$$\text{(I)}\ \begin{aligned}&\underbrace{\mathbf{R}_1, \mathbf{u}_1, \mathbf{R}, \mathbf{R}_2, \mathbf{u}_2, \mathbf{R}, \dots, \mathbf{R}_d, \mathbf{u}_d, \mathbf{R}}_{\mathbf{U}},\\ &\underbrace{\mathbf{R}_1, \mathbf{v}_1, \mathbf{R}, \mathbf{R}_2, \mathbf{v}_2, \mathbf{R}, \dots, \mathbf{R}_d, \mathbf{v}_d, \mathbf{R}}_{\mathbf{V}}\end{aligned} \tag{14}$$

Note that before updating $\mathbf{u}_t$ by Eq. (9) or $\mathbf{v}_t$ by Eq. (12), the tth residual matrix $\mathbf{R}_t = \mathbf{Z} - \mathbf{A}\sum_{j\neq t}\mathbf{u}_j\mathbf{v}_j^{\mathrm{T}} - \mathbf{E}$ needs to be updated at first, and once $\mathbf{u}_t$ or $\mathbf{v}_t$ is updated, the real residual matrix $\mathbf{R} = \mathbf{Z} - \mathbf{A}\sum_{i=1}^{d}\mathbf{u}_i\mathbf{v}_i^{\mathrm{T}} - \mathbf{E}$ needs to be calculated too by adding the updated $\mathbf{u}_t\mathbf{v}_t^{\mathrm{T}}$. The updates of $\mathbf{R}$, $\mathbf{R}_t$ are also included in the update sequence of Eq. (14). In this approach, each update of $\mathbf{u}_t$ and $\mathbf{v}_t$ will trigger the update of $\mathbf{R}_t$ and $\mathbf{R}$.

An alternative "feature-wise approach" is to update $\mathbf{u}_t$ in company with $\mathbf{v}_t$; thus, the update frequency of $\mathbf{R}$ and $\mathbf{R}_t$ is reduced in half. The detailed updating sequences are as follows:

$$\text{(II)}\ \underbrace{\mathbf{R}_1, \mathbf{u}_1, \mathbf{v}_1, \mathbf{R}}_{(\mathbf{U}^1,\mathbf{V}^1)}, \underbrace{\mathbf{R}_2, \mathbf{u}_2, \mathbf{v}_2, \mathbf{R}}_{(\mathbf{U}^2,\mathbf{V}^2)}, \dots, \underbrace{\mathbf{R}_d, \mathbf{u}_d, \mathbf{v}_d, \mathbf{R}}_{(\mathbf{U}^d,\mathbf{V}^d)}. \tag{15}$$

Moreover, between each update of $\mathbf{R}_t$ and $\mathbf{R}$, we can update $\mathbf{u}_t$, $\mathbf{v}_t$ by T times, and within these T inner updates, we do not need to update residual matrix, as explained in detail in Sect. 4.

Our experiments show that approach (II) demonstrates better clustering performance and higher efficiency than the approach (I) does. The algorithm outline of the approach (II) is presented in Algorithm 1.

2.3 Accelerate NLRR++

We accelerate the NLRR++ based on the approach (II) (in Eq. (15)) by controlling the number of inner iterations T for updating $\mathbf{u}_t$, $\mathbf{v}_t$ (Yu et al. 2012). NLRR++ could be slightly more efficient when $T > 1$ due to the benefit brought by the "delayed residual update." Note that $\mathbf{R}$ and $\mathbf{R}_t$ are fixed during NLRR++ iterations for each rank-one approximation problem in Eq. (8). Therefore, $\mathbf{R}$ and $\mathbf{R}_t$ are required to be updated only when we switch to the next subproblem corresponding to another feature. Moreover, the more the inner iterations we use, the better the approximation to subproblem in Eq. (8). Hence, a direct approach to accelerate NLRR++ is to increase T. But a large and fixed T might result in overhead on a single subproblem.

We close this section by a comparison between NLRR and NLRR++ in Fig. 1. Here, we compare four settings with the ImageNet dataset: Two settings are NLRR-1, and three others are NLRR++ with fixed $T = 1$ (NLRR++T1) (using the (II) approach), fixed $T = 4$ (NLRR++T4) (using the (II)

Algorithm 1 NLRR++ by Solving Eq. (7)

```
1: Input: Matrix Z, the parameters $d, \varepsilon, \beta, \lambda > 0$
2: Output: U, V, E.
3: Initialization: Generate $\mathbf{u}_t, \mathbf{v}_t$ from Gaussian distribution;
4: $\mathbf{R} = \mathbf{Z} - \mathbf{A}\sum_{i=1}^{d}\mathbf{u}_t^{\mathrm{T}}\mathbf{v}_t - \mathbf{E}$; $\mathbf{A}\mathbf{A}^{\mathrm{T}} = \mathbf{U}_*\boldsymbol{\Sigma}_*\mathbf{V}_*^{\mathrm{T}}$ ; $i = 1$ ; $\mathcal{L}_0 = g(\{\mathbf{u}_t\}_{t=1}^d, \{\mathbf{v}_t\}_{t=1}^d, \mathbf{E})$.
5: While not converged do
6:   for $t = 1 \cdots d$
7:   Update $\mathbf{R}_t = \mathbf{R} + \mathbf{A}\mathbf{u}_t\mathbf{v}_t^{\mathrm{T}}$;
8:     for $iter = 1 \cdots T$
9:       Update $\mathbf{u}_t$ in U using (10) and (11);
10:      Update $\mathbf{v}_t$ in V using (12);
11:    endfor
12:    Update $\mathbf{R} = \mathbf{R}_t - \mathbf{A}\mathbf{u}_t\mathbf{v}_t^{\mathrm{T}}$;
13:  endfor
14:  Update E via (13),
15:  Update object value:
16:  $\mathcal{L}_i = \frac{1}{2}\|\mathbf{R}\|_F^2 + \frac{1}{2}\beta(\|\mathbf{U}\|_F^2 + \|\mathbf{V}\|_F^2) + \lambda\|\mathbf{E}\|_1$
17:  If $\frac{\mathcal{L}_{i-1}-\mathcal{L}_i}{\mathcal{L}_0-\mathcal{L}_1} < \varepsilon$
18:    converged = true;
19:  endif
20:  $i = i + 1$;
21: end while
22: Return $\mathbf{U} = [\mathbf{u}_1 \cdots \mathbf{u}_d]$, $\mathbf{V} = [\mathbf{v}_1 \cdots \mathbf{v}_d]$, E
```

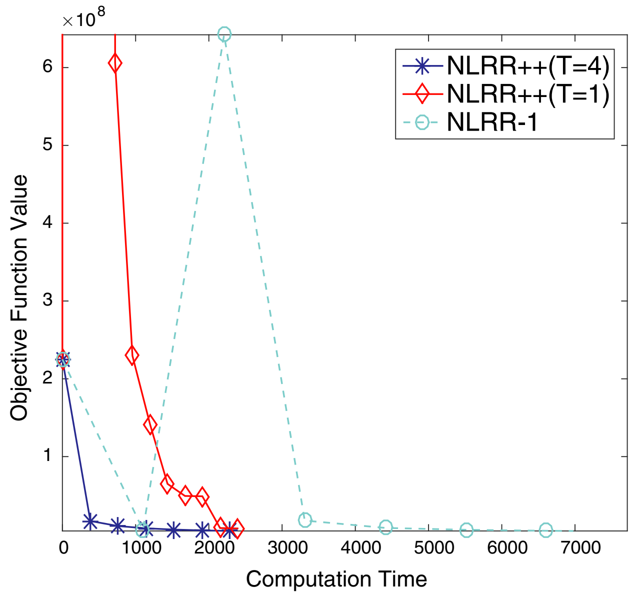

Fig. 1 The objective function value versus computation time with the comparison among NLRR++($T = 1$), NLRR++($T = 4$), NLRR++b (via the update approach (I) in Eq. (14)), NLRR-1 using ImageNet features ($N = 62{,}700$). Results show that our proposed update sequence has faster and more stable convergence

approach), NLRR++b (using the (I) approach), respectively. As shown in Fig. 1, the "feature-wise" update strategy by NLRR++ using approach (II), even when $T = 1$, converges faster than NLRR. We also observe that a larger T improves convergence of NLRR++.

Table 1 Comparison of time complexity memory cost of NLRR, NLRR++, OPRCA and OLRSC

	Time complexity	Memory cost
NLRR++	$\mathcal{O}(mnd)$	$\mathcal{O}(mn)$
NLRR (Shen and Li 2016)	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$
OLRSC (Shen et al. 2016)	$\mathcal{O}(mnd^2)$	$\mathcal{O}(md)$
ORPCA (Feng et al. 2013)	$\mathcal{O}(mnd^2)$	$\mathcal{O}(md)$

2.4 Time complexity and memory cost

Our NLRR++ via column-wise BCD is summarized in Algorithm 1. To analyze the computational and memory complexity, we assume $d \ll m < n$, since the sample dimension m is fixed, and the estimation d of subspace rank is generally smaller than the sample dimension. Only the number of samples n will increase with the increase in the dataset, and the rank d is usually less than 100.

In the inner iteration ($t = 1, \ldots, d$), both the updates of $\mathbf{R}_t$ and $\mathbf{R}$ in the step 7 and 12 cost $\mathcal{O}(mn)$. The updates of $\mathbf{u}_t$ and $\mathbf{v}_t$ in the step 9 and 10 cost $\mathcal{O}(3Tmn + 2Tm^2)$ and $\mathcal{O}(2Tmn)$, respectively, where T is a small fixed constant. Therefore, the time complexity of the inner iteration in Algorithm 1 can be summarized as $\mathcal{O}(dmn)$, and thus, the total time complexity of NLRR++ is $\mathcal{O}(T_1 mnd)$, where T_1 is the number of outer iterations scaling as $\mathcal{O}(\frac{1}{\varepsilon})$ (Richtrik and Tak 2014). In all the experiments shown in Sect. 4, T_1 is no more than 5. In terms of the memory cost, each of the steps in Algorithm 1 needs $\mathcal{O}(mn)$ memory.

We also compare our time complexity with other methods in Table 1 to clearly demonstrate the difference of time complexity and memory cost of NLRR++, NLRR (Shen and Li 2016), OLRSC (Shen et al. 2016) and ORPCA (Feng et al. 2013). Note that OLRSC and ORPCA are two online methods with the lowest memory cost which is in the sacrifice of the computational efficiency.

2.5 Theoretical convergence analysis

For NLRR++, $g(\{\mathbf{u}_t\}_{t=1}^d, \{\mathbf{v}_t\}_{t=1}^d, \mathbf{E})$ is convex in $\mathbf{u}_t$, $\mathbf{v}_t$ or $\mathbf{E}$ separately. Since we follow BCD update rule with $2t + 1$ blocks ($\{\mathbf{u}_t\}_{t=1}^d$, $\{\mathbf{v}_t\}_{t=1}^d$), and $\mathbf{E}$, their corresponding subproblems can be written as

$$\min_{\{\mathbf{u}_t\}_{t=1}^d,} g(\{\mathbf{u}_t\}_{t=1}^d, \{\mathbf{v}_t\}_{t=1}^d, \mathbf{E}), \tag{16}$$

$$\min_{\{\mathbf{v}_t\}_{t=1}^d} g(\{\mathbf{u}_t\}_{t=1}^d, \{\mathbf{v}_t\}_{t=1}^d, \mathbf{E}), \tag{17}$$

$$\min_{\mathbf{E}} g(\{\mathbf{u}_t\}_{t=1}^d, \{\mathbf{v}_t\}_{t=1}^d, \mathbf{E}). \tag{18}$$

According to the proposition 2.7.1 in Bertsekas (1999), if the solution of Eqs. (16), (17) and (18) is uniquely attained, then every limit point $\mathbf{u}_t^*$ of the sequences generated BCD is a stationary point. For Eqs. (16) and (17), due to the L2-regularization, the subproblems are strongly convex, so they clearly satisfy this condition. For Eq. (18), the solution given by the soft-thresholding is also uniquely attained. Therefore, our Algorithm 1 will converge to stationary points of problem in Eq. (7).

However, it is difficult to guarantee that the stationary point of non-convex optimization is globally optimal (Bertsekas 1999). Burer et al.'s work Burer and Monteiro (2005) shows that the local minimum solution obtained is also the global maximum solution when the data do not contain noise and the given rank is large enough. Usually, the algorithm stops at some stationary points instead of local minimum solution, but subsequent experiments show that the solution obtained by Algorithm 1 performs well in subspace clustering.

3 Clustering pipeline

The original pipeline of clustering is firstly using the representation matrix $\mathbf{X} \approx \mathbf{U}\mathbf{V}^{\mathrm{T}}$ to construct the n-by-n affinity matrix $\mathbf{W} = |\mathbf{V}||\mathbf{V}^{\mathrm{T}}|$, where $\mathbf{U}, \mathbf{V}$ are produced by the non-linear LRR approaches, and then feeding the affinity matrix to a spectral clustering algorithm (Ng et al. 2001) to get the final clustering result. However, the spectral clustering method is not practical when facing a n-by-n *dense* affinity matrix, because the memory cost and time complexity requirements are $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$, respectively. Therefore, the spectral clustering algorithm is not suitable for large datasets.

In this section, the K-means clustering method is implemented directly utilizing the n-by-d (d is the estimation rank of the dataset) matrix $\mathbf{U}, \mathbf{V}$ obtained from the non-convex low-rank representation model. Note that $\mathbf{X} \approx \mathbf{U}\mathbf{V}^{\mathrm{T}}$ is obtained from Eq. (1) or Eq. (3), and columns in $\mathbf{U} \in \mathbb{R}^{n\times d}$ denote the basis of $\mathbf{X}$ which spans the subspace of $\mathbf{X}$, and $\mathbf{V}$ denotes the corresponding coefficient matrix under the basis $\mathbf{U}$. Rather than directly feeding $\mathbf{V}$ to K-means algorithm as Shen and Li (2016) do, we introduce a dimension reduction step to avoid the curse of dimension especially when $d > 100$. We select k columns in $\mathbf{V}$ (assuming the cluster number k is known and usually $k \ll d$) which corresponds to the k principal components of $\mathbf{U}$ (measured by the column vector length) to construct a new dimension reduced representation matrix $\mathbf{V}_k$. Then, we normalize the rows of $\mathbf{V}_k$ and feed it to K-means clustering to obtain the final clustering result.

The comparison of the time-consuming and the clustering accuracy between the spectral clustering, K-means and our proposed methods is shown in Tables 2 and 3 where we feed $\mathbf{V}$ obtained by NLRR++ to the clustering methods. The spectral clustering computes SVD for an n-by-n similarity matrix which is quite slow and always dominates the running time of the whole task. The performance of K-means is not stable and not of high efficiency especially for large datasets.

Table 2 Time cost (seconds) of various post-processing schemes for subspace clustering

	USPS	Protein	M-20K	S-9K	I-30K	I-60K
Spectral	665.3	5307.1	1.7 h	764.6	2 h	7 h
K-means	10	28.52	28.43	14.22	65.58	1789.2
Ours	4	4.2	8.25	1.95	26.3	207.2

Here, we denote MNIST-20K by M-20K, SVHN-9K by S-9K and ImageNet-30K/60K by I-30K/I-60K

Table 3 Clustering accuracy (%) of various post-processing schemes for subspace clustering

	USPS	Protein	M-20K	S-9K	I-30K	I-60K
Spectral	51.76	45.0	63.1	33.25	90.3	76.7
K-means	67.9	42.76	25.01	42.1	65.2	28.7
Ours	66.25	43.2	79.1	41.0	95.69	90.89

Here, we denote MNIST-20K by M-20K, SVHN-9K by S-9K, and ImageNet-30K/60K by I-30K/I-60K

4 Experiments

Parameter settings and Environment We set $\beta = 0.8$ for both NLRR++ and NLRR (Shen and Li 2016), and number of inner iterations $T = 4$ for NLRR++. For the competing methods, i.e., OLRSC (Shen et al. 2016), ORPCA (Feng et al. 2013), LRR (Liu et al. 2013), SSC (Elhamifar and Vidal 2013) and CASS (Lu et al. 2014), we follow the default parameter settings specified in their papers.

All the experiments were executed on Intel Xeon E5-2640 2.4 GHz CPU with 64G RAM and Linux OS. Note that our algorithm can be easily parallelized since all the operations are matrix multiplications, and in fact, this can be automatically done in MATLAB if we assign more than one thread. Since all the other algorithms can also be parallelized by MATLAB,[2] we compare the algorithms under both single-thread and multi-thread settings.

4.1 Subspace clustering on simulation data

We use $four$ disjoint subspaces $\{\mathbf{S}_k\}_{k=1}^4 \in \mathbb{R}^m$, whose bases are denoted by $\{\mathbf{L}_k\}_{k=1}^4 \in \mathbb{R}^{m\times d_k}$. The data matrix $\mathbf{Z}_k \in S_k$ is then produced by $\mathbf{Z}_k = \mathbf{L}_k\mathbf{R}_k^{\mathrm{T}}$, where $\mathbf{R}_k \in \mathbb{R}^{n_k\times d_k}$. The entries of $\mathbf{L}_k$'s and $\mathbf{R}_k$'s are sampled from the normal distribution.

[2] Our source code is available at https://github.com/junwang929/subspace-clustering.

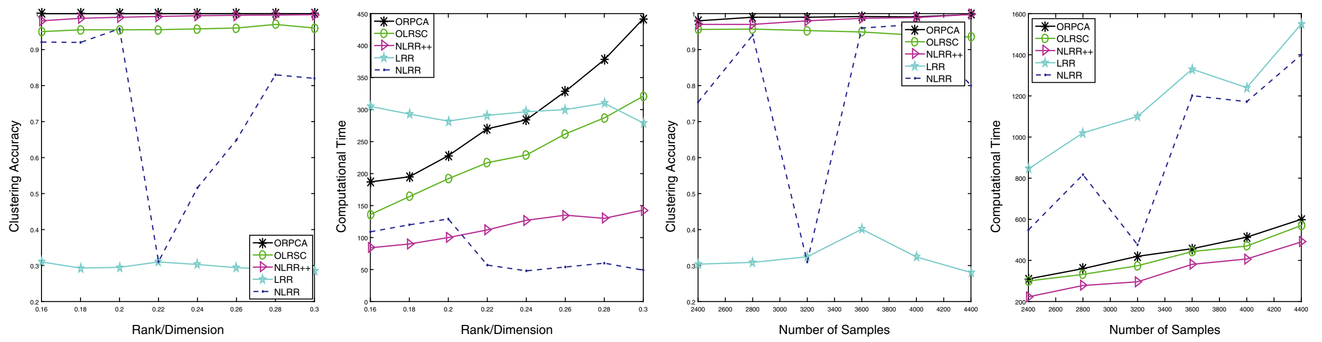

Fig. 2 Clustering accuracy and time complexity (in seconds). The first two subfigures show synthetic data with $m = 1000$, $k = 4$, $n_k = 1000$ and $\rho = 0.02$. The x-axis represents the *Rank/Dimension*. Note that NLRR takes less time while staying at bad stationary points with much lower clustering accuracy, especially when *Rank/Dimension* is larger than 0.2. The last two figures show synthetic data with $m = 1000$, $k = 4$, $d_k = 40$ and $\rho = 0.02$. The x-axis represents number of samples in each subspace. NLRR++ has a nice clustering performance and meanwhile has the lowest computational time

We generate the data with fixed dimension $m = 1000$, corruption rate $\rho = 0.02$ and set $\lambda = \frac{1}{\sqrt{m}}$ for both NLRR and NLRR++. In the first two subfigures of Fig. 2, we fix the number of samples in each subspace and show how the change of *Rank/Dimension*, which represents the ratio between the number of basis and the ambient dimension, affects the ACC (clustering accuracy) and the computational time (in seconds). As seen from the first two subfigures, the accuracy of NLRR++, OLRPCA and OLRSC is not much different, but it is far better than that of NLRR and LRR algorithms. In terms of the convergence speed, NLRR and NLRR++ algorithms are faster than other methods. Although NLRR is sometimes faster than NLRR++, its accuracy is poor due to the premature convergence of NLRR. In the last two subfigures of Fig. 2, we show the effect of the increase in sample number on the clustering accuracy and the running time of the algorithm. Fixed remaining parameters are $m = 1000$, $k = 4$, DK $= 40$ and $P = 0.02$. NLRR++ embodies higher clustering accuracy and the fastest convergence speed. The increase in sample size has a relatively small impact on the efficiency of NLRR++, OLRSC and OLRPCA algorithms. The convergence speed of LRR and NLRR methods decreases obviously with the increase in samples.

In Fig. 3, we plot the training curve of NLRR and NLRR++ in terms of objective function since they are solving the same optimization problem in Eq. (3). We set the *Rank/Dimension* as 16% and the number of samples in each subspace as 2800. The results clearly show that NLRR++ is much faster than NLRR.

4.2 Subspace clustering on real datasets

We examine the performance for subspace clustering on five real databases. The first four datasets can be downloaded

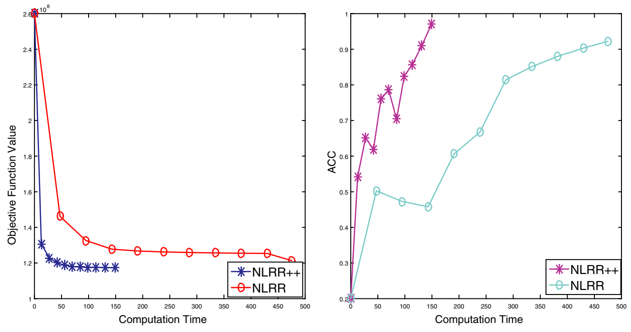

Fig. 3 Comparison of NLRR and NLRR++ on synthetic data (with $n_k = 2800$, Rank/Dimension $= 16\%$, $\rho = 0.02$). The x-axis is training time in seconds. We compare the objective function value Eq. (3) in the left figure and clustering accuracy in the right figure.

from the LibSVM website.[3] For MNIST and SVHN, we randomly select 22,700 samples and 9001 samples, to form MNIST-20K and SVHN-9K. We also evaluate our method on a real-world dataset, ImageNet.[4] ImageNet is an image database organized according to the WordNet hierarchy (currently only the nouns), in which each class contains hundreds or thousands of images. In this experiment, we use the features trained by convolutional neural network (Krizhevsky et al. 2012), to construct the sample set ImageNet-30K, ImageNet-60K and ImageNet-120K with 30K, 60K and 120K respectively.

In order to evaluate the clustering efficiency of the proposed method, we utilize two evaluation metrics accuracy (ACC) (Elhamifar and Vidal 2013) and F-measure (Larsen and Aone 1999). For NLRR and NLRR++, we fix the converge condition parameter $\varepsilon = 0.005$, $\beta = 0.8$ and the regularization parameter $\lambda = 40$ for all the datasets. We set the

[3] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/.

[4] http://www.image-net.org.

Table 4 Clustering F-measure with rank $d = \{50, 95, 90, 150, 250, 550\}$ for USPS, Protein, MNIST-20K, SVHN-9K, ImageNet-30K, ImageNet-60K

	NLRR	OLRSC	ORPCA	LRR	SSC	CASS	NLRR++
USPS	37.02	46.27	49.39	28.06	51.76	65.31	**66.37**
Protein	48.10	38.10	36.2	34.91	**49.27**	48.2	48.3
MNIST-20K	75.6	59.02	49.31	27.93	50.10	78.5	**78.80**
SVHN-9K	**39.7**	36.1	31.23	33.71	32.4	36.9	37.69
I-30K	87.71	93.71	34.66	6.20	–	92.1	**95.75**
I-60K	68.78	90.73	39.53	19.01	–	90.8	**93.70**

As for most of the datasets, NLRR++ has the best performance

Table 5 Clustering ACC (%) with rank $d = \{50, 95, 90, 150, 250, 550\}$ for USPS, Protein, MNIST-20K, SVHN-9K, ImageNet-30K, ImageNet-60K

	NLRR	OLRSC	ORPCA	LRR	SSC	CASS	NLRR++
USPS	37.19	48.72	53.35	27.38	45.78	66.14	**66.25**
Protein	43.71	38.12	36.30	35.82	**45.0**	42.9	43.2
MNIST-20K	76.21	60.0	49.6	28.1	30.7	78.6	**79.10**
SVHN-9K	38.7	**41.0**	34.12	36.45	31.2	38.04	38.38
I-30K	83.10	93.50	36.10	6.04	–	91.9	**95.69**
I-60K	64.9	88.43	33.10	19.20	–	90.02	**90.89**

As for most of the datasets, NLRR++ has the best performance

Table 6 Computational time (seconds) for each method by automatically parallelized with ten threads versus only one thread in MATLAB

	NLRR	OLRSC	ORPCA	LRR	SSC	CASS	NLRR++
USPS	56.1/286.0	14.4/15.5	15.7/16.2	77.5/140.35	71.2/310.38	76.5/130.55	**2.6/6**
Protein	84.1/505.6	62.0/76	74.3/88.0	260.1/595.2	6700/> 5 h	240.1/550.2	**16.25/48.3**
MNIST-20K	350.7/2113.6	292.1/516.7	233.1/481.6	481.6/2154.3	13,508/> 5 h	468.7/1955.4	**54.8/124.98**
SVHN-9K	**32.7/176.28**	168.9/468.4	174.4/440.3	914.0/4603.3	> 5 h/> 5 h	891.4/4308.5	78.6/248.57
I-30K	249.6/1615.4	5871.3/6174.4	6340.3/6749.6	2198.1/11,871	> 5 h/> 5 h	2108.4/11,673	**214.3/391.2**
I-60K	3306.41/32,349	37,930/48,826	49,129/55,199	3972.71/22,227	> 10 h/> 10 h	3729.71/22,170	**1306.5/2289.8**
I-120K	–	> 10 h/> 10 h	> 10 h/> 10 h	–	–	–	**1.2 h/2.2 h**

As for most of the datasets, NLRR++ has the best performance

rank to be $d = \{50, 95, 90, 150, 250, 550, 700\}$ for USPS, ImageNet-120K, MNIST-20K, SVHN-9K, ImageNet-30K, ImageNet-60K and Protein respectively. This rank is used for all the methods. Also note that following (Shen and Li 2016; Shen et al. 2016; Feng et al. 2013), we use the dataset itself as the dictionary $\mathbf{A}$. We use the proposed clustering pipeline mentioned in Sect. 3, which suffices to guarantee an appealing clustering result on the large-scale dataset, for all the methods except SSC. This is because the representation matrix obtained by SSC is too sparse to perform singular value decomposition, so the clustering strategy introduced in the previous section cannot be used directly. As a result, only the spectral clustering method can be applied.

We record the F-measure and the accuracy of NLRR, NLRR++, OLRSC, ORPCA, LRR and SSC in Tables 4 and 5. We also show the computational time via multiple threads and single thread in Table 6. Our algorithm NLRR++ significantly outperforms other state-of-the-art methods both in accuracy and efficiency on most of datasets. Although SSC is slightly better than OLRSC on *Protein* dataset, it takes almost 2 h, while NLRR++ only needs 16.25 s to obtain a good solution. As mentioned earlier, since SSC requires the spectral clustering for the final result, it is not applicable to large-scale clustering. Therefore, we do not report the accuracy of SSC on the dataset ImageNet-30K, ImageNet-60K because the affinity matrix is out of memory when using the spectral clustering algorithm.

Compared with NLRR, OLRSC and ORPCA, NLRR++ always achieves higher clustering accuracy and takes less running time. For example, on the USPS dataset, NLRR++ achieves the accuracy of 66.37%, while the second best algorithm can only achieve 51.76% accuracy. In terms of the computational time, NLRR++ uses only 6 s, which is much faster than all the other methods. For the ImageNet, MNIST and USPS datasets, NLRR++ achieves the best performance both in efficiency and accuracy. However, it is no better than NLRR in the SVHN dataset due to relatively smaller amount of data. Clearly, NLRR++ outperforms other methods in datasets with large amount of samples.

Finally, for the ImageNet-120K, NLRR, SSC and LRR run out of memory in their first iteration. The online methods OLRSC and ORPCA take more than 10 h to compute while NLRR++ only needs about 1.2 h. Apparently, NLRR++ is the only method that can scale to 120,000 samples.

5 Conclusions

In this paper, we propose a novel algorithm NLRR++ for scalable subspace clustering. It dramatically reduces the memory cost of NLRR from $\mathcal{O}(n^2)$ to $\mathcal{O}(mn)$ and the time complexity per iteration from $\mathcal{O}(n^3)$ to $\mathcal{O}(dmn)$. The two key techniques used in NLRR++ are the rank-one reformulation of the non-convex subspace recovery problem and the column-wise block coordinate descent that enables faster variable updates. we have also analyzed the time complexity and empirically demonstrated that our algorithm is computationally much more efficient compared with competing baselines. Our extensive experimental study on synthetic and realistic datasets also illustrates the robustness of NLRR++. The results show that NLRR++ is the only method that can solve an ImageNet problem with $120K$ samples in about 2 h with one single thread, while other methods either run out of memory or demand more than 10 h training time.

Acknowledgements This research was supported by the National Natural Science Foundation of China (Grant No. 51677042, 61402133).

Compliance with ethical standards

Conflict of interest All author declares that he/she has no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

Amin F, Fahmi A, Abdullah S (2019) Dealer using a new trapezoidal cubic hesitant fuzzy topsis method and application to group decision-making program. Soft Comput 23:5353–5366

Bertsekas DP (1999) Nonlinear programming. Athena scientific, Belmont

Bian W, Ding S, Yu X (2017) An improved fingerprint orientation field extraction method based on quality grading scheme. Int J Mach Learn Cybern 9(8):1–12

Burer S, Monteiro RDC (2005) Local minima and convergence in low-rank semidefinite programming. Math Program 103(3):427–444

Cheng D, Nguyen MN, Gao J, Shi D (2013) On the construction of the relevance vector machine based on bayesian ying-yang harmony learning. Neural Netw 48(6):173–179

Ding S, Xu X, Fan S (2018) Locally adaptive multiple kernel k-means algorithm based on shared nearest neighbors. Soft Comput 22:4573–4583

Du M, Ding S, Jia H (2016) Study on density peaks clustering based on k-nearest neighbors and principal component analysis. Knowl Based Syst 99:135–145

Du M, Ding S, Yu X (2017) A novel density peaks clustering algorithm for mixed data. Pattern Recognit Lett 97:46–53

Du M, Ding S, Yu X, Shi Z (2018) A novel density peaks clustering with sensitivity of local density and density-adaptive metric. Knowl Inf Syst 1:1–25

Elhamifar E, Vidal R (2013) Sparse subspace clustering: algorithm, theory, and applications. IEEE Trans Pattern Anal Mach Intell 35(11):2765–2781

Fahmi A, Abdullah S, Amin F, Khan MSA (2019) Trapezoidal cubic fuzzy number Einstein hybrid weighted averaging operators and its application to decision making. Soft Comput 23:5753–5783

Fan S, Ding S, Yu X (2016) Self-adaptive kernel k-means algorithm based on the shuffled frog leaping algorithm. Soft Comput 22(3):1–12

Fazel M, Hindi H, Boyd SP (2001) A rank minimization heuristic with application to minimum order system approximation. In: American control conference, vol 6. IEEE, pp 4734–4739

Feng J, Xu H, Yan S (2013) Online robust PCA via stochastic optimization. In: Advances in neural information processing systems, vol 26, pp 404–412

Hale ET, Yin W, Zhang Y (2008) Fixed-point continuation for l1 minimization: methodology and convergence. SIAM J Optim 19(3):1107–1130

Jia H, Ding S, Du M (2017) A nystrom spectral clustering algorithm based on probability incremental sampling. Soft Comput 21:5815–5827

Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems, vol 25, no 2, pp 1097–1105

Larsen B, Aone C (1999) Fast and effective text mining using linear-time document clustering. In: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp 16-22

Liu G, Lin Z, Yan S, Sun J, Yu Y, Ma Y (2013) Robust recovery of subspace structures by low-rank representation. IEEE Trans Pattern Anal Mach Intell 35(1):171–184

Lu C, Feng J, Lin Z, Yan S (2014) Correlation adaptive subspace segmentation by trace lasso. In: IEEE international conference on computer vision

Ng AY, Jordan MI, Weiss Y et al (2001) On spectral clustering: analysis and an algorithm. NIPS 14(2):849–856

Recht B, Fazel M, Parrilo PA (2010) Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. SIAM Rev 52(3):471–501

Richtrik P, Tak M (2014) Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. Math Program 144:1–38

Shen J, Li P (2016) Learning structured low-rank representation via matrix factorization. In: Proceedings of the 19th international conference on artificial intelligence and statistics (AISTATS), pp 500–509

Shen J, Li P, Xu H (2016) Online low-rank subspace clustering by basis dictionary pursuit. In: Proceedings of the 33rd international conference on machine learning (ICML), pp 622–631

Tang X, Wei G (2019) Multiple attribute decision-making with dual hesitant pythagorean fuzzy information. Cogn Comput 11(2):193–211

Tang X, Wei G, Gao H (2019) Models for multiple attribute decision making with interval-valued pythagorean fuzzy muirhead mean operators and their application to green suppliers selection. Informatica 30(1):153–186

Von Luxburg U (2007) A tutorial on spectral clustering. Stat Comput 17(4):395–416

Wall ME, Rechtsteiner A, Rocha LM (2003) Singular value decomposition and principal component analysis. In: Berrar DP, Dubitzky W, Granzow M (eds) A practical approach to microarray data analysis. Springer, Boston, MA, pp 91–109

Wang J, Shi D, Cheng D, Zhang Y, Gao J (2016) LRSR: low-rank-sparse representation for subspace clustering. Neurocomputing 214:S0925231216307573

Wang L, Peng JJ, Wang JQ (2018) A multi-criteria decision-making framework for risk ranking of energy performance contracting project under picture fuzzy environment. J Clean Prod 191:105–118

Wang R, Wang J, Gao H, Wei G (2019) Methods for madm with picture fuzzy muirhead mean operators and their application for evaluating the financial investment risk. Symmetry 11(6):1–21

Yu H-F, Hsieh C-J, Si S, Dhillon I (2012) Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In: IEEE 12th international conference on data mining (ICDM). IEEE, pp 765–774

Zhang S, Gao H, Wei G, Wei Y, Wei C (2019) Evaluation based on distance from average solution method for multiple criteria group decision making under picture 2-tuple linguistic environment. Mathematics 7(3):1–14

Zhou X, Yang C, Yu W (2013) Moving object detection by detecting contiguous outliers in the low-rank representation. IEEE Trans Pattern Anal Mach Intell 35(3):597–610