Appl Intell
DOI 10.1007/s10489-012-0418-0

Deflation-based power iteration clustering

Anh Pham The · Nguyen Duc Thang · La The Vinh · Young-Koo Lee · Sungyoung Lee

Abstract Spectral clustering (SC) is currently one of the most popular clustering techniques because of its advantages over conventional approaches such as K-means and hierarchical clustering. However, SC requires the use of computing eigenvectors, making it time consuming. To overcome this limitation, Lin and Cohen proposed the power iteration clustering (PIC) technique (Lin and Cohen in Proceedings of the 27th International Conference on Machine Learning, pp. 655–662, 2010), which is a simple and fast version of SC. Instead of finding the eigenvectors, PIC finds only one pseudo-eigenvector, which is a linear combination of the eigenvectors in linear time. However, in certain critical situations, using only one pseudo-eigenvector is not enough for clustering because of the inter-class collision problem. In this paper, we propose a novel method based on the deflation technique to compute multiple orthogonal pseudo-eigenvectors (orthogonality is used to avoid redundancy). Our method is more accurate than PIC but has the same computational complexity. Experiments on synthetic and real datasets demonstrate the improvement of our approach.

Keywords Spectral clustering · Deflation · Power iteration

1 Introduction

Clustering is an important task in data mining. It has many applications in various fields such as image processing [27, 30], social network analysis [21], biomedical data processing [12], and signal processing [4, 13]. The function of clustering is to identify the hidden structure inside the data so that better representation, stronger and more flexible security, and smaller compression can be achieved. Clustering is the task of dividing the data points into clusters so that similar data points are grouped in the same cluster, and dissimilar data points are grouped in different clusters.

Generally, conventional clustering algorithms can be divided into two categories: hierarchical clustering such as single linkage and CURE [34], and partitional clustering such as K-means. Hierarchical clustering algorithms produce a hierarchy of nested clusters, whereas partitional clustering algorithms directly output a one-level clustering solution. However, K-means, the best-known method in partitional clustering algorithms, has limited accuracy when applied to nonlinear data, as shown in Fig. 1. Furthermore, it can be trapped easily in a local optimal solution because of random initialization [24].

To overcome the limitation of K-means, spectral clustering (SC), which is the focus of this paper, has been proposed and developed. SC is the relaxation of the NP-hard normalized cut problem in graph theory. Whereas K-means works directly on data points, SC starts from an affinity matrix that shows the pairwise similarity of the data points. It then

A.P. The · L.T. Vinh · Y.-K. Lee (✉) · S. Lee
Department of Computer Engineering, Kyung Hee University, Seocheon-dong, Giheung-gu, Yongin-si, Gyeonggi-do, 446-701, South Korea
e-mail: yklee@khu.ac.kr

A.P. The
e-mail: phamtheanh@oslab.khu.ac.kr

L.T. Vinh
e-mail: vinhlt@oslab.khu.ac.kr

S. Lee
e-mail: sylee@oslab.khu.ac.kr

N.D. Thang
Department of Biomedical Engineering, International University, Ho Chi Minh City, Vietnam
e-mail: ndthang@hcmiu.edu.vn

Published online: 03 February 2013

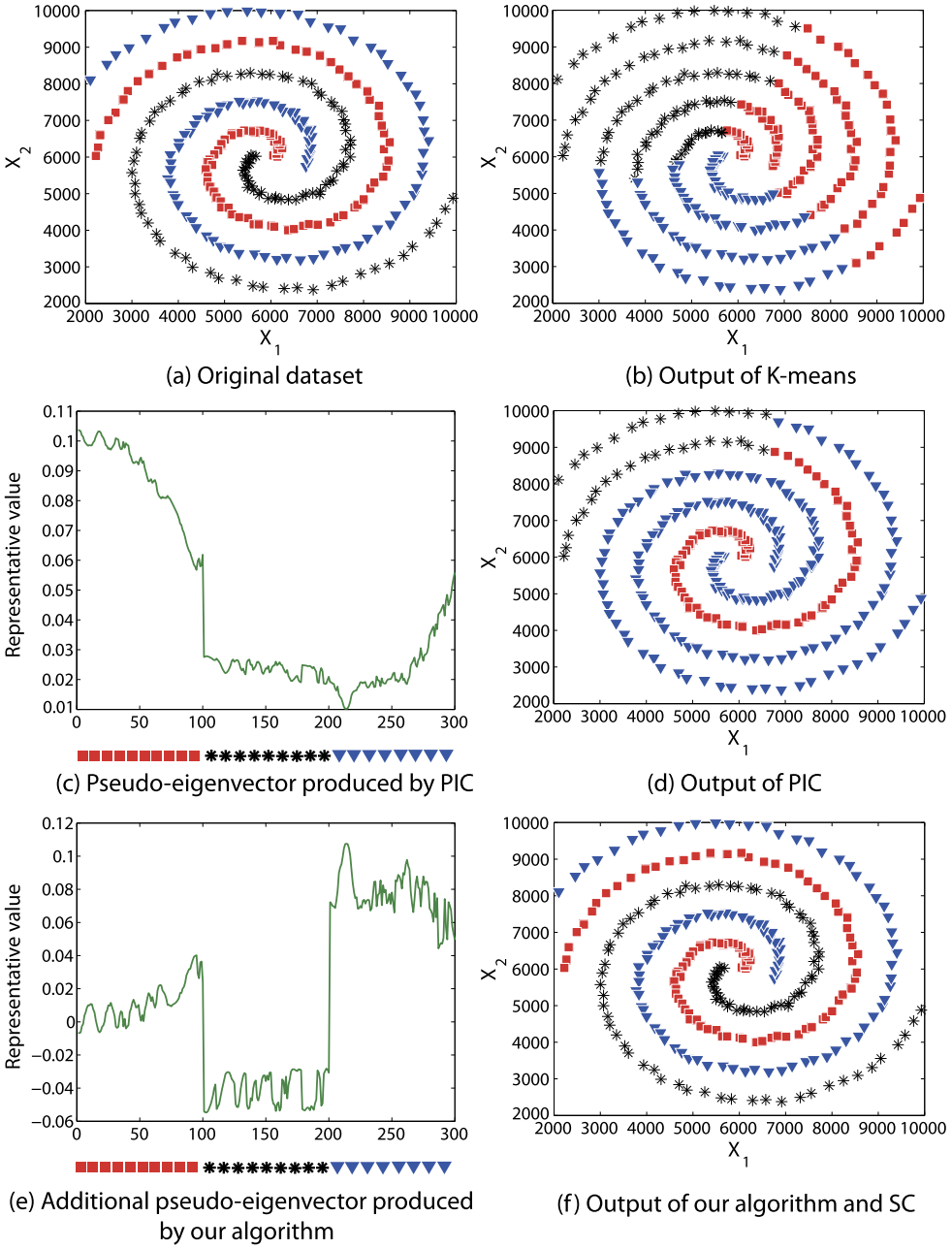


Fig. 1 The K-means' output on nonlinear data, and the PIC' limitation which is solved by our proposed additional pseudo-eigenvector. (**a**) A 3-spiral dataset; (**b**) the output of K-means; (**c**) the pseudo-eigenvector produced by PIC. The *horizontal axis* represents the indexes of all data points. The *vertical axis* shows the representative values of the data points in the pseudo-eigenvector. The data points are sorted in a sequence in which close data points are in the same cluster. The representative values of *red square cluster* and *black star cluster* are merged because of the *blue triangle cluster*; (**d**) the output of PIC using the PIC' pseudo-eigenvector; (**e**) the additional pseudo-eigenvector produced by our method is orthogonal to the PIC' pseudo-eigenvector, hence the square cluster and the star cluster are totally separated; Using (**c**) and (**e**), the exact clustering solution (**f**) is found (Color figure online)

computes the Laplacian matrix from the affinity matrix and finds the eigenvectors from this matrix. The effort required to compute the eigenvectors is relatively high, $O(n^3)$, where n is the number of data points. Given the high computational cost, several researchers have focused on developing fast SC techniques (using the sampling method [3, 28, 32, 33, 36], the Nyström method [5, 8, 38, 39], or the power iteration method [16, 20]).

Among all fast SC techniques, power iteration clustering (PIC) [16] is not only simple (it only requires a matrix-vector multiplication process), but is also scalable in terms of time complexity, $O(n)$ [17]. Using the assumption [22, 37] about the eigenvalues of the Laplacian matrix, instead of finding all the k biggest eigenvectors, PIC can simply and quickly compute their linear combination to construct a pseudo-eigenvector. Eigenvectors are mapping vectors that transform all data points from a linearly inseparable domain into a linearly separable domain in which data points in different clusters are most weakly connected. A pseudo-eigenvector is not a member of the eigenvectors but is created linearly from them. Therefore, in the pseudo-eigenvector, if two data points lie in different clusters, their values can still be separated. Different from the sampling method and the Nyström method, PIC does not modify the original data distribution; thus, no information is lost. Nev-

ertheless, PIC has a limitation when dealing with multi-class datasets, as shown in Fig. 1.

In this paper, we propose a novel algorithm that has better accuracy than PIC in multi-class datasets and is nearly equal in accuracy to that of the original implementation of SC. More importantly, the computational time of our algorithm is significantly smaller than that of the fast implementation of SC (i.e., SCIRAM [14]). Our contributions are twofold. First, by using the deflation method, instead of finding only one pseudo-eigenvector similar to PIC, we find additional pseudo-eigenvectors with interesting properties. Similar to the PIC' pseudo-eigenvector, each contains useful information for clustering. Pseudo-eigenvectors can be used together, and they are non-redundant because they are mutually orthogonal. Moreover, the computational time for finding each pseudo-eigenvector is as small as that for finding the PIC' pseudo-eigenvector. We also provide theoretical proofs for these properties. Second, we justify our method by conducting experiments on various handwritten digit, human face, document, and synthetic datasets. Specifically, in clustering the *sci* topic of a 20-newsgroup dataset, we improve the accuracy of PIC from 58 % to 69 % while keeping the computational time 16 times smaller than that of SCIRAM.

The rest of the paper is organized as follows. Section 2 discusses the related works. Section 3 presents the background of SC and PIC along with their limitations. Section 4 outlines our algorithm, and Sect. 5 examines its performance. Section 6 concludes the paper and summarizes future research directions.

2 Related works

Sampling-based SC A large number of studies have been conducted to reduce the computational time of SC. Most of these studies used the sampling method to determine some representative data points over the dataset [28, 32, 33, 36]. In particular, finding eigenvectors from n data points takes $O(n^3)$. However, if we use sampling, the number of representative points is $k \ll n$, and this time complexity is only $O(k^3)$. One of the sampling-based SC techniques is presented in [36] in which Yan *et al.* also provide an end-to-end error analysis for the method. In the first step, they use K-means to find k centroids as k representative points. In the second step, SC clusters these centroids together. In the post-processing step, each data point is classified in the same cluster with its centroid. The drawback of sampling-based methods is that the sampling step may not properly detect some representative data points for nonlinear datasets.

Nyström-based SC Another branch of fast SC research is the Nyström method [5, 8, 38, 39] that uses the rank reduction technique. This framework involves three steps. First, a sampling technique is used to select some data points. Second, by using the affinity matrix between these data points and all data points, the eigenvectors are found through the low-rank approximation technique. The last step is extrapolating these eigenvectors to a full set of data points. Typically, the works are different in the first step in which the issue is what sampling technique to use. The Nyström-based methods have two drawbacks: the memory requirement is high because a large number of high-dimension matrices have to be stored, and the sampling method in the first step may ignore all data points in a small cluster; thus, this cluster cannot be detected.

Power iteration-based SC Our approach lies in the third branch of fast SC algorithms. This direction is taken from the assumption in data distribution [22, 37] that, if the dataset has k clusters, the $(k+1)$th eigenvalue of the Laplacian matrix will be considerably larger than the kth. The wider the gap is, the faster the algorithm converges. Mavroeidis *et al.* [20] use this assumption to build a semi-supervised power iteration-based clustering algorithm. Using supervised labels for several data points from the original Laplacian matrix, they created a new Laplacian matrix with a wider gap between the two eigenvalues. Although the convergence time of this algorithm is significantly reduced, the drawback is the high cost of obtaining supervised labels.

Dominant set clustering Similar to SC, dominant set clustering [23] is a well-known clustering technique that utilizes the affinity matrix. Specifically, from the affinity matrix, without a normalization step like SC, dominant set clustering iteratively updates a vector that can be used to sequentially filter out the most compact clusters. Generally, similar to the rare class analysis [10], the dominant set method is effective in finding a strongly connected cluster even when it overlaps with other clusters. PIC, without the normalization step, can also be used to sequentially detect rare classes. However, we do not focus on this modified version of PIC in this paper. The limitation of the dominant set method is that only excessively compact data points can be obtained inside the compact clusters [23].

3 Background

3.1 Spectral clustering

Assuming that we have a set of vectors $X = \{\mathbf{x}_i\}_{i=1}^n$ and each $\mathbf{x}_i$ is in R^d space, which represents a data point in a dataset, we define a similarity function $s(\mathbf{x}_i, \mathbf{x}_j)$ to show the similarity between $\mathbf{x}_i$ and $\mathbf{x}_j$. A matrix $A = \{a_{ij}\}_{i,j=1}^n$, and each $a_{ij} = s(\mathbf{x}_i, \mathbf{x}_j)$, is called an affinity matrix. We define an affinity graph G as an undirected graph in which a vertex

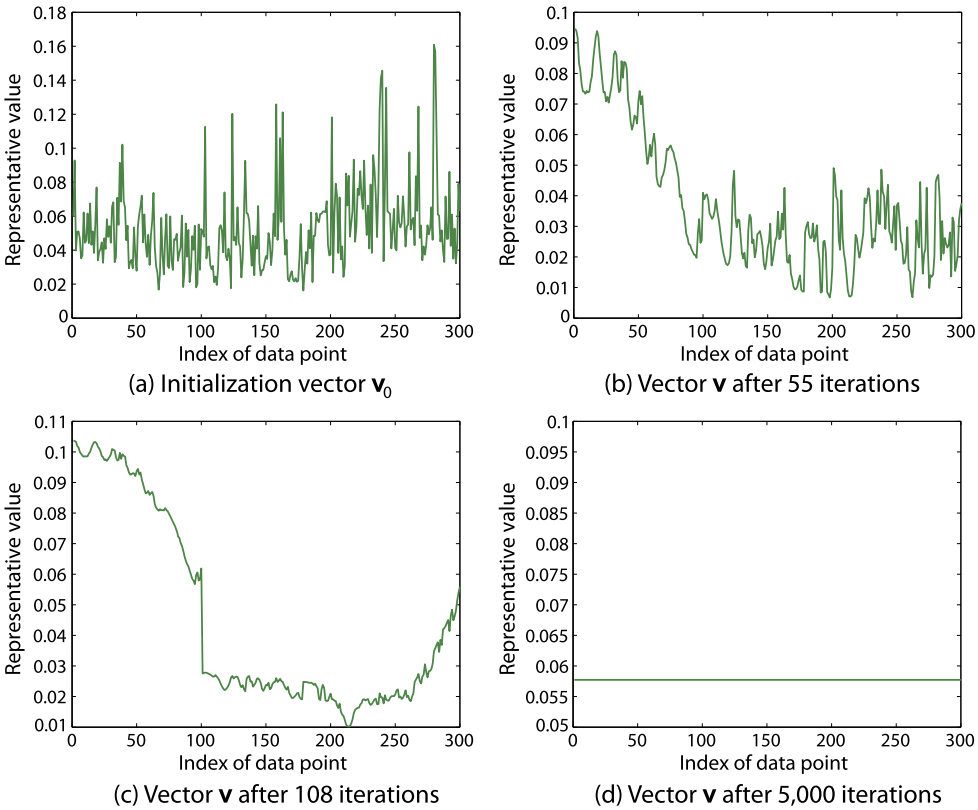


Fig. 2 PIC result for a dataset of 3 classes in Fig. 1. (**a**) is the initialization vector. (**b**) is the iterative vector after 55 iterations. (**c**) is the iterative vector after 108 iterations. (**d**) is the constant vector after 5,000 iterations. Around the 108th iteration is the local convergence phase. Around the 5,000th iteration is the global convergence phase. The transition from the initialization to the local convergence phase is rapid (107 iterations), but the transition from the local convergence phase to the global convergence phase is very slow (4,892 iterations)

v_i representing the data point $\mathbf{x}_i$. The weight of the edge between v_i and v_j represents the similarity between $\mathbf{x}_i$ and $\mathbf{x}_j$. The affinity matrix and the affinity graph can be sparse if we consider only the similarity between a data point $\mathbf{x}_i$ and its t nearest neighbors while setting the similarity between $\mathbf{x}_i$ and the others to zero. We define the diagonal matrix D, with $D_{ii} = \Sigma_j A_{ij}$, and the normalized affinity matrix W, with $W = D^{-1}A$. The Laplacian matrix is computed based on $L = I - W$, where I is the unit matrix.

The goal of SC is to divide the data into groups in which the data points in each group have a similar property. SC has several versions [22], but we focus only on the normalized cut version in which SC is a relaxation of the normalized cut problem in graph G. We find the k smallest eigenvectors of L, which are also the k largest eigenvectors of W. Hereafter, when we say k eigenvectors, we refer to the k largest eigenvectors of W. Based on these eigenvectors, we use K-means to obtain the final SC result.

3.2 Power iteration clustering

Instead of computing k individual eigenvectors, PIC finds only one pseudo eigenvector, which is a linear combination of the eigenvectors. Power iteration, a method to compute the largest eigenvector of W, is the main technique in PIC. This section describes the process of finding the largest eigenvector of W and uses this process to find the pseudo-eigenvector.

Initially, an iterative vector is set as equal to the random initialization vector $\mathbf{v}_0$. In each iteration, the iterative vector is updated based on the multiplication of W with itself. To prevent the vector from becoming too large in each iteration, we need a normalization step. Specifically,

$$\mathbf{v}^t = W * \mathbf{v}^{t-1}, \tag{1}$$

$$\mathbf{v}^t = \mathbf{v}^t / \|\mathbf{v}^t\|, \tag{2}$$

where $\mathbf{v}^t$ is the iterative vector. If there is no change between two continuous iterations, the iterative vector converges to the largest eigenvector of W. However, because W is a normalized matrix, its largest eigenvector is a constant vector that is useless for clustering [16]. Fortunately, Lin *et al.* [16] found that the aforementioned iteration process to become the largest eigenvector of W has two phases, as illustrated in Fig. 2.

- In the first phase, if two data points are in different clusters, their values in the iterative vector will be far apart. Therefore, this iterative vector is useful for clustering. This phase is called the local convergence phase, which is reached at an extremely fast pace from the beginning.
- In the second phase, the iterative vector slowly becomes the largest eigenvector, which is a constant vector. At this time, the iterative vector is useless. This phase is called the global convergence phase, which is reached at an extremely slow pace from the local convergence phase.

Algorithm 1 Power iteration

Input: Normalized Affinity Matrix W.
repeat
$\mathbf{v}^{t+1} = (W\mathbf{v}^t)/\|W\mathbf{v}^t\|_1$.
$\sigma^{t+1} = |\mathbf{v}^{t+1} - \mathbf{v}^t|$.
acceleration $= \|\sigma^{t+1} - \sigma^t\|$.
Increase t.
until acceleration $< \epsilon$
Output: Pseudo-eigenvector $\mathbf{v}^t$.

To determine when the local convergence phase should stop, [16] computed *acceleration*. When the *acceleration* is smaller than the predefined threshold, the iteration process is stopped. The most interesting property of the iterative vector, if we stop the iteration process at the end of the local phase, is its being a linear combination of the k largest eigenvectors of W [16]. Therefore, we call this vector the pseudo-eigenvector. The details of the power iteration of PIC are presented in Algorithm 1.

However, if we use PIC in the multi-class dataset, we will have an inter-class collision problem. As reported in Figs. 1 and 2, if we use the iterative vector $\mathbf{v}$ at the 108th iteration, we will not find which data points are in the red square cluster and which data points are in the black star cluster. In the following section, we show our solution to overcome this problem. Although there are emerging trends about parallel and distributed computing, we do not particularly design our algorithm for the parallel or distributed environment. However, such special deployment is an interesting topic for future research.

4 The proposed deflation-based power iteration clustering algorithm

Given the inter-class collision problem, we cannot use only one pseudo-eigenvector for clustering. We need more pseudo-eigenvectors, which are not only useful for clustering (i.e., they can be a linear combination of the k largest eigenvectors) but also do not contain redundant information (i.e., they can be mutually orthogonal). Moreover, as PIC is fast and requires simple computation and low memory, our algorithm must retain these advantages of PIC. To achieve this goal, we use the deflation technique and PIC together. We call our algorithm Deflation-based Power Iteration Clustering (DPIC). In the following subsections, we present the deflation technique and then describe the work flow and properties of our algorithm.

4.1 Deflation technique

Deflation is a technique that modifies a matrix by eliminating the effect of a given eigenvector. After deflation, all eigenvectors of the modified matrix are the same as those of the old matrix with similar eigenvalues. However, the given eigenvector, which is used for deflation, will have a new zero eigenvalue. Deflation is commonly used in sparse PCA [19] when determining the eigenvectors of a covariance matrix. Sequentially, we find the first eigenvector of this covariance matrix and then eliminate its effect on the matrix through deflation. Therefore, in the next step, the second eigenvector that we obtain contains non-redundant information on the first eigenvector.

A number of deflation techniques are used in [19] such as Hotelling deflation, projection deflation, and Schur complement deflation. Schur complement deflation is used in this paper because it is the only method that can ensure that the outputs of our algorithm are mutually orthogonal pseudo-eigenvectors.

Assuming that we have a matrix A_{t-1} and its eigenvector $\mathbf{v}_t$, the Schur complement deflation creates a new matrix A_t, which is computed by the following formula:

$$A_t = A_{t-1} - \frac{A_{t-1}\mathbf{v}_t\mathbf{v}_t^T A_{t-1}}{\mathbf{v}_t^T A_{t-1}\mathbf{v}_t}. \tag{3}$$

The eigenvalue of $\mathbf{v}_t$ on matrix A_t is zero because $A_t\mathbf{v}_t = 0$, indicating the elimination of the effect of $\mathbf{v}_t$ on A_{t-1} [19].

4.2 Deflation-based PIC

In this paper, we use Schur deflation to improve PIC accuracy while ensuring that the time and memory complexity does not change significantly. We propose a sequential method to find k mutually orthogonal pseudo-eigenvectors, where k is the number of clusters. First, from the original affinity matrix W_0, we still use power iteration as PIC to find the first pseudo-eigenvector $\mathbf{v}_1$. Then, we apply deflation to eliminate the effect of $\mathbf{v}_1$ on W_0 to obtain the second affinity matrix W_1. From W_1, we apply power iteration again to obtain the second pseudo-eigenvector $\mathbf{v}_2$. Then, we apply deflation again to eliminate the effect of $\mathbf{v}_2$ on W_1. This process is repeated k times. The flow of our algorithm is presented in Algorithm 2. Our algorithm has the following properties:

1. Each pseudo-eigenvector produced by our algorithm $(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k)$ is a linear combination of the k largest eigenvectors of the original affinity matrix W_0, indicating that each pseudo-eigenvector is useful in clustering the data. Proposition 2 proves this property.
2. These pseudo-eigenvectors are mutually orthogonal, which means that they do not contain redundant information about each other. Given the time it takes to compute a new pseudo-eigenvector, we do not compute new ones that contain redundant information about the old pseudo-eigenvectors (e.g., the addition of old pseudo-eigenvectors). The same idea is applied in PCA in which

Algorithm 2 Deflation-based Power Iteration Clustering (DPIC)

Input: Normalized Affinity Matrix W.
$W_0 = W$.
repeat
$\mathbf{v}_l$ = PowerIteration(W_{l-1}). *//Power iteration: find* $\mathbf{v}_l$ *from* W_{l-1}
$W_l = W_{l-1} - \frac{W_{l-1}\mathbf{v}_l\mathbf{v}_l^T W_{l-1}}{\mathbf{v}_l^T W_{l-1}\mathbf{v}_l}$. *//Deflation: remove the effect of* $\mathbf{v}_l$ *on* W_{l-1}
Increase l.
until $l > k$
Use K-Means on pseudo-eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$.
Output: Clusters $C_1, C_2, \ldots, C_k$.

only orthogonal principle components are found to avoid redundancy between the principle components. Proposition 3 proves this property.
3. The speed of computing every pseudo-eigenvector $(\mathbf{v}_2, \mathbf{v}_3, \ldots, \mathbf{v}_k)$ is the same as that of computing the first pseudo-eigenvector $\mathbf{v}_1$. Section 4.4 justifies this property.

Note that the original normalized affinity matrix W has eigenvectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ with the corresponding eigenvalues of $\lambda_1, \lambda_2, \ldots, \lambda_n$. These eigenvalues are sorted in descending order. References [22, 37] show that, if we have k clusters, the first k eigenvalues will almost be equal, and the gap between the kth and $(k+1)$th eigenvalue will be extremely large. Thus, to follow the proof more easily, we assume that the first k eigenvalues are equal to λ_b. All the remaining eigenvalues are equal to the $(k+1)$th eigenvalue, λ_s, which is considerably smaller than λ_b. (In reality, the remaining eigenvalues are significantly smaller than λ_s, justifying the following proof.)

Moreover, note also that different datasets have different eigenvectors and eigenvalues. Nevertheless, we have to prove that the abovementioned properties are true regardless of the datasets. Thus, we consider $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n, \lambda_b, \lambda_s$ as variables.

Let F be a group of vectors. Each vector is a linear combination of $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k)$. Specifically, $F = \{c_1\mathbf{x}_1 + c_2\mathbf{x}_2 + \cdots + c_k\mathbf{x}_k \mid \forall c_1, c_2, \ldots, c_k \in R\}$.

Let G be a group of vectors. Each vector is a multiplication of λ_b with a linear combination of $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k)$. Specifically, $G = \{\lambda_b(c_1\mathbf{x}_1 + c_2\mathbf{x}_2 + \cdots + c_k\mathbf{x}_k) \mid \forall c_1, c_2, \ldots, c_k \in R\}$.

Proposition 1 *The following conditions are true for each loop* l*th of DPIC*:

1. $W_{l-1}\mathbf{x}_i \in G$, *for* $i \le k$,
2. $W_{l-1}\mathbf{x}_i = \lambda_s\mathbf{x}_i$, *for* $i > k$,
3. $\mathbf{v}_l \in F$.

Proof We will prove this proposition by induction on l.

These conditions are true for the first loop: $W_0\mathbf{x}_i = \lambda_b\mathbf{x}_i \in G$ for $i \le k$; $W_0\mathbf{x}_i = \lambda_s\mathbf{x}_i$ for $i > k$; $\mathbf{v}_1 \in F$ [16].

Assuming that these three conditions are true until the lth loop, we will prove that they are true for the $(l+1)$th loop.

1. For $i \le k$

We multiply both sides of the Schur equation with $\mathbf{x}_i$

$$W_l\mathbf{x}_i = W_{l-1}\mathbf{x}_i - W_{l-1}\mathbf{v}_l\left(\frac{\mathbf{v}_l^T W_{l-1}\mathbf{x}_i}{\mathbf{v}_l^T W_{l-1}\mathbf{v}_l}\right). \tag{4}$$

According to the lth loop, $\mathbf{v}_l \in F$. As a result,

$$\mathbf{v}_l = c_1\mathbf{x}_1 + c_2\mathbf{x}_2 + \cdots + c_k\mathbf{x}_k. \tag{5}$$

We multiply both sides with W_{l-1}

$$W_{l-1}\mathbf{v}_l = c_1 W_{l-1}\mathbf{x}_1 + c_2 W_{l-1}\mathbf{x}_2 + \cdots + c_k W_{l-1}\mathbf{x}_k. \tag{6}$$

Each factor on the right side of (6) is in G so the left side is in G. Therefore,

$$W_{l-1}\mathbf{v}_l = \lambda_b(e_1\mathbf{x}_1 + e_2\mathbf{x}_2 + \cdots + e_k\mathbf{x}_k), \tag{7}$$

where $e_h \in R, \forall h \le k$.

Note that $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$ are in norm 2 and are mutually orthogonal [22], and that W_{l-1} is symmetric. Therefore, from (4), (5), and (7),

$$W_l\mathbf{x}_i = W_{l-1}\mathbf{x}_i - W_{l-1}\mathbf{v}_l\frac{e_i}{c_1e_1 + c_2e_2 + \cdots + c_ke_k}. \tag{8}$$

Both factors on the right side of (8) are in G. In sum,

$$W_l\mathbf{x}_i \in G, \quad \text{for } i \le k. \tag{9}$$

2. For $i > k$

Given that $W_{l-1}\mathbf{x}_i = \lambda_s\mathbf{x}_i$ for $i > k$, from (4),

$$W_l\mathbf{x}_i = \lambda_s\mathbf{x}_i - W_{l-1}\mathbf{v}_l\left(\frac{\mathbf{v}_l^T\lambda_s\mathbf{x}_i}{\mathbf{v}_l^T W_{l-1}\mathbf{v}_l}\right). \tag{10}$$

Note that $\mathbf{x}_i$ and $\mathbf{x}_j$ are orthogonal for $j \le k$. Therefore, from (5) and (10),

$$W_l\mathbf{x}_i = \lambda_s\mathbf{x}_i - W_{l-1}\mathbf{v}_l\left(\frac{0}{\mathbf{v}_l^T W_{l-1}\mathbf{v}_l}\right) = \lambda_s\mathbf{x}_i. \tag{11}$$

From (11), we have

$$W_l\mathbf{x}_i = \lambda_s\mathbf{x}_i, \quad \text{for } i > k. \tag{12}$$

3. Computing $\mathbf{v}_{l+1}$

According to the $(l+1)$th loop of Algorithms 1 and 2, to compute the $(l+1)$th pseudo-eigenvector, we start with a random initialization vector:

$$\mathbf{v}_0 = d_1\mathbf{x}_1 + d_2\mathbf{x}_2 + \cdots + d_k\mathbf{x}_k + d_{k+1}\mathbf{x}_{k+1} + \cdots + d_n\mathbf{x}_n,$$

where $d_h \in R,\ \forall h \leq n$.

After running the power iteration on $\mathbf{v}_0$, where t is a large number of iterations

$$\mathbf{v}_{l+1} = (W_l)^t(d_1\mathbf{x}_1 + d_2\mathbf{x}_2 + \cdots + d_k\mathbf{x}_k) + (W_l)^t(d_{k+1}\mathbf{x}_{k+1} + \cdots + d_n\mathbf{x}_n). \quad (13)$$

Considering (9), (12), we can derive the following from (13):

$$\mathbf{v}_{l+1} = \lambda_b^t \times \mathbf{f} + \lambda_s^t \times (d_{k+1}\mathbf{x}_{k+1} + \cdots + d_n\mathbf{x}_n), \quad \text{with } \mathbf{f} \in F. \quad (14)$$

Note that $\lambda_s^t \ll \lambda_b^t$, with t being large. The coefficient λ_b^t will disappear after the normalizing step in Algorithm 1. Therefore, $\mathbf{v}_{l+1} \approx \mathbf{f}$ and $\mathbf{v}_{l+1} \in F$. □

Proposition 2 *Each pseudo-eigenvector produced by our algorithm is a linear combination of the k largest eigenvectors of the original affinity matrix.*

Proof The proof is straightforward based on the third condition of Proposition 1. □

Proposition 3 *The pseudo-eigenvectors produced by our algorithm are mutually orthogonal.*

Proof From the Schur equation on the lth loop of Algorithm 2, we obtain

$$W_l = W_{l-1} - \frac{W_{l-1}\mathbf{v}_l\mathbf{v}_l^T W_{l-1}}{\mathbf{v}_l^T W_{l-1}\mathbf{v}_l}.$$

We multiply both sides with $\mathbf{v}_l$

$$W_l\mathbf{v}_l = W_{l-1}\mathbf{v}_l - \frac{W_{l-1}\mathbf{v}_l\mathbf{v}_l^T W_{l-1}\mathbf{v}_l}{\mathbf{v}_l^T W_{l-1}\mathbf{v}_l} = 0. \quad (15)$$

From the Schur equation on the $(l+1)$th loop of Algorithm 2,

$$W_{l+1} = W_l - \frac{W_l\mathbf{v}_{l+1}\mathbf{v}_{l+1}^T W_l}{\mathbf{v}_{l+1}^T W_l\mathbf{v}_{l+1}}.$$

We multiply both sides with $\mathbf{v}_l$. From (15), $W_l\mathbf{v}_l = 0$

$$W_{l+1}\mathbf{v}_l = W_l\mathbf{v}_l - \frac{W_l\mathbf{v}_{l+1}\mathbf{v}_{l+1}^T(W_l\mathbf{v}_l)}{\mathbf{v}_{l+1}^T W_l\mathbf{v}_{l+1}} = 0. \quad (16)$$

We prove this in the same manner as that in (15), (16), with $\forall s \geq l+1$. Therefore, we have

$$W_{s-1}\mathbf{v}_l = 0. \quad (17)$$

From Algorithm 2, the pseudo-eigenvector of the sth loop is

$$\mathbf{v}_s = (W_{s-1})^t\mathbf{v}_0, \quad (18)$$

where $\mathbf{v}_0$ is the initialization vector and t is the number of iterations.

Given that the Schur deflation preserves the symmetry and using (17), we obtain

$$\mathbf{v}_l^T\mathbf{v}_s = \left(\mathbf{v}_l^T W_{s-1}^T\right)(W_{s-1})^{t-1}\mathbf{v}_0 = 0. \quad (19)$$

In sum, the DPIC' pseudo-eigenvectors are mutually orthogonal:

$$\mathbf{v}_l^T\mathbf{v}_s = 0.$$ □

4.3 Computational time for each DPIC' pseudo-eigenvector

In this section, we show that the speed of computing each DPIC' pseudo-eigenvector is the same as that of computing the first pseudo-eigenvector. From Algorithm 2, to find the $(l+1)$th pseudo-eigenvector $\mathbf{v}_{l+1}$, we perform the power iteration on affinity matrix W_l. Similarly, to find the first pseudo-eigenvector $\mathbf{v}_1$, we perform the power iteration on affinity matrix W_0. The structure of W_l is different from that of W_0, however, we will show that it takes the same time to run the power iteration on them.

In general, the main computing cost of the power iteration comes from a number of matrix-vector multiplication steps between the affinity matrix and the current iterative vector. Therefore, first, we show that the computational time for each step between W_l and the iterative vector is equal to that between W_0 and the iterative vector in Sect. 4.3.1. Second, we show that the number of steps for W_l is equal to that for W_0 in Sect. 4.3.2.

4.3.1 Computational time for each step of power iteration for each DPIC' pseudo-eigenvector

This section shows that the time to perform a matrix-vector multiplication step between W_l and an iterative vector is equal to that between W_0 and the iterative vector. In Algorithm 2, although W_0 is sparse, all the other matrices from the second loop are not. Consider the Schur complement formula again:

$$W_l = W_{l-1} - \frac{W_{l-1}\mathbf{v}_l\mathbf{v}_l^T W_{l-1}}{\mathbf{v}_l^T W_{l-1}\mathbf{v}_l}.$$

Indeed, even if W_0 is sparse, $W_1, W_2, \ldots, W_l$ are not. We need $O(n^2)$ for a matrix-vector multiplication step if the matrix is dense. However, if the matrix is sparse, this step will only take $O(tn)$ with t as the number of non-zero items in one row of the matrix on average. W_1 has two parts: the sparse matrix W_0 and the multiplication of two vectors:

$$\mathbf{p} = \frac{W_0\mathbf{v}_1}{\mathbf{v}_1^T W_0 \mathbf{v}_1}, \qquad \mathbf{q}^T = \mathbf{v}_1^T W_0.$$

Even if W_1 is not sparse, we can perform the matrix-vector multiplication on it quickly. Let us refer to the current iterative vector as $\mathbf{v}_c$. Specifically, instead of multiplying $W_1\mathbf{v}_c$ directly, we compute two factors $W_0\mathbf{v}_c$ and $\mathbf{p}\mathbf{q}^T\mathbf{v}_c$ then perform subtraction:

$$W_1\mathbf{v}_c = W_0\mathbf{v}_c - \frac{W_0\mathbf{v}_1\mathbf{v}_1^T W_0}{\mathbf{v}_1^T W_0\mathbf{v}_1}\mathbf{v}_c = W_0\mathbf{v}_c - \mathbf{p}\mathbf{q}^T\mathbf{v}_c.$$

The computational time of $\mathbf{p}\mathbf{q}^T\mathbf{v}_c$ is $O(n)$. The computational time of $W_0\mathbf{v}_c$ is $O(tn)$. In this manner, the computational time for the $W_1\mathbf{v}_c$ multiplication is only $O((t+1)n) \approx O(tn)$, which is almost equal to the computational time for the $W_0\mathbf{v}_c$ multiplication. Based on this analysis, even if the new affinity matrix is not sparse, after deflation, we can guarantee that the speed of the matrix-vector multiplication between the new affinity matrix and the iterative vector remains unchanged.

4.3.2 Number of iterations for computing each DPIC' pseudo-eigenvector

In this section, we point out that the number of matrix-vector multiplication steps for computing each of our pseudo-eigenvectors is equal to that of the first pseudo-eigenvector. In other words, the number of iterations, t, for performing the power iteration on the lth loop of Algorithm 2 is equal to that in the first loop, $\forall l \geq 2$.

From (14), $\mathbf{v}_l = \lambda_b^t \times \mathbf{f} + \lambda_s^t \times (d_{k+1}\mathbf{x}_{k+1} + \cdots + d_n\mathbf{x}_n)$, $\forall l \geq 1$. Therefore, on the one hand, in the first loop, $l = 1$, we can stop the iteration process with a sufficiently large value of t to remove the second factor, as $\lambda_s^t \ll \lambda_b^t$. On the other hand, in the lth loop, the value of t is equal to that of the first loop because it is enough to make $\lambda_s^t \ll \lambda_b^t$. Specifically, this value of t is chosen when the *acceleration* is small, as in Algorithm 1. Therefore, we need the same number of matrix-vector multiplication steps to find every pseudo-eigenvector even though the matrices in each loop are mutually different.

4.4 Time and space complexity of DPIC

As proved in Sect. 4.3, the computational time for each of our pseudo-eigenvectors is equal to PIC', which is $O(n)$ [17]. Therefore, according to Algorithm 2, for a dataset having k classes, our algorithm computes k pseudo-eigenvectors, with an overall time complexity of $O(nk)$. Currently, alongside the original SC, which is cubic in time, is a fast implementation using the Implicitly Restarted Arnoldi Method (SCIRAM), with a time complexity of $O(n^2)$ [17]. Therefore, our algorithm is faster than SCIRAM and SC in general because the number of classes k is much smaller than the number of data points n.

Our memory complexity is linear in space. To compute pseudo-eigenvectors, first, we need to store one sparse matrix whose size is $O(nt)$, where t is the number of neighbors for each data point. The sparse matrix is updated to compute each pseudo-eigenvector. Second, we need to store one iterative vector that is $O(n)$ in size. Third, two intermediate vectors $\mathbf{p}$ and $\mathbf{q}$, which cost $O(2n)$, should be stored for each pseudo-eigenvector. Therefore, the total memory required to compute k pseudo-eigenvectors is $O(n(t+2k+1))$, where t and k are small with respect to n. In our experiments, on average, n, t, and k are 3123, 10.71, and 5.14, respectively.

5 Experiments

In this section, we conduct various experiments on realistic and synthetic datasets to empirically justify the advantages in speed and accuracy of our algorithm over some conventional implementations of SC. Our computing system is Intel Core i5 2.53 GHz, 2 GB of RAM, Matlab R2008a, and Windows 7. We compare DPIC with PIC and two versions of SC. The first version is the original implementation SC [22] which finds all eigenvectors and then selects k eigenvectors with the largest eigenvalues. The second version is the fast implementation SC called SCIRAM [14]. SCIRAM directly approximates k eigenvectors without finding all eigenvectors. Hence, the computational time of the second version is significantly smaller than that of the first version. The following sections will briefly introduce these datasets.

5.1 Datasets

We use document, human face, and handwritten digit datasets in our experiments to test our algorithm over a wide range of domains. These datasets are as follows:

- The first dataset is the USPS handwritten digit [31] that contains 9,298 images, each of which has a handwritten digit from 0 to 9. Every image in this dataset measures 16×16 pixels. Therefore, each image is modeled by a 256-attribute vector. We use three datasets from USPS. The first is USPS49 that contains 4 and 9 digit images; the second is USPS3568 that contains 3, 5, 6, and 8 digit images; the third is USPS0127 that contains 0, 1, 2, and 7 digit images.

- The second dataset is a 20-newsgroup dataset [26], which is a collection of 20,000 documents from 20 different newsgroups. Some of these newsgroups are related because they have the same topic, and some are completely unrelated. We use four datasets from the 20-newsgroup dataset. The first is Newsa that covers the *computer* topic including *ms-windows*, *ibm-hardware*, *mac-hardware*, and *window-x* newsgroups. The second is Newsb that covers the *rec* topic including *autos*, *motorcycles*, *baseball*, and *hockey* newsgroups. The third is Newsc that covers the *sci* topic including *crypt*, *electronics*, *med*, and *space* newsgroups. The last is Newsd that covers the *talk* topic including *guns*, *mideast*, *politics.misc*, and *religion.misc* newsgroups. Each document in these datasets is represented by a 26,214-attribute vector. Each attribute of this vector is the frequency of a word in this document.
- The third dataset is the MNIST handwritten digit [1] that contains 70,000 handwritten digit images from 0 to 9. The dataset is divided into 60,000 training images and 10,000 test images. We select the first 10,000 training images for the purpose of clustering in our experiment. We use three datasets from MNIST. The first is MNIST49 that contains 4 and 9 digit images; the second is MNIST3568 that contains 3, 5, 6, and 8 digit images; the last is MNIST0127 that contains 0, 1, 2, and 7 digit images.
- The fourth dataset is Reuters [15], a benchmark dataset for document clustering. All documents in Reuters appeared on the Reuters newswire in 1987. The dataset contains 21,578 documents in 135 categories. After removing documents from multiple categories, the dataset contains 8,293 documents in 65 categories. We use one dataset from this version that contains six categories, namely, *acquisitions*, *crude*, *trade*, *money*, *interest*, and *ship*. Each document in this dataset is represented by a 18,933-attribute vector.
- The fifth dataset is UMist [29], a dataset for the purpose of human face clustering. The dataset consists of 575 face images of 20 different persons in 20 labeled folders including $1a$, $1b$, $1c, \ldots,$ $1t$. Each image is a grayscale 8-bit, 112×92 pixel image that shows the face of a person in one view. Therefore, each image is represented by a 10,304-attribute vector.
- The sixth dataset is Yeast in the UCI machine learning repository that contains 1,484 labeled protein sequences in 10 classes. Each protein sequence is described by one vector having 8 attributes. We select one major class *nuclear* and three minor classes from Yeast, namely, *extracellular*, *vacuolar*, and *peroxisomal*.
- The last dataset is TDT2 [2, 11], a collection of documents taken from different sources in 1998 such as newswires (e.g., APW and NYT), radio programs (e.g., VOA and PRI), and television programs (e.g., CNN and ABC). The dataset contains 11,201 documents in 96 categories. After removing documents from multiple categories, the remaining documents total to 9,394 in 30 categories. We select 4,981 documents in six categories with corresponding labels, namely, 20015, 20002, 20013, 20070, 20044, and 20076. Each document in this dataset is represented by a 36,771-attribute vector.

The detailed information on our datasets is listed in Table 1.

5.2 Evaluation measure

We use three measures, namely, purity, normalized mutual information (NMI), and rand index (RI), to evaluate clustering performance. These measures calculate the similarity between the result of each clustering algorithm and the exact clustering solution of the dataset. By using these measures, we can determine how well each clustering algorithm performs on a dataset and compare their performances. These metrics are widely used to measure clustering quality [3, 12, 16, 25, 32, 35].

5.2.1 Clustering purity

Each cluster in the predicted clustering result is reassigned to the exact label that is most frequent in the cluster to compute for the purity. Then, we count the number of correctly assigned data points. After summarizing, we divide this number by the number of all data points in the dataset to obtain the clustering purity. Specifically,

$$\text{purity}(\Omega, C) = \frac{1}{N} \sum_{k} \max_{j} |w_k \cap c_j|, \tag{20}$$

where $\Omega = \{w_1, w_2, \ldots, w_k\}$ is the predicted clustering result, and $C = \{c_1, c_2, \ldots, c_k\}$ is the exact clustering solution. The larger the purity, the better the algorithm performance. The largest value of purity is 1, and the smallest value is 0.

5.2.2 Normalized mutual information

NMI is a popular information theory measure used to evaluate the clustering quality. If we have two random variables $\mathbf{X}$ and $\mathbf{Y}$, the NMI between them is computed as follows:

$$\text{NMI}(\mathbf{X}, \mathbf{Y}) = \frac{I(\mathbf{X}, \mathbf{Y})}{\sqrt{H(\mathbf{X}) H(\mathbf{Y})}}, \tag{21}$$

where $I(\mathbf{X}, \mathbf{Y})$ is the mutual information between $\mathbf{X}$ and $\mathbf{Y}$; $H(\mathbf{X})$ and $H(\mathbf{Y})$ are the entropies of $\mathbf{X}$ and $\mathbf{Y}$, respectively. The mutual information between $\mathbf{X}$ and itself is 1. The larger the NMI, the better the performance. Specifically, NMI can be computed as

$$\text{NMI} = \frac{\sum_{l=1}^{c} \sum_{h=1}^{c} n_{l,h} \log(\frac{n \times n_{l,h}}{n_l \hat{n}_h})}{\sqrt{(\sum_{l=1}^{c} n_l \log \frac{n_l}{n})(\sum_{h=1}^{c} \hat{n}_h \log \frac{\hat{n}_h}{n})}}, \tag{22}$$

Table 1 Detailed information on our datasets. For each dataset, the number of data points n, the data dimensionality d, the number of classes c, the class distribution, and the class label are provided

Dataset	n	d	c	Class distribution	Class label
USPS0127	4,543	256	4	1,553/1,269/929/792	0/1/2/7
USPS3568	3,082	256	4	824/716/834/708	3/5/6/8
USPS49	1,673	256	2	852/821	4/9
MNIST0127	4,189	784	4	1,001/1,127/991/1,070	0/1/2/7
MNIST3568	3,853	784	4	1,032/863/1,014/944	3/5/6/8
MNIST49	1,958	784	2	980/978	4/9
Newsa	3,918	26,214	4	985/982/963/988	ms-windows/ibm-hardware/ mac-hardware/window-x
Newsb	3,979	26,214	4	990/996/994/999	autos/motorcycles/baseball/ hockey
Newsc	3,952	26,214	4	991/984/990/987	crypt/electronics/med/space
Newsd	3,253	26,214	4	910/940/775/628	guns/mideast/politics.misc/ religion.misc
Reuters	3,258	18,933	6	2,055/321/298/245/ 197/142	acquisitions/crude/trade/ money/interest/ship
TDT2	4,981	36,771	6	1,828/1,222/811/ 441/407/272	20015/20002/20013/ 20070/20044/20076
Yeast	514	8	4	429/35/30/20	nuclear/extracellular/ vacuolar/peroxisomal
UMist	575	10,304	20	38/35/26/24/26/23/19/ 22/20/32/34/34/26/30/ 19/26/26/33/48/34	1a/1b/1c/1d/1e/1f/1g/ 1h/1i/1j/1k/1l/1m/1n/ 1o/1p/1q/1r/1s/1t

where n_l is the number of data points in cluster ω_l of the predicted clustering result, $\hat{n}_h$ is the number of data points in cluster c_h of the exact clustering solution, and $n_{l,h}$ is the number of data points in the intersection between cluster ω_l and cluster c_h.

5.2.3 Rand index

RI is a measure used to compute clustering quality based on counting the pair of data points that both the predicted clustering result and the exact clustering solution agree and disagree that the two data points are in the same cluster. Specifically, we assume that the exact clustering solution of the dataset is C and that the predicted clustering result of the dataset is Ω. If the size of the dataset is N, then the number of data pairs is $N(N-1)/2$. Assume that N_{00} is the number of pairs that are in different clusters in both C and Ω and that N_{11} is the number of pairs that are in the same cluster in both C and Ω. RI can be computed as follows:

$$\mathrm{RI} = \frac{2 \times (N_{00} + N_{11})}{N(N-1)}. \tag{23}$$

The larger the RI, the better the clustering result.

5.3 Parameter setting and experiment process

We need to address two parameters to use SC as mentioned in Sect. 2. The first parameter is k, the number of classes, which is required for all algorithms including PIC, DPIC, SC, SCIRAM, and K-means. We manually set this parameter to the true number of classes for each dataset.

The second parameter is t, the number of neighbors for each data point. If t is too small, the connectivity inside each cluster may be lost. Otherwise, if t is too large, separate clusters may be merged. Therefore, we describe how to choose t as follows. For each dataset, we perform a pre-experiment by running SC with t selected from {5, 15, 25, 35, 45, 55, 65, 75}. Then, we select the value of t which maximizes the purity result of SC. Finally, with selected t, we report the computational time, purity, NMI, and RI for remaining algorithms. The details of the experiments when t is varied for SC are shown in Fig. 3. The performances of all algorithms with the optimal t are reported in Tables 2, 3, 4, and 5.

In all the experiments, we use the cosine measure that calculates the similarity between two data points $\mathbf{x}_i$ and $\mathbf{x}_j$ as follows:

$$s(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{x}_i^T . \mathbf{x}_j}{\|\mathbf{x}_i\| \|\mathbf{x}_j\|}. \tag{24}$$

Table 2 Computational time (in milliseconds) results for PIC, the proposed DPIC, SCIRAM, and SC

Dataset	PIC	DPIC	SCIRAM	SC
USPS0127	**36**	***227***	1,558	278,810
USPS3568	**16**	***121***	859	85,845
USPS49	**15**	***39***	222	15,922
Newsa	**36**	***242***	5,532	161,133
Newsb	**39**	***246***	3,166	168,413
Newsc	**40**	***259***	4,052	161,539
Newsd	**27**	***261***	2,341	97,768
MNIST0127	**24**	***163***	861	203,674
MNIST3568	**26**	***187***	2,315	170,492
MNIST49	**10**	***30***	368	25,672
Reuters	**39**	***383***	3,063	100,101
TDT2	**71**	***543***	2,497	299,839
Yeast	**3**	***30***	205	590
UMist	**3**	237	***184***	942

Table 3 Clustering purity results for PIC, the proposed DPIC, SCIRAM, SC, and K-means

Dataset	PIC	DPIC	SCIRAM	SC	K-means
USPS0127	0.9725	**0.9802**	0.9797	0.9795	0.8831
USPS3568	0.8968	**0.9429**	0.7407	0.7744	0.7744
USPS49	**0.8739**	0.8601	0.5642	0.8523	0.6891
Newsa	0.2570	**0.4939**	0.3236	0.4775	0.2585
Newsb	0.5959	**0.6886**	0.4579	0.6790	0.2588
Newsc	0.5768	**0.6941**	0.4626	0.6675	0.2555
Newsd	0.4986	**0.6148**	0.5192	0.5799	0.3021
MNIST0127	0.7085	**0.9754**	0.8221	0.9709	0.9006
MNIST3568	0.5738	**0.8152**	0.7809	0.8084	0.6096
MNIST49	**0.7706**	0.5592	0.5306	0.5306	0.5074
Reuters	0.7909	**0.8539**	0.8422	0.8453	0.6906
TDT2	0.9217	**0.9851**	0.9825	0.9830	0.5725
Yeast	0.8560	0.9066	0.9105	**0.9105**	0.8618
UMist	0.3861	**0.6643**	0.6017	0.5687	0.5234

For a fair comparison between these algorithms, each algorithm is performed 100 times on each dataset. Afterward, we take the best performance. In each run, after finding the pseudo-eigenvectors, K-means is performed 100 times to avoid the local optimal solution of random initialization.

5.4 Computational time result

Table 2 presents the computational time results for PIC, DPIC, SCIRAM, and SC. PIC is the fastest among these algorithms. Its average computational time is lower than that of SC, especially in a large dataset with a size of almost 4,000 data points such as USPS0127, Newsa, Newsb, Newsc, MNIST0127, and TDT2.

DPIC is almost k times slower than PIC, where k is the number of classes. Nevertheless, DPIC is still faster than SCIRAM and SC because k is small (less than 10) and PIC is over 50 times faster than SCIRAM in most datasets. Our computational time is a combination of two factors. The first factor is for finding k pseudo-eigenvectors, and the second one is for finding some intermediate vectors $\mathbf{p}$ and $\mathbf{q}$, as in Sect. 4.3.1 (which can be less than or similar to the first factor in most cases). In particular datasets (e.g., USPS49, MNIST49, Reuters, and TDT2), our algorithm is almost k times slower than PIC. The experiment proves that the time to compute for every pseudo-eigenvector is almost equal to the time to compute for the PIC' pseudo-eigenvector. This result also means that the time to compute intermediate vectors $\mathbf{p}$ and $\mathbf{q}$ is negligible with respect to the time to compute for each pseudo-eigenvector. In 13 out of 14 datasets, our algorithm is faster than SCIRAM. Especially in the Newsa dataset, DPIC is 20 times faster than SCIRAM. However,

Table 4 NMI results for PIC, the proposed DPIC, SCIRAM, SC, and K-means

Dataset	PIC	DPIC	SCIRAM	SC	K-means
USPS0127	0.9037	**0.9264**	0.9249	0.9244	0.7784
USPS3568	0.7536	**0.8207**	0.6794	0.6781	0.5188
USPS49	**0.4533**	0.4176	0.0121	0.4028	0.1118
Newsa	0.0205	0.2667	0.0270	**0.2725**	0.0324
Newsb	0.4315	**0.5567**	0.3103	0.5563	0.0336
Newsc	0.3872	**0.5818**	0.2192	0.5554	0.0312
Newsd	0.2876	**0.4284**	0.3035	0.4248	0.0503
MNIST0127	0.6420	**0.9060**	0.7466	0.8988	0.7386
MNIST3568	0.4710	**0.6667**	0.6002	0.6600	0.3866
MNIST49	**0.2793**	0.0101	0.0027	0.0027	0.0018
Reuters	0.4931	**0.6704**	0.6051	0.6134	0.2497
TDT2	0.8460	**0.9415**	0.9348	0.9368	0.3602
Yeast	0.2434	0.3752	0.3917	**0.3917**	0.1839
UMist	0.4760	**0.7436**	0.7248	0.6865	0.6554

Table 5 RI results for PIC, the proposed DPIC, SCIRAM, SC, and K-means

Dataset	PIC	DPIC	SCIRAM	SC	K-means
USPS0127	0.9738	**0.9803**	0.9798	0.9796	0.8978
USPS3568	0.9114	**0.9457**	0.8110	0.8108	0.8206
USPS49	**0.7795**	0.7594	0.5082	0.7491	0.5716
Newsa	0.2589	**0.6473**	0.5702	0.6321	0.2594
Newsb	0.7409	**0.8004**	0.7045	0.7918	0.2606
Newsc	0.6827	**0.7760**	0.6620	0.7408	0.2558
Newsd	0.6264	**0.7085**	0.6220	0.5825	0.2781
MNIST0127	0.8160	**0.9759**	0.8709	0.9720	0.9073
MNIST3568	0.6433	**0.8481**	0.8171	0.8431	0.7114
MNIST49	**0.6465**	0.5070	0.5018	0.5018	0.5005
Reuters	0.7254	**0.9077**	0.8505	0.8508	0.6129
TDT2	0.9446	**0.9863**	0.9840	0.9844	0.5105
Yeast	0.5931	0.6920	0.6973	**0.6973**	0.4731
UMist	0.9150	**0.9425**	0.9312	0.9262	0.9322

in the UMist dataset, our computational time is longer than that of SCIRAM because the cost of computing intermediate vectors is high.

SCIRAM is faster than SC in all experiments; these algorithms are in the third and fourth places, respectively. The speed of SC depends on the number of data points, as it computes all eigenvectors. However, because SCIRAM quickly approximates k eigenvectors using some characters of the dataset, its speed also depends on the dataset structure. For example, even though the MNIST0127 dataset is larger than the MNIST3568 dataset, the computational time of SCIRAM on MNIST0127 is lower.

5.5 Accuracy result

The accuracy of PIC is the worst among these algorithms, except K-means, in 9 out of 12 multi-class datasets (especially in UMist, Reuters, MNIST0127, and Newsa) because of the inter-class collision problems. The results are shown in Tables 3, 4, and 5. However, without the problem, PIC achieves better accuracy than do other algorithms in two-class datasets (i.e., USPS49 and MNIST49). The better accuracy of PIC compared to SC is explained in [16]. SC selects the k largest eigenvectors in which there may be a "bad" eigenvector. PIC computes the pseudo-eigenvector

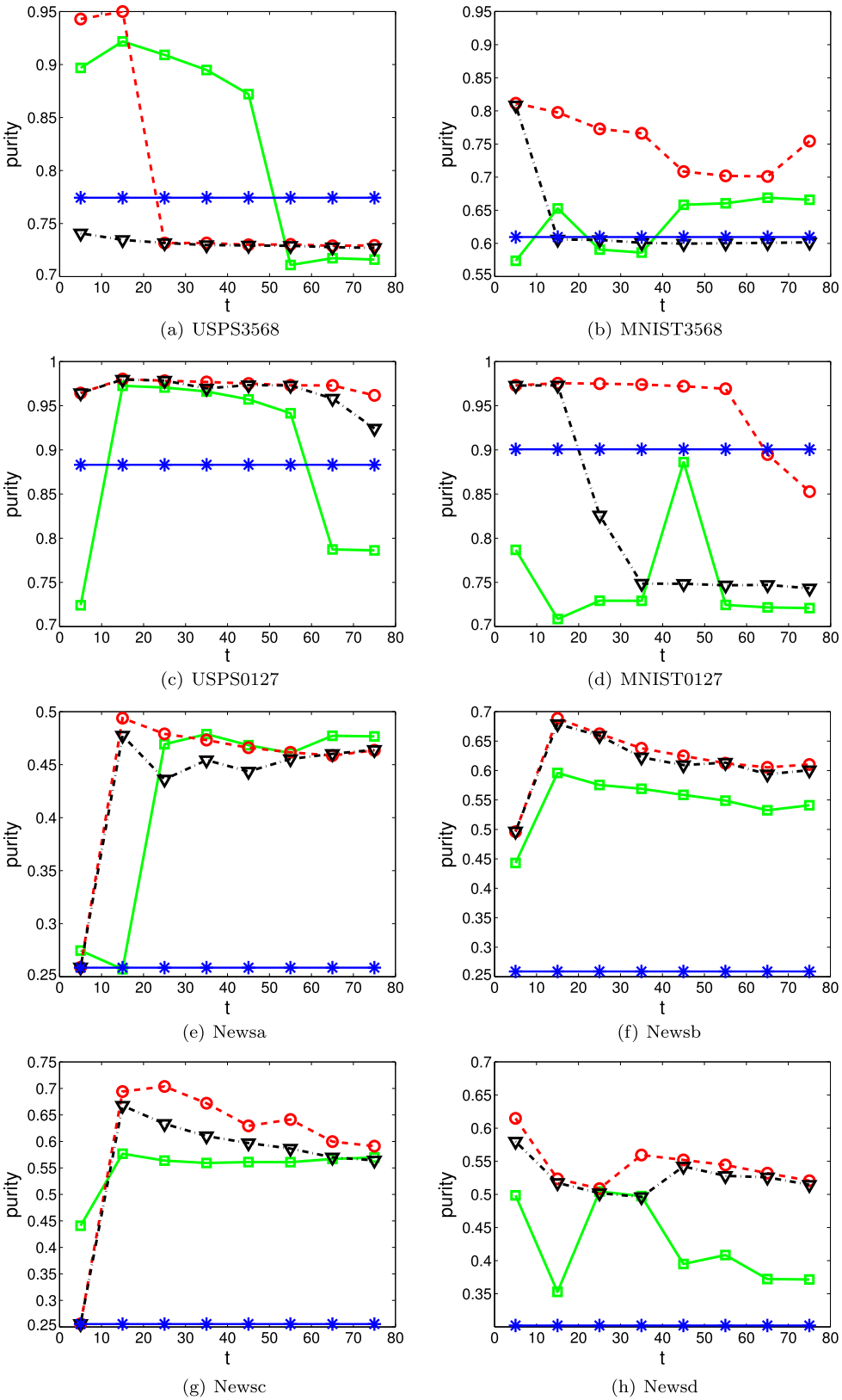
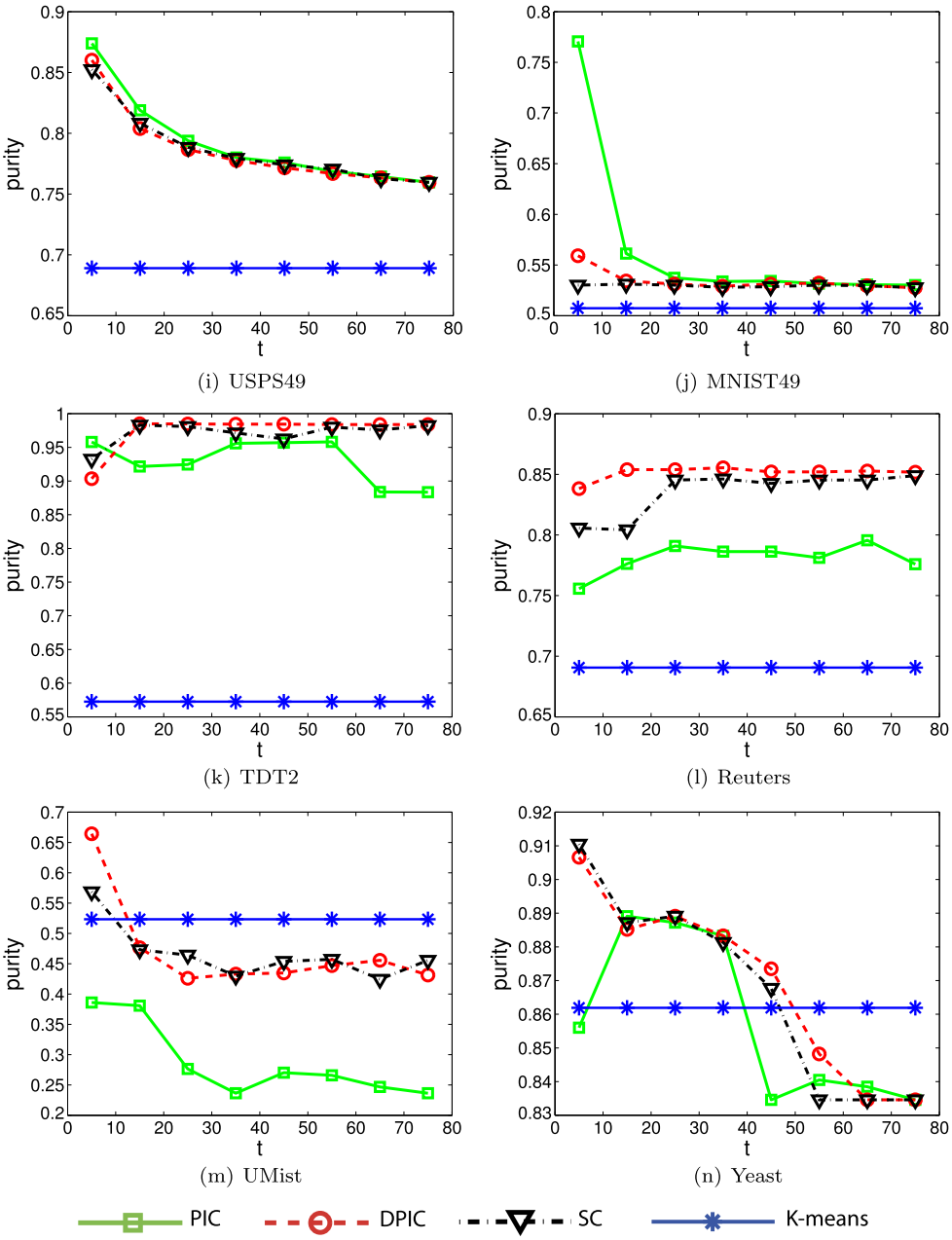


Fig. 3 Clustering purity results for PIC, DPIC, SC, and K-means on realistic datasets with different values of t

Fig. 3 (*Continued*)



which is a linear combination of the k largest eigenvectors. However, in the linear combination, an eigenvector corresponding to a small eigenvalue has a small weight. Luxburg [18] shows that good eigenvectors often have high eigenvalues. In consequence, even though PIC is an approximate algorithm, its pseudo-eigenvector is sometimes more useful than SC' eigenvectors.

DPIC achieves the best accuracy in 11 out of 14 datasets, especially in UMist, Reuters, MNIST3568, and USPS3568. We adopt the better accuracy of PIC compared to SC because our algorithm also computes pseudo-eigenvectors. In particular, the accuracy of DPIC in the USPS3568 dataset is 95 %, whereas the accuracy of SC is only 73 %. In the UMist dataset, the accuracy of DPIC is 66 %, whereas the accuracy of SC is only 57 %. Our algorithm has a significant improvement compared with PIC in the multi-class datasets (i.e., Newsa, Newsb, Newsc, Newsd, MNIST0127, MNIST3568, Reuters, and UMist), especially in UMist in which our accuracy is 66 % whereas that of PIC is only 39 %. The result proves that our pseudo-eigenvectors are useful for clustering and do not contain redundant information about each other. However, in USPS49 and MNIST49, the accuracy of DPIC is lower than that of PIC because one pseudo-eigenvector is enough for these two-class datasets. As DPIC computes two pseudo-eigenvectors, with the second one containing noises, its accuracy is low.

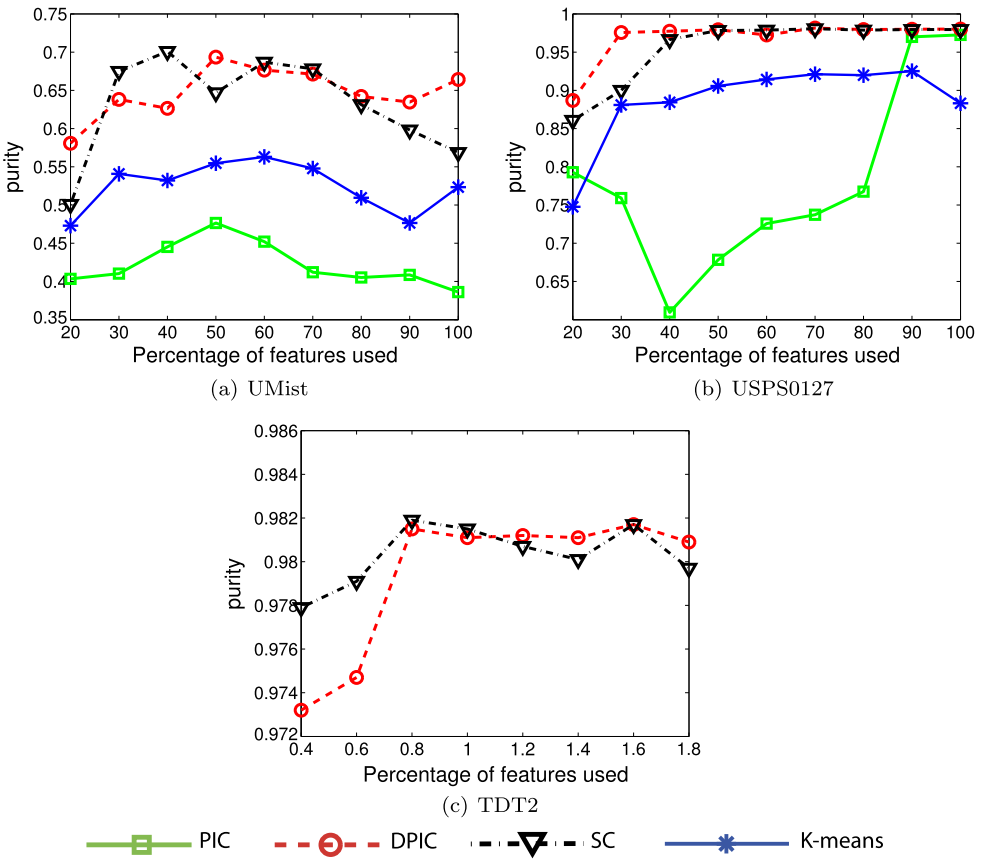


Fig. 4 Clustering purity results with feature selection for PIC, DPIC, SC, and K-means. Note that in the TDT2 documents dataset, we do not report the low result of PIC (86 %) and K-means (63 %) in order to see the evolution in accuracy of SC and DPIC

We also report the purity results of these algorithms on Yeast, a dataset with rare classes. Generally, a rare class is a minor but compact class [10]. Our focus is to identify minor classes, but better accuracy may come from classifying two major classes. Therefore, we only input one major class of Yeast. The purity result of PIC is low at 85.6 % because of three minor classes. Conversely, SC and DPIC achieve a better solution at 90.66 % by computing multiple vectors to solve the inter-class collision problem between the three minor classes.

Although SCIRAM is not the best algorithm in terms of accuracy, it is a good approximation of SC because it is faster while maintaining the same clustering quality. However, running SCIRAM on some datasets, such as Newsa, Newsb, Newsc, and MNIST0127, is unsafe because the accuracy is significantly reduced.

Lastly, we report the performance of every algorithm under different numbers of neighbors t for each data point in Fig. 3. The accuracy of our proposed DPIC is better than that of PIC and almost similar to that of SC in almost all datasets. The accuracy of SCIRAM is the same as or lower than that of SC in most datasets for every t so it is not reported here. Each dataset has an optimal value of t with which SC, DPIC, and PIC work best. For example, in Newsa, Newsb, and Newsc, these algorithms achieve good results when t is 15, which is not too small or too large.

5.6 DPIC with feature selection

This section conducts experiments to determine whether the accuracy of DPIC can be improved when incorporating feature selection as a pre-processing step. Our algorithm is an unsupervised method that departs from an affinity matrix with the similarity between data points and their local nearest neighbors. Therefore, we use the Laplacian score method [9] for the feature selection step because of its unsupervised manner and local neighborhood preserving advantage. We perform experiments on three datasets USPS0127, TDT2, and especially UMist because of its huge number of features (10,304) and low number of data points (575).

In the experiments, we use a value of t with which SC works best without the feature selection for each dataset. At the beginning, we use the Laplacian score to rank all features and then select some top-ranked features. Finally, we report the clustering accuracy by varying the number of selected top-ranked features from 20 % to 100 % of the number of all features in the dataset, as shown in Fig. 4.

Figure 4 shows that the accuracies of SC and DPIC with the Laplacian feature selection can be at least equal to those

Table 6 Computational time (in milliseconds), maximum purity (MaxPurity), and average purity (AvgPurity) for the proposed DPIC and SCIRAM on synthetic graphs

Number of nodes	SCIRAM			DPIC		
	Time	MaxPurity	AvgPurity	Time	MaxPurity	AvgPurity
10,000	Out of memory	0.000	0.000	**747**	**1.000**	**1.000**
9,000	Out of memory	0.000	0.000	**624**	**1.000**	**1.000**
8,000	Out of memory	0.000	0.000	**475**	**1.000**	**1.000**
7,000	Out of memory	0.000	0.000	**368**	**1.000**	**1.000**
6,000	Out of memory	0.000	0.000	**254**	**1.000**	**1.000**
5,000	17,721	0.9936	0.7700	**170**	**1.000**	**1.000**
4,000	9,433	0.9978	0.8608	**104**	**1.000**	**1.000**
3,000	5,188	1.000	0.9721	**79**	**1.000**	**1.000**
2,000	1,645	0.9975	0.7569	**40**	**1.000**	**1.000**
1,000	1,131	0.8980	0.6580	**18**	**1.000**	**1.000**

without feature selection. Feature selection shows its essential role particularly in the UMist dataset because, if we use just enough features (40 % to 60 %), we will achieve better accuracy than using a full set of features. Although we cannot obtain good results for the USPS0127 and TDT2 datasets compared with the UMist dataset, at least we can compress the number of features to low levels (40 % and 0.8 %, respectively) while retaining equivalent accuracy.

5.7 Experiments on synthetic datasets

First, this section describes the experiments performed on synthetic graphs to show the memory usage advantage of our algorithm over SCIRAM. Second, we show that DPIC can achieve better accuracy than a straightforward algorithm using k random initializations of PIC. To generate a large, sparse, and multi-class graph, we use a modified version of Erdős, Rényi [7] random network model. The model is widely applied to generate a synthetic network [6, 16].

5.7.1 Comparison of memory usage between DPIC and SCIRAM

We create 10 synthetic graphs, each having four clusters, with the number of nodes n ranging from 1,000 to 10,000 to compare the memory usage of our algorithm and that of SCIRAM. In each synthetic graph, every node connects with t neighbor nodes when t is 2 % of n. Each cluster is a strongly connected cluster, as 80 % of the edges connect two nodes in the same cluster, and only 20 % of the edges connect two nodes from different clusters. In each graph, we run each algorithm 10 times and take the highest purity score and the average purity score. The performances of SCIRAM and DPIC on these synthetic graphs are shown in Table 6.

SCIRAM consumes a huge amount of memory. When the number of nodes is greater than 6,000, it cannot produce the clustering result because of the *out of memory* problem. By contrast, DPIC can perform well in all synthetic graphs because it stores only one sparse matrix and $(2k + 1)$ intermediate vectors. Therefore, DPIC is linear in space, $O(n(t + 2k + 1))$, where t is smaller than 200 and k is equal to four in all graphs. Although a strong connectivity exists inside every cluster, SCIRAM cannot achieve the highest purity score. Conversely, DPIC can achieve 100 % purity in all synthetic graphs.

5.7.2 Comparison of accuracy between DPIC and k random PIC

A straightforward solution to overcome the inter-class collision problem is to run PIC k times, with each time having different initialization vector v_0. We call this solution the k-random PIC. The limitation of this solution is that it computes redundant pseudo-eigenvectors that are not mutually orthogonal. We create 10 synthetic graphs, with the number of nodes n ranging from 1,000 to 10,000, to show that our algorithm, which produces multiple orthogonal pseudo-eigenvectors, is more accurate than this straightforward solution.

We generate two groups of synthetic graphs. In the first group, we set t, the number of neighbors for each node, to $0.01n$ for all graphs. In the second group, we flexibly set t to be equal to $0.01n$ for graphs having less than 3,000 nodes, $0.005n$ for graphs having 4,000 to 6,000 nodes, and $0.002n$ for graphs having more than 7,000 nodes.

The purity results of DPIC and k-random PIC in the first group are shown in Fig. 5. As we set t to be directly proportional to n, the number of edges increases quadratically with the number of nodes. In consequence, the gap between the number of edges connecting two nodes in the same cluster and the number of edges connecting two nodes from different clusters also increases quadratically with the number of nodes. Therefore, finding the clustering solution for large

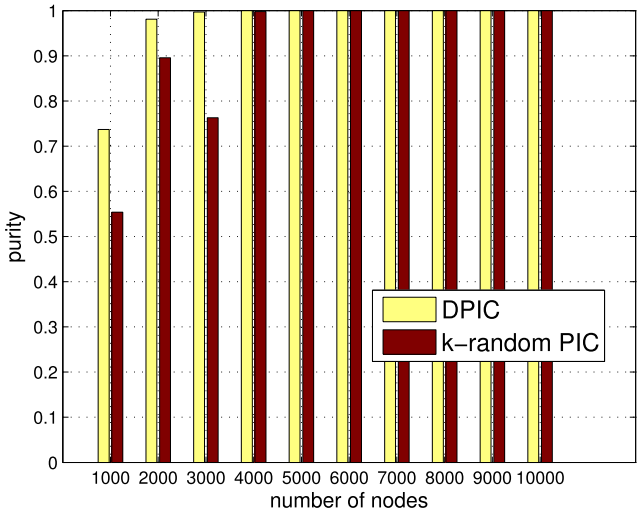


Fig. 5 Clustering purity results for DPIC and k random initializations PIC (k-random PIC) on the first group

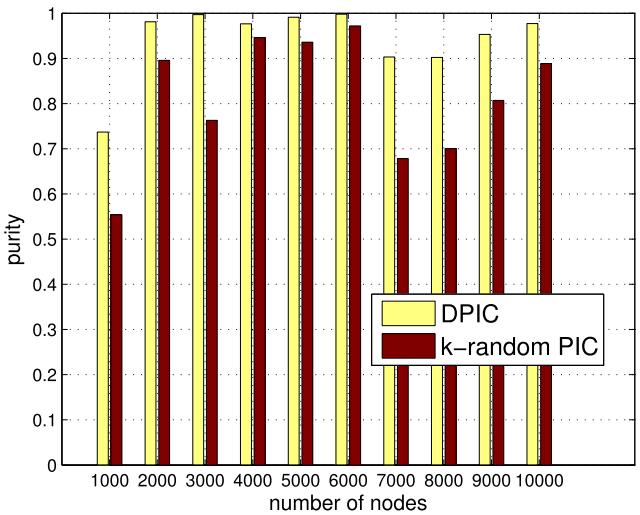


Fig. 6 Clustering purity results for DPIC and k random initializations PIC (k-random PIC) on the second group

graphs that have more than 4,000 nodes seems easy. In these graphs, both DPIC and k-random PIC achieve best results.

The purity results of DPIC and k-random PIC in the second group are shown in Fig. 6. As we slightly decrease t for large graphs, the number of edges does not increase too fast with respect to the number of nodes. Indeed, k-random PIC achieves lower purity than DPIC in these cases. The results experimentally show that, in difficult situations, DPIC produces a better solution than k-random PIC.

6 Conclusion

In this paper, we propose a solution to overcome the inter-class collision problem in PIC. Our method uses the deflation technique to find more pseudo-eigenvectors. Therefore, our DPIC algorithm has better accuracy than PIC and smaller complexity than the state-of-the-art SCI-RAM. Our method only finds mutually orthogonal pseudo-eigenvectors; therefore, the method avoids finding redundant pseudo-eigenvectors. Our experiments on classical and realistic document, human face, and handwritten digit datasets justify our idea in theory. DPIC is suitable for large, sparse, and multi-class datasets. However, the pseudo-eigenvectors still contain noise along with the classification information because of the method used to compute pseudo-eigenvectors. After using deflation on these pseudo-eigenvectors, the inner structure of the dataset can be degraded. In the future, we will examine how to reduce this noise so that the deflation has a stronger effect on PIC.

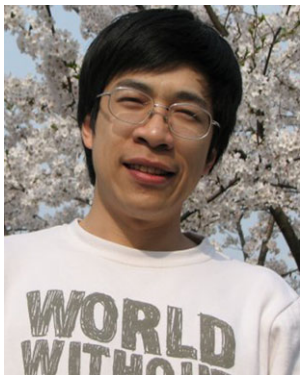
Acknowledgements This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2010-0013689).

References

1. Cai D Mnist dataset. URL http://www.cad.zju.edu.cn/home/dengcai/Data/MNIST/10kTrain.mat
2. Cai D Tdt2 dataset. URL http://www.cad.zju.edu.cn/home/dengcai/Data/TDT2/TDT2.mat
3. Chen X, Cai D (2011) Large scale spectral clustering with landmark-based representation. In: Proceedings of the twenty-fifth AAAI conference on artificial intelligence, San Francisco, California, pp 313–318
4. Chen L, Mao X, Wei P, Xue Y, Ishizuka M (2012) Mandarin emotion recognition combining acoustic and emotional point information. Appl Intell 37(4):602–612
5. Drineas P, Mahoney MW (2005) On the Nyström method for approximating a gram matrix for improved kernel-based learning. J Mach Learn Res 6:2153–2175
6. Durrett R (2010) Some features of the spread of epidemics and information on a random graph. Proc Natl Acad Sci 107(10):4491–4498
7. Erdős P, Rényi A (1960) On the evolution of random graphs. Magy Tud Akad Mat Kut Intéz Közl 5:17–61
8. Fowlkes C, Belongie S, Chung F, Malik J (2004) Spectral grouping using the Nyström method. IEEE Trans Pattern Anal Mach Intell 26(2):214–225
9. He X, Cai D, Niyogi P (2005) Laplacian score for feature selection. Adv Neural Inf Process Syst 18:507–514
10. He J, Tong H, Carbonell J (2010) Rare category characterization. In: Proceedings of the 10th IEEE international conference on data mining, Sydney, Australia, pp 226–235
11. Hu X, Zhang X, Lu C, Park EK, Zhou X (2009) Exploiting Wikipedia as external knowledge for document clustering. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining, Paris, France, pp 389–396
12. Jain AK (2010) Data clustering: 50 years beyond k-means. Pattern Recognit Lett 31(8):651–666
13. Keshet J, Bengio S (2009) Automatic speech and speaker recognition: large margin and kernel methods. Wiley Online Library
14. Lehoucq RB, Sorensen DC (1996) Deflation techniques for an implicitly restarted Arnoldi iteration. SIAM J Matrix Anal Appl 17(4):789–821

15. Lewis DD Reuters-21578 dataset. URL http://www.daviddlewis.com/resources/testcollections/reuters21578/
16. Lin F, Cohen WW (2010) Power iteration clustering. In: Proceedings of the 27th international conference on machine learning, Haifa, Israel, pp 655–662
17. Lin F, Cohen WW (2010) A very fast method for clustering big text datasets. In: Proceedings of the 19th European conference on artificial intelligence, Lisbon, Portugal, pp 303–308
18. Luxburg UV (2007) A tutorial on spectral clustering. Stat Comput 17(4):395–416
19. Mackey L (2008) Deflation methods for sparse pca. Adv Neural Inf Process Syst 21:1017–1024
20. Mavroeidis D (2010) Accelerating spectral clustering with partial supervision. Data Min Knowl Discov 21(2):241–258
21. Mishra N, Schreiber R, Stanton I, Tarjan RE (2007) Clustering social networks. In: Proceedings of the 5th international conference on algorithms and models for the web-graph, San Diego, CA, pp 56–67
22. Ng AY, Jordan MI, Weiss Y (2001) On spectral clustering: analysis and an algorithm. Adv Neural Inf Process Syst 14:849–856
23. Pavan M, Pelillo M (2007) Dominant sets and pairwise clustering. IEEE Trans Pattern Anal Mach Intell 29(1):167–172
24. Peña JM, Lozano JA, Larrañaga P (1999) An empirical comparison of four initialization methods for the k-means algorithm. Pattern Recognit Lett 20(10):1027–1040
25. Peng W, Li T (2011) On the equivalence between nonnegative tensor factorization and tensorial probabilistic latent semantic analysis. Appl Intell 35(2):285–295
26. Rennie J 20 newsgroups. URL http://qwone.com/~jason/20Newsgroups/
27. Saha S, Bandyopadhyay S (2011) Automatic MR brain image segmentation using a multiseed based multiobjective clustering approach. Appl Intell 35(3):411–427
28. Shang F, Jiao LC, Shi J, Wang F, Gong M (2012) Fast affinity propagation clustering: a multilevel approach. Pattern Recognit 45(1):474–486
29. Sheffield Face database. URL http://www.sheffield.ac.uk/eee/research/iel/research/face
30. Shi J, Malik J (2000) Normalized cuts and image segmentation. IEEE Trans Pattern Anal Mach Intell 22(8):888–905
31. Smola A, Schölkopf B Datasets for benchmarks and applications. URL http://www.kernel-machines.org/data/
32. Taşdemir K (2012) Vector quantization based approximate spectral clustering of large datasets. Pattern Recognit 45(8):3034–3044
33. Tung F, Wong A, Clausi DA (2010) Enabling scalable spectral clustering for image segmentation. Pattern Recognit 43(12):4069–4076
34. Wu S, Chow TWS (2004) Clustering of the self-organizing map using a clustering validity index based on inter-cluster and intra-cluster density. Pattern Recognit 37(2):175–188
35. Wu M, Schölkopf B (2006) A local learning approach for clustering. Adv Neural Inf Process Syst 19:1529–1536
36. Yan D, Huang L, Jordan MI (2009) Fast approximate spectral clustering. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining, Paris, France, pp 907–916
37. Zelnik-Manor L, Perona P (2004) Self-tuning spectral clustering. Adv Neural Inf Process Syst 17:1601–1608
38. Zhang K, Kwok JT (2009) Density-weighted Nyström method for computing large kernel eigensystems. Neural Comput 21(1):121–146
39. Zhang K, Tsang IW, Kwok JT (2008) Improved Nyström low-rank approximation and error analysis. In: Proceedings of the 25th international conference on machine learning, Helsinki, Finland, pp 1232–1239

Anh Pham The received his B.S. in Computer Science from Hanoi University of Technology, Vietnam, in 2010. He got his M.S. in Computer Engineering from Kyung Hee University, Korea, in 2012. His research interests include data mining, pattern recognition, and machine learning.



Nguyen Duc Thang received his B.E. degree in Computer Engineering from Posts and Telecommunications Institute of Technology, Vietnam. He got his M.S. and Ph.D. degrees in the Department of Computer Engineering at Kyung Hee University, South Korea. His research interests include artificial intelligence, computer vision, and machine learning.

La The Vinh received his B.S. and M.S. from Hanoi University of Science and Technology, Vietnam, in 2004 and 2007, respectively. He has completed his Ph.D. degree at the Department of Computer Engineering at Kyung Hee University, Korea, 2012. Currently, he is working as a lecturer at the School of Information and Communication Technology, Hanoi University of Science and Technology. His research interests include digital signal processing, pattern recognition and artificial intelligence.

Young-Koo Lee got his B.S., M.S. and Ph.D. in Computer Science from Korea advanced Institute of Science and Technology, Korea. He is a professor in the Department of Computer Engineering at Kyung Hee University, Korea. His research interests include ubiquitous data management, data mining, and databases. He is a member of the IEEE, the IEEE Computer Society, and the ACM.

Sungyoung Lee received his B.S. from Korea University, Seoul, Korea. He got his M.S. and Ph.D. degrees in Computer Science from Illinois Institute of Technology (IIT), Chicago, Illinois, USA in 1987 and 1991 respectively. He has been a professor in the Department of Computer Engineering, Kyung Hee University, Korea since 1993. He is a founding director of the Ubiquitous Computing Laboratory, and has been affiliated with a director of Neo Medical ubiquitous-Life Care Information Technology Research Center, Kyung Hee University since 2006. He is a member of ACM and IEEE.