FUSE: Full Spectral Clustering

Wei Ye[†], Sebastian Goebl[†], Claudia Plant[§], Christian Böhm[†]
[†]Ludwig-Maximilians-Universität München, Munich, Germany
[§] University of Vienna, Vienna, Austria
{ye, goebl, boehm}@dbs.ifi.lmu.de
claudia.plant@univie.ac.at

ABSTRACT

Multi-scale data which contains structures at different scales of size and density is a big challenge for spectral clustering. Even given a suitable locally scaled affinity matrix, the first k eigenvectors of such a matrix still cannot separate clusters well. Thus, in this paper, we exploit the fusion of the cluster-separation information from all eigenvectors to achieve a better clustering result. Our method **FU**ll **S**pectral Clust**E**ring (FUSE) is based on Power Iteration (PI) and Independent Component Analysis (ICA). PI is used to fuse all eigenvectors to one pseudo-eigenvector which inherits all the cluster-separation information. To conquer the cluster-collision problem, we utilize PI to generate p ($p > k$) pseudo-eigenvectors. Since these pseudo-eigenvectors are redundant and the cluster-separation information is contaminated with noise, ICA is adopted to rotate the pseudo-eigenvectors to make them pairwise statistically independent. To let ICA overcome local optima and speed up the search process, we develop a self-adaptive and self-learning greedy search method. Finally, we select k rotated pseudo-eigenvectors (independent components) which have more cluster-separation information measured by *kurtosis* for clustering. Various synthetic and real-world data verifies the effectiveness and efficiency of our FUSE method.

Keywords

Spectral clustering; Power iteration; ICA; Givens rotation; Multi-scale data

1. INTRODUCTION

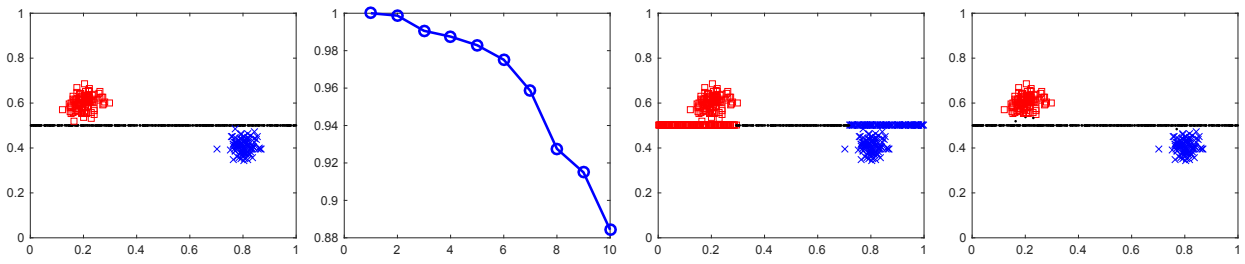
Clustering is a basic technique in data analysis and mining. Two commonly used methods are k-means and Expectation Maximization clustering (EM) which pre-assume that data fits a Gaussian model. Such model-based clustering methods perform well if data fits the model. However, in most cases, we do not know the distribution of data. It is hard to decide which model to adopt. Spectral clustering, on the other hand, does not pre-assume any model. It only uses local information (point to point similarity) to achieve global clustering. Thus it is very elegant and popular in data ming and machine learning. Spectral clustering transforms the clustering of a set of data points with pairwise similarities into a graph partitioning problem, i.e., partitioning a graph such that the intra-group edge weights are high and the inter-group edge weights are low. There are three kinds of similarity graphs, i.e., the ε-neighborhood graph, the k-nearest neighbor graph and the fully connected graph [20]. Luxburg [20] emphasized that "*theoretical results on the question how the choice of the similarity graph influences the spectral clustering result do not exist*". However, the parameters (ε,k,σ) of these similarity graphs highly affect the clustering results, especially in cases where data contains structures at different scales of size and density. One usually used objective function in spectral clustering is *normalized cut* [16]. As pointed out in [14], the *normalized cut* criterion does not always work even given a proper affinity matrix.

Consider three clusters of different geometry shapes and densities in Figure 1(a). Both Gaussian clusters have 100 data points. The rectangular stripe cluster has 400 data points sampled from a uniform distribution. Conventional spectral clustering algorithms tend to fail on this multi-scale data. Self-tuning spectral clustering (ZP) [23] proposes to use the locally scaled affinity matrix to solve the limitation. Further, ZP rotates the eigenvectors to create the maximally sparse representation to estimate the number of clusters automatically. However, such proposals still do not work on multi-scale data because of the unsuitability of the *normalized cut* criterion only using local information. Such an argument can be inferred from Figure 1(c). ZP fails to correctly separate the three clusters. Both cuts are along the stripe. Intuitively, it is not difficult to understand. The *normalized cut* criterion tries to make clusters "balanced" as measured by the number of vertices or edge weights. Since each of the two Gaussian clusters only has 100 data points and they are so close to the stripe cluster, cuts between the Gaussian clusters and the stripe cluster have a higher penalty than those along the stripe.

Differing from other spectral clustering algorithms, our method combines the cluster-separation information from all eigenvectors to achieve a better clustering result. As can be seen from Figure 1(d), only some controversial data points lying on the boundaries are clustered incorrectly. The fusion of the cluster-separation information from all eigenvectors is accomplished by exploiting truncated Power Iteration (PI). To yield good clustering, spectral clustering uses the first k eigenvectors of the graph Laplacian matrix. Similarly, we use PI to generate p ($p > k$) pseudo-eigenvectors. Each pseudo-eigenvector is a linear combination of all original eigenvectors, including the information not only from the "informative" eigenvectors but from the "noise" eigenvectors. Note that the pseudo-eigenvectors are redundant to each other. One main question is how to make the information from the "informative" eigenvectors stand out and suppress the information from the

KDD '16, August 13-17, 2016, San Francisco, CA, USA

DOI: http://dx.doi.org/10.1145/2939672.2939845



(a) SYN1 **(b)** eigenvalues **(c)** ZP **(d)** FUSE

Figure 1: Clustering results of ZP and FUSE on our SYN1 data ((b) gives the top 10 eigenvalues of the normalized affinity matrix).

"noise" eigenvectors? In this paper, we use Independent Component Analysis (ICA) to reduce the redundancy, i.e., to make the pseudo-eigenvectors statistically independent (non-redundant) to each other. After whitening (more details in Section 3.1), ICA rotates the pseudo-eigenvectors to find the direction in which the entropy is minimized. Subsequently, a *kurtosis*-based selection strategy is exploited. Such a minimum-entropy rotation plus a *kurtosis*-based selection improve the cluster separation. Our contributions are as follows,

Contributions

1) **We achieve the eigenvector-fusion by using Power Iteration (PI)**. The generated pseudo-eigenvectors include information from all eigenvectors.

2) **We improve the cluster separation by applying ICA combined with a *kurtosis*-based selection strategy**. Since the generated pseudo-eigenvectors are redundant to each other, which is not beneficial to good clustering, we apply ICA to make them statistically independent. Then, a *kurtosis*-based selection strategy is exploited to improve the cluster separation. To the best of our knowledge, we are the first to apply ICA on spectral clustering.

3) **We develop a greedy search method to render searching for statistically independent components more efficient and effective**. The greedy search strategy discriminates the search order to let ICA not get easily trapped into local optimal. In addition, during the search process, the greedy search makes use of self-adaptive and self-learning strategies to balance the efficiency and effectiveness.

2. PRELIMINARIES

In this section, we give some major notations used in this paper and some background techniques our algorithm is based on to make this paper self-contained.

2.1 Notations

We use lower-case Roman letters (e.g. a, b) to denote scalars. Upper-case Roman letters (e.g. X, Y) are used for continuous random variables. We denote vectors (row) by boldface lower case letters (e.g. $\mathbf{x}$). Matrices are denoted by boldface upper case letters (e.g. $\mathbf{X}$). We denote entries in a matrix by non-bold lower case letters, such as x_{ij}. Row i of matrix $\mathbf{X}$ is denoted by the vector $\mathbf{x}_{i\cdot}$, column j by the vector $\mathbf{x}_{\cdot j}$. We use $[x_1, \cdots, x_n]$ to denote a row created by stacking n continuous random variables; similarly, we use $\mathbf{X} = [\mathbf{x}_1; \cdots; \mathbf{x}_m]$ to denote creating a matrix by stacking $\mathbf{x}_i$ along the rows. A set is denoted by calligraphic capital letters (e.g. $\mathcal{S}$). A cluster $\mathcal{C}$ is a set of data objects, i.e., $\mathcal{C} = \{\mathbf{o}_1, \mathbf{o}_2, \cdots, \mathbf{o}_n\}$. An undirected graph is denoted by $\mathsf{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of graph vertices and $\mathcal{E}$ is a set of graph edges. An affinity matrix of the vertices is denoted by $\mathbf{A} \in \mathbb{R}^{n \times n}$ with $a_{ij} \geq 0$, $a_{ij} = a_{ji}$ (the graph is undirected). The degree matrix $\mathbf{D}$ is a diagonal matrix associated with $\mathbf{A}$ with $d_{ii} = \sum_j a_{ij}$. A normalized affinity matrix $\mathbf{W}$ is defined as $\mathbf{D}^{-1}\mathbf{A}$. Thus, a normalized graph random-walk Laplacian matrix is $\mathbf{L} = \mathbf{I} - \mathbf{W} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{A}$ according to Meila and Shi [13]. We list common symbols used throughout this paper in Table 1.

Table 1: Common symbols used throughout the paper

Symbol	Description
$\mathbf{I}$	Identity matrix
$\mathbf{A}$	Affinity matrix
$\mathbf{W}$	Normalized affinity matrix
$\mathbf{D}$	Degree matrix
$\mathbf{L}$	Graph Laplacian matrix
$\mathbf{U}$	Eigenvector matrix
$\mathbf{\Lambda}$	Eigenvalue matrix
$\mathbf{V}$	Pseudo-eigenvector matrix
$\mathbf{E}$	Fused eigenvector matrix
$\mathbf{M}$	Demixing matrix
$I(X;Y)$	Mutual information between two random variables

2.2 Power Iteration

Although spectral clustering has gained its popularity and success in data mining and machine learning fields, its high time complexity ($\mathcal{O}(n^3)$ for computing the eigenvectors of the graph Laplacian matrix $\mathbf{L}$) limits its practical use in real-world data. To address the difficulty, Lin and Cohen [11] used truncated power iteration to find a pseudo-eigenvector on the normalized affinity matrix $\mathbf{W}$ with time complexity $\mathcal{O}(n)$, which is very efficient and attractive. Note that the k largest eigenvectors of $\mathbf{W}$ are also the k smallest eigenvectors of $\mathbf{L}$. Power Iteration (PI) is an efficient and popular method to compute the dominant eigenvector of a matrix. PI starts with a random vector $\mathbf{v}^0$ and iteratively updates as follows [11],

$$\mathbf{v}^t = \frac{\mathbf{W}\mathbf{v}^{t-1}}{\|\mathbf{W}\mathbf{v}^{t-1}\|_1} \tag{1}$$

Suppose $\mathbf{W}$ has eigenvectors $\mathbf{U} = [\mathbf{u}_1; \mathbf{u}_2; \cdots; \mathbf{u}_n]$ with eigenvalues $\mathbf{\Lambda} = [\lambda_1, \lambda_2, \cdots, \lambda_n]$, where $\lambda_1 = 1$ and $\mathbf{u}_1$ is constant. We have $\mathbf{W}\mathbf{U} = \mathbf{\Lambda}\mathbf{U}$ and in general $\mathbf{W}^t\mathbf{U} = \mathbf{\Lambda}^t\mathbf{U}$. When ignoring renormalization, Equation 1 can be written as

$$\begin{aligned} \mathbf{v}^t &= \mathbf{W}\mathbf{v}^{t-1} = \mathbf{W}^2\mathbf{v}^{t-2} = \cdots = \mathbf{W}^t\mathbf{v}^0 \\ &= c_1\mathbf{W}^t\mathbf{u}_1 + c_2\mathbf{W}^t\mathbf{u}_2 + \cdots + c_n\mathbf{W}^t\mathbf{u}_n \\ &= c_1\lambda_1^t\mathbf{u}_1 + c_2\lambda_2^t\mathbf{u}_2 + \cdots + c_n\lambda_n^t\mathbf{u}_n \end{aligned} \tag{2}$$

According to Equation 2, we have

$$\frac{\mathbf{v}^t}{c_1\lambda_1^t} = \mathbf{u}_1 + \frac{c_2}{c_1}\left(\frac{\lambda_2}{\lambda_1}\right)^t \mathbf{u}_2 + \cdots + \frac{c_n}{c_1}\left(\frac{\lambda_n}{\lambda_1}\right)^t \mathbf{u}_n \quad (3)$$

So the convergence rate of PI towards the dominant eigenvector $\mathbf{u}_1$ depends on the significant terms $\left(\frac{\lambda_i}{\lambda_1}\right)^t (2 \leqslant i \leqslant n)$. PI will finally converge to the dominant eigenvector $\mathbf{u}_1$ which is of little use in clustering. PIC [11] defines the velocity at t to be the vector $\boldsymbol{\delta}^t = \mathbf{v}^t - \mathbf{v}^{t-1}$ and defines the acceleration at t to be the vector $\boldsymbol{\epsilon}^t = \boldsymbol{\delta}^t - \boldsymbol{\delta}^{t-1}$ and stops PI when $\|\boldsymbol{\epsilon}^t\|_{max}$ is below a threshold $\hat{\epsilon}$.

2.3 ICA

Let $\mathbf{s}_1, \cdots, \mathbf{s}_m$ be m one-dimensional independent sources. Each has n i.i.d samples denoted by $\mathbf{s}_i = [s_{i1}, \cdots, s_{in}] (1 \leq i \leq m)$. Let $\mathbf{S} = [\mathbf{s}_1; \cdots; \mathbf{s}_m] \in \mathbb{R}^{m \times n}$ and we assume $\mathbf{S}$ is hidden and only a matrix $\mathbf{X}$ of mixed sources can be observed. The task of ICA is to find a demixing matrix $\mathbf{M} \in \mathbb{R}^{m \times m}$ such that $\mathbf{S} = \mathbf{MX}$ and every two components $\mathbf{s}_i$ and $\mathbf{s}_j$ $(1 \leq i, j \leq m, i \neq j)$ are statistically independent. Without loss of generality, we assume data has been whitened, which means (1) the expectation value is zero and the covariance matrix is an identity matrix ($\mathbf{I}$), (2) the demixing matrix is square, orthogonal ($\mathbf{M} \cdot \mathbf{M} = \mathbf{I}$) and full rank.

3. FULL SPECTRAL CLUSTERING

3.1 Fusing eigenvectors

For real-world data, a single pseudo-eigenvector is not enough when the number of clusters is large. The reason is we need more eigenvectors to discriminate clusters when the cluster count increases. Thus, the cluster-collision problem may happen on one-dimensional pseudo-eigenvector. However, using PI p $(p > k)$ times with random generated starting vectors to generate p pseudo-eigenvectors is not sufficient either, which can only alleviate the situation a little because of the redundant information provided by these pseudo-eigenvectors. It is just the first step of the eigenvector fusion. We also need to reduce the redundancy in these pseudo-eigenvectors. Our goal is twofold: 1) generate p pseudo-eigenvectors, 2) reduce redundancy to make the cluster-separation information stand out and suppress the noise information. The goal can also be rephrased as fusing the cluster-separation information from all original eigenvectors to improve clustering. Why do we need to fuse the information from all eigenvectors? The analysis in [14] shows that "*when confronted with clusters of different scales, corresponding to a multi-scale landscape potential, standard spectral clustering which uses the first k eigenvectors to find k clusters will fail*". Even given a locally scaled affinity matrix [23], ZP still cannot overcome the limitation if clusters have comparable densities.

For example, Figure 2(a) demonstrates the eigenvector space (consists of the eigenvectors associated with the top three minimum eigenvalues) found by ZP (other spectral clustering algorithms yield similar ones) on the running example (SYN1) in Section 1. It demonstrates that the three clusters are connected together in the eigenvector space, which is the reason for k-means' difficulty in separating them. The cluster-separation information is not provided by the first three eigenvectors. However, we can see from Figure 2(b) that the blue and red clusters have fewer overlapped data points with the black cluster. If we fuse the information from the eigenvectors in Figure 2(b) to those in Figure 2(a), we can achieve a better clustering result. In this paper, we use truncated Power Iteration (PI) to fuse eigenvectors. Figure 2 (c) shows the four pseudo-eigenvectors returned by running PI four times with randomly generated starting vectors. The resulting pseudo-eigenvectors are very similar and redundant, e.g., the pseudo-eigenvectors $\mathbf{v}_1^t$ and $\mathbf{v}_4^t$. Thus, the cluster-separation information is not standing out. Figure 2(d) gives the pseudo-eigenvector space returned by our algorithm. In such space, the blue and red clusters have much fewer close data points to the black cluster compared to those in Figure 2(a), which makes k-means easily distinguish them from each other.

Consider that each pseudo-eigenvector generated by PI is a linear combination of all eigenvectors of the normalized affinity matrix $\mathbf{W}$ and every pair of distinct pseudo-eigenvectors is redundant. One way to reduce redundancy is to make p pseudo-eigenvectors statistically independent, which can be accomplished by ICA. Mathematically speaking, the problem formulation is as follows,

Problem: Statistically Independent Pseudo-eigenvectors. *Given a pseudo-eigenvector matrix $\mathbf{V} \in \mathbb{R}^{p \times n}$ generated by running PI p times, find a demixing matrix $\mathbf{M} \in \mathbb{R}^{p \times p}$ such that $\mathbf{E} = \mathbf{MV}$ and the sum of mutual information between pairwise components of $\mathbf{E}$ is minimized, where $\mathbf{E} \in \mathbb{R}^{p \times n}$ is a resulting independent pseudo-eigenvector matrix.*

$$J_I(\mathbf{M}) := \min \sum_{1 \leq i,j \leq p, i \neq j} I(\mathbf{e}_{i\cdot}; \mathbf{e}_{j\cdot}) \quad (4)$$

The demixing matrix $\mathbf{M}$ can be derived by determining the directions of minimal entropy. To find such directions, ICA requires to whiten data, i.e., the expectation value of data is zero and the covariance matrix of data is an identity matrix, by applying PCA. The demixing matrix $\mathbf{M}$ is orthonormal in white space. After whitening data, ICA finds those directions of minimal entropy by rotating the whitened data. In this paper, instead of using fastICA [6], we use Jacobi ICA [8, 7] to find statistically independent pseudo-eigenvectors for the reasons that 1) we can choose different kinds of contrast functions, 2) we can make it escape from local optima more easily. Note that Equation 4 can be solved by iteratively optimizing every pairwise mutual information. We rewrite Equation 4 as the following objective function:

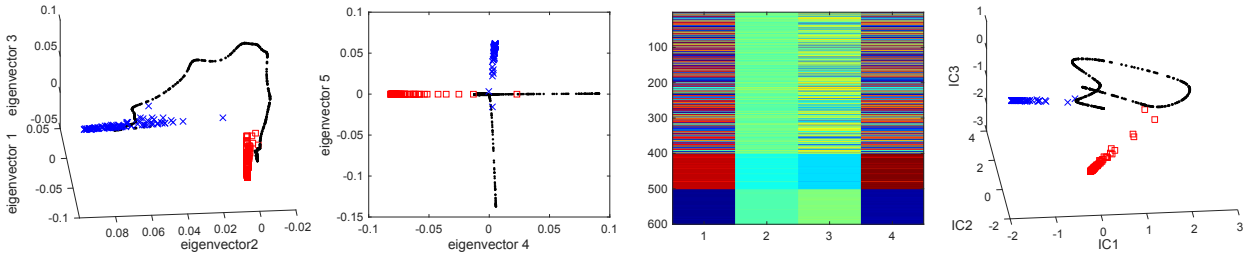
$$\begin{aligned} &\min I(\mathbf{e}_{i\cdot}; \mathbf{e}_{j\cdot}) \\ &\text{subject to } \mathbf{E} = \mathbf{MV},\ 1 \leq i, j \leq p,\ i \neq j \end{aligned} \quad (5)$$

Now it comes to how to select k independent components. Since ICA is interested in searching for non-Gaussian directions, in which negentropy is minimized. Non-Gaussian may be super-Gaussian as well as sub-Gaussian. We are only interested in sub-Gaussian directions, in which clusters are as much separated as possible. Such directions can be best modeled by uniform distributions [2]. In this paper, we use *kurtosis* to measure the distance of the probability distribution of an independent component to Gaussian distribution. The kurtosis is the fourth *standardized moment*, defined as,

$$\text{Kurt}(X) = \frac{\mu_4}{\sigma^4} = \frac{E\left((X-\mu)^4\right)}{\left(E\left((X-\mu)^2\right)\right)^2} \quad (6)$$

where μ_4 is the fourth moment of the mean and σ is the standard deviation.

The kurtosis of any univariate Gaussian distribution is 3. The kurtosis of any sub-Gaussian distribution is below 3 and the kurtosis of any sup-Gaussian distribution is above 3. We prefer the independent components associated with the top k minimum kurtosis values.



(a) **(b)** **(c)** **(d)**

Figure 2: Demonstration of the (pseudo-)eigenvector space generated by ZP and FUSE on SYN1 data. (a) the eigenvector space (consists of the first three eigenvectors) returned by ZP, (b) the eigenvector space consists of the fourth and fifth eigenvectors returned by ZP, (c) four pseudo-eigenvectors generated by running PI four times with random initial vectors, (d) the pseudo-eigenvector space returned by FUSE, in which the clusters are well separated.

3.2 Givens Rotation

The objective function in Equation 5 is difficult to solve. Inspired by Learned-Miller et. al [8] and Kirshner et. al [7] in which they used Givens rotation to estimate a demixing matrix for independent component analysis by sequentially rotating every two mixture components. The reason behind Givens rotation is: 2d-pairwise distances are not changing after rotation, thus joint distribution remains the same, whereas marginal distributions change after rotation. Thus, any metric based on joint distribution and marginal distributions varies when the rotation angle θ varies. We can find the maximal or minimal values of metrics with respect to θ. For d-dimensional data, a Givens rotation of angle θ for dimensions i and j is defined as:

$$G(i,j,\theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cos\theta & \cdots & -\sin\theta & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & \sin\theta & \cdots & \cos\theta & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

where $\sin\theta$ is on the j-th row and i-th column and $-\sin\theta$ is on the i-th row and j-th column of G.

The demixing matrix $\mathbf{M}$ can be estimated as

$$\mathbf{M} = \prod G(i,j,\theta^*), 1 \le i,j \le p, i \ne j \qquad (7)$$

where $G(i,j,\theta^*)$ is a Givens rotation of the best angle θ^* for optimizing the mutual information of the dimensions i and j of a data matrix.

4. ALGORITHM

4.1 Greedy Search

Learned-Miller et. al [8] and Kirshner et. al [7] optimized Equation 5 by exhaustively search over $K = 150$ values of θ in the range $\left[0, \frac{\pi}{2}\right]$ which is time-consuming. Besides, they did not discriminate the order of optimization for different pairwise dimensions, which results in getting easily trapped in local optima. Considering the two limitations, we propose a new optimization method which is very efficient and effective.

To speed up the search process, we do not exhaustively search over $K = 150$ values of θ in the range $\left[0, \frac{\pi}{2}\right]$. In contrast, we use the history search as anchors to guide the search process. From this perspective, the greedy search is a self-learning method based on its learned knowledge. The strategy of adjusting the search resolution (see the following) makes our greedy search self-adaptive. Note that we only need to consider θ in the interval $\left[0, \frac{\pi}{2}\right]$ because the effectiveness of any 90 degree rotation is equivalent as explained in [8]. In this paper, we adopt kernel generalized variance (KGV) proposed by Bach and Jordan [1] to estimate pairwise mutual information considering its linear complexity and especially its smoothness w.r.t. log function. For a detailed description, please see [1, 17]. Now we give an example to demonstrate our greedy search method. Because in practical use we can choose different contrast functions, here we just give a generalized function f to demonstrate the main idea. The curve in Figure 3 (a) is a function of θ. The goal is to find the best θ^* achieving the maximal objective function f in the interval $\left[0, \frac{\pi}{2}\right]$ (the minimal f can be achieved by finding the maximal $-f$). Our method is described as follows:

Case 1: As depicted in Figure 3 (a), we set the ascending and descending step size to $\frac{\pi}{2K}$. We choose three different initial θ (in our example is θ_1, θ_2 and θ_3) with the same interval (e.g. $\frac{\pi}{2K}$) and compute their objective function ($f(\theta_1)$, $f(\theta_2)$ and $f(\theta_3)$). If $f(\theta_3) \le f(\theta_2) \le f(\theta_1)$, we assume that the function is continuously decreasing and we multiply the descending step size by $\tau = 2$ (the search resolution) and update θ_3 for the following iteration. If τ is two large, the method may skip some important search niche, while if τ is too small, the efficiency will be decreased. We update $\theta_1 = \theta_2$, $\theta_2 = \theta_3$, $f(\theta_1) = f(\theta_2)$ and $f(\theta_2) = f(\theta_3)$ as history reference values for the following search.

Case 2: We compute $f(\theta_3)$ as depicted in Figure 3 (b). If $f(\theta_2) \le f(\theta_3)$ and $f(\theta_2) \le f(\theta_1)$, we assume the function is continuously increasing. We set the descending step size to its initial value and multiply the ascending step size by τ and update θ_3. Also, we update $\theta_1 = \theta_2$, $\theta_2 = \theta_3$, $f(\theta_1) = f(\theta_2)$ and $f(\theta_2) = f(\theta_3)$ as history reference values for the following search. If $f(\theta_3) \le f(\theta_2) \le f(\theta_1)$, go to case 1.

Case 3: We compute $f(\theta_3)$ as depicted in Figure 3 (c). If $f(\theta_1) \le f(\theta_2) \le f(\theta_3)$, we assume the function is continuously increasing. We set the descending step size to its initial value and multiply the ascending step size by τ and update θ_3. Also, we update $\theta_1 = \theta_2$, $\theta_2 = \theta_3$, $f(\theta_1) = f(\theta_2)$ and $f(\theta_2) = f(\theta_3)$ as history reference values for the following search. if not, go to case 4.

Case 4: In this case (Figure 3 (d)), $f(\theta_1) \le f(\theta_2)$ and $f(\theta_3) \le f(\theta_2)$, we assume there may be some peaks in the interval. We exhaustively search in the interval $[\theta_1, \theta_3]$ with a step size $\frac{\pi}{2K}$. Finally, we update $\theta_1 = \theta_2$, $\theta_2 = \theta_3$, $f(\theta_1) = f(\theta_2)$, $f(\theta_2) = f(\theta_3)$ and set the ascending step size to its initial value. Since now $f(\theta_2) \le f(\theta_1)$, we assume the function is continuously decreas-

ing. We multiply the descending step size by τ and update θ_3. If $f(\theta_3) \leq f(\theta_2)$, go to case 1; if $f(\theta_3) \geq f(\theta_2)$, go to case 2.

We repeat the above four cases until $\theta_3 \geq \frac{\pi}{2}$. Note that, in each case, we update the best objective function value f_b and θ^*.

4.2 FUSE

As said before, in [8] and [7], the authors do not differ between the order of optimizing pairwise dimensions which results in the algorithm's getting easily trapped in local optima. In this paper, we use a greedy selection method, i.e., computing the objective function values for each pairwise dimensions and then optimizing pairs according to their objective function values from the worst to the best, to make our method not easily get trapped in local optima. Our pseudo-code is given in Algorithm 1.

Steps 1 – 2 initialize the demixing matrix $\mathbf{M}$ to an identity matrix, compute the affinity matrix $\mathbf{A}$ and normalize it to $\mathbf{W}$. Steps 3 – 8 generate $p = k + 1$ pseudo-eigenvectors using power iteration (PI). In step 4, the starting vector is randomly generated following a Gaussian distribution with mean 0 and variance 1. In Steps 5 – 8, each starting vector is iteratively updated by power iteration until the acceleration threshold $\hat{\epsilon}$ or the maximum iteration number is reached. Step 10 whitens the pseudo-eigenvector matrix $\mathbf{V}$ to let it have zero expectation value and an identity covariance matrix. In step 11, $\mathbf{c}$ includes the indices of each pairwise components in $\mathbf{E}$ and their mutual information values ($a_i \in \{1, 2, \cdots, p\}$). To let the algorithm escape from local optima, step 14 sorts $\mathbf{c}$ in descending order w.r.t. mutual information values. Steps 15 – 19 use the greedy search to find the best θ^* for each pairwise components of $\mathbf{E}$ to make them statistically independent and update components' pairwise mutual information value stored in c_{j3}, and also update $\mathbf{E}$ and $\mathbf{M}$. We set the mutual information threshold to 0.1 for a balance between the efficiency and effectiveness. If we set it lower, the efficiency will be decreased but effectiveness will be increased and vice versa. Step 20 returns the selected independent components which will be fed to k-means to find clusters.

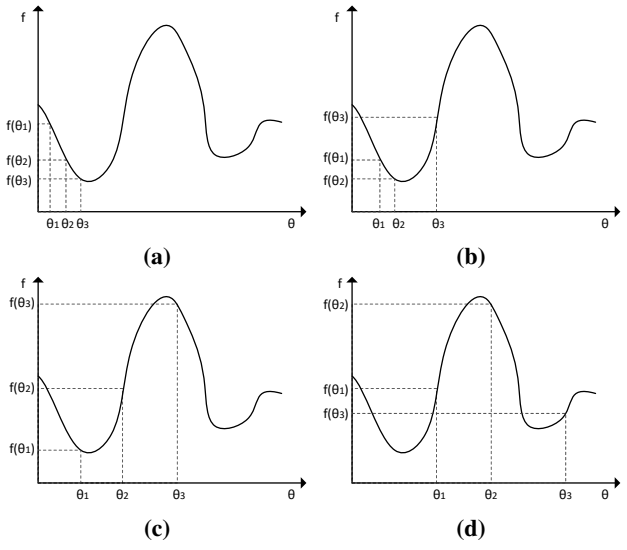


Figure 3: Greedy search strategy

4.3 Complexity Analysis

We omit the runtime for computing the affinity matrix which is a common step in all spectral clustering methods. The analysis of runtime complexity of FUSE is as follows: In steps 3 – 8, generating one pseudo-eigenvector costs $\mathcal{O}(n)$ time [12], and thus the runtime complexity to generate $p = k + 1$ pseudo-eigenvectors by power iteration is $\mathcal{O}((k+1)n)$. In step 10, as a preprocessing step, whitening data costs $\mathcal{O}((k+1)^2 n)$ time. In this paper, we adopt Kernel Generalized Variance (KGV) using incomplete Cholesky decomposition proposed in [1] to estimate mutual information. The complexity of KGV is $\mathcal{O}(m^2 M^2 n)$[1], where m is data dimension and M is the maximal rank considered by the low-rank decomposition algorithms for the kernels. In step 11, the computation time for all pairwise mutual information values is $\frac{k(k+1)}{2} \cdot \mathcal{O}(2^2 M_1^2 n) = \mathcal{O}(k^2 M_1^2 n)$. To make FUSE escape from local optimal, we sort $\mathbf{c}$ using quick sort algorithm. The runtime complexity of the ordering process is $3(k+1) \cdot \mathcal{O}(l \log l) = 3(k+1) \cdot \mathcal{O}(\frac{k(k+1)}{2} \log \frac{k(k+1)}{2}) = \mathcal{O}(k^4 \log k)$. The time cost of finding independent pseudo-eigenvectors is $3(k+1) \cdot \frac{k(k+1)}{2} \cdot K \cdot \mathcal{O}(2^2 M_2^2 n) = \mathcal{O}(k^3 M_2^2 n)$. Finally, we use k-means to cluster on the selected independent pseudo-eigenvectors. The total runtime complexity of FUSE is $\mathcal{O}\left(k^2(M_1^2 n + knM_2^2 + k^2 \log k)\right)$ plus the time complexity of k-means, i.e., $\mathcal{O}(nk) \times (\#\ k\text{-means iterations})$ [3].

Algorithm 1: FUSE

Input: Data $\mathbf{X} \in \mathbb{R}^{m \times n}$
Output: cluster indicator vectors

1 $T \leftarrow 1000$, *levels* $\leftarrow 3$, *sweeps* $\leftarrow p$; /* $p = k + 1$ */
2 Initialize $\mathbf{M} \in \mathbb{R}^{p \times p}$ to an identity matrix and compute the normalized affinity matrix $\mathbf{W} \in \mathbb{R}^{n \times n}$;
3 **for** $j \leftarrow 1$ **to** p **do**
4 $\quad t \leftarrow 0, \mathbf{v}_j^0 \leftarrow$ **randn** $(1, n)$; /* $\mathbf{v}_j \in \mathbb{R}^{1 \times n}$ */
$\quad$ /* power iteration */
5 $\quad$ **repeat**
6 $\quad\quad \mathbf{v}_j^{t+1} \leftarrow \frac{\mathbf{W}\mathbf{v}_j^t}{\|\mathbf{W}\mathbf{v}_j^t\|_1}$;
7 $\quad\quad \boldsymbol{\delta}^{t+1} \leftarrow |\mathbf{v}_j^{t+1} - \mathbf{v}_j^t|$;
8 $\quad\quad t \leftarrow t + 1$;
9 $\quad$ **until** $\|\boldsymbol{\delta}_j^{t+1} - \boldsymbol{\delta}_j^t\|_{\max} \leq \hat{\epsilon}$ **or** $t \geq T$;
10 $\mathbf{V} = [\mathbf{v}_1; \ldots; \mathbf{v}_p]$;
11 $\mathbf{V} \leftarrow$ **whiten** $(\mathbf{V})$,$\mathbf{E} \leftarrow \mathbf{M}\mathbf{V}$;
/* $\mathbf{c}$ has $l = \binom{p}{2}$ tuples */
12 $\mathbf{c} \leftarrow ((a_1, a_2, I_1 = I(\mathbf{v}_{a_1 \cdot}; \mathbf{v}_{a_2 \cdot})), \ldots, (a_{l-1}, a_l, I_l = I(\mathbf{v}_{a_{l-1} \cdot}; \mathbf{v}_{a_l \cdot})))$;
13 **for** *level* $\leftarrow 1$ **to** *levels* **do**
14 $\quad$ **for** *sweep* $\leftarrow 1$ **to** *sweeps* **do**
15 $\quad\quad \mathbf{c} \leftarrow$ **order_descending_by_I_value** $(\mathbf{c})$;
$\quad\quad$ /* minimize pairwise mutual information */
16 $\quad\quad$ **for** $j \leftarrow 1$ **to** l **do**
17 $\quad\quad\quad$ **if** $c_{j3} > 0.1$ **then**
18 $\quad\quad\quad\quad [\theta^*, c_{j3}] \leftarrow$ **greedy_search** $(c_{j1}, c_{j2}, \mathbf{E})$;
19 $\quad\quad\quad\quad \mathbf{E} \leftarrow G(c_{j1}, c_{j2}, \theta^*)\mathbf{E}$;
20 $\quad\quad\quad\quad \mathbf{M} \leftarrow G(c_{j1}, c_{j2}, \theta^*)\mathbf{M}$;
21 compute kurtosis of each pseudo-eigenvector in $\mathbf{E}$ and return the pseudo-eigenvectors associated with the top k minimum values;

5. EXPERIMENTAL EVALUATION

Competitors: To evaluate the performance of FUSE, we adopt three spectral clustering methods NCut [16], NJW [15] and ZP

[23] and power-iteration-based clustering methods PIC [11], PIC-k [10], DPIC [18] and DPIE [5] as competitors. FUSE and all the comparison methods are written in Matlab. All experiments are run on the same machine with an Intel Core Quad i7-3770 with 3.4 GHz and 32 GB RAM. The code of FUSE and all synthetic and real-world data used in this paper are available at the website[1].

Parameters: The parameters for spectral clustering, power-iteration-based clustering methods are set according to their original papers. For FUSE, we set $\hat{\epsilon}_i = i \cdot \lceil \log(k) \rceil \cdot \frac{1e-5}{n}$ as adopted by DPIE [5]. For text data, we use cosine similarity ($\frac{\mathbf{x}_i \cdot \mathbf{x}_j}{\|\mathbf{x}_i\|_2 \|\mathbf{x}_j\|_2}$) to compute the affinity matrix. For network data, the element a_{ij} of the affinity matrix $\mathbf{A}$ is simply 1 if blog i has a link to j or vice versa, otherwise $a_{ij} = 0$. For all other data, the locally scaled affinity matrix is computed as the way proposed in [23] with $K_{NN} = 7$. The original ZP method automatically chooses the number of clusters. For a fair comparison, we give ZP the correct number of clusters.

Since all the comparison algorithms use k-means in the last step, in each experiment k-means is run 100 times with random starting points and the most frequent cluster assignment is used [11]. We run each experiment 50 times and report the mean and standard deviation of Adjusted Mutual Information (AMI) [19]. For AMI, higher value means better clustering.

5.1 Synthetic Data

5.1.1 Quality

Synthetic dataset SYN1 is the running example used in Section 1. SYN1 has three clusters. Each of the two Gaussian clusters has 100 data points and the stripe cluster has 400 data points. We have shown the results before.

Synthetic dataset SYN2 has three clusters as well depicted in Figure 4(a). Both Gaussian clusters have 100 data points and the rectangular cluster has 400 data points. Both Gaussian clusters have some very close data points to the rectangular cluster making them hard to be separated correctly. The mean AMI of FUSE is 0.750 and the highest of the comparison algorithms' is 0.574 achieved by PIC-k. ZP only has a value 0.483. Figure 4(b)– (d) give us an intuitive demonstration. FUSE just wrongly clustered a few data points in the magenta rectangular cluster to the blue Gaussian cluster, while ZP and PIC-k wrongly clustered about a half of the data points in the magenta rectangular cluster to the blue Gaussian cluster. Our algorithm is superior to the competing algorithms.

SYN3 in Figure 5(a) also has three clusters. Two Gaussian clusters each has 90 and 92 data points, respectively. The blue ring cluster has 130 data points. SYN3 is very interesting because the density of the blue ring cluster is lower than those of the Gaussian clusters. And the ring cluster is very close to the Gaussian clusters, which could make the $K_{NN} = 7$ neighbors of some points in the blue ring cluster be belonging to the Gaussian clusters. SYN3 is also difficult to cluster correctly. However, FUSE achieved the best compared to all the comparison algorithms. PIC-k even clustered several data points belonging to two Gaussian clusters to the ring cluster, which did not make sense.

SYN4 in Figure 6(a) contains five clusters, each of the two Gaussian clusters has 100 data points, each of the two square clusters has 82 and 100 data points respectively and the ring cluster has 56 data points. The ring cluster is very close to the square clusters and even has some overlap with the two Gaussian clusters. Still, FUSE achieved the best result, only not distinguishing between the overlapped data points from the Gaussian clusters and the ring cluster. ZP wrongly detected a half of the data points in the ring cluster to the green Gaussian cluster. PIC-k wrongly clustered several data points in the ring cluster to the black square cluster although the densities of these two clusters are significantly different.

For all these multi-scale synthetic datasets, our algorithm FUSE outperforms all the competing algorithms. FUSE is even superior to the spectral clustering algorithms ZP, NCut and NJW, which proves that the *normalized cut* criterion is not alway suitable for clustering. FUSE-E is our algorithm using the exhaustive search strategy over θ proposed in [8, 7] which does not discriminate the order of optimization of the pseudo-eigenvectors generated by PI. We can see that sometimes it gets trapped in local optimal (the result on SYN3). Our algorithm FUSE adopting the greedy search strategy achieves quite similar or even better results than using the exhaustive search strategy.

5.1.2 Scalability

In this experiment, we want to test the runtime against the number of data points using data SYNC5. Synthetic dataset SYNC5 is generated as follows: Firstly, we generate two 2D clusters sampled from uniform distributions with the number of data points 4,000 and 1,000 respectively. The two clusters are our basis clusters. Then at each step we increase the data points in each cluster by the size of its basis. Finally, we have data points varying from 5,000 to 30,000 by a step size 5,000. We feed each algorithm with the same affinity matrix. Thus, the runtime does not include the computation time for the affinity matrix. The results are demonstrated in Figure 7. Since the runtime of NJW and NCut are similar, here we only show the runtime of NCut for a clearer demonstration. Figure 7 shows that the runtime of FUSE becomes much lower than that of ZP, NCut and DPIC when increasing the number of data points. Compared to PIC, we can see their slope variances are quite similar. The runtime difference between FUSE and PIC is owing to that FUSE needs to determine the directions in which the entropy of pseudo-eigenvectors is minimized. Compared with PIC-k, the runtime of FUSE becomes close to that of PIC-k when the number of data points increases to 30,000. Note that FUSE-E is our algorithm using the exhaustive search strategy. FUSE is faster than FUSE-E as can be seen from the figure. Our algorithm is of more practical use than ZP, NCut, NJW and DPIC.

Table 2: AMI on Synthetic Data (mean $\pm$ standard deviation)

AMI	SYNC1	SYNC2	SYNC3	SYNC4
FUSE	0.715±0.131	**0.750±0.127**	**0.702±0.048**	**0.891±0.016**
FUSE-E	**0.735±0.142**	**0.750±0.120**	0.688±0.044	0.886±0.018
ZP	0.374±0	0.483±0	0.528±0	0.882±0
NCUT	0.370±0	0.479±0	0.522±0	0.874±0.002
NJW	0.379±0	0.451±0	0.533±0	0.879±0.002
PIC	0.355±0.088	0.309±0	0.494±0.059	0.840±0.034
PIC-k	0.324±0.048	0.574±0.105	0.508±0.055	0.8544±0.022
DPIC	0.324±0.094	0.465±0.113	0.499±0.107	0.482±0.064
DPIE	0.350±0.056	0.128±0.097	0.310±0	0.630±0

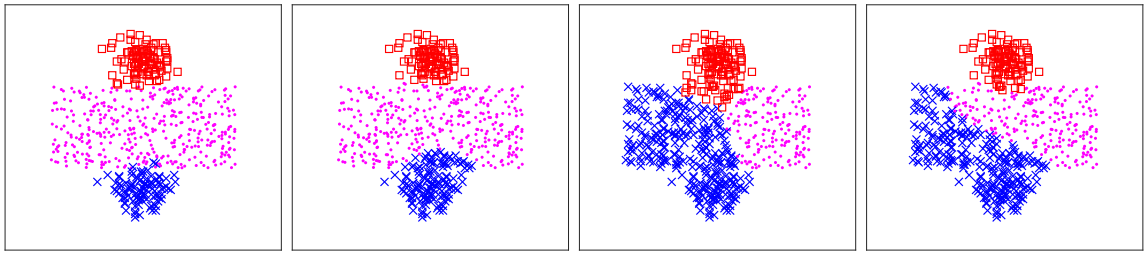
5.2 Real-world Data

5.2.1 Clustering

Now we demonstrate the effectiveness of our FUSE on seven real-world datasets. PENDIGITS is available from UCI machine learning repository[2]. The original datasets MNIST, 20NEWSGROUPS, REUTERS21578, TDT2 and RCV1 are available at this

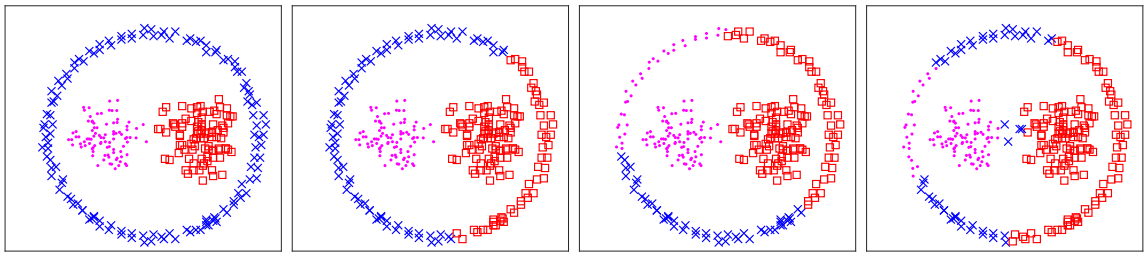
[1] https://github.com/yeweiysh/FUSE

[2] http://archive.ics.uci.edu/ml/



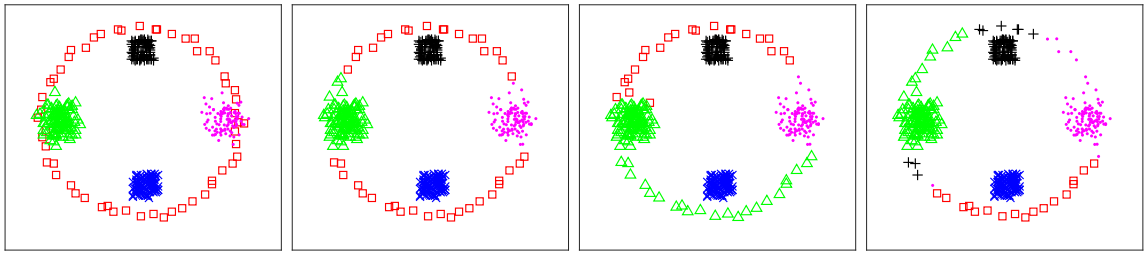
(a) SYN2 **(b)** FUSE **(c)** ZP **(d)** PIC-k

Figure 4: Clustering results on SYNC2 (shown are the most frequent clusterings)



(a) SYN3 **(b)** FUSE **(c)** NJW **(d)** PIC-k

Figure 5: Clustering results on SYNC3 (shown are the most frequent clusterings)



(a) SYN4 **(b)** FUSE **(c)** ZP **(d)** PIC-k

Figure 6: Clustering results on SYNC4 (shown are the most frequent clusterings)

website[3]. 20NGD from [11] is a subset of 20NEWSGROUPS, and MNIST0127 from [18] is a subset of MNIST. TDT2_3CLASSES, REUTERS_4CLASSES and RCV1_4CLASSES are samples from original REUTERS21578, TDT2 and RCV1 corpus using random indices from the website[3]. AGBLOG is a connected network dataset of 1222 liberal and conservative political blogs mined from blog homepages [11]. For text datasets, we use the preprocessed document-term matrix to compute the TF-IDF matrix. Then each feature vector is normalized to have unity norm. Finally, we use cosine similarity to compute the affinity matrix. The statistics of all datasets are given in Table 3.

From Table 4, we can see that on all datasets, FUSE achieves the best results, even outperforms self-tuning spectral clustering algorithm (ZP) and the conventional spectral clustering algorithms (NCut and NJW). Compared to PIC and PIC-k, FUSE improves AMI on each dataset. The most likely reason is the pseudo-eigenvectors found by FUSE are statistically independent (non-redundant), which make every cluster stand out in each pseudo-eigenvector. Compared to DPIC and DPIE which also aim at reducing redundancy in pseudo-eigenvectors generated by PI, FUSE improves AMI much on most datasets. One reason is that finding directions in which the entropy is minimized is much more beneficial to clustering.

[3]http://www.cad.zju.edu.cn/home/dengcai/Data/data.html

Table 3: Statistics of Datasets

Dataset	#instances	#features	#clusters
PENDIGITS	7494	16	10
MNIST0127	4189	784	4
AGBLOG	1222	498	2
20NGD	800	26214	4
TDT2_3CLASSES	314	36761	3
REUTERS_4CLASSES	649	18933	4
RCV1_4CLASSES	1000	29985	4

Two most interesting results are on AGBLOG and TDT2_3CLASSES datasets. All comparison methods except PIC and PIC-k fail on AGBLOG dataset. AGBLOG dataset has two balanced clusters with the number of instances 586 and 636, respectively. We show the most frequent data embeddings of FUSE and ZP (the results of NJW and NCut are very similar) in Figure 8(a), (b) and (c). We can see that most data points (blue ones in Figure 8(b) and (c)) are assigned to one cluster, which

makes the results of ZP and PIC-k not appealing. However, our algorithm finds the embedding space in which two clusters are separated evenly.

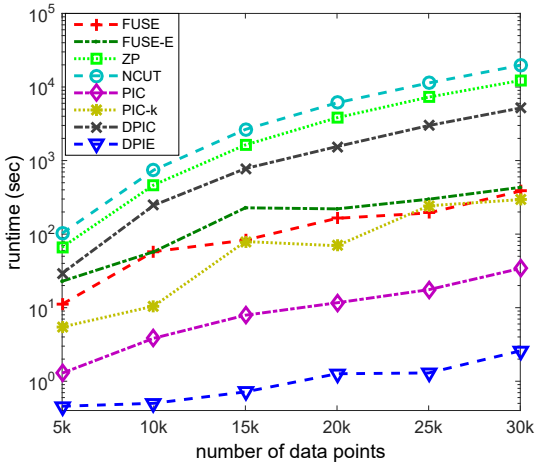


Figure 7: Runtime comparison

Figure 9 shows the clustering results on the TDT2_3CLASSES dataset. Figure 9(b) demonstrates the eigenvector space found by ZP, in which the red square cluster and the dot magenta cluster are connected together. ZP only achieves 0.673 in terms of AMI. PIC-k found two pseudo-eigenvectors. Also in its found embedding space, two clusters (blue and magenta) are not well separated. However, in the embedding space detected by our algorithm, three clusters are well separated, which makes the value of AMI much higher than those of the competing methods as can be seen in Table 4. Thus, the embedding space found by our algorithm is much more attractive and effective.

If we look into Table 5, we can find that we have six datasets on which the runtime of our method is much lower than that of NCUT and NJW. And we also have four datasets on which our method is faster than ZP. Our method is very efficient and promising for practical use. However, on PENDIGITS dataset, FUSE is slower than the conventional spectral clustering algorithms because the maximal rank M considered by KGV is close to n which costs much time to compute the pairwise mutual information.

5.2.2 *Image Segmentation*

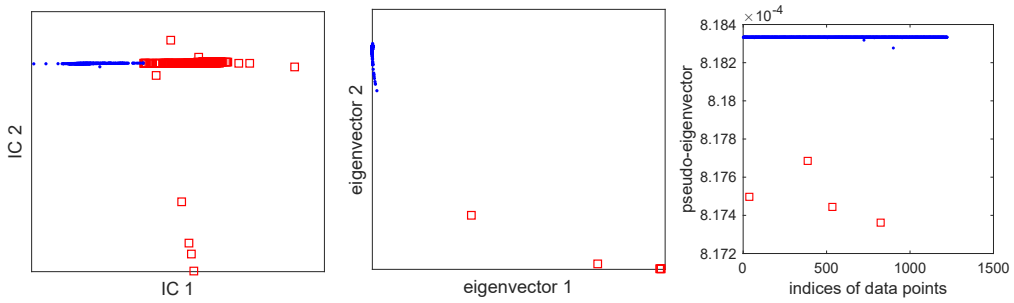
In this section, we apply our algorithm on image segmentation. Figure 10 shows two examples from the Berkeley Segmentation Dataset and Benchmark [4]. Each pixel is represented as a five dimensional vector of its pixel coordinates x and y, and the color intensities [3]. We set the number of clusters in Figure 10(a) three and four for Figure 10(e). Since the results returned by ZP, NCut and NJW are very similar, we only show the results of ZP here. For other methods, we show such methods whose results are more interpretable. Figure 10(b) shows that FUSE separates the deer, the grass and the forest very well. ZP correctly segments the grass, but does not distinguish the deer from the forest, while PIC-k recognizes the deer but not the grass or the forest. Compared to Figure 10(a), Figure 10(e) demonstrates a more challenging task because the two elephants are very similar in terms of the color and position. However, our method FUSE successfully distinguishes these two similar elephants and also recognizes the sky. ZP cannot separate the two elephants, but the sky is well segmented. DPIE can also distinguish the two elephants but the segmentation is worse than FUSE's. In addition, DPIE does not segment the sky well.

[4]https://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/BSDS300/html/dataset/images.html

6. RELATED WORK

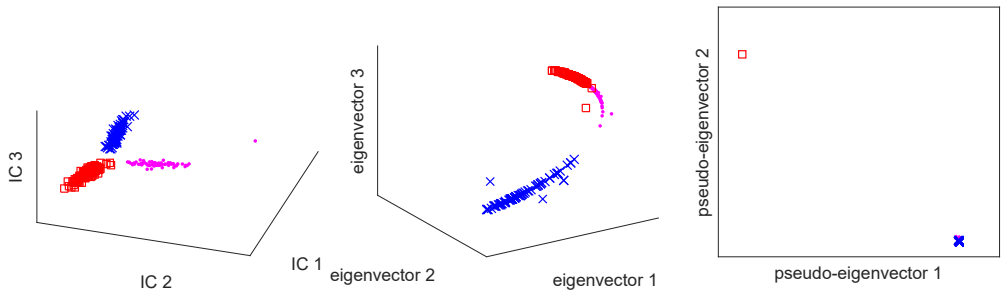
Spectral clustering. Spectral clustering is very popular in data ming owing to its ability to detect arbitrary-shape clusters in data spectrum space. Spectral clustering can be divided into three categories by the type of Laplacian matrix, i.e., unnormalized spectral clustering, normalized spectral clustering proposed by Shi and Malik (NCut) [16] and another normalized spectral clustering proposed by Ng, Jordan and Weiss (NJW) [15]. After deciding the type of Laplacian matrix, it computes the first k eigenvectors of the Laplacian matrix and then uses k-means to cluster in the space formed by these eigenvectors. Spectral clustering is very elegant. However, the computation cost is very high for large-scale data. Finding eigenvectors takes $\mathcal{O}(n^3)$ in general. Recently, researchers have proposed many fast approximating techniques, such as IRAM and sampling techniques [4, 22]. Spectral clustering assumes that the "informative" eigenvectors are those associated with the smallest k eigenvalues, which seems not to be successful on some real-world data with much noise or multi-scale density. And this promotes many researchers work on how to select much informative eigenvectors, and how to estimate the local scale of data with varying densities, shapes and levels of noise [23, 9, 21, 3]. ZP [23] is a representative of all these algorithms. ZP takes local scaling into consideration and constructs a locally scaled affinity matrix which proves beneficial to clustering especially for multi-scale data or data with irregular background clutter. ZP also exploits the structure of eigenvectors to improve clustering. As NCut, NJW and other spectral-based clustering methods, ZP only uses the first k eigenvectors to cluster, which is not appropriate in some cases. However, our algorithm FUSE exploits all "informative" eigenvectors and fuses all their information to accomplish better clustering.

Power-iteration-based clustering. Power Iteration Clustering (PIC) [11] uses truncated power iteration on a normalized affinity matrix of the data points to find a very low-dimensional data embedding which is a linear combination of the major eigenvectors for clustering. It is very elegant and efficient. However, the assumptions it bases on are very strict and it returns only one pseudo-eigenvector which prevents its performance on data with large number of clusters, where cluster-collision problem is easy to happen. PIC-k [10] has been proposed to alleviate the situation but actually it still cannot solve the cluster-collision problem due to much similarity exists in the returned pseudo-eigenvectors. Another clustering algorithm based on power iteration is Deflation Power Iteration Clustering (DPIC)[18] which uses Schur complement deflation to generate multiple orthogonal pseudo-eigenvectors. However, the pseudo-eigenvectors still contain noise together with cluster-separation information. Diverse Power Iteration Clustering (DPIE) [5] normalizes the residue (regression) error which is obtained by subtracting the effects of the already-found DPIEs from the embeddings returned by PIC. However, DPIE cannot guarantee to find diverse embeddings in every iteration and it bases on the assumption that clear eigen-gap exists between every two successive eigenvalues which is also very strict. Our method FUSE does not make any assumptions and finds statistically independent pseudo-eigenvectors, each of which is a different linear combination of the original eigenvectors. Besides, each statistically independent pseudo-eigenvector eliminates noise and only keeps cluster-separation information which makes FUSE much more advanced and effective.



(a) FUSE **(b)** ZP **(c)** PIC-k

Figure 8: The embedding space found by FUSE, ZP and PIC-k on AGBLOG data.



(a) FUSE **(b)** ZP **(c)** PIC-k

Figure 9: The embedding space found by FUSE, ZP and PIC-k on TDT2_3CLASSES data.

Table 4: AMI on Real-world Data (mean ± standard deviation)

AMI	PENDIGITS	MNIST0127	AGBLOG	20NGD	TDT2_3CLASSES	REUTERS_4CLASSES	RCV1_4CLASSES
FUSE	**0.828±0.009**	**0.594±0.043**	**0.729±0.001**	**0.348±0.017**	**0.951±0.005**	**0.593±0.031**	**0.493±0.015**
ZP	0.813±0	0.444±0	0.017±0	0.293±0	0.673±0	0.585±0.02	0.452±0
NCUT	0.800±0	0.418±0	0.002±0	0.325±0.022	0.670±0	0.539±0.004	0.405±0
NJW	0.800±0	0.496±0.040	0.017±0	0.265±0	0.670±0	0.567±0.005	0.402±0
PIC	0.680±0	0.446±0.034	0.226±0.317	0.318±0.004	0.308±0.287	0.310±0	0.335±0.034
PIC-k	0.773±0.024	0.456±0.006	0.227±0.288	0.284±0.051	0.400±0.300	0.366±0.069	0.351±0.021
DPIC	0.616±0.044	0.331±0.006	0.330±0	0.046±0.028	0.606±0.019	0.234±0.028	0.150±0.044
DPIE	0.624±0.034	0.011±0.003	0.050±0	0.271±0.046	0.135±0.197	0.434±0.198	0.404±0.043

Table 5: Runtime (sec) on Real-world Data (mean ± standard deviation)

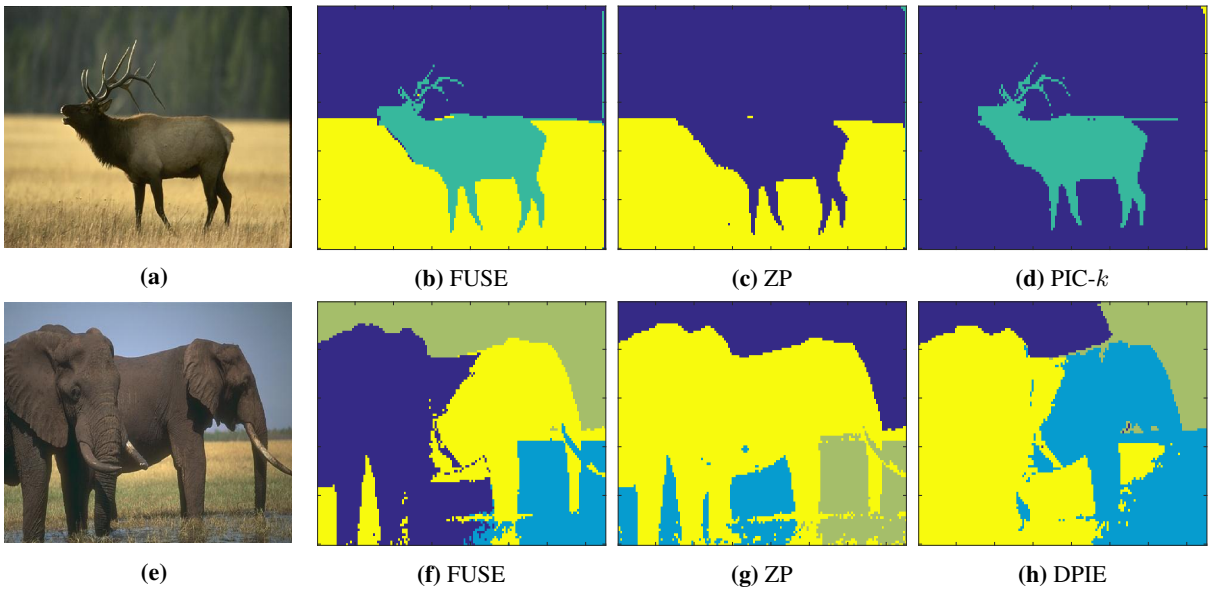
Runtime	PENDIGITS	MNIST0127	AGBLOG	20NGD	TDT2_3CLASSES	REUTERS_4CLASSES	RCV1_4CLASSES
FUSE	90.644±13.336	3.831±0.890	0.325±0.074	0.474±0.132	0.826±0.376	0.394±0.201	0.391±0.148
ZP	50.250±3.105	41.188±0	0.781±0	0.271±0	0.032±0	0.438±0	52.714±4.310
NCUT	50.375±2.066	62.697±0	2.318±0	0.957±0	51.429±3.824	51±3.512	52.143±3.891
NJW	51±3.406	59.940±0	0.374±0	0.807±0	50.286±3.302	51±3.873	52.143±4.100
PIC	5.197±0.017	0.249±0.031	0.063±0.010	0.013±0.003	0.035±0.042	0.010±0.006	0.003±0
PIC-k	13.745±5.020	0.263±0.036	0.035±0.006	0.002±0.002	0.005±0.015	0.002±0	0.003±0
DPIC	299.556±30.740	20.270±1.933	0.521±0.047	0.468±0.078	0.099±0.078	0.231±0.042	0.502±0.074
DPIE	0.099±0.520	1.816±3.822	0.003±0.002	0.097±0.009	0.183±0.019	0.104±0.015	3.287±0.041

7. CONCLUSION

We have proposed FUSE to handle multi-scale data on which the *normalized cut* criterion tends to fail even given a suitable locally scaled affinity matrix. FUSE exploits PI and ICA to fuse all "informative" eigenvectors to yield better clustering. Since the pseudo-eigenvectors fused by PI are redundant and the cluster-separation information does not stand out, ICA is adopted to reduce the redundancy. Then, a *kurtosis*-based selection strategy is used to improve cluster separation. To speed up the search process, we have developed a greedy search method which learns from its history search records and also adaptively adjusts its search resolution. Extensive experiments and evaluations on various synthetic and real-world data show FUSE's promising in dealing with multi-scale data. Future work would go to explore how to adaptively select the number of pseudo-eigenvectors for different datasets.

Acknowledgement

Warm thanks to Hao Huang for his DPIE code and anonymous reviewers. This work has been supported by the China Scholarship Council (CSC).



(a) **(b)** FUSE **(c)** ZP **(d)** PIC-k

(e) **(f)** FUSE **(g)** ZP **(h)** DPIE

Figure 10: Image segmentation (shown are the most frequent clusterings of each method)

8. REFERENCES

[1] F. R. Bach and M. I. Jordan. Kernel independent component analysis. *The Journal of Machine Learning Research*, 3:1–48, 2003.

[2] C. Böhm, C. Faloutsos, and C. Plant. Outlier-robust clustering using independent components. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 185–198. ACM, 2008.

[3] C. D. Correa and P. Lindstrom. Locally-scaled spectral clustering using empty region graphs. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1330–1338. ACM, 2012.

[4] C. Fowlkes, S. Belongie, F. Chung, and J. Malik. Spectral grouping using the nystrom method. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(2):214–225, 2004.

[5] H. Huang, S. Yoo, D. Yu, and H. Qin. Diverse power iteration embeddings and its applications. In *Data Mining (ICDM), 2014 IEEE International Conference on*, pages 200–209. IEEE, 2014.

[6] A. Hyvärinen. Fast and robust fixed-point algorithms for independent component analysis. *Neural Networks, IEEE Transactions on*, 10(3):626–634, 1999.

[7] S. Kirshner and B. Póczos. Ica and isa using schweizer-wolff measure of dependence. In *Proceedings of the 25th international conference on Machine learning*, pages 464–471. ACM, 2008.

[8] E. G. Learned-Miller et al. Ica using spacings estimates of entropy. *The Journal of Machine Learning Research*, 4:1271–1295, 2003.

[9] Z. Li, J. Liu, S. Chen, and X. Tang. Noise robust spectral clustering. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[10] F. Lin. *Scalable methods for graph-based unsupervised and semi-supervised learning*. PhD thesis, Carnegie Mellon University, 2012.

[11] F. Lin and W. W. Cohen. Power iteration clustering. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 655–662, 2010.

[12] F. Lin and W. W. Cohen. A very fast method for clustering big text datasets. In *ECAI*, pages 303–308, 2010.

[13] M. Meila and J. Shi. A random walks view of spectral segmentation. 2001.

[14] B. Nadler and M. Galun. Fundamental limitations of spectral clustering. In *Advances in Neural Information Processing Systems*, pages 1017–1024, 2006.

[15] A. Y. Ng, M. I. Jordan, Y. Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.

[16] J. Shi and J. Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888–905, 2000.

[17] Z. Szabó, B. Póczos, and A. Lőrincz. Undercomplete blind subspace deconvolution. *The Journal of Machine Learning Research*, 8:1063–1095, 2007.

[18] N. D. Thang, Y.-K. Lee, S. Lee, et al. Deflation-based power iteration clustering. *Applied intelligence*, 39(2):367–385, 2013.

[19] N. X. Vinh, J. Epps, and J. Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *The Journal of Machine Learning Research*, 11:2837–2854, 2010.

[20] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[21] T. Xiang and S. Gong. Spectral clustering with eigenvector selection. *Pattern Recognition*, 41(3):1012–1029, 2008.

[22] D. Yan, L. Huang, and M. I. Jordan. Fast approximate spectral clustering. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 907–916. ACM, 2009.

[23] L. Zelnik-Manor and P. Perona. Self-tuning spectral clustering. In *Advances in neural information processing systems*, pages 1601–1608, 2004.