Float Greedy-search-based Subspace Clustering

Lingxiao Song[1,2] Man Zhang[1,2] Qi Li[1,2] Zhenan Sun[1,2,3] Ran He[1,2,3]
[1]Center for Research on Intelligent Perception and Computing, CASIA
[2]National Laboratory of Pattern Recognition, CASIA
[3]Center for Excellence in Brain Science and Intelligence Technology, CAS
{lingxiao.song, zhangman, qli, znsun, rhe}@nlpr.ia.ac.cn

Abstract

Many kinds of efficient greedy subspace clustering methods have been proposed to cut down the computation time in clustering large-scale multimedia datasets. However, these methods are easy to fall into local optimum due to the inherent characteristic of greedy algorithms, which are step-optimal only. To alleviate this problem, this paper proposes a novel greedy subspace clustering strategy based on floating search, called Float Greedy Subspace Clustering (FloatGSC). In order to control the complexity, the nearest subspace neighbor is added in a greedy way, and the subspace is updated by adding an orthogonal basis involved with the newly added data points in each iteration. Besides, a backtracking mechanism is introduced after each iteration to reject wrong neighbors selected in previous iterations. Extensive experiments on motion segmentation and face clustering show that our algorithm can significantly improve the clustering accuracy without sacrificing much computational time, compared with previous greedy subspace clustering methods.

1. Introduction

Subspace clustering, which seeks to cluster data into different subspaces and finds a low-dimensional subspace fitting each group of points, has been extensively studied over the past few decades. Different from the traditional clustering, subspace clustering is based on the assumption that high-dimensional data often lie around low-dimensional subspaces instead of being uniformly distributed across the ambient space. As a result, data from multiple classes can be well modeled by unions of low-dimensional subspaces, *e.g.*, images of the same subject under different illumination lie close to a union of 9D subspaces [4]. Hence, subspace clustering finds wide applications in computer vision and machine learning, such as image segmentation [19], motion segmentation [15], face clustering [6], image representation and compression [7], and multimedia analysis [20, 16].

Existing subspace clustering algorithms can be divided into four main categories: algebraic, iterative, statistical, and spectral clustering-based methods. Factorization-based algebraic methods obtain the segmentation of data based on the factorization of data matrix such as [1, 8]. However, these methods are effective only when the subspaces are independent. Another typical example of algebraic methods is Generalized Principal Component Analysis (GPCA) [15], which uses a polynomial function to fit the given point. The GPCA is able to handle both independent and dependent subspaces, but its computational complexity increases exponentially when the dimension of data grows. Iterative approaches, such as [6, 21], iteratively refine subspaces of each cluster and assign points to the closest subspace. These methods can be applied to both linear and affine subspaces, but they are easy to fall into a local optimum, and thus several restarts are often needed. Statistical approaches, such as [18, 13], model both data and noise under explicit assumptions of the probabilistic distribution of data and noise in each subspaces. The main limitations of these methods are that they generally need the number and dimensions of the subspaces, and they are sensitive to outliers.

In recent years, a number of spectral clustering based methods spring out, to which most state-of-the-art subspace clustering algorithms belong, *e.g.*, Sparse Subspace Clustering (SSC) [3, 4], Low-Rank Representation (LRR) [10], Low-Rank Representation via Correntropy(CLRR) [23, 22] and Low-Rank Sparse Subspace Clustering (LRSSC) [17]. The basic idea of SSC is that a data point can be represented as a linear or affine combination of other data points in the same subspace under an l_1-minimization constraint. A similar optimization based method, called LRR, minimizes nuclear norm instead of the l_1-norm in SSC to guarantee a low-rank affinity matrix. CLRR proposed by Zhang *et al*. [23, 22] attempts to maximize the correntropy between data points and their reconstruction, and an efficient solution of the optimization problem based on half-quadratic minimization is given in their paper. Wang *et al*. [17] propose a hybrid algorithm termed LRSSC by combining l_1-

norm and nuclear norm, assuming that the representation matrix is often not only sparse but also low-rank.

Since the volume of multimedia data generated and accumulated in our daily life becomes bigger and bigger, the complexity of processing these raw data is on the rapid rise, even though the computing power of nowadays computers is increasing fast too. In order to conquer this problem, several greedy-like spectral clustering-based approaches have been proposed in recent years. Dyer *et al*. [2] induce a greedy method for sparse signal recovery called orthogonal matching pursuit (OMP), which is used to replace the l_1-minimization in the SSC. Heckel *et al*. [5] propose a simple algorithm based on thresholding the correlation between data points (TSC). Park *et al*. [11] present Nearest Subspace Neighbor (NSN), which constructs neighbors via incrementally selecting point that is closest to the spanned subspace. All the greedy approaches share a common advantage of computationally less demanding. However, a major disadvantage of greedy-like methods is that they cannot cope with complex situation where the subspaces intersect or lots of noise exists.

In this paper, we aim to improve the robustness of NSN in the presence of large corruptions and outliers. The float greedy subspace clustering algorithm (FloatGSC) is proposed which exploits the superiority of greedy algorithm as well as Floating Search method. Firstly, the nearest subspace neighbors are added to make up a neighbor set in a greedy way. Then, a backtracking exclusion stage is induced aiming to remove wrong neighbors selected in the forward greedy selection stage. Extensive experiments on real-word applications demonstrate that our algorithm can reach state-of-the-art clustering performance, with better robustness and comparatively low computational cost.

2. Proposed method

In this section, we illustrate the proposed method in details. As mentioned before, The proposed method is mainly inspired by the Nearest Subspace Neighbor (NSN)[11]. In order to better understand our model, a brief review of NSN is described firstly. Then, we propose the FloatGSC procedure.

For convenience, the uniform writing rules are used for formula if no special notes in the remaining paper. Uppercase letters and lowercase letters denote matrices and vectors respectively. And some parameters and sets are also denoted by uppercase letters. Specifically, U represents subspace spanned by a set of data points: $U = span(\Omega)$; $\text{proj}_U(x)$ is defined as the projection of data point x to the subspace U; $\mathbb{I}\{.\}$ is the indicator function; $A^\dagger$ is the Moore-Penrose pseudoinverse of matrix A; $\|.\|$ denote Euclidean norm for vectors; and $\langle . \rangle$ is the inner product operator.

2.1. Preliminaries about NSN

Given a data matrix $X \in \mathbb{R}^{D\times N}$ whose each column represents a data point $x_i \in \mathbb{R}^D$, the task of subspace clustering is to recover the union of K subspaces $S = S_1 \cup S_2 \cup ... \cup S_K$, to which the data points belong. Spectral clustering-based methods follow a basic procedure of two steps to solve this problem:(a) computing an affinity matrix $W \in \mathbb{R}^{N\times N}$ whose entry W_{ij} measures the similarity between points x_i and x_j, then (b) deriving clusters from data by applying spectral clustering to this affinity matrix. That means that all of the spectral clustering-based methods are only different in the first affinity matrix construction step.

NSN constructs a neighbor set Ω_i that contains no more than M elements for each data point x_i, thus an affinity matrix whose entries are either 0 or 1 is computed accordingly. The basic idea behind NSN is to find neighbors that are most likely to be on the same subspace. In each iteration, it greedily adds the closest point to the subspace spanned by neighbors selected in early iterations into the neighbor set, until enough neighbors are selected.

That is to say, NSN chooses the nearest neighbor as the first member of the neighbor set to construct subspace. Apparently, NSN has difficulties in dealing with points near the intersection of two subspaces because the neighborhood of a point can contain points from different subspaces. If a wrong neighbor is selected at the very beginning, it becomes very difficult to correctly recover the underlying subspace and then find correct same-class neighbors. Furthermore, there is no way to remove wrong neighbors that are selected in previous iterations. Thus, we propose a floating search method to alleviate this problem.

2.2. Float Greedy Subspace Clustering

As mentioned before, in NSN, neighbors are selected one by one greedily and cannot be discarded in following process. In order to delete wrong neighbors, a backtracking mechanism is introduced in our algorithm. The idea of backtrack is inspired by the Floating Search which is originally proposed in [12] for feature selection.

Inspired by the idea mentioned above, we propose a float greedy subspace clustering algorithm termed as FloatGSC. FloatGSC uses backtrack to remove unfavorable neighbors from the neighbor set, to achieve a better subspace spanning.

Our method mainly involves the following steps:

1) Initial subspace construction: The first step is to construct a good initial subspace. For greedy method is step-optimal only, a good initialization can often results in preferable local optimum. Thus, seeking a good initialization is important. The first neighbor is selected based on the theory of sparse representation. For each data point x, L nearest neighbors around x in the ambient space are chosen

to made up a base. And then the first neighbor can be determined by solving the following l_1-optimization problem

$$\min \|c\|_1 \qquad s.t. \quad x = X^{(L)}c, \tag{1}$$

where $X^{(L)}$ is a subset of the data matrix X which contains the L nearest neighbors of x. Obviously, when L is not large, this l_1-optimization problem can be efficiently solved. The data point corresponding to the max entry of c is the first neigShbor added to x's neighbor set Ω. Then, the initial subspace U can be computed accordingly.

2) Forward inclusion: After constructing the initial subspace, the closest point to the subspace is greedily added into the neighbor set Ω_i in each iteration as NSN. For the sake of computational simplification, the subspace U is represented by a set of orthogonal bases: $\eta_m, m = 1, 2, \ldots$. Therefore, the closet data point, which is equivalent to the point that has maximum projection to the subspace, can be easily derived by the following functions.

$$j^* = \arg \max_{j \in [N] \setminus \Omega} \text{proj}_U(x_j), \tag{2}$$

$$\text{proj}_U(x)^2 = \sum_k \langle x, \eta_k \rangle^2. \tag{3}$$

It is worth noting that we only need to compute the inner product with the newest added orthogonal basis in each iteration, if no exclusion happens in last iteration.

3) Conditional exclusion: This step is the core of the Floating Search idea. In this step we introduce backtrack to remove the neighbor that has least affinity to point x in the spanned subspace. The affinity of all the selected neighbors to x in the spanned subspace can be computed by

$$c_j = (X_\Omega^\dagger)_j x, \tag{4}$$

where X_Ω is submatrix of X indexed by the neighbor set Ω, $X_\Omega^\dagger$ is the Moore-Penrose pseudoinverse of X_Ω, and $(X_\Omega^\dagger)_j$ means the j-th row of $X_\Omega^\dagger$.

$$\min_{k \in \Omega} c_k < c_{j^*}. \tag{5}$$

Least affinity to x means least similarity to x in the built subspace. Thus, if there exists a data point whose similarity is lower than the newest one as the exclusion condition in Eq.(5), it should be replaced by the newest added point. In fact, there may be more than one neighbor that has lower affinity than the newest neighbor, but we only discard one neighbor at one conditional exclusion step for the sake of simplification. Once the exclusion condition holds, the neighbor set as well as spanned subspace should be updated, which involves a series of computations. Consequently, introducing the backtracking step helps to find better neighbors at the cost of sacrificing computational efficiency.

4) Subspace updating: In every iteration, the spanned subspace is updated if the dimension of existing subspace is not higher than the maximum dimension P, which is set to control the complexity of the recovered subspaces. According to the Gram-Schmidt process, updating the subspace U can be simplified to adding new orthogonal bases involved with newly added data points. The newly added orthogonal base can be derived as Eq.(6).

$$\eta_{m+1} = x_{j^*} - \sum_{k=1}^{m} \langle x_{j^*}, \eta_k \rangle \eta_k. \tag{6}$$

The last three steps iterate until enough neighbors are selected. Then, a binary neighbor matrix can be computed based on each data points' neighbor set. Finally, spectral clustering is applied to find the clusters as in other spectral-based algorithms. The whole procedure of our method is shown in Algorithm 1.

Algorithm 1 Float Greedy Subspace Clustering(FloatGSC)

Input: Data matrix $X \in \mathbb{R}^{D \times N}$, number of classes K, maximum subspace dimension P, number of neighbors M.

Output: Neighbor matrix $Z \in \{0,1\}^{N \times N}$, estimated class labels $\hat{c}_1, \hat{c}_2, ..., \hat{c}_N$.

Normalize all data points: $x_i \leftarrow x_i / \|x_i\|$;
for each x_i **do**
  Initialization: Construct the initial subspace U and initial neighbor set Ω_i by solving Eq. (1), $m = 2$.
  Iteration: Find neighbors for x_i.
  while $m < M$ **do**
    Forward Inclusion:
      $j^* = \arg\max_{j \in [N] \setminus \Omega_i} \text{proj}_U(x_j)$,
      $\Omega_i \leftarrow \Omega_i \cup \{j^*\}$,
      $m = m + 1$;
    Conditional Exclusion:
      $k^* = \arg\min_{k \in \Omega_i} (X_{\Omega_i}^\dagger)_k x_i$;
      if $j^* \neq k^*$, **then**
        $\Omega_i \leftarrow \Omega_i \setminus \{j^*\}$;
        $m = m - 1$;
    if $m < P$, **then** $U \leftarrow span\{x_j : j \in \Omega_i\}$;
  end while
  Update the neighborhood matrix: $Z_{ij} = 1, \forall j \in \Omega_i$;
end for
Construct the affinity matrix: $W_{ij} = Z_{ij} + Z_{ji}$;
Apply spectral clustering to (W, K).

3. Experiments

In order to validate the proposed method, we perform experiments on two real-world applications in this section: motion segmentation and face clustering, and compare it with a series of state-of-the-art methods. For all the baseline methods, the MATLAB codes provided by their authors are used.

3.1. Experimental settings

Three evaluation metrics are used in our experiments: clustering error (CE), neighbor selection error (NSE) and

run time (CT). CE is defined as

$$CE = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}(c_i \neq \hat{c}_i), \tag{7}$$

where c_i is the class label, $\hat{c}_i$ is the estimated class label. Since the definite label index of estimated class may be not consistent with its real index, every permutation of estimated class label should be calculated in CE, and the minimum among all the permutation is adopted.

NSE is the proportion of points that do not have all correct neighbors,

$$NSE = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}(j|W_{ij} \neq 0, c_i \neq \hat{c}_i), \tag{8}$$

it measures the extent that algorithms misconnect data points from different subspaces in the adjacency matrix. Besides, we compare the average run time (RT) to evaluate our algorithm's efficiency.

3.2. Motion segmentation results

The Hopkins155 dataset[1] [14] is used to test the performance of motion segmentation, which comprises 155 video sequences of 2 or 3 motions. This test is aiming to segment the tracked points in a frame into different motion clusters. All experiments are directly done on the raw data downloaded from the database webpage without any preprocessing.

In our method, there are two parameters that need to be set: the maximum subspace dimension P and the number of neighbors M for each data point. In general, the choice of M depends on the prior knowledge of the amount of data: the larger the data volume is, the more the neighbors can be. As for the subspace dimension P, in theory, the value of P should be set equal to the underlying subspace where the data lies in. However, existence of corruption and noise will cause to the gap between theory and practice. So choosing the subspace dimension directly equal to its theoretical value is suboptimal. According to the affine projection model, all the trajectories associated with a single rigid motion live in an affine subspace of dimension 3, so we wet P around 3. After several trails under different P, we choose P = 5 as a tradeoff between clustering accuracy and computational efficiency, and set $M = P = 5$ empirically.

Table 1 shows the results over all sequences in the Hopkins155. Since there is no much corruption or missing data in the Hopkins155 dataset, most baseline algorithms can achieve quite good performance. And the optimization-based methods obtain best results in terms of the CE and NSE measures. As can be seen, FolatGSC performs comparable to the optimization-based methods, and much better than other greedy algorithms. Moreover, the CE and NSE of FloatGSC are much smaller than NSN, which demonstrates the effectiveness of the Floating Search idea in our method.

[1]The Hopkins155 database is available online at http://www.vision.jhu.edu/data/hopkins155/.

Table 1. Results on Hopkins155 dataset

	Algotithms	SSC	LRR	OMP	TSC	NSN	**FloatGSC**
2 Motions	Mean CE(%)	**1.53**	2.13	17.25	18.44	3.62	2.62
	Median CE(%)	**0**	**0**	13.33	16.92	**0**	**0**
	Mean NSE(%)	**1.09**	6.03	37.61	2.86	2.91	2.56
	Mean RT(s)	0.50	0.96	0.17	0.16	**0.06**	0.42
3 Motions	Mean CE(%)	4.40	**4.03**	27.61	28.58	8.28	7.56
	Median CE(%)	**0.56**	1.43	23.79	29.67	2.76	1.11
	Mean NSE(%)	**2.44**	10.56	78.03	7.42	8.30	7.02
	Mean RT(s)	1.03	1.33	0.28	0.38	**0.12**	0.72
All	Mean CE(%)	**2.18**	2.56	19.59	20.73	4.67	3.74
	Median CE(%)	**0.13**	0.32	15.69	19.80	0.62	0.25
	Mean NSE(%)	**1.39**	7.05	46.74	3.89	4.13	3.57
	Mean RT(s)	0.62	1.04	0.19	0.21	**0.07**	0.48

3.3. Face clustering results

We evaluate the face clustering performance of FloatGSC and state-of-the-art methods on the Extended Yale-B database[2][9]. The Extended Yale-B database contains frontal face images of 38 subjects under 9 poses and 64 varying illumination conditions.

In our experiments, only the frontal images of the Extended Yale-B dataset are used. Besides, to reduce the computational time and memory cost, we use the cropped images and resize them to 48×42 pixels, then concatenate the raw pixels value into a 2016-dimensional vector for each data point. Similar to the motion segmentation experiment, we set $P = M = 15$ after numerous comparative experiments of different value of P and M. A series of experiments under different number of subjects are taken to validate our method's performance. We randomly sample 100 combinations for different number of subjects respectively, and all the experimental results listed in Table 2 are average value of the 100 random trials.

Table 2. Results on the Extended Yale-B dataset

	Algotithms	SSC	LRR	OMP	TSC	NSN	**FloatGSC**
2 Subjects	Mean CE(%)	2.67	4.29	7.41	12.53	1.84	**1.37**
	Median CE(%)	**0**	0.78	1.57	2.36	0.78	0.78
	Mean NSE(%)	8.14	5.59	13.86	10.04	2.63	**1.79**
	Mean RT(s)	17.13	2.59	0.21	**0.15**	0.28	2.22
3 Subjects	Mean CE(%)	4.16	5.60	5.12	20.02	3.32	**2.19**
	Median CE(%)	**1.04**	**1.04**	2.08	13.54	2.6	2.08
	Mean NSE(%)	20.99	9.95	39.07	19.04	4.72	**2.92**
	Mean RT(s)	21.99	4.89	**0.35**	0.36	0.95	3.82
5 Subjects	Mean CE(%)	4.72	5.72	9.26	29.58	6.18	**3.36**
	Median CE(%)	**2.81**	2.90	5.00	31.25	5.31	3.13
	Mean NSE(%)	53.76	17.53	87.00	27.56	7.65	**4.31**
	Mean RT(s)	32.14	10.48	0.76	**0.49**	1.86	7.62
10 Subjects	Mean CE(%)	11.75	11.65	16.15	41.68	13.63	**7.77**
	Median CE(%)	11.02	12.73	17.19	42.66	12.03	**5.86**
	Mean NSE(%)	76.46	33.47	94.54	36.56	12.44	**6.92**
	Mean RT(s)	61.52	41.16	3.24	**1.56**	7.91	27.76

Obviously, FloatGSC obtains the smallest mean CE as well as mean NSE in most cases, while SSC and LRR perform best on the median CE. One possible reason is that these optimization-based methods are more powerful to handle most general conditions, but they fail in some difficult cases. As a consequence, SSC and LRR

[2]The Extended Yale-B database is available online at http://vision.ucsd.edu/ leekc/ExtYaleDatabase/ExtYaleB.html.

can obtain very little clustering error in most cases, but cannot do well in clustering hard samples. While the proposed method performs more robust. Compared to the other greedy algorithms, OMP, TSC and NSN, FloatGSC performs significantly better in clustering errors at a sacrifice of computational time. In a word, the proposed method can achieve state-of-the-art clustering performance with comparable low computational cost.

4. Conclusions

This paper studied a float strategy of greedy subspace clustering methods. In summary, FloatGSC constructs the neighbor set of each data point in a Floating Search idea: forward greedy selection and backward floating exclusion. In the forward inclusion stage, the neighbor nearest to subspaces built by selected neighbors is added greedily. And in the conditional exclusion stage, the neighbor that has least affinity to point x in the spanned subspace is removed, which helps to discard potential different-class neighbors. Experimental results show that FloatGSC achieves better performance than other greedy subspace clustering methods, which indicates the helpfulness of the float strategy. Meanwhile FloatGSC maintains comparable low computational cost due to the forward greedy strategy.

Acknowledgement

This work is funded by the Youth Innovation Promotion Association, Chinese Academy of Sciences (Grant No. 2015190), and the National Natural Science Foundation of China (Grant No. 61473289).

References

[1] J. P. Costeira and T. Kanade. A Multibody Factorization Method for Independently Moving Objects. *International Journal of Computer Vision*, 29(3):159–179, 1998. 1

[2] E. L. Dyer, A. C. Sankaranarayanan, and R. G. Baraniuk. Greedy feature selection for subspace clustering. *The Journal of Machine Learning Research*, 14(1):2487–2517, 2013. 2

[3] E. Elhamifar and R. Vidal. Sparse subspace clustering. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2790–2797, 2009. 1

[4] E. Elhamifar and R. Vidal. Sparse subspace clustering: Algorithm, theory, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(11):2765–2781, 2013. 1

[5] R. Heckel and H. Bölcskei. Subspace clustering via thresholding and spectral clustering. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3263–3267, 2013. 2

[6] J. Ho, M.-H. Yang, J. Lim, K.-C. Lee, and D. Kriegman. Clustering appearances of objects under varying illumination conditions. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages I11–I18, 2003. 1

[7] W. Hong, J. Wright, K. Huang, and Y. Ma. Multiscale hybrid linear models for lossy image representation. *IEEE Transactions on Image Processing*, 15(12):3655–3671, 2006. 1

[8] K. Kanatani. Motion segmentation by subspace separation and model selection. In *IEEE International Conference on Computer Vision*, volume 2, pages 586–591, 2001. 1

[9] K.-C. Lee, J. Ho, and D. J. Kriegman. Acquiring linear subspaces for face recognition under variable lighting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(5):684–698, 2005. 4

[10] G. Liu, Z. Lin, S. Yan, J. Sun, Y. Yu, and Y. Ma. Robust recovery of subspace structures by low-rank representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):171–184, 2013. 1

[11] D. Park, C. Caramanis, and S. Sanghavi. Greedy subspace clustering. In *Advances in Neural Information Processing Systems*, pages 2753–2761, 2014. 2

[12] P. Pudil, J. Novovičová, and J. Kittler. Floating search methods in feature selection. *Pattern Recognition Letters*, 15(11):1119–1125, 1994. 2

[13] M. E. Tipping and C. M. Bishop. Mixtures of probabilistic principal component analyzers. *Neural computation*, 11(2):443–482, 1999. 1

[14] R. Tron and R. Vidal. A benchmark for the comparison of 3-d motion segmentation algorithms. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2007. 4

[15] R. Vidal, R. Tron, and R. Hartley. Multiframe motion segmentation with missing data using powerfactorization and gpca. *International Journal of Computer Vision*, 79(1):85–105, 2008. 1

[16] D. Wang, Q. Yin, R. He, L. Wang, and T. Tan. Multi-view clustering via structured low-rank representation. In *ACM International Conference on Information and Knowledge Management*, 2015. 1

[17] Y.-X. Wang, H. Xu, and C. Leng. Provable subspace clustering: When LRR meets SSC. In *Advances in Neural Information Processing Systems*, pages 64–72, 2013. 1

[18] A. Y. Yang, S. R. Rao, and Y. Ma. Robust statistical estimation and segmentation of multiple subspaces. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 99–99, 2006. 1

[19] A. Y. Yang, J. Wright, Y. Ma, and S. S. Sastry. Unsupervised segmentation of natural images via lossy data compression. *Computer Vision and Image Understanding*, 110(2):212–225, 2008. 1

[20] Q. Yin, S. Wu, R. He, and L. Wang. Multi-view clustering via pairwise sparse subspace representation. *Neurocomputing*, 156:12–21, 2015. 1

[21] T. Zhang, A. Szlam, and G. Lerman. Median K-flats for hybrid linear modeling with many outliers. In *Computer Vision Workshops (ICCV Workshops)*, pages 234–241, 2009. 1

[22] Y. Zhang, Z. Sun, R. He, and T. Tan. Robust low-rank representation via correntropy. In *Asian Conference on Pattern Recognition*, pages 461–465, 2013. 1

[23] Y. Zhang, Z. Sun, R. He, and T. Tan. Robust subspace clustering via half-quadratic minimization. In *IEEE International Conference on Computer Vision*, pages 3096–3103, 2013. 1