Matrix Organization Problems via Bi-Folderings

Julien Clancy

December 18, 2017

1 Introduction

In a series of papers [1],[2],[3] Coifman et al recently formulated a generalization of multiscale analysis to spaces of points equipped with a set of functions, *assuming* smoothness of the functions to build a metric on the points. Such a set of functions can be conveniently organized into a matrix, if the functions are taken to be the columns and the points are the rows. This induced metric on the rows can then be used to build a metric on the columns, this time considering the *rows* as functions on the point set of the *columns*. Alternating in this fashion, we obtain a pair of coupled geometries on the rows and columns, which hopefully reveal some structure of the matrix. The inventors' work focuses on "real data", e.g. [4], starting with a kind of earth-mover's distance between the rows or columns. We hope to apply these methods to solve problems relating to recovering low-rank blocks in matrices. The aim of this project is to use these methods on "planted low-rank block" problems to gauge their performance, and combine them with techniques more traditionally used for low-rank matrices. We obtain empirical solutions to open problems in the area, and cheap and simple solutions to existing problems.

2 The Problems

2.1 Subspace Clustering

The most elementary version of the problem at hand is subspace clustering (see [5] and references therein). We are given a $n \times m$ matrix X with the property that there are subspaces V_i such that all X_j (columns of X) are in some V_i, and $\sum \dim V_i \ll m$; the goal is to recover V_i and the assignment

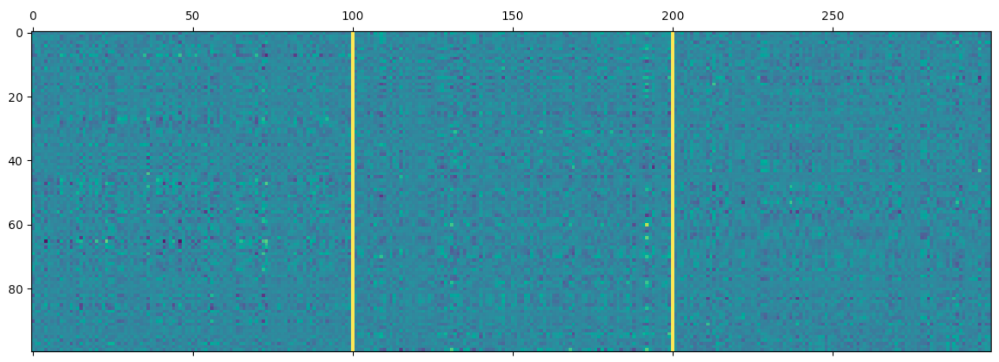

Figure 1: An instance of the subspace clustering problem on three subspaces, with the correct permutation. Subspaces and points in them were chosen randomly. Notice the repetitive rectangular patterns in the blocks, which are consistent within but inconsistent across blocks.

of each X_j to some V_i (or, in the case where there are also outlying points that belong to no subspace, just the subspaces V_i, from which the non-outlier membership can be easily recovered). Equivalently, we are guaranteed that after a permutation the columns of X are in contiguous low-rank blocks, and we must recover the permutation. A matrix (with the correct column ordering) generated according to this problem is shown in figure 1, with outlines for relief.

What [5] analyzed was the "sparse subspace clustering" (SSC) algorithm, where we take each point X_i and try to write it as a linear combination of other points X_j in the sparsest way possible. Specifically, they solve

$$\min \|C\|_1 \text{ s.t. } XC = X, C_{i,i} = 0$$

and use the sparsity pattern of C as a similarity matrix for spectral clustering. They prove that with overwhelming probability over a natural generative model (random subspaces with random vectors, ρ vectors per subspace) C only contains nonzeros where two vectors truly are in the same subspace. The generalization to the noisy case, given in [6], is the Lasso-SSC program

$$\min \frac{1}{2}\|X - XC\|_F^2 + \lambda\|C\|_1 \text{ s.t. } C_{i,i} = 0$$

One of the goals of this project (and our ongoing research) is to integrate this penalty into tree-based clustering to improve performance.

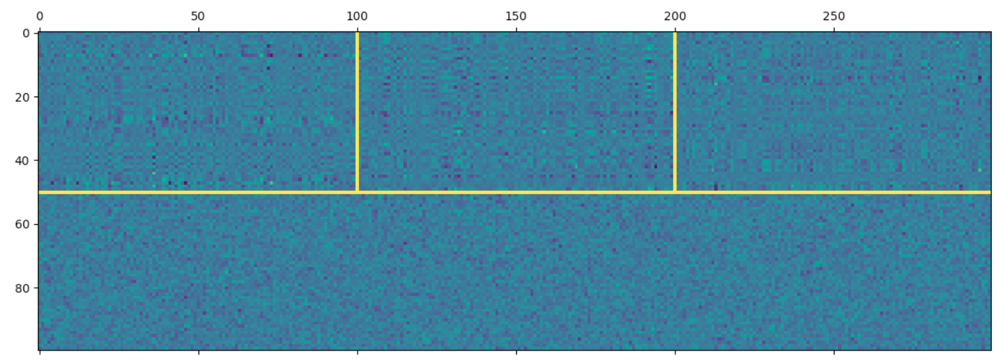

Figure 2: Subspace clustering with irrelevant features (shown in "planted" form, without permutations on the rows and columns).

2.2 Subspace Clustering with Irrelevant Features

In [7] the authors introduced and solved a more complex version of subspace clustering, where some one the features (rows) are meaningless, or in the model, replaced with noise. Otherwise the task is the same as above. Looking at each point in the Lasso-SSC program individually, expanding the Frobenius norm, and assuming that $\|x_i\| = 1$, we see that it is equivalent to

$$\min \frac{1}{2}\|X_{\sim i}c_i\|_2^2 - \langle x_i, X_{\sim i}c_i\rangle + \lambda\|c_i\|_1$$

Their trick is to replace inner products $\langle a, b\rangle = \sum_{i=1}^n a_i b_i$ by a robust counterpart, where instead of summing all $a_i b_i$ we take only the k smallest entries, for some $k < n$. This allows us to redefine the norm and inner product above and run the minimization. What the authors show is that indeed this works.

2.3 Subspace-Dependent Irrelevant Features

In [7] the authors also bring up a related problem, where which features are relevant depends on the subspace. This setup is shown in figure 3. The major goal of this project was to solve this problem, hopefully integrating the methods of [5] with those of [2].

3 Bi-Foldering

Here we describe the simplified version of the algorithm in [2] for matrix organization. We take a matrix to be a function $M\colon X \times Y \to \mathbb{R}$, where X is the set of rows and Y is the set of columns. A tree $\mathcal{T}$ is a hierarchy

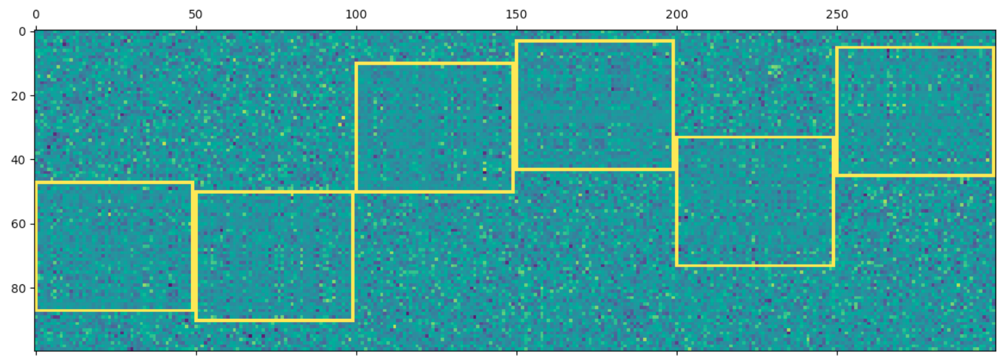

Figure 3: Recovering planted low-rank blocks, where the blocks do not intersect across columns. As before this matrix is shown without permutations for clarity.

of partitions on a point space which has the property that each "folder" in a partition intersects only one folder in the partition immediately above it — think of a binary tree where the first level is just the whole space X, the second is a bipartition of X, the third is a division of X into four disjoint subsets, and so on, with all partitions nested. Given a tree $\mathcal{T}$ we say its ℓth level is $\mathcal{T}^\ell$ and the ith folder on that level is $\mathcal{T}_i^\ell$. Notice that a tree naturally induces a geometry on the space that its folders partition, via the tree metric.[1] Denoting by $d_\mathcal{T}$ this metric, let $W_\mathcal{T}(x_1, x_2) = d_\mathcal{T}(x_1, x_2)$ be its affinity matrix. It is well-known that the top eigenvectors of affinity matrices embed their metrics in low-dimensional space, so if we do this, we could hopefully recover the tree via recursive 2-means clustering. It is not particularly useful to recover a tree from its embedding, but it is useful to point out that trees and affinity matrices are in some sense one and the same.

Given a partition tree $\mathcal{T}$ on a set X and a collection of functions $f_j \colon X \to \mathbb{R}$, we can define new affinity matrix on $Y = \{f_j\}$ by

$$W_Y(f_j, f_k) = \sum_\ell \sum_i \frac{1}{|\mathcal{T}_i^\ell|^\alpha} \left| \langle f_j, f_k \rangle_{\mathcal{T}_i^\ell} \right|$$

where $\langle f, g \rangle_S = \sum_{i \in S} f_i g_i$ and α is some parameter that represents scale-sensitivity of the metric — we might wish to undercount small folders because their contents are noisy or simply because there are so many of them, corresponding to $\alpha \leq 1$, or we might want to treat every folder independently of its size by choosing $\alpha = 0$, or with the central limit theorem in mind we might take $\alpha = 1/2$. Throughout all of the experiments recorded here we

[1] This is the distance of two points to their least common ancestor in the tree. One might think that if two people live in the same house their tree distance is, say, 2, while if they live only in the same city it is 3, and if they live only in the same state it is 4.

used $\alpha = 1$, but changing to $\alpha = 1/2$ did not substantively change the results. Using the procedure outlined above we embed the set $\{f_j\}$ in in Euclidean space using the eigenvectors of W_Y, then construct a binary tree $\mathcal{S}$ using recursive 2-means. Since the functions f_j are actually the columns of M, we then use $\mathcal{S}$ to organize the functions given by the *rows* of M, and alternate in this fashion until convergence (which usually means 5 to 10 iterations).

4 Experiments

As a sanity check we ran this algorithm on the first example problem above, subspace clustering. Its performance was competitive with the state of the art. We also ran experiments on the problem of subspace clustering with irrelevant features, and this time achieved significantly better performance than the state of the art — while the method in [7] can reportedly identify the subspaces with up to half of the features corrupted, our method works at least down to 90% corruptions, and possily more. On the problem of finding low-rank blocks our method performed extremely well, outperforming reasonable expectations. For example, in our experiments we started with a 256×1024 matrix of noise, and planted eight 64×128 blocks containing random samples from 8-dimensional subspaces. We ensured that the statistics of noise and non-noise patches matched by scaling the noise levels, and ran the simple algorithm above. Our metric for recovery of a block was if one level in the column partition tree very closely matched with which columns were in some block, since given this information we can easily recover the block itself. Indeed we found that every subspace had a level in the final partition tree that contained its and only its columns, modulo an error of no more than 5%.

We also implemented the Lasso-SSC algorithm using FISTA [8] to construct affinity matrices at each level. While more experiments are necessary to calibrate scaling constants relating to different levels in the tree and the sparsity-fit tradeoff in the coefficient matrix C, we found that it performed slightly worse than the tree-based method and required much more computation time (since the affinity solution step is itself a convex solve, not a matrix multiply). In the future we plan to implement d'Aspremont et al's acceleration methods in [9] or the restart schemes described in [10] to further speed up this solve. We believe that a SSC-type method has the potential to be very successful in this area, but further work is needed.

If we are dealing with very small numbers of subspaces then looking at the permuted matrix is enough to recognize whether the algorithm worked. However, especially in the case of irrelevant features that depend on the subspace, it is difficult to identify optically whether the algorithm actually succeeded. A better way is to allow for the user to explore the tree organization of the columns of the matrix. While Julia's plotting is quite good, we found that the most intuitive solution involves user interaction. This is much simpler to do in JavaScript using the D3.js library, so we installed the HTTPServer.jl package to interface the algorithms we implemented with the D3 visualization. The tree visalization in the lower-left is largely based off of the examples of Mike Bostock (the primary D3 developer). The code operates on a randomly permuted version of the example matrix (generated from one of the models above), but the interface shows the matrix in the correct order, so that contiguous blocks of columns being in the same folder corresponds to algorithmic correctness. The three-dimensional scatterplot shows the columns plotted according to the top three[2] eigenvectors of the affinity matrix generated by the learned tree. To navigate the data, single-click or double-click nodes in the tree to highlight the corresponding columns or expand a node's children.

5 Code Structure

The server is currently configured to generate and analyze an instance of the third, hardest model. The visualization is shown at 127.0.0.1:8000.

The algorithm itself operates by traversing partition trees, so the workhorses of our code are types for Folders, Partitions, and PartitionTrees. The functions for generating the tree-based similarity matrix and deriving a dual geometry from it proceed as explained above; see [2] for more details. The sparse subspace clustering code uses an ℓ^1 minimization solver based on the accelerated gradient descent algorithm derived in[8]. We also implemented a fast top eigenvector solver desribed in [11], but found it to be slightly inferior in runtime compared to Julia's built-in svds function at a given accuracy level (benchmarked against an unlimited runtime full svd). We believe memory is the key culprit here, which is an unavoidable feature of the algorithm — theoretically it is only a log factor worse than Lanczos iteration, but the constants are probably unfavorable. We also implemented unit tests for the portions of our code amenable to it.

Our implementation of the main algorithm is based off of one in MATLAB due to Ronald Coifman and his collaborators, which we found difficult to modify without reimplementation.

[2]Actually, the top four minus the first, which is almost constant beacause affinity matrices are typically close to being Markov.

6 Future Work

The feeling of the author is that subspace clustering should be present somehow in any solution of the low-rank blocks problem — this is the most prominent avenue for future research. We found a naive integration with tree geometries to be unsuccessful, but it could be that it would work better without a multiscale framework, say with selective masking of entries.

The models considered here are in some sense toys. There are many real-world examples where low-rank blocks appear continuously rather than with well-defined boundaries, for instance in hierarchical matrices aririsisng from potential operators [12]. Matrices with arbitrary low-rank block decompositions do not necessarily fall on a perfect tensor grid, and so the bi-tree setup is probably unsuitabe to analyze them. This is the other direction in which we intend to continue research.

References

[1] M. Gavish and R. R. Coifman, "Sampling, denoising and compression of matrices by coherent matrix organization," *Applied and Computational Harmonic Analysis*, vol. 33, no. 3, pp. 354 – 369, 2012.

[2] R. R. Coifman and M. Gavish, *Harmonic Analysis of Digital Data Bases*, pp. 161–197. Boston: Birkhäuser Boston, 2011.

[3] G. Mishne, R. Talmon, I. Cohen, R. Coifman, and Y. Kluger, "Data-driven tree transforms and metrics," 2017. https://arxiv.org/pdf/1708.05768.pdf.

[4] G. Mishne, R. Talmon, R. Meir, J. Schiller, M. Lavzin, U. Dubin, and R. R. Coifman, "Hierarchical coupled-geometry analysis for neuronal structure and activity pattern discovery," *IEEE Journal of Selected Topics in Signal Processing*, vol. 10, pp. 1238–1253, Oct 2016.

[5] M. Soltanolkotabi and E. J. Candes, "A geometric analysis of subspace clustering with outliers," *Ann. Statist.*, vol. 40, no. 4, p. 2195 2238, 2012.

[6] M. Soltanolkotabi, E. Elhamifar, and E. Candes, "Robust subspace clustering," *Ann. Statist.*, vol. 42, no. 2, pp. 669 – 699, 2014.

[7] C. Qu and H. Xu, "Subspace clustering with irrelevant features via robust dantzig selector," in *Advances in Neural Information Processing Systems 28* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 757–765, Curran Associates, Inc., 2015.

[8] A. Beck and M. Teboulle, “A fast iterative shrinkage-thresholding algorithm for linear inverse problems,” *SIAM J. Img. Sci.*, vol. 2, pp. 183–202, Mar. 2009.

[9] D. Scieur, A. d’Aspremont, and F. Bach, “Regularized nonlinear acceleration,” *Advances in Neural Information Processing Systems 29*, pp. 712–720, 2016.

[10] B. O’Donoghue and E. Candès, “Adaptive restart for accelerated gradient schemes,” *Foundations of Computational Mathematics*, vol. 15, pp. 715–732, Jun 2015.

[11] C. Musco and C. Musco, “Randomized block krylov methods for stronger and faster approximate singular value decomposition,” in *Advances in Neural Information Processing Systems 28* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 1396–1404, Curran Associates, Inc., 2015.

[12] W. Hackbusch, *Hierarchical Matrices: Algorithms and Analysis.* Springer Publishing Company, Incorporated, 1st ed., 2015.