Robust Principal Component Analysis

Yi Zheng, Yichuan Sun

May 28, 2022

Contents

1 Introduction

Principal component analysis (PCA) is the process of computing the principal components and using them to perform a change of basis on the data, sometimes using only the first few principal components and ignoring the rest, by which people can successfully reduce the dimension of the data. In short, PCA tries to find the best projection on a low-dimensional subspace.

Consider a common situation where we have a signal matrix M containing the useful structural information as well as the noise, and we want to decompose it into the summation of two matrices, L and S, such that

$$M = L + S$$

where L is the matrix of useful information and S is the matrix of noise. Often, L is low-rank (due to some intrinsic structure of the information, the columns or rows may be linearly dependent) and S is sparse. When we applied PCA, we assume that the noises are Gaussian. But if there are some non-Gaussian noise or big outliers, this assumption may not be satisfied and PCA may have trouble working properly. Unfortunately, these kinds of gross errors are very common in many applications, arising for example from corrupted data, sensor failures or corrupted samples in repetitive measurement tasks in biology applications. Thus, Robust PCA was introduced to solve this problem, which is a promising way even in the presence of such gross but sparse corruptions. It combines the two popular heuristics of nuclear norm minimization (used to encourage low-rankness) and ℓ_1-norm minimization (used to encourage sparsity) and casts the problem of separating a low-rank ”data” component from a sparse ”noise” component into a convex optimization problem.

Over the past decade there has been an explosion in terms of the massive amounts of highdimensional data in almost all fields of science and engineering, which is largely due to its success in numerous application domains, ranging from bioinformatics, statistics, and machine learning to image and video processing in computer vision. In such applications, researches today routinely deal with data that lie in thousands or even billions of dimensions, with a number of samples sometimes of the same order of magnitude. Therefore, it makes sense to explore this method. In this project, we implemented two classic robust PCA algorithms, *Inexact Augmented Lagrange Multiplier (IALM) Method* and *Alternating Direction Method of Multipliers (ADMM)* to solve the problem of Robust PCA. Numerical results are attached.

2 Restatement of the Problem

One particular formulation of RPCA can be stated as follows:

$$\min_{L,S\in R^{m\times n}} \operatorname{rank}(L) + \rho\|S\|_0 \quad \text{s.t. } L + S = M \tag{1}$$

where $\|S\|_0$ is called the ℓ_0-norm 1 of S and counts the number of nonzero entries of S, and $\rho > 0$ is a trade-off parameter. However, this form of the problem is NP-hard and thus not that tractable.

Luckily, it was proved later that under certain conditions, (1) is equivalent to the following convex program with high probability:

$$\min_{L,S\in R^{m\times n}} \|L\|_* + \rho\|S\|_1 \quad \text{s.t. } L + S = M \tag{2}$$

where $\|L\|_*$ is called the nuclear norm of L and equal to the sum of the singular values of L, and $\|S\|_1 := \sum_{ij} |S_{ij}|$ is called the ℓ_1-norm of S. The optimization problem in (2) is called robust principal component pursuit (RPCP).

Besides the RPCP formulation in (2), the so-called stable PCP (SPCP) tries to solve this problem:

$$\min_{L,S \in R^{m\times n}} \operatorname{rank}(L) + \rho\|S\|_0 \quad \text{s.t. } \|L+S-M\|_F \leq \sigma \tag{3}$$

Since (3) satisfies the Slater's condition, it is equivalent to the following unconstrained problem for an appropriately chosen parameter $\mu > 0$ depending on σ:

$$\min_{L,S \in R^{m\times n}} \operatorname{rank}(L) + \rho\|S\|_0 + \frac{\mu}{2}\|L+S-M\|_F^2 \tag{4}$$

Formulation (2) and (4) are used in this project.

3 Generation of Test Matrix

The object of robust is to decompose a low-rank matrix with sparse noise. To simulate the problem, we get the target matrix with the following steps.

- Generate an orthogonal matrix for the required n.
- Cut off the matrix to the required rank, get an $n \times$ rank matrix.
- Amplify each column with random numbers.
- Multiply this matrix with its transpose, get the $n \times n$ low-rank matrix.
- generate S unique random numbers that refers to the location of the noise.
- Add random numbers to the location generated by the former step in the low-rank matrix, get the final matrix.

4 Statement of Algorithms

4.1 Notations

Some notations that might be used in the 2 algorithms are listed as below.

- $J(D)$: the dual norm of a matrix D.
- $\mathcal{S}_\tau[x]$: shrinkage operator, the result is $\max(|x| - \tau, 0)$.
- $D_\tau(X)$: the result is $U\mathcal{S}_\tau[\Sigma]V$, where $X = U\Sigma V$ for singular value decomposition.

4.2 Inexact Augmented Lagrange Multiplier

Before explaining IALM method, we cannot avoid introducing exact ALM (EALM) first, which not only appeared earlier but laid a solid foundation for the appearance of IALM.
For formulation (2), we may apply the augmented Lagrange multiplier method by identifying:

$$X = (L, S), \quad f(X) = \|L\|_* + \lambda\|S\|_1, \quad \text{and} \quad h(X) = M - L - S$$

Then the Lagrangian function is:

$$l(L, S, Y, \mu) = \|L\|_* + \lambda\|S\|_1 + \langle Y, M - L - S\rangle + \frac{\mu}{2}\|M - L - S\|_F^2$$

When not converged, EALM tries to iteratively solve the sub-problem:

$$\left(L_{k+1}^*, S_{k+1}^*\right) = \arg\min_{L,S} l\left(L, S, Y_k^*, \mu_k\right)$$

As the SVD accounts for the majority of the computational load, the choice of $\{\mu_k\}$ should be judicious so that the total number of SVDs is minimal.
Fortunately, as it turns out, we do not have to solve the sub-problem

$$\left(L_{k+1}^*, S_{k+1}^*\right) = \arg\min_{L,S} l\left(L, S, Y_k^*, \mu_k\right)$$

exactly. Rather, updating L_k and S_k once when solving this sub-problem is sufficient for L_k and S_k to converge to the optimal solution of the RPCA problem. This leads to inexact ALM (IALM) method.
Set the convergence tolerance $tol = 1e - 7$, and maximum iteration $maxiter = 1000$, then the algorithm can be described in the following code:

- initialize $Y_0 = D/J(D); E_0 = 0; \mu_0 > 0; \rho > 1; k = 0.$
- while $||M - L - S||_F > tol$ and $iter < maxiter$:
-   $(U, S, V) = \text{svd}\left(D - E_k + \mu_k^{-1}Y_k\right)$
-   $A_{k+1} = U\mathcal{S}_{\mu_k^{-1}}[S]V^T$
-   $E_{k+1} = \mathcal{S}_{\lambda\mu_k^{-1}}\left[D - A_{k+1} + \mu_k^{-1}Y_k\right]$
-   $Y_{k+1} = Y_k + \mu_k\left(D - A_{k+1} - E_{k+1}\right); \mu_{k+1} = \rho\mu_k$
- end while

4.3 Alternating Direction Method of Multipliers

As is shown in the article, $\min_L l(L, S, Y)$ and $\min_S l(L, S, Y)$ have simple solutions so we do not need to solve convex problems.
Initialize the hyperparameters $\lambda = 1/\sqrt{\max(M, N)}$, $\mu = 10 \times \lambda$. Set the convergence tolerance $tol = 1e - 7 \times ||M||_F$, and maximum iteration $maxiter = 10^5$. The algorithm can be described in the following loop.

```
• while $||M - L - S||_F > tol$ and $iter < maxiter$:
•     $L_{k+1} = D_{1/\mu}(M - S_k + \mu^{-1}Y_k)$
•     $S_{k+1} = \mathcal{S}_{\lambda/\mu}[M - L_{k+1} + \mu^{-1}Y_k]$
•     $Y_{k+1} = Y_k + \mu(M - L_{k+1} - S_{k+1})$
• end while
```

5 Numerical Simulation Results

5.1 Method

In order to test the performance of the algorithm, we choose the Dimension $n = 100, 200, 500, 1000$, with $rank = 0.1n$ and $0.05n$, Sparse matrix entry $= 0.01n^2$ and $0.05n^2$. For each parameter, we run the algorithm 20 times, and get the average value of each parameter.

5.2 Result

5.2.1 Result of IALM

Dimension n	rank (L_0)	$\|S_0\|_0$	rank$(\hat{L})$	$\|\hat{S}\|_0$	#SVD	Time(s)
100	5	100	5.00	100.00	16.65	0.13
200	10	400	10.00	399.95	16.55	0.46
500	25	2500	25.00	2499.70	16.90	2.91
1000	50	10000	50.00	9998.20	19.00	18.80

rank $(L_0) = 0.05 \times n, \|S_0\|_0 = 0.01 \times n^2$.

Dimension n	rank (L_0)	$\|S_0\|_0$	rank$(\hat{L})$	$\|\hat{S}\|_0$	#SVD	Time(s)
100	10	100	10.15	116.10	21.05	0.15
200	20	400	20.00	399.95	19.40	0.58
500	50	2500	50.00	2499.45	18.85	3.36
1000	100	10000	100.00	9997.30	20.60	22.52

rank $(L_0) = 0.1 \times n, \|S_0\|_0 = 0.01 \times n^2$.

Dimension n	rank (L_0)	$\|S_0\|_0$	rank$(\hat{L})$	$\|\hat{S}\|_0$	#SVD	Time(s)
100	5	500	5.00	500.00	20.30	0.12
200	10	2000	10.00	1999.55	20.05	0.54
500	25	12500	25.00	12497.80	20.85	3.82
1000	50	50000	50.00	49990.50	22.05	21.06

rank $(L_0) = 0.05 \times n, \|S_0\|_0 = 0.05 \times n^2$.

Dimension n	rank (L_0)	$\|S_0\|_0$	rank$(\hat{L})$	$\|\hat{S}\|_0$	#SVD	Time(s)
100	10	500	10.20	530.40	26.30	0.20
200	20	2000	20.10	2020.45	24.10	0.68
500	50	12500	50.00	12497.05	23.25	4.45
1000	100	50000	100.00	49988.20	23.55	22.46

rank $(L_0) = 0.1 \times n, \|S_0\|_0 = 0.05 \times n^2$.

5.2.2 Result of ADMM

Dimension n	rank (L_0)	$\|S_0\|_0$	rank$(\hat{L})$	$\|\hat{S}\|_0$	#SVD	Time(s)
100	5	100	5.00	100.00	111.65	0.73
200	10	400	10.00	399.90	98.7	4.43
500	25	2500	25.00	2499.60	113.25	25.04
1000	50	10000	49.90	9998.20	107.85	115.77

rank $(L_0) = 0.05 \times n, \|S_0\|_0 = 0.01 \times n^2$.

Dimension n	rank (L_0)	$\|S_0\|_0$	rank$(\hat{L})$	$\|\hat{S}\|_0$	#SVD	Time(s)
100	10	100	10.00	99.95	127.9	1.10
200	20	400	20.00	399.95	115.25	4.95
500	50	2500	49.95	2499.75	124.1	27.85
1000	100	10000	99.95	9998.10	165.9	166.61

rank $(L_0) = 0.1 \times n, \|S_0\|_0 = 0.01 \times n^2$.

Dimension n	rank (L_0)	$\|S_0\|_0$	rank$(\hat{L})$	$\|\hat{S}\|_0$	#SVD	Time(s)
100	5	500	5.00	499.80	220.85	1.59
200	10	2000	10.00	1999.35	212.85	9.02
500	25	12500	25.00	12498.10	201.0	45.65
1000	50	50000	50.00	49988.50	198.6	192.41

rank $(L_0) = 0.05 \times n, \|S_0\|_0 = 0.05 \times n^2$.

Dimension n	rank (L_0)	$\|S_0\|_0$	rank$(\hat{L})$	$\|\hat{S}\|_0$	#SVD	Time(s)
100	10	500	9.95	499.50	279.15	1.97
200	20	2000	19.95	1999.65	244.45	10.49
500	50	12500	50.00	12497.75	215.1	48.25
1000	100	50000	100.00	49987.60	249.85	249.89

rank $(L_0) = 0.1 \times n, \|S_0\|_0 = 0.05 \times n^2$.

6 Conclusions and discussions

All the above two algorithms are augmented Lagrange multiplier based algorithms, though the penalty parameter does not approach infinity, both two algorithms have shown high accuracy.
Coded by matlab, the IALM algorithm performs much faster than the ADMM algorithm, which is coded in python. By inspection, we can observe that IALM have much less iterations than ADMM. However, when rank is relatively low(n=100), the IALM method gives out somehow inaccurate result (refering to the rank of L). This may be caused by the low-level implementation of matrix calculation.
Hyperparameter influences the speed of the algorithm significantly, a proper choice of Hyperparameter can speed up the algorithm by reduce the number of iteration. Further experiment is needed to determine the optimal value of hyperparameter for each matrix size.

7 Contribution

Yi implemented the IALM method and Yichuan implemented the ADMM method. Equal contribution for the report.

8 References

1. Ma, S., and Aybat, N. S. (2018). Efficient optimization algorithms for robust principal component analysis and its variants. Proceedings of the IEEE, 106(8), 1411-1426.

2. Maximilian Balandat, Walid Krichene, Chi Pang Lam, Ka Kit Lam. EE227 Project Report Robust Principal Component Analysis, 2012.

3. Candés, E. J., Li, X., Ma, Y., and Wright, J. (2011). Robust principal component analysis?. Journal of the ACM, 58(3), 1-37.

4. J. Wright, A. Ganesh, S. Rao, Y. Peng, and Y. Ma. (2009). Robust principal component analysis: Exact recovery of corrupted low-rank matrices via convex optimization. Advances in Neural Information Processing Systems, vol. 22. Available: http://papers.nips.cc/paper/3704-robust-principalcomponent-analysis-exact-recovery-of-corruptedlow- rank-matrices-via-convex-optimization.pdf.

5. Sobral, Andrews and Bouwmans, Thierry and Zahzah, El-hadi. LRSLibrary: Low-Rank and Sparse tools for Background Modeling and Subtraction in Videos. Robust Low-Rank and Sparse Matrix Decomposition: Applications in Image and Video Processing. CRC Press, Taylor and Francis Group, 2015.