STA663 Project: Biclustering via Sparse Singular Value Decomposition

Jishen Yin, Ziang Wang

April 2020

1 Abstract / Instructions

In this project, the statistical algorithm used in the paper "Biclustering via Sparse Singular Value Decomposition" by Lee, Shen, Huang and Marron is re-implemented in python environment. This algorithm(SSVD) imposes adaptive lasso penalty parameter for sparsity to make both the left and right singular vectors from SVD of the data matrix be sparse. This algorithm is tested with both 2 cases of simulated datasets and one real-world credit fraud transaction dataset. This algorithm is also compared with other two competing algorithms which are SVD and Sparse PCA. It's found that SSVD has good performance for both simulated and real datasets, and it performs even better than these two competitors.

The code and other related documents are available at: https://github.com/YinJishen/ssvd-package. The package has been uploaded to TestPypi, available at https://test.pypi.org/project/ssvd-pkg-JY-WZ/. One can install it with the command pip install -i https://test.pypi.org/simple/ ssvd-pkg-JY-WZ .

The real-world dataset used in this project is available at: https://www.kaggle.com/mlg-ulb/creditcardfraud.

This project was finished by Ziang Wang and Jishen Yin. The plain and optimized code of Sparse SVD was written and tested by both of us. The rest parts were split and finished separately. Jishen Yin finished "application on simulated data set" and "comparison among competing algorithm" while Ziang Wang finished "application on real data set" and wrapped up the final report.

2 Background

It's not a new observation that high dimensional data becomes much more common in a lot of applications. The statistical analysis would be much harder with additional dimension in the data. Multivariate analysis such as building multivariate normal model with Bayesian analysis is considered as classical way. However, the low sample size always becomes an issue. With the situation that high-dimension low sample size (HDLSS) offers new challenges, unsupervised learning is much more important.

The Biclustering method, a collection of unsupervised methods, that could simultaneously two patterns (Rows and Columns) in data matrix. This paper mainly focuses on sparse singular value decomposition (SSVD), which is a new tool for biclustering. The singular value decomposition refers to a matrix X that is $n * d$ whose rows represent samples and columns represent variables. The usual singular value decomposition (SVD) is generalized as the following:

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathbf{T}} = \sum_{k=1}^{r} s_k \mathbf{u_k} \mathbf{v_k^T}$$

where r is the rank of $\mathbf{X}$, $\mathbf{U}$ is a matrix of orthonormal left singular vectors, $\mathbf{V}$ is a matrix of orthonormal right singular vectors, and $\mathbf{D}$ is the diagonal matrix whose diagonals are positive singular values ranked from the largest to smallest. Each $s_k \mathbf{u_k} \mathbf{v_k^T}$ refers to a SVD layer. The sparse SVD means one only prefers SVD layers whose S_k are large. The rest of layers whose S_k are small would be considered noise and not much useful. With the first K large singular values we prefer, we have the following rank K aapproximation to $\mathbf{X}$, which is called the sparse singular decomposition:

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathbf{T}} \approx \mathbf{X}^{(\mathbf{K})} = \sum_{k=1}^{K} s_k \mathbf{u_k} \mathbf{v_k^T}$$

With such process, vectors $\mathbf{u_k}, \mathbf{v_k}$ are sparse, containing many 0 entries. The key algorithm of this paper is how to find the resulting components of the SSVD by adding sparsity-inducing penalties to the minimization objective, using adaptive lasso penalty.

It's also found that $\mathbf{X^{(k)}}$ minimizes the squared Frobenius norm, such that:

$$\mathbf{X^{(k)}} = \underset{\mathbf{X^* \in A_k}}{\arg\min} ||\mathbf{X} - \mathbf{X}^*||_F^2 = \underset{\mathbf{X^* \in A_k}}{\arg\min}\, tr\{(\mathbf{X} - \mathbf{X}^*)(\mathbf{X} - \mathbf{X}^*)^T\}$$

where A_k is the set of all $n \times d$ matrices of rank K.

The Biclustering via SSVD method could be widely used in medical and biological fields such as text categorization, medical imaging, and microarray gene expression analysis. The example given in this paper is to implement this method to find sets of biologically relevant genes that are expressed for certain cancer types with lung cancer data.

The main advantage of SSVD is that it could deal with the lack of samples. Also, it's easy to convert the data to a large matrix, with rows representing records or individuals and columns representing variables in the data frame. The key point of this SSVD algorithm is that unlike just taking large singular values and find corresponding vectors, it adds some penalties and constraints. It could take into account potential row-column interactions. With such a step, it would take much longer time to find results.

This algorithm provides a new aspect to the singular value decomposition. We could always implement this algorithm when trying to decompose the data in biological and medical field. It also inspires us to find if other penalties or constraints may work better in this algorithm.

3 Description of Algorithm

In this algorithm, it focuses on extracting the first SSVD layer, and it's iterative to continue updating the desired terms. First, it applies the standard SVD to X. Then it updates the s, $\mathbf{u}$ and $\mathbf{v}$ step by step until the update doesn't make significant changes. Then the final s, $\mathbf{u}$ and $\mathbf{v}$ would be the results of the sparse singular value decomposition. When updating $\mathbf{u}$ and $\mathbf{v}$ for sparsity, it solves the lasso penalty by using the sparsity-inducing penalty terms by the adaptive lasso penalties.

Based on the equation that the sparse matrix minimizes the squared frobenius norm and the focus on the first SSVD layer, we continue minimizing it by the following penalized criterion:

$$||\mathbf{X} - s\mathbf{u}\mathbf{v}^T||_F^2 + \lambda_u P_1(s\mathbf{u}) + \lambda_v P_2(s\mathbf{v})$$

where $P_1(s\mathbf{u})$ and $\lambda_v P_2(s\mathbf{v})$ are sparsity-inducing penalty terms.

More specifically, for fixed $\mathbf{u}$ with $\tilde{\mathbf{v}} = s\mathbf{v}$,

$$||\mathbf{X} - \mathbf{u}\mathbf{v}^T||_F^2 + \lambda_v P_2(s\mathbf{v}) = ||\mathbf{Y} - (\mathbf{I_d} \otimes \mathbf{u})\tilde{\mathbf{v}}||^2 + \lambda_v P_2(s\mathbf{v})$$

where $\mathbf{Y} = (\mathbf{x_1}^T, ...\mathbf{x_d}^T)^T \in R^{nd}$ with $\mathbf{x_j}$ being the j^{th} column of $\mathbf{X}$, and $\otimes$ being the Kronecker product. In the adaptive lasso penalties,

$$P_2(s\mathbf{v}) = s\sum_{j=1}^{d} w_{2,j}|v_j|$$

where

$$w_2 = (w_{2,1}, ..., w_{2,d})^T = |\hat{\tilde{\mathbf{v}}}|^{-\gamma_2}$$

where γ_2 is a known negative number, and $|\hat{\tilde{\mathbf{v}}}|^{-\gamma_2}$ is an operation to each component of the vector $|\hat{\tilde{\mathbf{v}}}|$. This vector is the OLS estimate of $\tilde{\mathbf{v}}$, which is $\mathbf{X}^T\mathbf{u}$. We could get the similar equations for for fixed $\mathbf{v}$ with $\tilde{\mathbf{u}} = s\mathbf{u}$.

It uses the BIC to find the penalty term, which is the following:

$$\mathbf{BIC}(\lambda_v) = \frac{||\mathbf{Y} - \hat{\mathbf{Y}}||^2}{nd\hat{\sigma}^2} + \frac{log(nd)}{nd} df(\lambda_v)$$

where $df(\lambda_v)$ is the degree of sparsity of v with λ_v as the penalty parameter, and $\hat{\sigma}^2$ is the OLS estimate of the error variance from the last equation. Similarly, use BIC to get λ_u.

To get the solution, we also need the following lemma:

Lemma 1 *The minimizer of* $\beta^2 - 2y\beta + 2\lambda|\beta|$ *is* $\hat{\beta}$ $=sign(y)(|y| - \lambda)_+$. *That is: if* $y > \lambda$, *then* $\hat{\beta} = y - \lambda$; *if* $y < -\lambda$, *then* $\hat{\beta} = y + \lambda$; *otherwise,* $\hat{\beta} = 0$.

The iterative algorithm is the following with parameters defined all above:

1. Apply the standard SVD to $\mathbf{X}$. Let $\{s_{old}, \mathbf{u}_{old}, \mathbf{v}_{old}\}$ be the first SVD componenet.
2. Update as the following:
   - Set $\tilde{v_j} = sign\{(\mathbf{X}^T\mathbf{u}_{old})\}_j(|\mathbf{X}^T\mathbf{u}_{old})_j| - \lambda_v w_{2,j}/2)_+$, where $j = 1, 2, ...d$ and λ_v is the minimizer of BIC(λ_v). Let $\tilde{\mathbf{v}} = (\tilde{v_1}, ...\tilde{v_d})^T, s = ||\tilde{v}||$ and $\mathbf{v}_{new} = \tilde{v}/s$.
   - Set $\tilde{u_j} = sign\{(\mathbf{X}^T\mathbf{v}_{new})\}_j(|\mathbf{X}^T\mathbf{v}_{new})_j| - \lambda_u w_{1,j}/2)_+$, where $j = 1, 2, ...d$ and λ_u is the minimizer of BIC(λ_u). Let $\tilde{\mathbf{u}} = (\tilde{u_1}, ...\tilde{u_d})^T, s = ||\tilde{u}||$ and $\mathbf{u}_{new} = \tilde{u}/s$.
   - Set $\mathbf{u}_{old} = \mathbf{u}_{new}$ and repeat step 2 until convergence.
3. Set $\mathbf{u} = \mathbf{u}_{new}, \mathbf{v} = \mathbf{v}_{new}, s = \mathbf{u}_{new}^T\mathbf{X}\mathbf{v}_{new}$ at convergence.

The final s, $\mathbf{u}$ and $\mathbf{v}$ are what this algorithm is looking for given the matrix $\mathbf{X}$.

4 Describe optimization for performance

The algorithm is implemented in plain Python as it is in the code notebook. Starting from the plain sparse SVD algorithm, we finished two types of modification:

1. **Use of vectorization**: The original algorithm updates each element of v_{new} and u_{new} by using the loop, the modified version vectorized this process.
2. **Use of better algorithm or data structures**: When finding the penalty parameter λ_u and λ_v, the plain version conducted a rough search followed by a precise search. However, when implementing this algorithm, we found that the rough search is enough for it to find the appropriate λ. So we deleted the precise search procedure, which improved the speed a lot but did not change the accuracy.

We then tested their performance in terms of the calculating speed. As the result shows, it takes about 22.7 seconds per loop for the original algorithm, while it takes about 9.62 seconds per loop for the optimized algorithm.

5 Applications to simulated datasets

It doesn't show the evidence that there are specific inputs that give known outputs.

In this section, we report two simulated study of sparse SVD. For both part, we set $\gamma_1 = \gamma_2 = 2$ and update λ_u and λ_v in each step with respect to BIC.

5.1 Case 1: Rank-1 matrix approximation

Consider a one-rank true signal matrix $X^* = suv^T$ with
$\tilde{u} = [10, 9, 8, 7, 6, 5, 4, 3, r(2, 17), r(0, 75)]^T, u = \tilde{u}/||\tilde{u}||$,
$\tilde{v} = [10, 10, 8, 8, 5, 5, r(3, 5), r(3, 5), r(0, 34)]^T, v = \tilde{v}/||\tilde{v}||$,
$s = 50$

A data matrix X is randomly created by adding X^* and a noise matrix ϵ, where each element is independently generated from standard normal distribution. This process is repeated 100 times. We mainly assessed the model's accuracy of distinguishing zero and nonzero entity in u and v.

Here's the summary of the results:

Items	u	v
Number of zeros	72.87	32.8
Correctly Labeled zero	97%	96.5%
Correctly Labeled non-zero	99.7%	100%
Misspecification rate	2.29%	2.4%

Table 1: Results of Case 1 of simulated datasets

The results shows that the average number of zeros in u is about 73, and the algorithm correctly labeled 97 percent of zeros in u. The number of zeros in is about 33 in average, and the algorithm correctly labeled 96.4 percent of zeros in v. It turns out that this SSVD algorithm performs pretty well in this rank-1 case. For example, in terms of correctly identifying the true zero and nonzero entries, on average it only misclassifies 1.5 percent and 1.0 percent of the entries in u and v, respectively.

5.2 Case 2: Higher rank approximation

Consider a true signal matrix X^* with shape 50×100, where each element $X^*_{ij} = T_{ij} I_{(|T_{ij}|>1)}$ with

$$T_{ij} = \begin{cases} [26^2 - (i-25)^2 - (j-50)^2]/100 & \text{if } 26 \leq j \leq 75 \\ 0 & \text{otherwise} \end{cases}$$

X^* is almost rank-2 in that its eigenvalues are almost zero except the first two. The figure below shows the structure of X^*, where the positive entries are red, the negative ones are blue, and the zeros are white.

A data matrix X is randomly created by adding X^* and a noise matrix ϵ, where each element is independently generated from standard normal distribution. This process is repeated 100 times. We mainly assessed the model's accuracy of distinguishing zero and nonzero entity in u and v as well as the distance of fitted matrix to the original matrix as the number of layers increases.

There are some figures that we generated to show results.

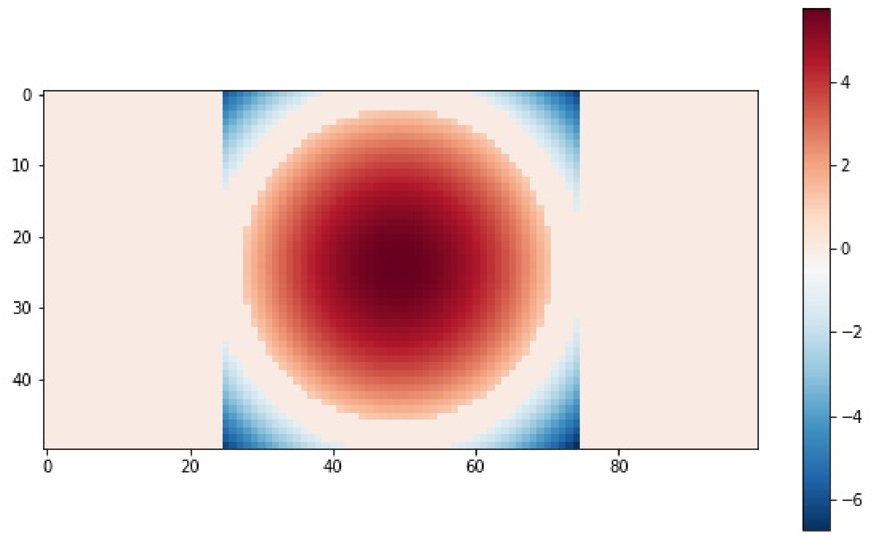

Figure 1: True Signal

Figure 1 highlights the structure of $\mathbf{X}^*$. The positive entries are red, the negative entries are blue, and the white part is zero.

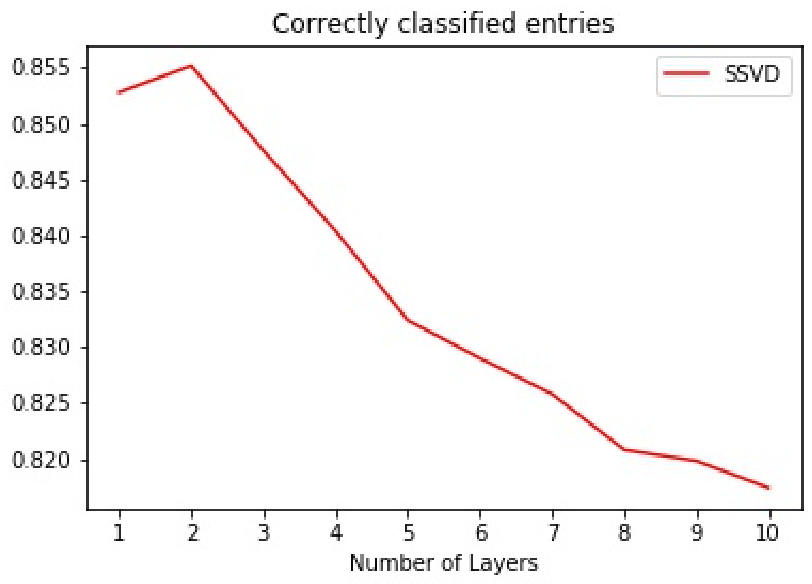

Figure 2: Proportion of the correclty identified entries

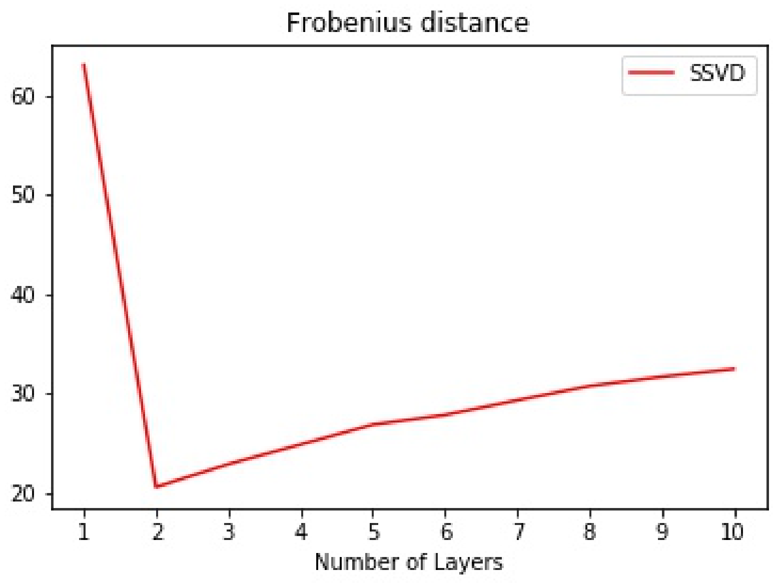

Figure 3: Frobenius distance

Figure 2 plots the proportion of the correctly identified entries of the SSVD algorithm. Figure 3 plots the Frobennius distance between the true signal matrix and the estimate, when the number of layers used in the estimation increases.

In this case, the Sparse SVD algorithm gives the best approximation when the number of layers is 2. After the minimum point, this model starts to overfit the data.

6 Applications to real data sets

The real datasets ”Lung Cancer data” used in the paper could not be found. So we provide another example of the real-world data that’s not in the paper.

This dataset contains credit card transactions made by European cardholders in September 2013. Some of them are fraud transactions. For privacy reason, this dataset only has numerical inputs and we don't know exactly what each variable represents. [2]

For convenience, we subset only first 100 rows of records to make it as the test matrix X. The shape of X is 100 by 28. Each row represents each record of transaction, while each column represents a feature.

Here's the summary of results:

Items	u	v
Number of zeros	45.07	15.55
Correctly Labeled zero	97.4%	96.8%
Correctly Labeled non-zero	80.7%	80.2%
Misspecification rate	13.8%	12.1%

Table 2: Results of the real-world dataset

Following the same testing procedure as we did in rank-1 approximation in the simulated datasets, we found that the performance is good, but not as good as what we have in the simulated datasets. The misspecification rates in u and v are about 0.14 and 0.12. We could see that this algorithm has a high accuracy of identifying 0 in both vectors.

When there're more 0 in both u and v, then it looks that this algorithm has a better performance in terms of labeling non-zero elements.

In overall, for both simulated datasets and real datasets, this SSVD shows evidence of good performance.

7 Comparative analysis with competing algorithms

In this section, We compared Sparse SVD algorithm's performance in simulated datasets with two competing methods, which are SVD and Sparse PCA algorithm. We followed the same testing algorithm used in previous two sections.

7.1 Case 1 analysis

Here is the summary of results:

Algorithms	No.of zeros(true)	Correctly identified zero	Correctly identified non-zero	misspecification rate
SSVD u	72.87(75)	97.1%	96.5%	2.30%
SSVD v	32.80(35)	99.7%	100.0%	2.40%
SPCA(u) u	75.58	99.7%	96.8%	1.04%
SPCA(u) v	0(35)	0.0%	100.0%	68%
SPCA(v) u	0(75)	0.0%	100.0%	75%
SPCA(v) v	34.28(35)	100.0%	98.3%	0.56%
SVD u	0(75)	0.0%	100.0%	75%
SVD v	0(35)	0.0%	100.0%	68%

Table 3: Comparison with competing algorithms for case 1

As one can see, SSVD performs much better than the competitors in all four metrics. The classic SVD can only identify non-zero entity without the ability to identify zero entities. For both SPCA(u) and SPCA(v), they can only demonstrate a compelling result in one dimension, either u or v. And they perform poorly in the other dimension.

7.2 Case 2 analysis

Here are some figures that show results:

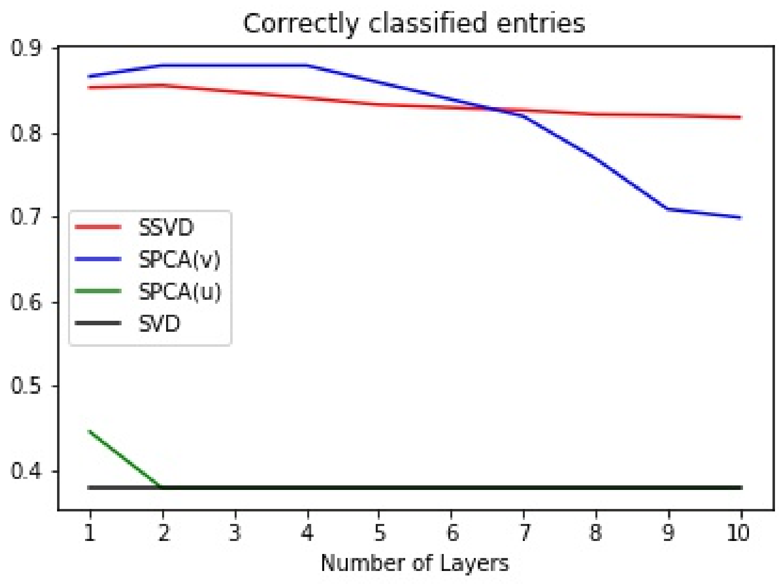

Figure 4: Proportion of the correctly identified entries for competing algorithms

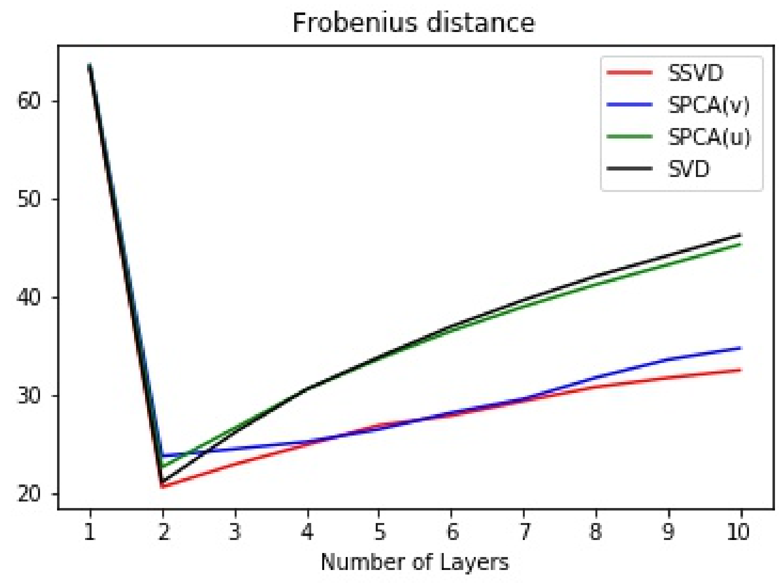

Figure 5: Frobenius distance for competing algorithms

In rank-2 case, SSVD and SPCA(u) performs well when classifying entities while SVD and SPCA(v) has trouble in detecting the sparse structure. Both SSVD and SPCA(u) arrive at maximum accuracy with 2 layers and start to overfit the data with more layers. The accuracy of SPCA(u) decreses rapidly after it exceeds 7 layers. With respect to Frobenius distance, all four methods perform well in approximating the true signal with minimum distance arrived at 2 layers. SSVD has the smallest distance with true signal. In conclusion, SSVD is the best option among these competing algorithms.

8 Discussion / Conclusion

This algorithm modified the singular value decomposition of a data matrix by imposing a sparse parameter on the left and right singular vectors. The sparsity of SVD could help to select important rows and columns and could also consider interactions between these rows and columns.

Based on the analysis on simulated datasets and the real-world dataset on credit card fraud transactions, we could see that this SSVD algorithm is quite effective and useful. This algorithm fulfills the need to research on row-column associations within high-dimensional data matrices when one seeks a low-rank structured matrix approximation. Researchers in Biology and medical fields may often need it.

There are still some issues with this algorithm. It uses the adaptive lasso penalty when doing the lasso regression. One may consider using other types of penalty for sparsity to seek for better performance. In addition, we've seen that this algorithm performs much better when there are many 0 entries in the left and right singular vectors. One may consider improving this algorithm to deal with the case when vectors don't have many 0 entries.

References

[1] Mihee Lee, Haipeng Shen, Jianhua Z. Huang and J.S. Marron (2010), *Biclustering via Sparse Singular Value Decomposition*, The International Biometric Society. Retrieved from: https://www.ncbi.nlm.nih.gov/pubmed/20163403

[2] Real Dataset, Machine Learning Group (ULB) (2018), Retrieved from: https://www.kaggle.com/mlg-ulb/creditcardfraud

[3] Shen, H. and Huang, J. Z. (2008). Sparse principal component analysis via regularized low rank matrix approximation. Journal of Multivariate Analysis 99, 1015–1034.