DUKE UNIVERSITY

Stats 663 Final Project
Implemention of Biclustering via Sparse Singular Value Decomposition

Rui Wang
Yan Zhao

1 ABSTRACTION

We implement sparse singular value decomposition algorithm(ssvd), which was proposed by Lee, Shen and Jianhua as a tool for clustering rows and columns, in other words bi-clustering. Applied on high-dimensional data matrices,this algorithm aims to make both left and right singular vectors sparse(having many zero entries).The adaptive lasso penalties are used to produce sparse singular vectors, as the author treated singular vectors as regression coefficients vectors for certain linear regressions. In this paper,we first introduce this algorithm's mathematical derivation. Then, based on using python to reproduce the original algorithm, we also tried different methods to optimize it, such as just in time compilation(JIT), multiprocess and ipyparallel. In addition, we apply this method on both simulated dataset and real lung cancer dataset, along with plots for visualization. In the end, we compare ssvd with other algorithms.

2 INTRODUCTION

In many applications such as text categorization, medical imaging and micro-array gene expression analysis, high dimensional dataset, which means high variable size, low sample size, appears to be very common. Under this scenario, the traditional statistical multivariate analysis does not work. In contrast, non-supervising methods perform more effective and efficient.

Non-supervising methods aim to group subjects that have similar features. Specifically, in data matrices,bi-cluster method allows to group both rows and columns simultaneously and to identify row-column associations. The paper"Bi-clustering via Sparse Singular Value Decomposition" introduces a new method to do bi-cluster by making each singular vector sparse.

The ssvd algorithm was applied on two real dataset in the original paper. However, as we can only access to one data set, we only implement the algorithm on that one. The lung cancer data is a typical high-dimensional dataset, with 12625 genes as variable size, but with only 56 patients as sample size. We do visualization to show that by using ssvd algorithm, groups of co-regulated genes for different cancer types can be identified.

Let X be the input dataset of a n*d matrix. The rows represent samples and columns represent variable genes. The singular value decomposition of X can be written as,

$$X = \mathrm{UDV}^T = \sum_{k=1}^{r} s_k \boldsymbol{u}_k \boldsymbol{v}_k^T \tag{2.1}$$

SVD decomposes X into a summation of rank one matrices, $\sum_{k=1}^{r} s_k \boldsymbol{u}_k \boldsymbol{v}_k^T$, each of which we call an SVD layer. We want layer with large s_k, and treat smalls_k as noise. If we take the first K rank-one summation in (2.1), we obtain the following rank K approximation to X:

$$X = UDV^T = \sum_{k=1}^{K} s_k \boldsymbol{u}_k \boldsymbol{v}_k^T \tag{2.2}$$

We want the approximation to minimize the squared Frobenius norm, which is

$$X^{(K)} = \underset{X^* \in A_K}{\operatorname{argmin}} \ \|X - X^*\|_F^2 = \underset{X^* \in A_K}{\operatorname{argmin}} \, tr\{(X - X^*)(X - X^*)^T\} \tag{2.3}$$

The ssvd not only seeks a low-rank matrix approximation to X, but also produce u_k and v_k to be sparse.

The rest of the paper is composed with following components. We first illustrate that ssvd algorithm's math basis, followed by implementation through python. Next, We invest on the running time of different functions in the python script through profiling and find parts that take long time. We then use different methods to optimize it. After we achieve a more optimal algorithm, we apply it on simulated data set to access its prediction accuracy. We also apply it on the lung cancer data set, doing visualization. Last but not least, we compare it with other algorithms that is designed for biclustering.

3 Description of algorithm

SSVD seeks a low-rank matrix approximation to the original matrix X similar to SVD, but with the requirement that the vector u_k and v_k are sparse, which means they both have many zero entries. In order to achieve sparsity, SSVD adds penalty to the objective function and implement iterative algorithm to minimize objective function. Without loss of generality, we take extracting first SSVD layer as an example to present the algorithm. Subsequent layers can be extracted sequentially from the residual matrices after removing the proceeding layers.

3.1 A penalized sum-of-squares criterion

We know that using SVD, $s_1 u_1 v_1^T$ is the best rank-one matrix approximation of X under the Frobenius norm,

$$(s_1, \boldsymbol{u_1}, \boldsymbol{v_1}) = \underset{s,u,v}{\operatorname{argmin}} \ \|X - s\boldsymbol{u}\boldsymbol{v}^T\|_F^2, \tag{3.1}$$

In order to achieve sparsity, SSVD method adds additional penalty to the objective function and minimize the following function

$$\|X - s\boldsymbol{u}\boldsymbol{v}^T\|_F^2 + \lambda_u P_1(s\boldsymbol{u}) + \lambda_v P_2(s\boldsymbol{v}), \tag{3.2}$$

where λ_u and λ_v are two non-negative penalty parameters that control shrinkage, and $P_1(s\boldsymbol{u})$ and $p_2(s\boldsymbol{v})$ are sparsity-inducing penalty terms. Two penalty parameters are used to allow different levels of sparsity imposed on $\boldsymbol{u}$ and $\boldsymbol{v}$. When $\lambda_u = \lambda_v$=0, the objective function is the same as that of SVD.

In this method, adaptive lasso penalties are used as sparsity-inducing penalty terms,

$$p_1(s\boldsymbol{u}) = s\sum_{i=1}^{n} \omega_{1,i}|\boldsymbol{u}_i| \qquad \text{and} \qquad p_2(s\boldsymbol{v}) = s\sum_{j=1}^{d} \omega_{2,i}|\boldsymbol{v}_j|, \tag{3.3}$$

where $\omega_{1,i}$ and $\omega_{2,i}$ are possibly data-driven weights. Weights can be chosen as

$$\omega_1 = (\omega_{1,1}, ..., \omega_{2,n})^T = |\tilde{\boldsymbol{u}}|^{-\gamma_1},$$

$$\omega_2 = (\omega_{2,1}, ..., \omega_{2,d})^T = |\tilde{\boldsymbol{v}}|^{-\gamma_2},$$

γ is an operation to each component of the vector $\tilde{\boldsymbol{v}}$. There are some candidates for $gamma$, for example, $gamma = 0$ corresponds to lasso fit. In this method, Bayesian information criterion(BIC) are introduced to select the penalty parameter γ. $\tilde{\boldsymbol{v}}_{\boldsymbol{j}} = s\boldsymbol{v}$

3.2 Iterative Algorithm

With the adaptive lasso penalty, the minimizing objective for SSVD can be written as

$$\|X - s\boldsymbol{u}\boldsymbol{v}^T\|_F^2 + s\lambda_u \sum_{i=1}^{n} \omega_{1,i}|\boldsymbol{u}_i| + s\lambda_v \sum_{j=1}^{d} \omega_{2,i}|\boldsymbol{v}_j|, \tag{3.4}$$

This objective function can be minimized with respect to $\boldsymbol{u}$ and $\boldsymbol{v}$ separately, which indicates an iterative algorithm.

For fixed $\boldsymbol{u}$, minimization of the original objective function is equivalent to minimizing

$$\|X - \boldsymbol{u}\tilde{\boldsymbol{v}}^T\|_F^2 + \lambda_v \sum_{j=1}^{d} \omega_{2,j}|\tilde{\boldsymbol{v}}_j| = \|X\|_F^2 + \sum_{j=1}^{d} \{\tilde{\boldsymbol{v}}_j^2 - 2\tilde{\boldsymbol{v}}_j (X^T\boldsymbol{u})_j + \lambda_v \omega_{2,j}|\tilde{\boldsymbol{v}}_j|\} \tag{3.5}$$

A closed-form solution to this minimization problem is $\tilde{\boldsymbol{v}}_j = sign\{(X^T\boldsymbol{u})_j\}(|(X^T\boldsymbol{u})_j| - \lambda_v\omega_{2,j}/2)_+$. Then we scale $\tilde{\boldsymbol{v}}_{\boldsymbol{j}}$ by letting $\boldsymbol{v} = \tilde{\boldsymbol{v}}/\|\tilde{\boldsymbol{v}}\|$

Similarly, for fixed $\boldsymbol{u}$, objective function can be wrote as

$$\|X - \tilde{\boldsymbol{u}}\boldsymbol{v}^T\|_F^2 + \lambda_u \sum_{i=1}^{n} \omega_{1,i}|\tilde{\boldsymbol{u}}_i| = \|X\|_F^2 + \sum_{i=1}^{n} \{\tilde{\boldsymbol{u}}_i^2 - 2\tilde{\boldsymbol{u}}_i (X\boldsymbol{v})_i + \lambda_u \omega_{1,i}|\tilde{\boldsymbol{u}}_i|\} \tag{3.6}$$

The closed-form solution to this minimization problem is $\tilde{\boldsymbol{u}}_j = sign\{(X\boldsymbol{v})_i\}(|(X\boldsymbol{v})_i| - \lambda_u\omega_{1,i}/2)_+$. Then we scale $\tilde{\boldsymbol{u}}_i$ by letting $\boldsymbol{u} = \tilde{\boldsymbol{u}}/\|\tilde{\boldsymbol{u}}\|$

3.3 Penalty parameter selection

We use BIC as criterion to select the penalty parameters.

For penalized regression with fixed $\boldsymbol{u}$,

$$BIC(\lambda_v) = \frac{\|Y - Y\|^2}{nd \cdot \sigma^2} + \frac{\log(nd)}{nd} df(\lambda_v), \tag{3.7}$$

For penalized regression with fixed $\boldsymbol{v}$,

$$BIC(\lambda_u) = \frac{\|Z - Z\|^2}{nd \cdot \sigma^2} + \frac{\log(nd)}{nd} df(\lambda_u), \tag{3.8}$$

$df(\lambda_v)$ is defined as the degree of sparsity, which is the number of zero components in $\boldsymbol{v}$ and similar for $\boldsymbol{u}$.

3.4 THE SSVD ALGORITHM

Step 1. Apply the standard SVD to X. Let $\{s_{old}, \boldsymbol{u}_{old}, \boldsymbol{v}_{old}\}$ denote the first SVD triplet.

Step 2. (a) Set $\tilde{\boldsymbol{v}}_j = sign\{(X^T \boldsymbol{u_{old}})_j\}(|(X^T \boldsymbol{u_{old}})_j| - \lambda_v \omega_{2,j}/2)_+$ where λ_v is the minimizer of $BIC(\lambda_v)$. Let $\boldsymbol{v}_{new} = \tilde{\boldsymbol{v}}/\|\tilde{\boldsymbol{v}}\|$

(b) Set $\tilde{\boldsymbol{u}}_j = sign\{(X\boldsymbol{v}_{new})_i\}(|(X\boldsymbol{v}_{new})_i| - \lambda_u \omega_{1,i}/2)_+$ where λ_u is the minimizer of $BIC(\lambda_u)$. Let $\boldsymbol{u}_{new} = \tilde{\boldsymbol{u}}/\|\tilde{\boldsymbol{u}}\|$

(c) Set $\boldsymbol{u}_{old} = \boldsymbol{u}_{new}$ and repeat until converge.

Step 3. Set $\boldsymbol{u} = \boldsymbol{u}_{new}$, $\boldsymbol{v} = \boldsymbol{v}_{new}$ and $s = \boldsymbol{u}_{new}^T X \boldsymbol{v}_{new}$ at convergence.

4 ALGORITHM OPTIMIZATION

In order to optimize the algorithm, we first use profiling to identify bottlenecks in the python script. Then we apply several optimization method such as JIT, parallelism and distributed computing.

4.1 PROFILING

Table 4.1: profiling of pure python code

function	ncalls	tottime	percall	cumtime	percall
SSVD	1	444.129	444.129	490.654	490.654
thred	76098	11.913	0.000	11.913	0.000
np.sum	152209	2.811	0.000	33.299	0.000
np.zeros	76098	1.198	0.000	1.198	0.000
la.svd	1	0.103	0.103	0.105	0.105

Table 4.1 reports the results of profiling. As we can see, the function that takes up the most running time is the main SSVD function. In order to speed up the main function, we can break the main function into several small functions and optimize each function separately. After that, 'thred' and 'np.sum' are functions that are called most frequently. This result suggests that two for loops in our main SSVD function cost much time and optimize them can improve the main function significantly.

4.2 JUST-IN-TIME COMPILATION

We first add a single jit decorator to the thred function. When tested on the simulation data, the performance is shown the table 4.2. As we can see, the improvement is not significant in this case. We also tried to rewrite the for loop as a function and used parallelism in numba to speed it up. However, it seems inappropriate since it's difficult to turn our main ssvd function into non python code to achieve parallelism.

Table 4.2: Just-in-time compilation

function	runtime(ms)
pure python	72.8
JIT	65.1

4.3 IPyParallel

We use IPyParallel to speed up two time-consuming loops in our main ssvd function. We send data to remote engines with push and use the dictionary interface to distribute a large lookup table to all engines once instead of repeatedly as a function argument. When tested on the lung dataset, the performance is shown in table 4.3. As we can see from the table, the runtime is reduced about 40%.

Table 4.3: IpyParallel

function	runtime(ms)
pure python	15min33s
IPyParallel	9min54s

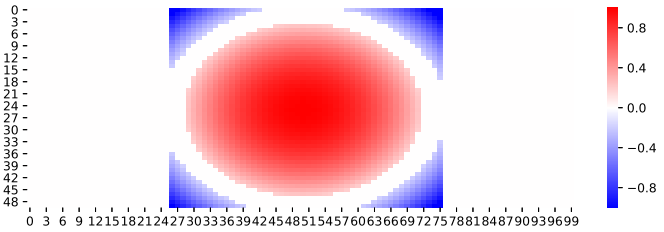

Figure 5.1: True Signal X

5 Applications to simulated data sets

In this section, we implement two simulation studies to investigate the performance of SSVD and to compare it with the standard SVD. The SSVD method performs much better than standard SVD in both rank-1 approximation as well as higher rank approximation.

5.1 Rank-1 Approximation

First, we consider a rank-1 matrix X^*. Let $X^* = s\boldsymbol{u}\boldsymbol{v}^T$ be a 100×50 rank-1 matrix with $s = 50$ and

$$\tilde{\boldsymbol{u}} = [10, 9, 8, 7, 6, 5, 4, 3, r(2,17), r(0,75)]^T, \quad \boldsymbol{u} = \tilde{\boldsymbol{u}}/\|\tilde{\boldsymbol{u}}\|,$$

$$\tilde{\boldsymbol{v}} = [10, -10, 8, -8, 5, -5, r(3,5), r(-3,5), r(0,34)]^T, \quad \boldsymbol{v} = \tilde{\boldsymbol{v}}/\|\tilde{\boldsymbol{v}}\|,$$

In this simulated dataset, $\boldsymbol{u}$ and $\boldsymbol{v}$ contain 25 and 16 nonzero entries respectively. A noise matrix ϵ is added to the matrix X^* to generate the data matrix X and the simulation is repeated 100 times. This model estimation is challenging in that the nonzero entries of X^* take on several small distinct values.

For SSVD, we use BIC to choose the degree of sparsity in the updating step, and use weight parameters $\gamma_1 = \gamma_2 = 2$ in deciding the weight vector ω_1 and ω_2.

As the estimation results show, SSVD correctly identified over 99% zeros and nonzero entries in both direction. More importantly, the misclassification rate is only around 1.00%.

5.2 Higher rank Approximation

In this simulation study, the true signal matrix X^* is a 50 by 100 matrix whose elements are given by $X^*_{ij} = T_{i,j}1(|T_{i,j}| > 1)$, where

$$T_{i,j} = \begin{cases} \{24^2 - (i-25)^2 - (j-50)^2\}/100, & 26 \le j \le 75 \\ 0, & \text{otherwise} \end{cases}$$

Figure 5.1 shows the complex structure of X^* where positive entries are red, negative ones are blue and zeros are white.

This matrix is more complex in that it does not have a multiplicative structure. Also, X^* is almost rank-2 in that its eigenvalues are almost zero except the first two.

The simulation results suggest that for standard SVD method, it can only correctly classified 38% entries no matter how many number of layers are extracted. For SSVD, it can correctly classified 78% of entries only with 2 layers.

6 Lung Cancer Data

We used ssvd algorithm on micro array gene expression dataset. Most of this type of data is high dimensional low sample size, because genes normally have thousand categories, d, while only few samples, n are available for measurement. The goal in this type of study is to identify several significant genes among a large pool of genes that are significantly expressed for certain diseases.

Specifically, the available data set is a 56*12625 matrix, with rows representing 56 patients, each one is either healthy or has one of the three types of Cancer(Carciniod, Colon and SmallCell), while column representing 12625 kinds of genes. The patients are grouped together according to their cancer type. We want to use ssvd to find which group of genes are significantly expressed in certain types of cancer.

We only evaluate on first three singular layers $s_k \boldsymbol{u}_k \boldsymbol{v}_k^T$, obtained through ssvd algorithm, because the first three singular values are much bigger than the rest.The results are sequentially shown in Figure 6.1. For each layer, all entries are scaled with value between[-1,1] by dividing all entries by its maximum entry. Rows and columns are both reordered to better visualize grouping. In each layer, subjects are reordered within its cancer type according to values of left sparse singular matrix $\boldsymbol{u_k}$ in a descending order, and Genes are reordered according to values of right sparse singular vectors $\boldsymbol{v_k}$ in a descending order.Gene selections are performed through ssvd by returning significant genes having corresponding value that is not zero. 3205,2511 and 1221 genes are selected in each of the three layers.The white area between two vertical dotted line in the graph represents genes that are omitted.The three horizontal dotted lines differentiate groups of cancer types

The figure 6.1 clearly shows different groups of genes are expressed among different types of cancer. For example, in layer one, Carcinoid and Small Cell have positive expression level for the first 1463 genes, while Colon has negative expression level for the same group of genes.In contrast, Carcinoid and Small Cell are negatively expressed for the gene that starts from 10883, while Colon has positive expression here. The obvious grouping pattern also presents in the 2nd layer, showing a contrast between Carcinoid/Normal and Colon/SmallCell. The third layer zeros out normal cell.

We also use scatter plots in Figure two to visualize grouping. The first graph show the relationship between first two sparse left singular vectors $\boldsymbol{u_k}$from first two layers. We identify three distinct cluster, which is correspond to three cancer type groups: Carciniod, Colon/SmallCell, and Normal.The second and third graph visualize the relationship between $\boldsymbol{u_1}$, $\boldsymbol{u_3}$ and $\boldsymbol{u_2}$, $\boldsymbol{u_3}$, respectively. Interestingly, these two graphs show that Carcinoid cancer has two clusters. In a word, those three sparse left singular vectors efficiently separate four types of subjects.

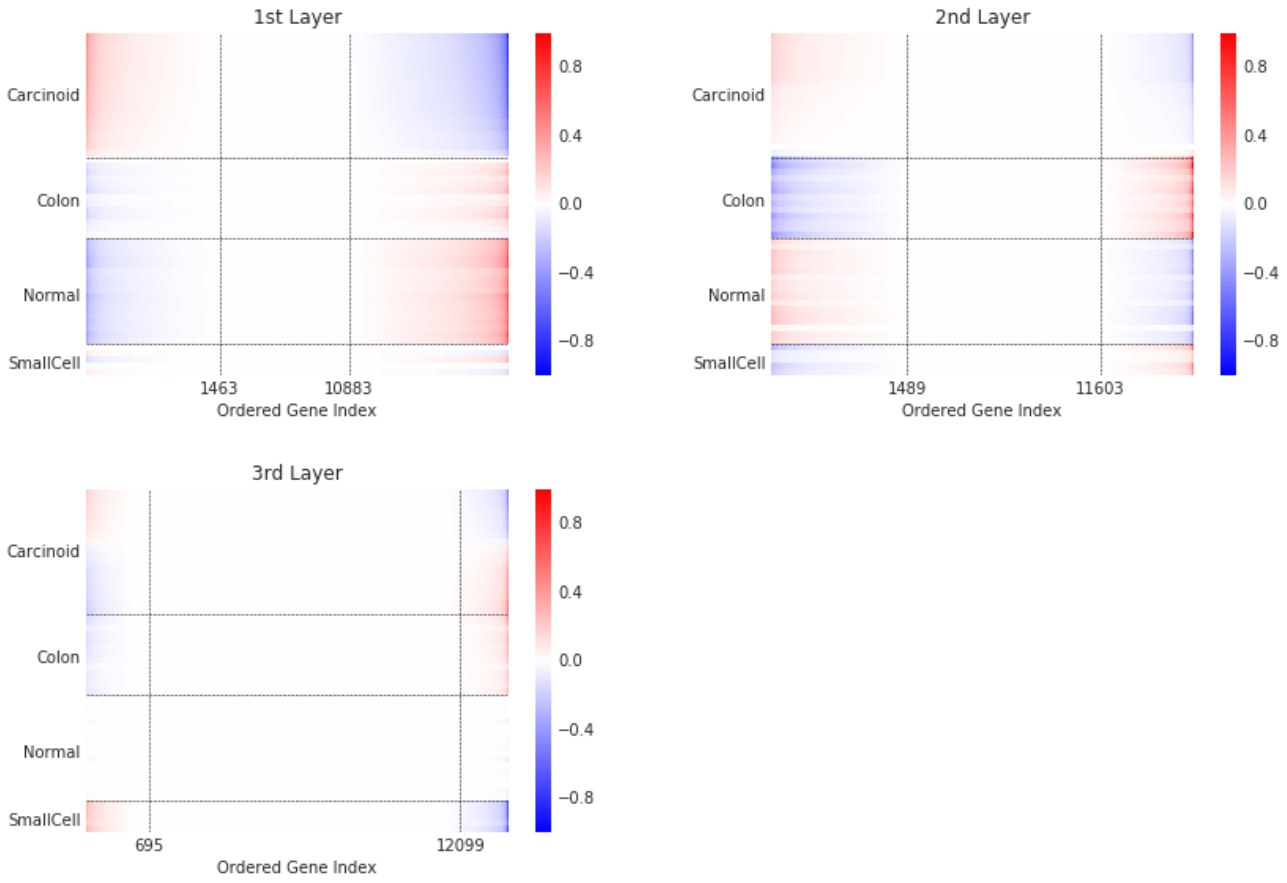

Figure 6.1: Heatmap of first three ssvd layers

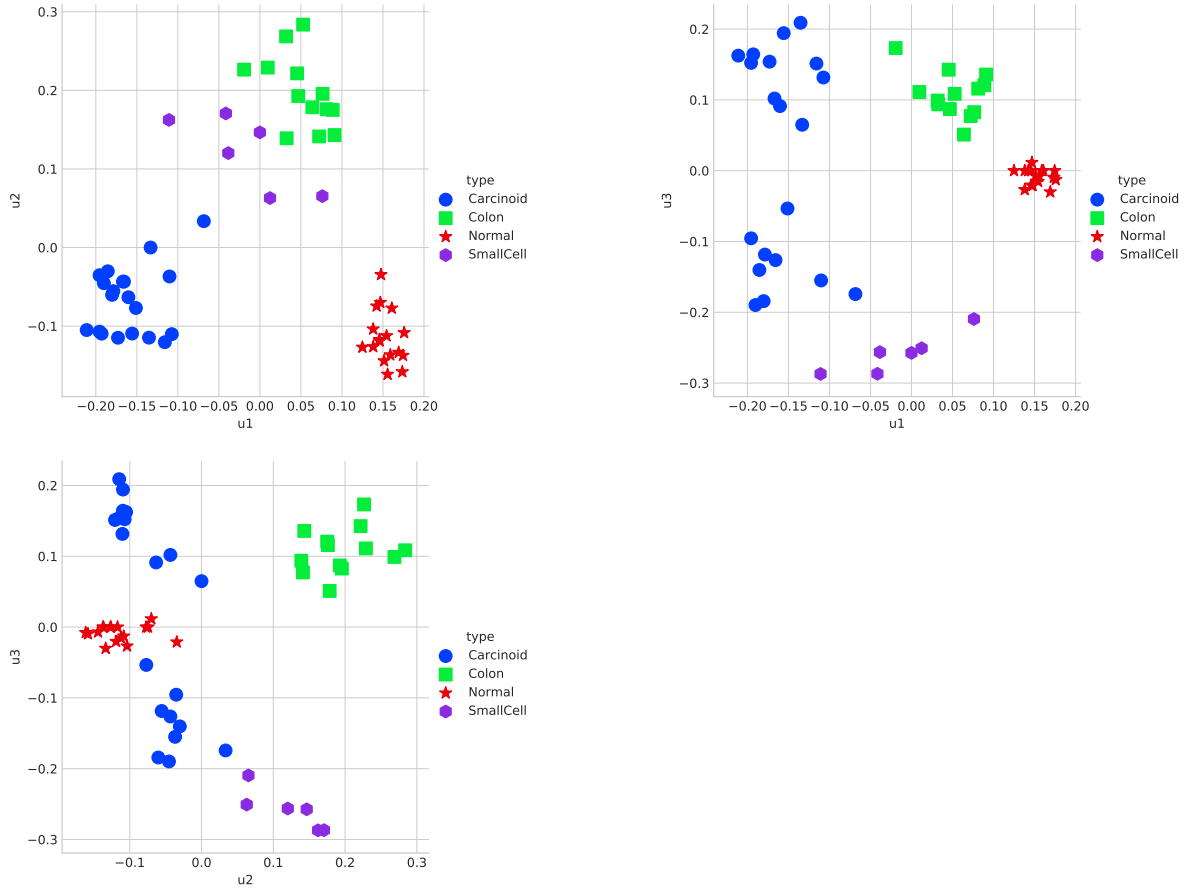

Figure 6.2: Scatter plots to show the first three left sparse singular vectors

7 Comparative analysis with competing algorihtms

There are another two existing biclustering methods, Plaid and ROBiC. Both methods assume that the data matrix can be approximated by a structure that is similar to that provided by SVD.

Plaid assumes that $X_{i,j} = \sum_{k=1}^{K} \theta_{ijk}\rho_{ik}\kappa_{jk} = \sum_{k=1}^{K} (\mu_k + \alpha_{ik} + \beta_{jk})\rho_{ik}\kappa_{jk}$, where K is the number of layers, and θ_{ijk} specifies the contribution of the kth bicluster. Plaid also employs an iterative procedure to obtain the layers, and the parameters are estimated by maximizing the reduction in the sum of squares.

RoBiC starts with the best rank-1 approximaion of the data matrix obtained from applying SVD. To obtain the first layer of biclustering, it looks at the first pair of singular vectors. The entries of the first left singular vector are ordered decreasingly in absolute value and then are fit by a hinge function. RoBiC uses an ad-hoc procedure to select the biclusters and hinge function model imposes an unnecessarily strict constraint that might limit the types of parse structures to be detected.

When applied to the simulation dataset, RoBiC and Plaid both detect too many zeroes. The problem is that the model used by these two methods have trouble separating the small nonzero entries from being zero. However, SSVD directly balances the goodness-of-fit measure with the sparsity and connects naturally with the variable selection.

Table 7.1: Comparison between SSVD and SVD

	Direction	Avg. # of zeros(true)	Age # of correctly identified zeros	Avg. # of correctly identified nonzeros	Misclassification rate
SSVD	$\boldsymbol{u}$	74.02	73.73(98.31%)	24.71(98.84%)	1.56%
SSVD	$\boldsymbol{v}$	33.76	33.76(99.29%)	16.00(100.0%)	0.48%
SVD	$\boldsymbol{u}$	0.00	0.00(0.00%)	25.00(100.0%)	75.00%
SVD	$\boldsymbol{v}$	0.00	0.00(0.00%)	16.00(100.0%)	68.00%

Table 7.1 reports the performance of SSVD and SVD on simulation dataset. We mainly care about the average number of zero elements in the estimated 100 singular vectors in both directions, the average number of correctly identified nonzero and zero entries as well as misclassification rate. As we can see, SSVD performs much better than the standard SVD. For example, in terms of correctly identifying the true zero and non zero entries, SSVD method only misclassifies 1% and 0.% on average. In general, SSVD is more rigorous and allows more general sparse structures.

8 Conclusion

We use python to implement SSVD algorithims. Using both JIT and parallel computing, we optimize this algorithm to make it run faster. Through not only applying the algorithm on both simulated dataset and real dataset, but also comparing it with other algorithms of same type, we conclude that this algorithm efficiently zero out entries, allowing us to do dimension reduction on high dimension low sample size data. You can find our pure python code, optimization code and reproducible examples on https://github.com/yzha0802/sparse-svd

References

[1] Lee M, Shen H, Huang J Z, et al. Biclustering via sparse singular value decomposition[J]. Biometrics, 2010, 66(4): 1087-1095.