pcamix2: An efficient R package of principal component method for a mixture of qualitative and quantitative variables.

Hsuan-yu Yeh[a], **Zheng Li**[a], **Qianhui Huang**[a]
[a]University of Michigan

Abstract. *PCAmixdata* is an R package for PCAMIX, a principal component method for a mix data of qualitative and quantitative variables. It is a new presentation of PCAMIX method based on Kiers' (1991) interpretation where the principal components and the squared loadings are obtained from a Singular Value Decomposition in a computationally efficient manner. However, the performance of the method does not work well with large datasets, especially when the number of variables (p) is large. In this context, we propose a modified version of this R package *pcamix2* with C++ implementation to improve the computational efficiency with large dataset while retaining the computational accuracy. A simulation study shows a good time efficiency of the proposed modification. An application on a real dataset with detailed presentation helps users to understand the input and the output of each R function. All source codes are available in the R package *pcamix2* with detailed help documentation.

1 Introduction

Multivariate data analysis refers to descriptive statistical methods used to analyze data arising from more than one variable. These variables can be either numerical or categorical. In general, we will use principal component analysis (PCA) for numerical variables and multiple correspondence analysis (MCA) for categorical variables. But for the datasets with both categorical and numerical variables, another approach needs to be employed. The R package *PCAmixdata* is dedicated to mixed data and provides the function PCAmix (PCA of a mixture of numerical and categorical variables), PCArot (rotation after PCAmix), MFAmix (multiple factor analysis of mixed multi-table data).[1] Here, we are focusing on the function **PCAmix**. It implements an algorithm based on generalized singular value decomposition and generates a squared loading matrix for downstream analysis such as varimax rotation and clustering classification.[2] Compared with the original method proposed by Kiers,[3] this package is more computationally efficient. However, in our simulation, the performance of this package does not work well with large datasets (especially when the number of variables is larger than the number of observations).[4] Here, we propose a re-implementation of the **PCAmix** function in C++ and aim to improve its computational efficiency in large datasets. In addition, we include a new feature to the original package by introducing a function **pcasimu** to generate a random dataset with both quantitative and qualitative variables for simulation studies. All modified functions and the new function are available in R package *pcamix2* we have complied.

2 Method and Algorithm

We will first introduce some notations used in this section.
Define:

- n as number of observation units, p_1 as the number of quantitative variables (p_2 for qualitative).
- m is the number of categories in p_2 variables.
- $\mathbf{z_j}$ is column vector containing standardized score of j-th quantitative variable.

- Let $\mathbf{G}$ be the matrix of the indicator variables of the m categories of the p_2 qualitative variables and let $\mathbf{D}$ be the diagonal matrix of frequencies of the m categories. Let $\mathbf{J}$ be the centering operator.
- $\mathbf{G = (G_1|\cdots|G_j|\cdots|G_{p2})_{n,m}} \quad \mathbf{D = diag(D_1,\cdots,D_j,\cdots,D_{p2})_{m,m}} \quad \mathbf{J = I_n - 11'/n}$

In the two following subsections, we will introduce two formulations of the PCAMIX method from Kiers'[3] and the package *PCAmixdata*.[1] We will highlight and focus on the differences in how they get the squared loading matrix. Squared loading matrix is defined as the squared correlation for quantitative variables and as the correlation ratio for qualitative variables with components. Squared loading matrix is used to plot the quantitative and qualitative variables on the same graph.

2.1 The Original PCAMIX Procedure

Let:

$$\mathbf{S} = \sum_{\mathbf{j=1}}^{\mathbf{p}} \mathbf{S_j}$$

where $\mathbf{S_j} = \frac{\mathbf{1}}{\mathbf{n}}\mathbf{z_j z_j^T}$ for $j \in$ quantitative and $\mathbf{S_j = JG_jD^{-1}G^TJ}$ for $j \in$ qualitative. Matrix $\mathbf{C}$ of the squared loadings of the p variables on k components will be $\mathbf{c_{jl}} = \frac{\mathbf{1}}{\mathbf{n}}\mathbf{x_l'S_j x_l}$, where $\mathbf{X'X = nI_k}$ (k be the number of first k eigenvectors of $\mathbf{S}$ obtained by Eigenvalue Decomposition) and $\frac{\mathbf{1}}{\mathbf{n}}\mathbf{x_l'S_j x_l}$ is the variance of the l-th component ($l : 1,\cdots,k$).
Note that, the squared loading matrix is obtained from the matrices $\mathbf{X}$ and $\mathbf{S}$ in Kiers' approach, which means that it requires the construction and the storage of p matrices of dimension nxn which can lead to memory size problem when n and p increases.

2.2 The SVD Based PCAMIX Procedure

Step 1: Pre-processing

$$\mathbf{Z} = \frac{\mathbf{1}}{\sqrt{\mathbf{n}}}(\mathbf{Z_1}|\mathbf{Z_2})$$

where $\mathbf{Z_1} = (\mathbf{z_1}|\cdots|\mathbf{z_j}|\cdots|\mathbf{z_{p_2}})$ and $\mathbf{Z_2 = JGD^{-1/2}}$.

Step 2: Factor coordinates processing with SVD

$$\mathbf{Z = U\Lambda V'} \quad \mathbf{A = V_k\Lambda_k} = \begin{bmatrix} A_1^* \\ A_2^* \end{bmatrix}$$

where $\mathbf{A_1^*}$ contains the factor coordinates of the p_1 numerical variables, $\mathbf{DA_2^*}$ contains the factor coordinates of the m levels(k be the number of first k eigenvectors of $\mathbf{U}$ obtained by Singular value decomposition).

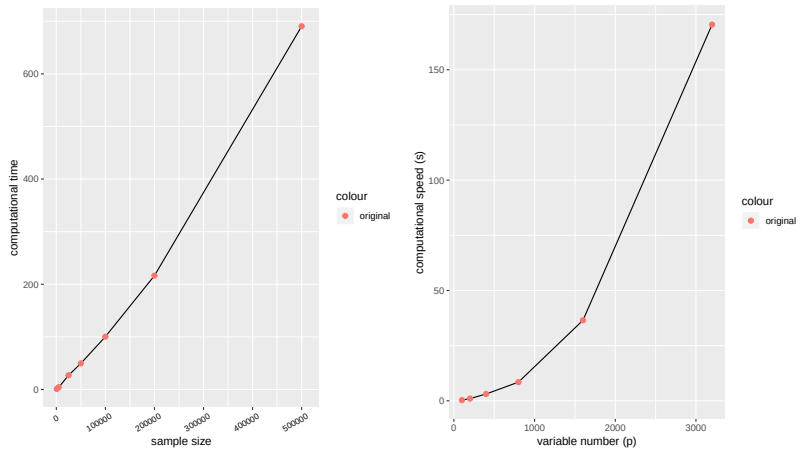
Step 3: Squared loading process

Matrix $\mathbf{C}$ of the squared loadings of the p variables on k components will be

$$\begin{cases} \mathbf{c_{jl} = a_{jl}^2} & \text{if variable } j \text{ is quantitative} \\ \mathbf{c_{jl} = \sum_{s\in I_j} a_{sl}^2} & \text{if variable } j \text{ is qualitative} \end{cases}$$

The squared loading matrix in the package *PCAmixdata* is calculated here from the only matrix $\mathbf{A}$ obtained with the SVD of $\mathbf{Z}$, which is more computationally efficient than Kiers' method. However, it still performs slow in large dataset in our simulation.

3 Modification



(a) (b)

Fig 1: Computation Time (a) versus sample size n; (b) versus variable number p.

Scatter plots and line charts are plotted for sample size (n) versus computational time and covariates number (p) versus computational time (Fig 1). From the figure, we can find in terms of n, the computational time increases in a linear trend whereas in terms of p, there exists an exponential or higher order relationship with time. With that in mind, we especially want to improve the performance of the algorithm with respect to p.

To speed up the algorithm, we firstly used "Rprof" function in R to get a sense about which parts of the algorithm are the most time-consuming. In the result, we find quite a large number of matrix manipulation steps (ex. apply, sweep, missing value checking, svd, qr etc.) and the "recodquant" function in our algorithm costs most part of the computation time. We re-implemented these functions using *Rcpp* and *RcppEigen* and compared their performance. As for basic matrix arithmetic, we also compared the performance of *Rcpp* vs *RcppEigen*. It turns out that in terms of simple matrix manipulations such as "apply", "sweep" or "missing value checking", *Rcpp* outperforms *RcppEigen* in time.

Also, we compared the time efficiency of R default "svd" and "qr" functions with those implemented in *RcppEigen*. In Eigen library, other than the standard JacobiSVD decomposition method, a much faster method BDCSVD is provided. It turns the input matrix to a bidiagonal form and utilizes the idea of divide and conquer to conduct a faster decomposition. Also in Eigen library, a column pivoting rank revealing method is provided to conduct QR decomposition. It more efficiently reveals the rank of a matrix than the default qr function in R and its advantage emerges especially when p is very large.

4 Numerical Results

In this section, the algorithms are compared on simulated data. Then, an application on a real data example illustrates the usage of *pcamix2* package.

4.1 Simulation

Dataset Generation:

```
library(pcamix2)
n <- c(5)
p <- c(6)
pcasimu(n,p)
```

```
##            X1          X2          X3      X4      X5      X6
## 1 -0.23597007 -0.02576136 -0.22389807 group41 group51 group61
## 2  0.24458615  0.24071311  0.27179533 group43 group53 group63
## 3  0.29806528  0.04380518  0.28840821 group43 group53 group63
## 4 -0.07880255  0.01265064 -0.07228501 group42 group51 group62
## 5 -0.30543484 -0.29779623 -0.35994498 group41 group52 group61
```

Fig 2: Simulated Dataset with n = 5 and p = 6

The computation time is compared from simulated datasets with various n and p. Since the original article did not provide a function for dataset generation. We built our own function to generate simulated datasets and conducted automatic simulation. Fig 2 shows an example of simulated data. The strategy of dataset construction is as follows:

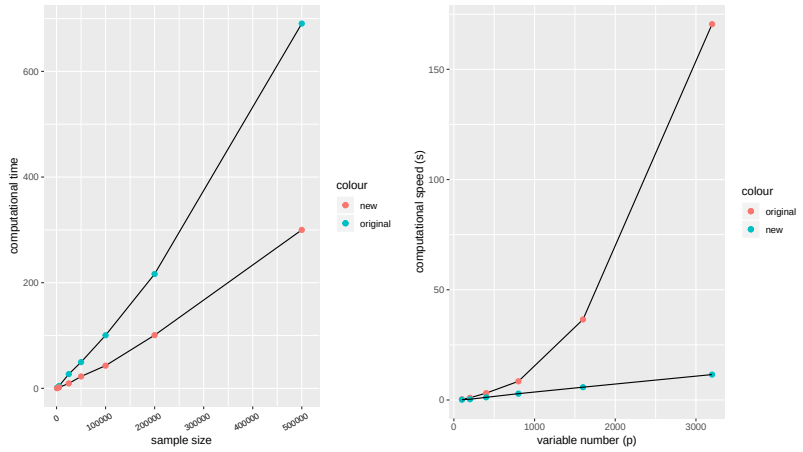
Step1: A dataset with n observations and p variables is generated based on a multivariate normal distribution with mean 0 and covariance matrix Σ. $\Sigma = Q^T Q$ where Q is sampled from a uniform distribution $Uni(0.2, 0.4)$.

Step2: The last $p/2$ covariates are converted from quantitative data to qualitative with each variable having three categories to generate a mixed dataset. The 0.33 quantile and 0.67 quantile are used to divide each variable into three groups. As a result, we can get a dataset with p/2 quantitative variables and $p/2$ qualitative variables and a total of $m = 3 * p/2$ categories.

Unit: (s)

		p = 100	p = 200	p = 400	p = 800	p = 1600	p = 3200
n = 1000	Original	0.32	1.01	3.1	8.47	36.52	170.50
n = 1000	New	0.13	0.33	1.2	2.87	5.8	11.53
n = 5000	Original	1.51	4.31	15.21	59.83	237.95	833.92
n = 5000	New	0.63	1.70	5.46	21.58	74.34	238.91
n = 25000	Original	7.92	26.98	79.94	279.81	1210.34	5459.13
n = 25000	New	4.03	9.34	31.64	112.31	416.42	1667.79
n = 50000	Original	16.77	49.59	157.96	577.52	2468.31	error
n = 50000	New	6.76	22.45	67.34	231.89	867.45	
n = 100000	Original	33.99	100.41	331.88	1224.02	error	
n = 100000	New	16.30	42.99	143.74	484.80		
n = 200000	Original	74.74	216.47	1056.36	error		
n = 200000	New	33.70	100.72	314.26			

Fig 3: Simulation Results



(a) (b)

Fig 4: Computation Time Comparison with (a) p fixed; (b) n fixed.

Performance Comparison: The numerical summary of the computational performance comparison is compiled in the table (Fig 3) and two plots (Fig 4) are drawn with fixed n (= 1000) and fixed p (= 200) respectively to compare the computation time in terms of each parameter. In the case of fixing p to compare n versus time, our modified algorithm is about 60% faster than the original one whereas in the case of fixing n to compare p versus time, our modified algorithm largely reduces the computation time especially when p is very large. As can be shown in the plot (Fig 4b), the computation time goes from an exponential or high order trend down to a linear trend. When p is large (= 3200), the computation time becomes 1/15 compared to the original algorithm. Notably, when n reaches 50000 and p reaches 3200, an error occurs due to memory exhaustion. Also, when n reaches 100000 and p reaches 1600, a similar error occurs. These two simulation results give a sense about the scope of our modified algorithm.

4.2 Real Data Application

```
library(PCAmixdata)
data(gironde)
head(gironde$housing)
```

```
##                    density primaryres   houses owners council
## ABZAC                131.70      88.77  inf 90%  64.23  sup 5%
## AILLAS                21.21      87.52  sup 90%  77.12  inf 5%
## AMBARES-ET-LAGRAVE   531.99      94.90  inf 90%  65.74  sup 5%
## AMBES                101.21      93.79  sup 90%  66.54  sup 5%
## ANDERNOS-LES-BAINS   551.87      62.14  inf 90%  71.54  inf 5%
## ANGLADE               63.82      81.02  sup 90%  80.54  inf 5%
```

```
split <- pcamix2::splitmix(gironde$housing)
x1 <- split$X.quanti
x2 <- split$X.quali
res <- pcamix2::PCAmix(x1,x2, graph = FALSE)
res$eig
```

```
##       Eigenvalue Proportion Cumulative
## dim 1  2.5268771  50.537541   50.53754
## dim 2  1.0692777  21.385553   71.92309
## dim 3  0.6303253  12.606505   84.52960
## dim 4  0.4230216   8.460432   92.99003
## dim 5  0.3504984   7.009968  100.00000
```

To illustrate the procedure of "PCAmix" function, we take the data table *housing* from the dataset *gironde*, which is contained in the original PCAmixdata package. There are n = 542 towns/villages in Gronde(France) and informations in the population density($population), the percentages of primary residences($primaryres), of houses($houses), of home owners living in their primary residence($owner) and of council housing($council) with $p_1 = 3$ numerical variables and $p_2 = 2$ categorical with a total of $m = 4$ levels.

First, the splitmix function splits the data set into two datasets, one with the numerical variables and one with the categorical variables. Then, we use the two data sets as the inputs of "PCAmix" function. The total number of eigenvalues for this dataset equal to $p_1 + m - p_2 = 5$ and the first two dimensions explain 71% of the total variation. To visualize the results, we take the first two eigenvectors as the coordinates.

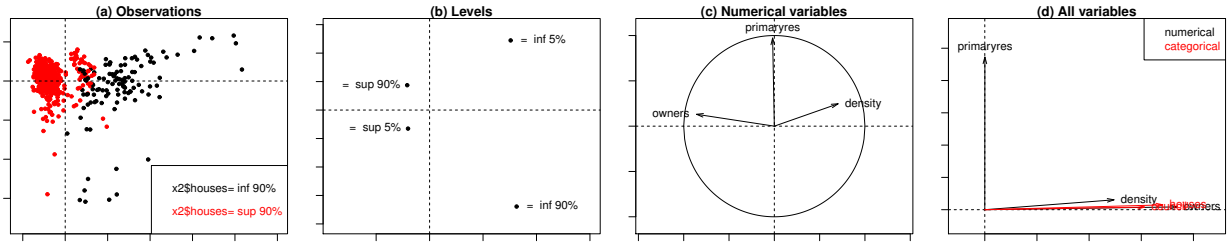


Fig 6: Graphical outputs of PCAmix applied to the data table housing.

In Fig (6a) we colored the towns by their percentage of houses(less than 90%, more than 90%). The first dimension highlights towns with large proportions of privately-owned properties on the left hand side. Fig (6b) confirms this interpretation and suggests that towns with high proportion of houses have a low percentage of council housing. Fig (6c) indicates that the percentage of home owners and the density is negatively correlated. Fig (6d) shows that the two numerical variables density and owners and the two categorical variables houses and council are linked to the first component. On the contrary, the variable primaryres is clearly orthogonal to these variables and associated with the second component.

In summary, towns on the right hand side of the two coordinates map have the most accommodation being rented. Towns on the left hand side are mostly composed of home owners living in their primary residence. The percentage of primary residences also stands important structuring role in the characterization of these towns by the second dimension, which means there may be some famous resorts in the towns with small values in the second coordinates.

5 Conclusion

Our new implemented package *pcamix2* is a more computationally efficient tool for analyzing large mixed-type-variables dataset, especially for large p cases, while retaining the same loading results as original package. The results of this *pcamix2* package can also be used in other clustering and modeling packages such as *ClustofVar* for dimension reduction.

Further improvement in terms of time efficiency for this package could be the varimax rotation as well as the multiple factor analysis part. The "PCArot" rotates the result of the PCA result in order to improve the explanation ratio of the first few eigenvalues. The "MFAmix" also uses the similar matrix decomposition method as "PCAmix". In addition, our new implementation still has limitation in terms of memory exhaustion (Fig 3), which could also be improved in the future.

References

1 M. Chavent, V. Kuentz, and J. Saracco, "Orthogonal rotation in pcamix.," *Advances in Data Analysis and Classification* **vol.6, no.2**, 131–146 (2012). [doi:10.1007/s11634-012-0105-3].

2 V. Chavent, M.and Kuentz and J. Liquet B., Saracco, "Clustofvar: An r package for the clustering of variables," *Journal of Statistical Software* **vol.50**, 1–16 (2012).

3 K. H. A.L., "Simple structure in component analysis techniques for mixtures of qualitative and quantitative variables," *Psychometrika* **vol.56, no.2**, 197–212 (June 1991). [doi:10.1007/bf02294458].

4 M. Chavent, V. Kuentz, and J. Saracco, "Multivariate analysis of mixed type data: The pcamixdata r package," *arXiv* (2012). [1411.4911v1].