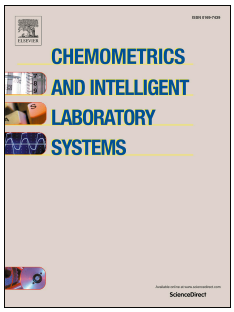
Hervás-Marín, D.; Prats-Montalbán, JM.; Lahoz Rodríguez, AG.; Ferrer, A. (2018). Sparse N-way partial least squares with R package sNPLS. Chemometrics and Intelligent Laboratory Systems. 179:54-63. https://doi.org/10.1016/j.chemolab.2018.06.005

Accepted Manuscript



Sparse N-way partial least squares with R package sNPLS

D. Hervás, J.M. Prats-Montalbán, A. Lahoz, A. Ferrer

PII: S0169-7439(18)30238-7

DOI: 10.1016/j.chemolab.2018.06.005

Reference: CHEMOM 3641

To appear in: *Chemometrics and Intelligent Laboratory Systems*

Received Date: 18 April 2018

Revised Date: 11 June 2018

Accepted Date: 16 June 2018

Please cite this article as: D. Hervás, J.M. Prats-Montalbán, A. Lahoz, A. Ferrer, Sparse N-way partial least squares with R package sNPLS, *Chemometrics and Intelligent Laboratory Systems* (2018), doi: 10.1016/j.chemolab.2018.06.005.

Sparse N-way Partial Least Squares with R Package sNPLS

Hervás, D.[a]; Prats-Montalbán, J.M.[b*], Lahoz A.[c*] and Ferrer, A.[b]

a) Biostatistics Unit, Health Research Institute La Fe, Valencia, Spain
b) Multivariate Statistical Engineering Group, Universitat Politècnica de València, Valencia, Spain
c) Biomarkers and Precision Medicine Unit, Health Research Institute La Fe, Valencia, Spain

* Corresponding author: José M. Prats-Montalbán. Departamento de Estadística e IO Aplicadas y Calidad. Universidad Politécnica de Valencia. Cno. De Vera s/n, Edificio 7A, 46022, Valencia, Spain. Tlf: +34.96.387.70.07 ext. 74949, Fax: +34.96.387.74.99. E-mail: jopramon@eio.upv.es

* Corresponding author: Agustín Lahoz. Biomarkers and Precision Medicine Unit, Analytical Unit (Metabolomics). Health Research Institute La Fe, Torre A-6-19. Avda. Fernando Abril Martorell, 106. 46026, Valencia, Spain. Tlf +34.96.124.66.52, Fax +34.96.124.66.20. E-mail: agustin.lahoz@uv.es

Abstract

We introduce the **R** package **sNPLS** that performs *N*-way partial least squares (*N*-PLS) regression and Sparse (L1-penalized) *N*-PLS regression in three-way arrays. *N*-PLS regression is superior to other methods for three-way data based in unfolding, thanks to a better stabilization of the decomposition. This provides better interpretability and improves predictions. The sparse version also adds variable selection through L1 penalization. The sparse version of *N*-PLS is able to provide lower prediction errors and to further improve interpretability and usability of the *N*-PLS results. After a short introduction to both methods, the different functions of the package are presented by displaying their use in simulated and a real dataset.

Keywords: *N*-PLS, LASSO, Sparse matrices

1. INTRODUCTION

N-way analysis refers to the analysis of data indexed and arranged in a *N*-dimensional array **X** (*I* x *J* x *K* x ...), instead of the regular two modes, which correspond to a two-dimensional matrix **X** (*I* x *J*). Several methods have been developed for dealing with three-way data, such as the Tucker3 (T3) model [1], the Candecomp/Parafac (CP) model [2] and the *N*-PLS model [3]. Of these, only *N*-PLS produces score vectors that maximize covariance with any *N*-way response array **Y**, so it is the only method that can be considered for prediction purposes.

A lot of development efforts have been put in the last years on improving and extending the properties of L1-penalized regression models [4, 5]. These models perform variable selection simultaneously to the fit, thus improving the interpretability of the results and also greatly reducing the variables involved in performing new predictions from the model.

The aim of this work is to show the different functions of the **R** package **sNPLS** for performing analysis of three-way data. In Section 2, we provide a methodological background for the *N*-PLS and sNPLS methods. Then, Section 3 presents the main functions `sNPLS()` and `cv.SNPLS()` and show how they work in a real dataset in Sections 4. We also present all the other supplementary functions of the package for plotting results and performing repeated cross-validation. Finally, we conclude with some remarks about the current status of the package and its future, as well as an external validation of the toolbox are made on Sections 5 and 6, respectively.

2. METHODOLOGICAL BACKGROUND

2.1. *N*-PLS

N-PLS studies relationships between some three-way (or *N*-way) **X** data structure and any **Y** data structure. It is the natural extension of PLS to *N*-way structures, which tries to maximize the covariance between **X** and **Y** data arrays.

Considering $\underline{\mathbf{X}}$ a three-way array of dimensions (I x J x K) and $\mathbf{X}$ (I x JK) its unfolded version, N-PLS tries to find latent spaces $\mathbf{W}^J$ and $\mathbf{W}^K$ that maximize the covariance between $\mathbf{X}$ and $\mathbf{Y}$, so it can be expressed as:

$$\mathbf{X} = \mathbf{T}(\mathbf{W}^K|\otimes|\mathbf{W}^J)^T + \mathbf{R} \quad (1)$$

Afterwards decomposing $\underline{\mathbf{X}}$ from $\mathbf{X}$ using the improved N-PLS version expression [6], in order to obtain residuals with better statistical properties:

$$\mathbf{X} = \mathbf{TGu}(\mathbf{W}^K \otimes \mathbf{W}^J)^T + \mathbf{R}' \quad (2)$$

In the same way, $\underline{\mathbf{Y}}$ can be decomposed by unfolding $\underline{\mathbf{Y}}$ (I x L x M) into $\mathbf{Y}$ (I x LM) as:

$$\mathbf{Y} = \mathbf{U}(\mathbf{Q}^M|\otimes|\mathbf{Q}^L)^T + \mathbf{R}'' \quad (3)$$

In this case, $\mathbf{W}^K$ and $\mathbf{W}^J$ refer to the weights of the third mode and of the second mode, respectively; whereas $\mathbf{T}$ matrix gathers the scores of the samples at each component extracted, in the 1st mode. $|\otimes|$ is the Khatri-Rao product and $\otimes$ the Kronecker product, which forbid or allow (respectively) to take interactions between the different modes components into account. **Gu** is the core array (unfolded) of a Tucker3 decomposition when using $\mathbf{T}$, $\mathbf{W}^K$ and $\mathbf{W}^J$ as loadings, in order to obtain a better (or at least not worse) approximation of the $\underline{\mathbf{X}}$ array [7]. Finally, **R'** incorporates the residuals. Analogously, **U** refers to the **Y** scores, and $\mathbf{Q}^M$ and $\mathbf{Q}^L$ to the loadings of the array $\underline{\mathbf{Y}}$, and **R''** the corresponding residuals.

Finally, from the scores $\mathbf{T}$ and $\mathbf{U}$, as well from the $\mathbf{W}$ weights, a $\mathbf{B}_{PLS}$ regression matrix can be obtained [8] so

$$\mathbf{Y} = \mathbf{XB}_{PLS} + \mathbf{R}''' \quad (4)$$

being **R'''** the final residuals.

2.2. Lasso

In linear models, where it first was developed for, L1 penalization consists in minimizing the usual sum of squared errors, with a bound on the sum of the absolute values of the coefficients [9]. It shrinks some coefficients and sets others to 0, and hence tries to retain the good features of both subset selection (interpretation) and ridge regression (stability and precision in estimations). The original LASSO for least squares is as follows:

$$\hat{\beta}^{lasso} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{P} x_{ij}\beta_j)^2 \tag{5}$$

Subject to the restriction:

$$\sum_{j=1}^{p} |\beta_j| \leq s$$

Increasing the penalization by reducing *s* forces the parameters to zero, producing a simpler model by deselecting some features. Thus, assuming data are standardized, Lasso automatically selects the most relevant features and discards the others.

To introduce the L1 penalization in the *N*-PLS algorithm we follow the approach of Lê Cao et al. [10] and make use of the soft-thresholding operator, which can be derived as a solution of the Lasso lagrangian form:

$$\hat{\beta}_i^{lasso} = sgn(\hat{\beta}_i^{LS})(|\hat{\beta}_i^{LS}| - \lambda)^+ \tag{6}$$

We introduce this operator at the $\mathbf{w}^K$ and $\mathbf{w}^J$ determination right after the SVD to achieve sparse versions of $\mathbf{w}^K$ and $\mathbf{w}^J$. A more detailed description of the algorithm can be found in [11].

3. MAIN FUNCTIONS IN THE R PACKAGE sNPLS

The package `sNPLS` provides functions for fitting *N*-PLS and sNPLS models, tuning the models using repeated cross-validation and plotting the results. It also provides

functions for extracting coefficients and performing predictions from new data. Below, we discuss the different functions providing code and different examples of use.

3.1 sNPLS function

Function sNPLS is used to fit *N*-PLS and sNPLS models to three-way data, depending on the input settings of the algorithm. The following R code shows an example of a model fit to a simulated three-way dataset.

```
R> library("sNPLS")
R> X_npls <- array(rpois(7500, 10), dim=c(50, 50, 3))
R> Y_npls <- matrix(2 + 0.4*X_npls[,5,1] + 0.7*X_npls[,10,1] –
+    0.9*X_npls[,15,1] + 0.6*X_npls[,20,1] - 0.5*X_npls[,25,1] +
+    rnorm(50), ncol=1)
R> fit <- sNPLS(X_npls, Y_npls, ncomp=3, keepJ = rep(2,3),
+    keepK = rep(1,3))
```

In this case, **X** is a three way array of dimension (50, 50, 3) formed by a Poisson distribution of lamda=10. On the other hand, **y** is a vector formed by a linear combination from **X**. Finally, sNPLS fits a sNPLS model with 3 components, with 2 variables per component retained in the second mode and 1 variable per component retained in the third mode.

Note that the function sNPLS needs a *N*-way array for **X** and any *N*-way array for **Y** as inputs. If data is in another format the function will throw an error. In the following, a generic call with an explanation of each of the arguments of the function is presented. Arguments with a defined value in the generic call will take that value as default if no other value is given when calling the function.

```
sNPLS(XN, Y, ncomp = 2, conver = 1e-16, max.iteration = 10000,
+    keepJ = rep(ncol(XN),ncomp), keepK = rep(rev(dim(XN))[1], ncomp),
+    scale.X = TRUE, center.X = TRUE, scale.Y = TRUE, center.Y = TRUE,
+    silent = F)
```

XN	*N*-dimensional array containing the predictors
Y	Array containing the response(s)
Ncomp	Number of components to use in the projection
conver	Convergence criterion
max.iteration	Maximum allowed number of iterations to achieve convergence
keepJ	Number of variables to keep at each component. If all variables are kept, *N*-PLS regression is performed, if any variable is removed then sNPLS is performed.
keepK	Number of elements of the third mode to keep at each component.
scale.X	Should unit variance scaling on X be performed?
center.X	Should mean centering on X be performed?
scale.Y	Should unit variance scaling on Y be performed?
center.Y	Should mean centering on Y be performed?
silent	Allows to choose if information regarding number of iterations should be displayed

The function `sNPLS` produces an S3 `sNPLS` object with defined `coef`, `predict` and `plot` methods that will be discussed later. The object consists of a list containing the following components: (1-6) the **T**, $\mathbf{W}^J$, $\mathbf{W}^K$, **B** (regression coefficients between $\underline{\mathbf{X}}$ and $\underline{\mathbf{Y}}$), **Y** and **Q** matrices, (7-8) **P** and **Gu** (the unfolded **G** core array of the Tucker decomposition), (9) The number of components, (10) Fitted values, (11) Squared error, (12) Scale and centering information performed on $\underline{\mathbf{X}}$ and **Y**.

3.2 `cv_snpls` function

Selecting parameter values for the `sNPLS` function requires choosing values for the number of components, the number of variables to select and the number of elements of the third mode to select. An appropriate way of selecting these parameters is performing a grid search with cross-validation. The function cv_snpls performs cross-validation on a grid of different `ncomp`, `keepJ` and `keepK` values estimating RMSE (Root Mean Square Error) for each combination of values and selecting the best set producing the lowest RMSE.

```
R> X_npls <- array(rpois(7500, 10), dim=c(50, 50, 3))
```

```
R> Y_npls <- matrix(2 + 0.4*X_npls[,5,1] + 0.7*X_npls[,10,1] –
+    0.9*X_npls[,15,1] + 0.6*X_npls[,20,1] - 0.5*X_npls[,25,1] +
+    rnorm(50), ncol=1)
R> cv1 <- cv_snpls(X_npls, Y_npls, ncomp=1:2, keepJ = 1:10,
+    keepK = 1:3, parallel = FALSE)
```

The generic call for `cv_snpls` is the following:

```
R> cv_snpls(X_npls, Y_npls, ncomp = 1:3, keepJ = 1:ncol(X_npls),
+   keepK = 1:dim(X_npls)[3], nfold = 10, parallel = FALSE)
```

It contains the same parameters as `sNPLS` but now admits vectors of values for each parameter in `ncomp`, `keepJ` and `keepK`. The number of subsets for the cross-validation is determined by the parameter `nfold`. This `nfold` parameter allows the specification of the number of folds of the cross-validation procedure, with 10-fold being the default Since grid search can be computationally intensive, it makes use of the parallel package for R. Being able to perform computations in parallel greatly reduces running times. Parallel mode is activated by setting `parallel` argument to `TRUE` and selecting a suitable number of free cores. In the case of `cv_snpls`, the parallelization applies to the grid search, not to the different folds of the cross-validation. Figure 1 shows the improvements in computing times by using parallelization with different number of cores. To further reduce computation times and improve memory use, sparse matrices from the R package Matrix [12] are used whenever possible in the matrix multiplication steps of the function. Sparse matrices achieve these goals by using an alternative representation to that of dense matrices: instead of being stored as two-dimensional arrays, only their non-zero values are stored, along with an index linking these values with their location in the matrix. The function `cv_snpls` returns a `cvsnpls` object which is a list with a component containing the best combination parameters and other components containing information about the grid and its corresponding RMSE. `cv_snpls` objects can be plotted for better interpretation of the results. The resulting plot is a grid of scatterplots with the different combinations of `keepJ`, `keepK` and `ncomp` and their resulting cross-validation errors (Figure 2).

[INSERT FIG. 1 ABOUT HERE]

A known issue of cross-validation is the variance in its results [13] which, on one hand entails that different runs of the cross-validation procedure can yield different results regarding the estimated best set of parameters (variability) and, on another hand, that the estimation of the best set of parameters is prone to overfitting (bias-variance tradeoff). A reasonable solution is performing repeated cross-validation and selecting the most frequently selected set of parameters along a round of different runs. The function `repeat_cv` performs repeated cross-validation by calling the function `cv_snpls` repeated times and storing each result in a `data.frame` object. The syntax is the same as in `cv_snpls` with an extra argument `times` for specifying the number of repetitions to perform. In the case of `repeat_cv` the parallelization applies to the replicates instead of applying to the grid search. The following code explains how to perform repeated cross-validation with `repeat_cv` (10 times in this case) varying from 1 to 2 components, and from 1 to 10 variables in the second mode, and from 1 to 3 in the third mode.

```
R> repcv <- repeat_cv(X_npls, Y_npls, ncomp=1:2, keepJ = 1:10,
+    keepK = 1:3, parallel = FALSE, times=10)
```

[INSERT FIG. 2 ABOUT HERE]

The result of the `repeat_cv` call is a `data.frame` storing the results of each cross-validation repetition (Table 1).

[INSERT TABLE 1 ABOUT HERE]

This output can be presented as a cross-tabulation table (Table 2) with the absolute frequencies of each combination of the different parameters. In our example, the table shows that the combination `ncomp`=2, `keepJ`=9 and `keepK`=1 is the most recurrent (3 out of 10 repetitions).

[INSERT TABLE 2 ABOUT HERE]

Results of the `repeat_cv` function can also be plotted to obtain a kernel density plot, which can be one-, two- or three-dimensional depending of the number of constant parameters obtained in the repeated cross-validation procedure. This density plot depicts the most frequently selected parameters (Figure 3). In this example, according to the plot, the most likely combination after all repetitions of the cross-validation was between 9 and 10 in the case of `keepJ` and `keepK`=1. Since the optimal number of components did not vary along all the repetitions, this parameter is not represented in the plot. Instead, the message `'ncomp is(are) constant with a value of 2'` is returned by the function. The density plot leads to a similar interpretation of the results as the cross-tabulation table, but the smoothing can result in more sensible estimates in the case of multiple and/or wide spread modes. It also provides a visual measure of the uncertainty in the selection of the best combination of parameters.

[INSERT FIG. 3 ABOUT HERE]

4. REAL DATASET ANALYSIS

To exhibit all package functionality, next we provide a complete analysis of the `bread` dataset [14] included in the **sNPLS** package. This dataset consists on data of five different breads that were baked in duplicate giving a total of ten samples. Eight different judges assessed the breads with respect to eleven different sensorial attributes. The data can be regarded as a three-way array (10 × 11 × 8). The data are quite noisy as opposed to, e.g., spectral data. The salt content of each bread was also measured, and it is considered as the response variable **y**.

```
R> data(bread)
R> Xbread <- bread$Xbread
R> Ybread <- bread$Ybread
R> cv_bread <- repeat_cv(Xbread, Ybread, ncomp=1:3, keepJ = 1:11,
+    keepK = 1:8, parallel = TRUE, times=50, nfold=3)
```

In this case, the number of components tested vary from 1 to 3, whereas all possible number of variables kept are considered both in the second and in the third mode. Repeated cross-validation is performed on the data to select the optimal parameters. 50 repetitions should be enough to get stable estimates, but using such a large number of repetitions can increase computing times dramatically. Therefore, using the option `parallel=TRUE` when performing repeated cross-validation is recommended. Results of this procedure yield a cross-tabulation table with the appearance frequencies for each combination of parameters. One can also adjust the number of folds of the cross-validation procedure with the `nfold` parameter, with 10-fold being the default. Table 3 shows the frequency table derived from the data.frame obtained in this case.

[INSERT TABLE 3 ABOUT HERE]

This table shows that there are two possible combinations of parameters that are almost equally selected by cross-validation. One is `ncomp`=1, `keepJ`=1 and `keepK`=4 and the other is `ncomp`=2, `keepJ`=3 and `keepK`=8, both with 4 appearances out of 50 repetitions. Also, most of the other combinations are close to one of these two mentioned 'hotspots'. A plot of the `cv_bread` object (Figure 4) reveals a similar information with a representation of a sliced three-dimensional kernel density estimate.

[INSERT FIG. 4 ABOUT HERE]

Next step would be fitting the sNPLS model using the `sNPLS` function with the selected set of parameters. As discussed before, our results point to two possible combinations and, based on the density plot, the option with `ncomp`=1, `keepJ`=1 and `keepK`=4 seems more likely. Nevertheless, since we are using the data for demonstration of the different functions of the package, we will use the combination with `ncomp`=2 to get working examples of the `plot` function (which needs, at least, two dimensions). It is important to take into account that `keepJ` and `keepK` have to be specified for each component, so they must be a vector of length equal to the number of components. Note that, in this case, we have chosen the same number of attributes and judges for each component, although this is not necessarily the case.

```
R> fit <- sNPLS(Xbread, Ybread, ncomp = 2, keepJ = rep(3, 2),
```

```
+    keepK = rep(8, 2), silent = FALSE)
```

The summary of the fit (Table 4) shows the number of components, the estimated squared error and a matrix with rows corresponding to attributes (second mode) and columns corresponding to judges (third mode). In this matrix there are five rows with non-zero coefficients which correspond to the 4th, 6th 7th 8th and 9th attributes from all the judges (no selection is performed on the third mode).

[INSERT TABLE 4 ABOUT HERE]

To better understand and interpret the results, the plot function can be used to display different visualizations of the results. Values of the **T** (Figure 5), **U** (Figure 6), $\mathbf{W}^J$ (Figures 7 and 9), and $\mathbf{W}^K$ (Figures 8 and 10) matrices can be plotted by changing the `type` parameter of the function. 1st and 2nd components are plotted by default, but they can be changed using the `comps` parameter.

```
R> plot(fit, type="T", cex.axis=1.2, cex.lab=1.2, cex=1.2, las=1,
+    bty="L")
R> plot(fit, type="U", cex.axis=1.2, cex.lab=1.2, cex=1.2, las=1,
+    bty="L")
```

[INSERT FIG. 5 ABOUT HERE]

Figure 5 shows how the 10 samples spread over the two first components. It can be seen how the samples evolve every two observations, from left to right, which seems reasonable attending to their different composition. Figure 6 is similar, but related to the **y** scores.

[INSERT FIG. 6 ABOUT HERE]

```
R> plot(fit, type="Wj", cex.axis=1.2, cex.lab=1.2, cex=1.2, las=1,
+    bty="L")
R> plot(fit, type="Wk", cex.axis=1.2, cex.lab=1.2, cex=1.2, las=1,
+    bty="L")
```

[INSERT FIG. 7 ABOUT HERE]

On the other hand, Figure 7 presents the attributes and Figure 8 the judges that help in predicting the **y** variable, for each of the two components. It can be seen that the first component of the attributes mode is related to attributes 7, 8 and 6 (in descending order in absolute values); whereas attributes 4, 9 and 6 are related to the second component. This can be also derived from Figure 9, which produces an equivalent graph. In this case, when trying to see what kind of relationship these attributes have with the judges' mode, Figure 10 provides easier-to-interpret results.

[INSERT FIG. 8 ABOUT HERE]

```
R> plot(fit, type="variables", cex.axis=1.2, cex.lab=1.2, cex=1.2,
+    las=1, bty="L", lwd=2)
R> plot(fit, type="time", cex.axis=1.2, cex.lab=1.2, cex=1.2, las=1,
+    bty="L", lwd=2, xlab="Judge")
```

[INSERT FIG. 9 ABOUT HERE]

It can be seen how, for the first component, there is approximately the same effect of all judges with respect variables 7, 8 and 6 (those related to the first component in the attributes mode) when trying to predict the salt content. In this case, since the judges' weights show positive values, the higher the value of attributes 7 and 6, and the lower the value of attribute 8, the higher the salt content. For the second component, attributes 4, 9 and 6 are more influenced by judge 5, even though the rest of judges also have a similar effect on the salt content scoring (**y** variable). Since the weights of the judges' mode are all negative for the second component, the higher the value of attributes 4, 9 and 6, the lower the salt content. In this case, Figure 8 is less interpretable. However, depending on the case, one representation or another might provide easier-to-interpret results; so it is decided to keep both graphs in the package.

[INSERT FIG. 10 ABOUT HERE]

As seen on these examples, all plots produced by the `plot` function are fully customizable using base R `plot` parameters such as `las`, `cex`, `bty`, etc.

The `predict` function can be used to make predictions using new data. Here we use it to make a prediction from a new **X** array with 6 new observations. This function also has an optional parameter `scale` which defaults to `TRUE` for controlling the final scale of the predictions (original vs. scaled). Table 5 shows the prediction performed by the model.

```
R> newX <- array(sample(0:5, 6*11*8, replace=TRUE), dim=c(6, 11, 8))
R> predict(fit, newX)
```

[INSERT TABLE 5 ABOUT HERE]

The output of the function is a **Y** matrix with the 8 predicted values.

4.1 Standard *N*-PLS analysis

For comparison, we also can perform a standard *N*-PLS analysis on this dataset. By calling the `sNPLS` function with its default arguments, no L1-penalization will be applied to the $\mathbf{w}^J$ and $\mathbf{w}^K$ vectors and the $\mathbf{W}^J$ and $\mathbf{W}^K$ matrices will be dense. Table 6 shows the corresponding coefficients.

```
R> fit2 <- sNPLS(Xbread, Ybread, silent = F)
R> summary(fit2)

sNPLS model with 2 components and squared error of 0.077
```

[INSERT TABLE 6 ABOUT HERE]

As expected, the coefficients' matrix is now dense, since no L1-penalization was applied. The different plots show also the same effect, with no estimation being equal to zero (Figure 11).

[INSERT FIG. 11 ABOUT HERE]

By comparing Figures 11(a) and 11(b) with Figs 5 and 6, very similar evolution of the scores along the first component (the one relevant) can be seen, mostly in the **U** scores (Figs 5 and 11(b)). However, when inspecting Figure 11(c) or 11(e), the relevance of the different attributes is not so easy. When comparing it with Fig 7 or 9, it can be seen that the most relevant attributes are 7 and 6 in the positive part of the plot, and 8 in the negative part. Nevertheless, attribute 6 is quite close to attribute 5, and 8 to 10, etc. So interpretation or selection of attributes is not direct. In this case, for the judges, very similar results are drawn from Figures 11(d) and 11(f) when compared to Figures 8 and 10. This is due to the fact that no selection was already performed in the judges' mode. However, in other problems, this might not be the case.

5. FINAL REMARKS

The most relevant features of the R package sNPLS have been presented. The package offers a complete set of functions for tuning, fitting and interpreting *N*-PLS and sNPLS models. As tuning is a computationally intensive method, all cross-validation functions in the package allow the use of parallelization through the parallel package and also use sparse matrices from the Matrix package. These two optimizations allow for speedups of up to 20 times faster computation times compared to non-parallelized computations with dense matrices. This paper refers to the version 0.3.31 of the sNPLS package, which is available at CRAN (The Comprehensive R Archive Network). Future versions of the package will implement different regularization methods apart from *L1*-penalty and also provide more tools to analyze three-way data.

6. VALIDATION

Dr. Marco Calderisi. Administratore unico. Kode Srl. Via Nino Pisano 14 - 56124 Pisa.

The **sNPLS** R package introduced in this article allows for the application of the *N*-way partial least squares algorithm to three way arrays, with the addition of the possibility to use sparse (L1 penalized) models. The package is very straightforward to install and use. All the functions are well documented, and the examples provided make it easy to understand how to use all of the functionality of the package. The main function of the package, **sNPLS**(), can be used to perform a regularized L1-penalized *N*-PLS analysis on a given three way dataset. If the optimal configuration of the model parameters is not known, the package also provides the functions cv_npls() and repeat_cv(), that can be used to perform cross validation on the data, and automatically pick the best combination of those parameters. Furthermore, these functions allow for the parallel computation of the parameter grid search, which can be very advantageous, as this operation can quickly become computationally expensive on real world problems. The package also comes with a large set of plotting functions, which proved to be very useful when interpreting the modelling and cross validation outputs.

Dr. Leonardo Ramirez-Lopez. NIR Data Analytics Manager, BUCHI Labortechnik AG. Meierseggstr. 40, 9230 Flawil, Switzerland. □

In the context of the R programming language, the uniqueness of the package **sNPLS** lies on the fact that it offers tools to perform three-way PLS regression in conjunction with L1 regularization in the cases where variable selection is required. Following the spirit of R, this new package is publicly available and it can be downloaded from the CRAN repository. The main functions are implemented in a user-friendly way by keeping only the arguments that are unquestionably relevant. I installed the package on my personal computer and used it to build three-way PLS models. For this, I used two datasets (which were provided by the authors): a synthetic one and the bread data described in Bro (1996, also included in the package). The modeling results I obtained were easily interpretable especially with the help of the plotting functions included in the package. The computational time seems not to be an issue since the main functions include parallel computing functionality. In this respect, the package would be suitable for processing large datasets. I believe this package is a worthy contribution to R and to the expansion of the continuously growing chemometrics community of R users. □

Conflict of interests

There is no conflict of interest.

Acknowlegments

Research in this study was partially supported by the *Conselleria de Educación, Investigación, Cultura y Deporte de la Generalitat Valenciana* under the project PROMETEO/2016/093

REFERENCES

[1] L.R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika* 31(3) (1966), 279-311.

[2] R.A. Harshman. Foundations of the parafac procedure: models and conditions for an "explanatory" multimodal factor analysis. *UCLA Working Papers in Phonetics*, 16(1) (1970), 1-84.

[3] R. Bro. Multiway calibration. Multilinear PLS. *J. Chemom*. 10(1) (1996), 47-61.

[4] T. Hesterberg, N.H. Choi, L. Meier, C. Fraley. Least angle and ℓ1 penalized regression: A review. *Statistics Surveys*. 2 (2008), 61-93.

[5] R. Lockhart, J. Taylor, R.J. Tibshirani, R. Tibshirani. A significance test for the lasso. *Ann. Stat*. 42(2) (2014), 413-468.

[6] R. Bro, A.K. Smilde, S. de Jong. On the difference between low-rank and subspace approximation: improved model for multi-linear PLS regression. Chemom. Intell. Lab. Syst. 58 (1) (2001), 3-13.

[7] A. Smilde, R. Bro, P. Geladi. *Multi-way analysis: applications in the chemical sciences.* John Wiley & Sons (2005).

[8] R. Bro. *Multi-way analysis in the food industry: models, algorithms, and applications.* Royal Veterinary and Agricultural University, Copenhagen, Denmark. (2002).

[9] R. Tibshirani. Regression shrinkage and selection via the lasso. *J. Roy. Stat. Soc. B Met*. 58(1) (1996), 267-288.

[10] K.A. Lê Cao, D. Rossouw, C. Robert-Granié, P. Besse. A sparse PLS for variable selection when integrating omics data. *Stat. Appl. Genet. Mo. B.* 7(1) (2008), 1-29.

[11] D. Hervás, J.M. Prats-Montalbán, A. Lahoz, A. Ferrer. Variable selection in N-PLS by L1 penalization. Submitted to *Chemom. Intell. Lab. Syst*. (2018)

[12] D. Bates, M. Maechler. Matrix: Sparse and Dense Matrix Classes and Methods. R package version 1.2-9 (2017), URL https://CRAN.R-project.org/package=Matrix

[13] D. Krstajic, L.J. Buturovic, D.E. Leahy, S. Thomas. Cross-validation pitfalls when selecting and assessing regression and classification models. *J. Cheminformatics*. 6(1):10 (2014).

Figure Captions

Figure 1: Computing times of `cv_snpls` under different conditions of grid length and number of cores. The function scales efficiently up until 8 cores.

Figure 2: Results of cross-validation. Lines depicting the cross-validated error (CVE) for each combination of the parameters are presented in a grid layout combining KeepJ values (number of variables selected), KeepK values (number of elements of the third mode) and Ncomp values (number of components).

Figure 3: Kernel density plot with the result of the repeated cross-validation. Highest density lies around keepJ=9 and keepK=1. Points scaled in size by frequency of appearance are additionally included on top of the density plot. The number of components is not represented since it was constant at 2 in all repetitions of the cross-validation.

Figure 4: Results of the repeated cross-validation function performed on the bread dataset. The plot consists on the faceted representation of a three-dimensional density

plot sliced in different planes (one per number of components). The kernel density estimation is created with the observed frequencies of the combinations of the different parameters. Additionally, points scaled in size by frequency of appearance are added to the density plots.

Figure 5: Score plot of the two first components in the **T** matrix.

Figure 6: Score plot of the two first components in the **U** matrix

Figure 7: Weights Plot of the $\mathbf{W}^{J}$ weights matrix

Figure 8: Weights Plot of the $\mathbf{W}^{K}$ weights matrix

Figure 9: Plot of the second mode

Figure 10: Plot of the third mode

Figure 11: Plots of the standard *N*-PLS model

Table 1. Results of each cross-validation repetition gathered in the data.frame structure.

```
R> repcv

   ncomp keepJ keepK
1      2     9     1
2      2     9     1
3      2    10     1
4      2     9     1
5      2    11     2
6      2    12     1
7      2     7     1
8      2    10     1
9      2     8     1
10     2    12     2
```

Table 2. Frequency table derived from the data.frame structure.

```
R> ftable(table(repcv))

            keepK 1 2
ncomp keepJ
2     7           1 0
      8           1 0
      9           3 0
      10          2 0
      11          0 1
      12          1 1
```

Table 3. Frequency table derived from the data.frame structure.

```
R> ftable(table(cv_bread))

            keepK 1 2 3 4 5 6 7 8
ncomp keepJ
1     1           1 2 2 4 3 3 1 1
      2           0 0 1 1 2 1 0 2
2     1           0 0 0 0 0 0 0 1
      2           0 0 0 0 0 0 0 2
      3           0 0 0 0 0 1 0 4
      4           0 0 0 0 0 0 1 1
      5           1 0 0 0 0 0 0 1
      6           0 0 0 0 0 0 1 0
      7           0 0 0 0 0 0 1 0
      8           0 0 0 0 0 0 0 2
      9           0 0 0 0 0 0 0 1
3     1           0 0 1 0 0 2 0 1
      2           0 0 0 0 0 0 1 1
      3           0 0 0 0 0 0 0 1
      5           0 0 0 0 0 0 0 1
      8           0 0 0 0 0 0 0 1
```

Table 4. Summary of the sNPLS model fitted with the finale selection (2 components, 3 variables per component in the second mode and 8 in the third mode)

```
R> summary(fit)

sNPLS model with 2 components and squared error of 0.047

Coefficients:
        Z.1    Z.2    Z.3    Z.4    Z.5    Z.6    Z.7    Z.8
X.1   0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
X.2   0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
X.3   0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
X.4  -0.041 -0.041 -0.043 -0.049 -0.074 -0.040 -0.042 -0.029
X.5   0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
X.6   0.019  0.026  0.024  0.027  0.021  0.025  0.018  0.036
X.7   0.070  0.089  0.086  0.095  0.087  0.088  0.069  0.115
X.8  -0.030 -0.038 -0.037 -0.041 -0.038 -0.038 -0.030 -0.050
X.9  -0.009 -0.009 -0.010 -0.011 -0.017 -0.009 -0.009 -0.006
X.10  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
X.11  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
```

Table 5. Predictions performed by the sNPLS model

	Y.1
[1,]	1.4196482
[2,]	0.7819288
[3,]	0.9619593
[4,]	1.0523621
[5,]	1.2585099
[6,]	0.7243910

Table 6. Summary of the standard NPLS model fitted with 2 components.

Coefficients:

	Z.1	Z.2	Z.3	Z.4	Z.5	Z.6	Z.7	Z.8
X.1	0.015	0.017	0.017	0.017	0.017	0.015	0.014	0.021
X.2	-0.019	-0.019	-0.020	-0.017	-0.023	-0.014	-0.019	-0.021
X.3	-0.006	-0.008	-0.008	-0.011	-0.005	-0.010	-0.004	-0.012
X.4	-0.025	-0.024	-0.026	-0.022	-0.030	-0.018	-0.025	-0.027
X.5	0.010	0.013	0.013	0.016	0.009	0.015	0.007	0.019
X.6	0.024	0.028	0.028	0.031	0.026	0.028	0.020	0.036
X.7	0.054	0.058	0.061	0.060	0.061	0.053	0.049	0.073
X.8	-0.023	-0.027	-0.028	-0.030	-0.024	-0.028	-0.019	-0.036
X.9	-0.034	-0.034	-0.037	-0.033	-0.039	-0.028	-0.032	-0.040
X.10	-0.007	-0.010	-0.009	-0.013	-0.006	-0.013	-0.004	-0.015
X.11	0.017	0.019	0.020	0.020	0.020	0.017	0.016	0.024

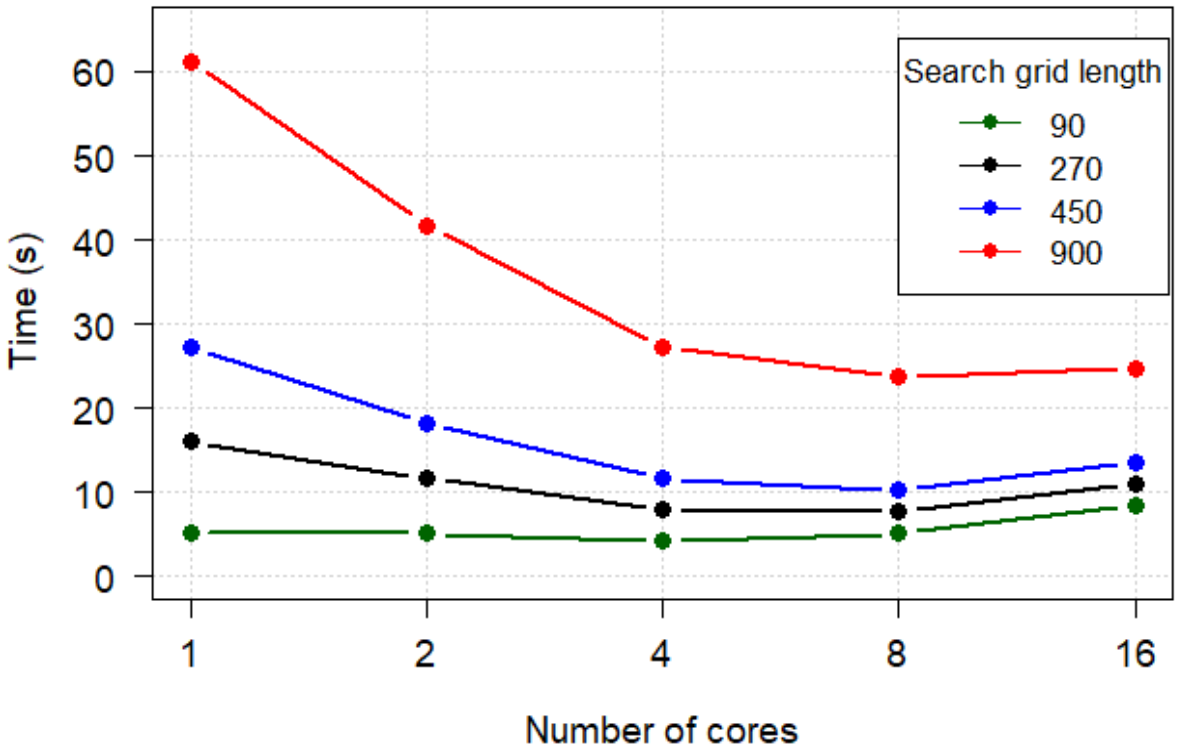


Figure 1: Computing times of cv_snpls under different conditions of grid length and number of cores. The function scales efficiently up until 8 cores.

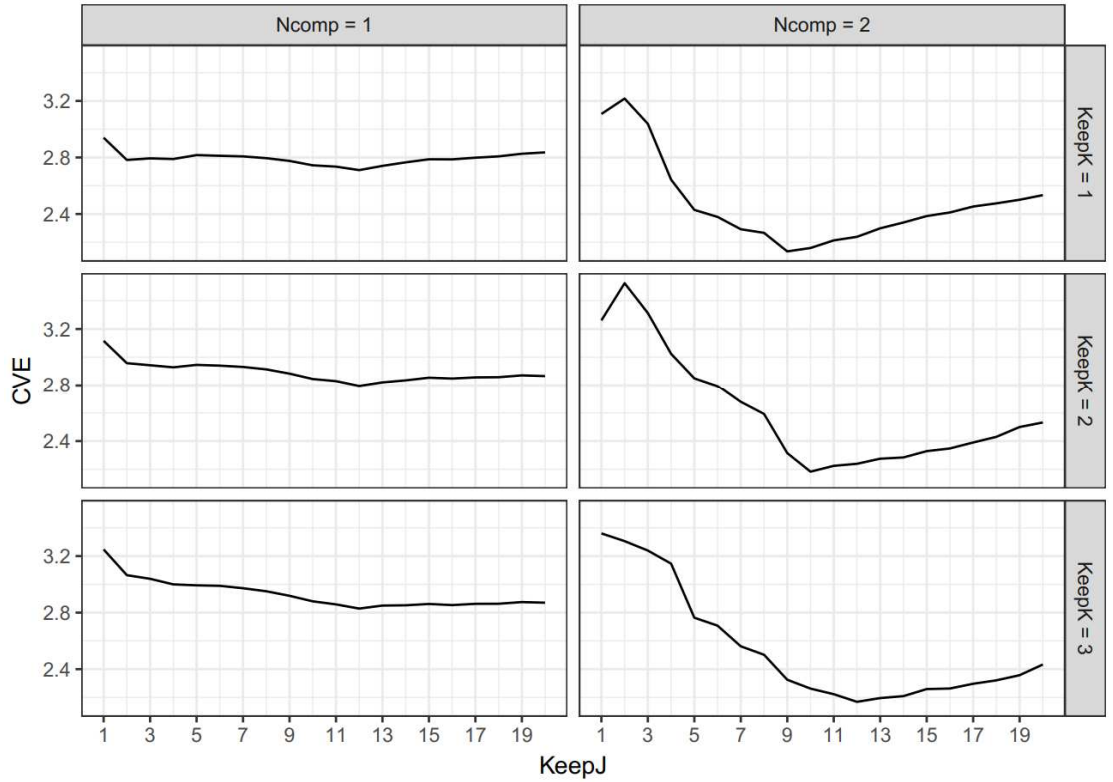


Figure 2: Results of cross-validation. Lines depicting the cross-validated error (CVE) for each combination of the parameters are presented in a grid layout combining KeepJ values (number of variables selected), KeepK values (number of elements of the third mode) and Ncomp values (number of components).

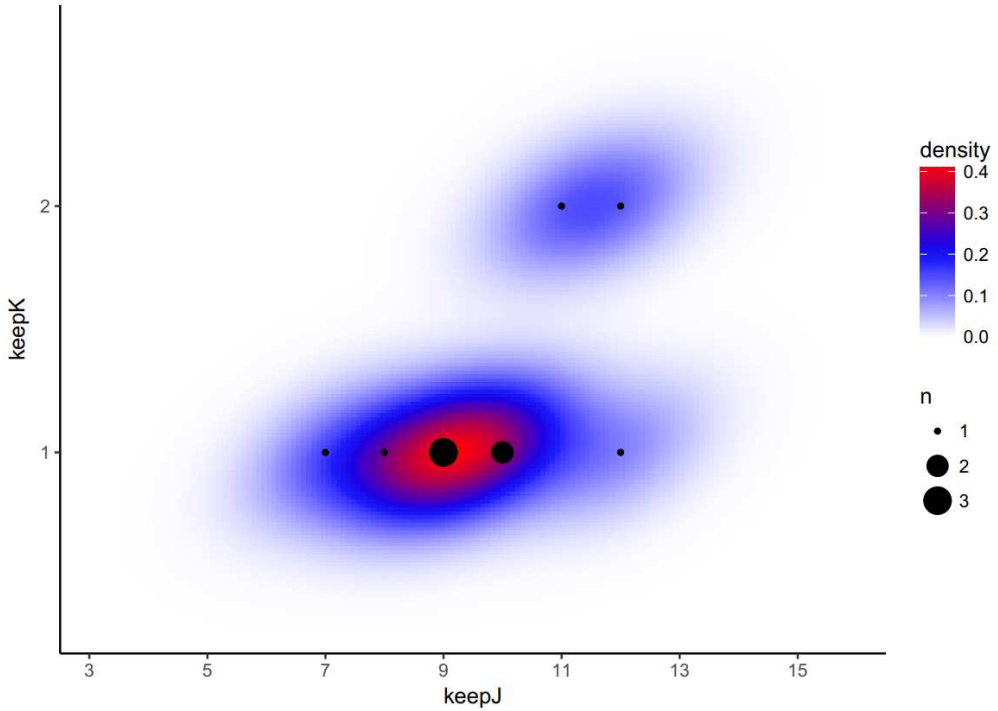


Figure 3: Kernel density plot with the result of the repeated cross-validation. Highest density lies around keepJ=9 and keepK=1. Points scaled in size by frequency of appearance are additionally included on top of the density plot. The number of components is not represented since it was constant at 2 in all repetitions of the cross-validation.

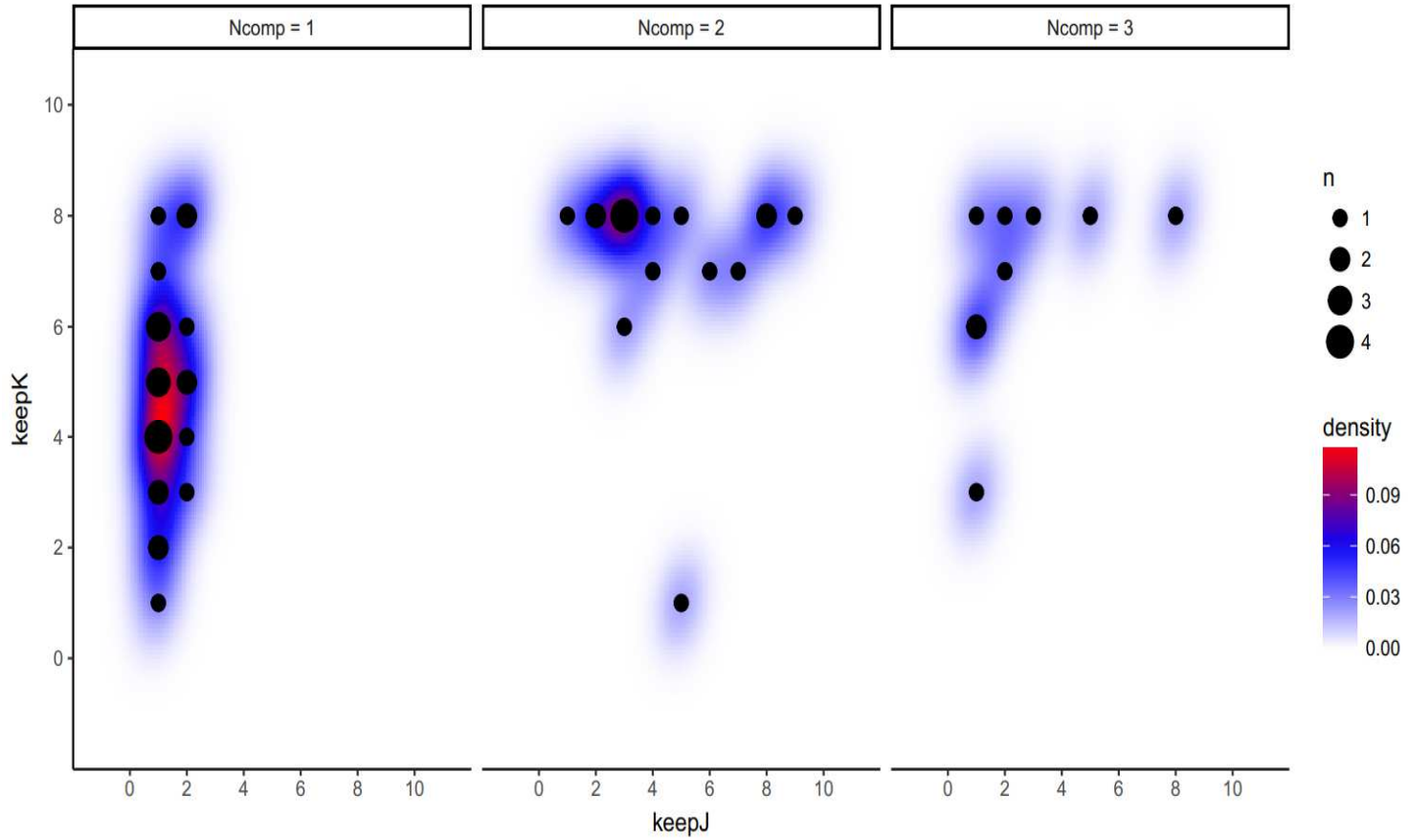


Figure 4: Results of the repeated cross-validation function performed on the bread dataset. The plot consists on the faceted representation of a three-dimensional density plot sliced in different planes (one per number of components). The kernel density estimation is created with the observed frequencies of the combinations of the different parameters. Additionally, points scaled in size by frequency of appearance are added to the density plots.

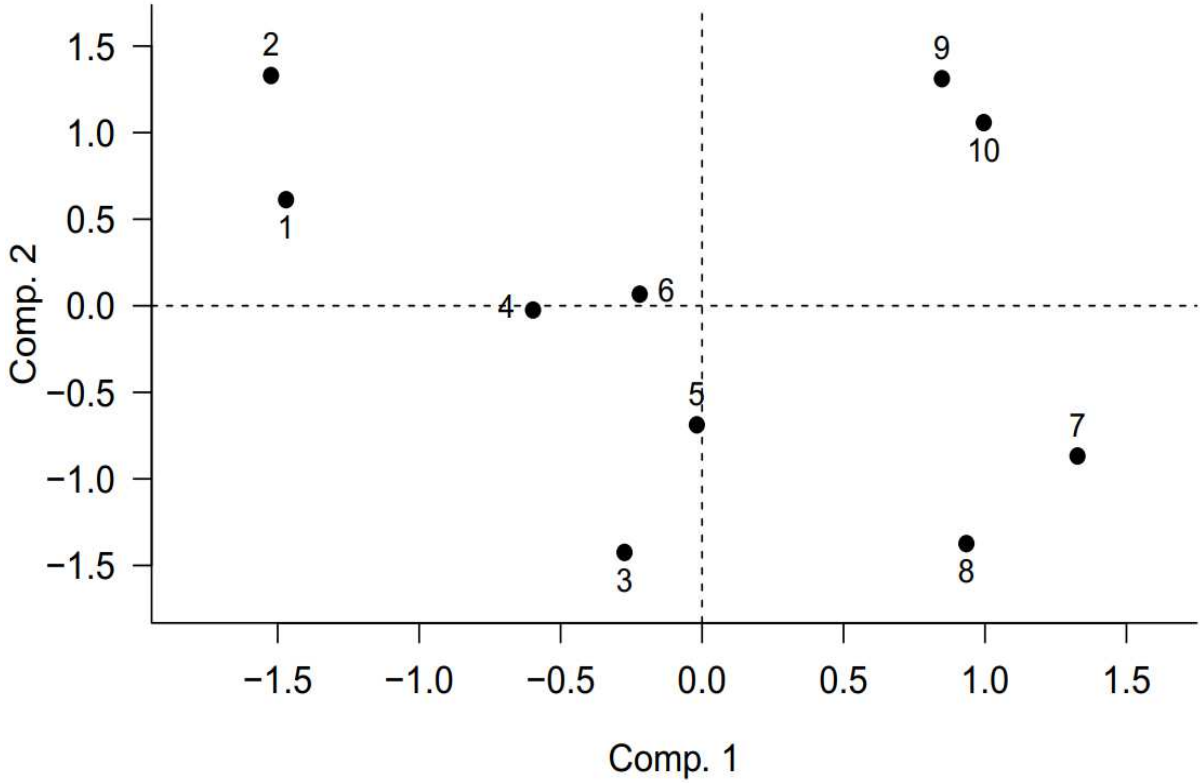


Figure 5: Score plot of the two first components in the **T** matrix.

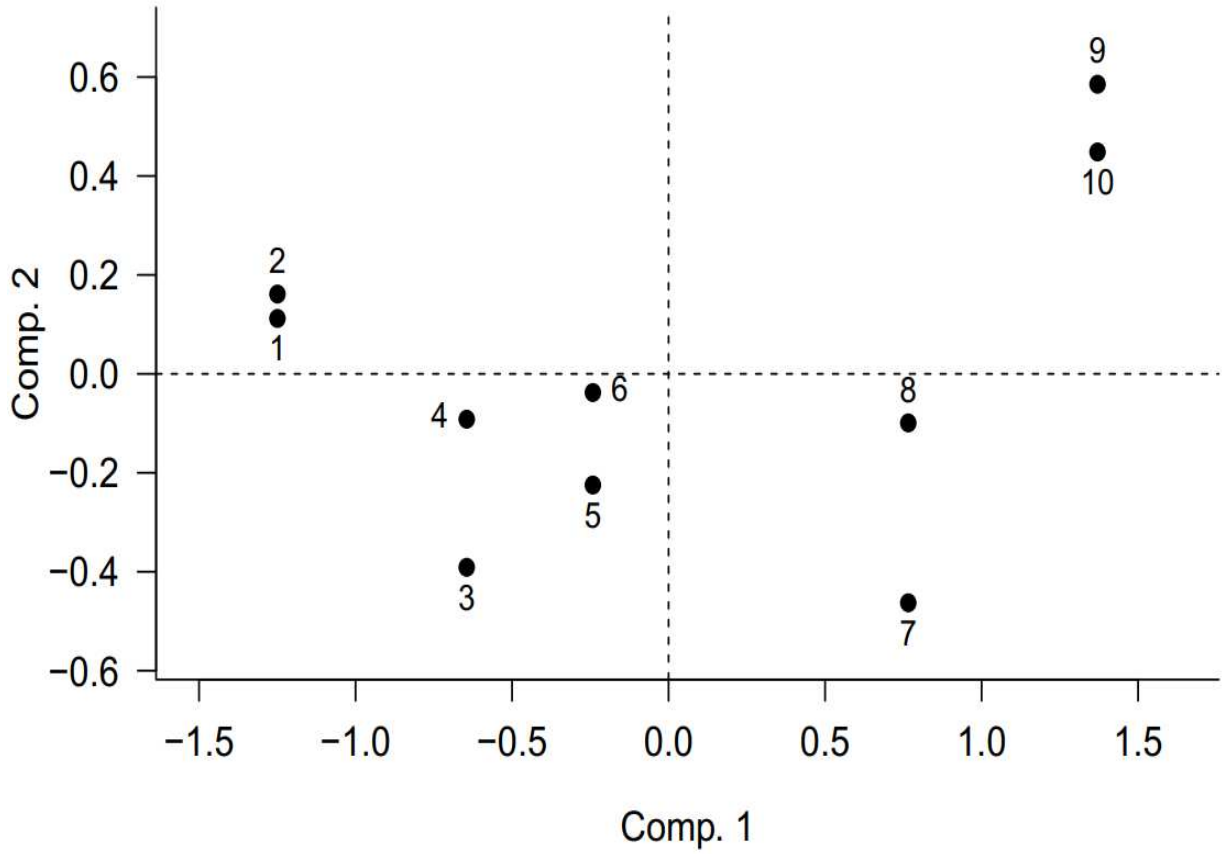


Figure 6: Score plot of the two first components in the **U** matrix

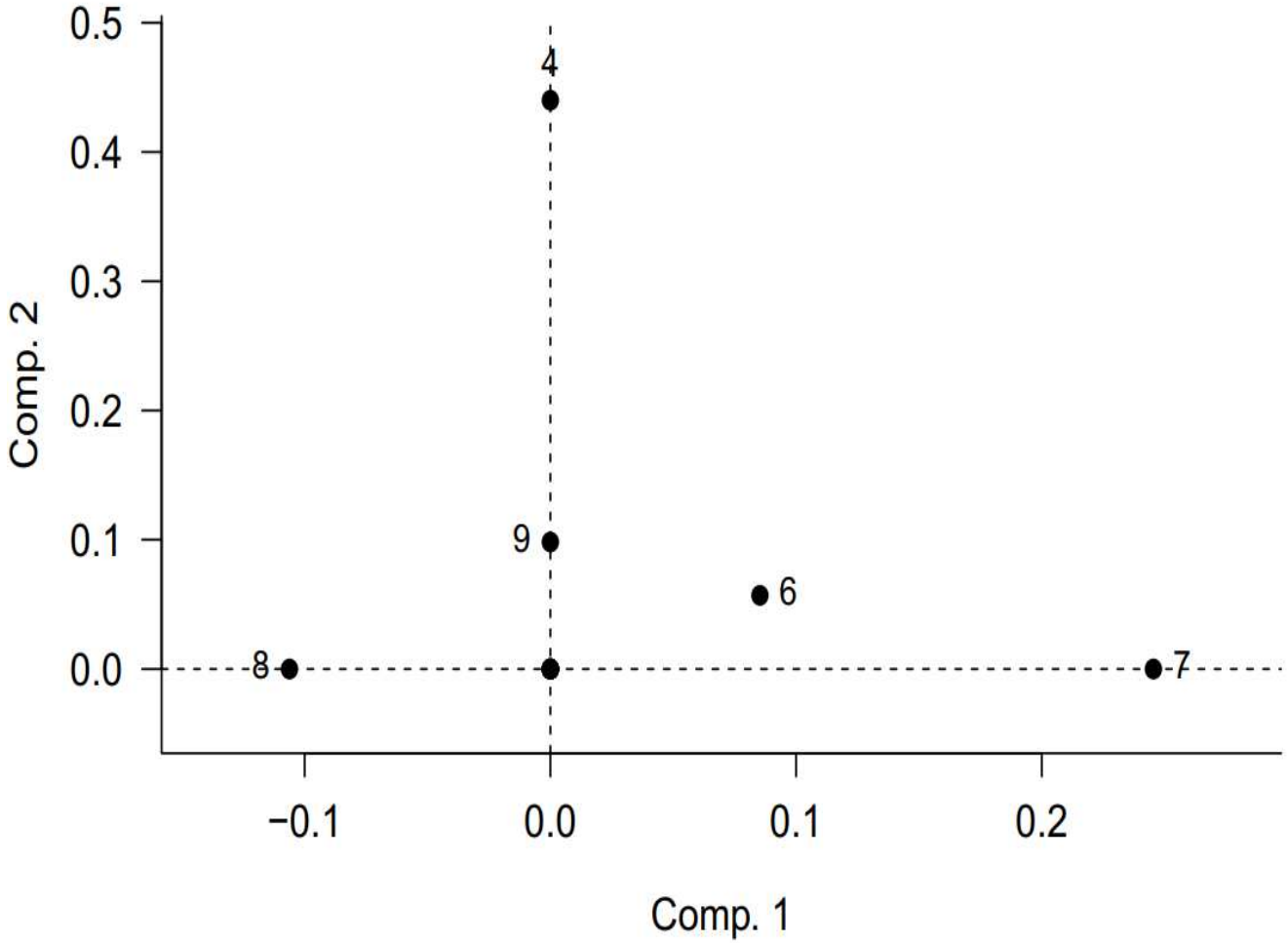


Figure 7: Weights Plot of the $\mathbf{W}^{J}$ weights matrix

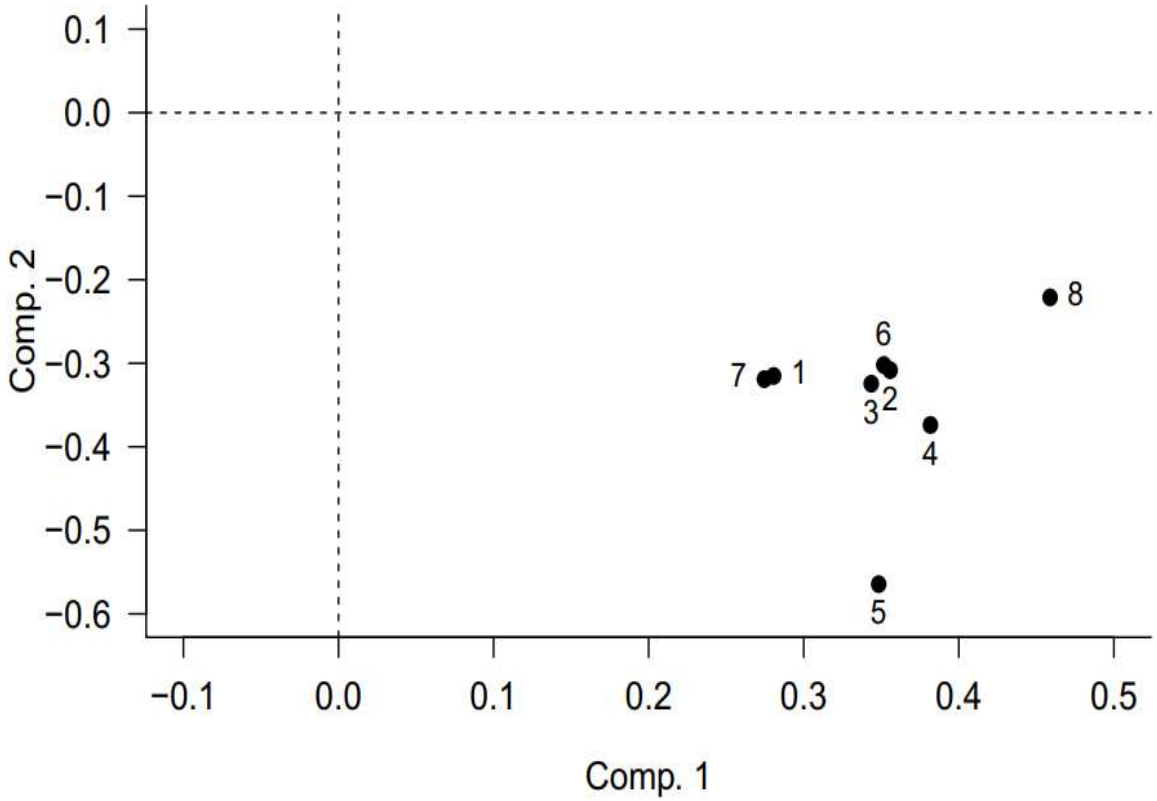


Figure 8: Weights Plot of the $\mathbf{W}^K$ weights matrix

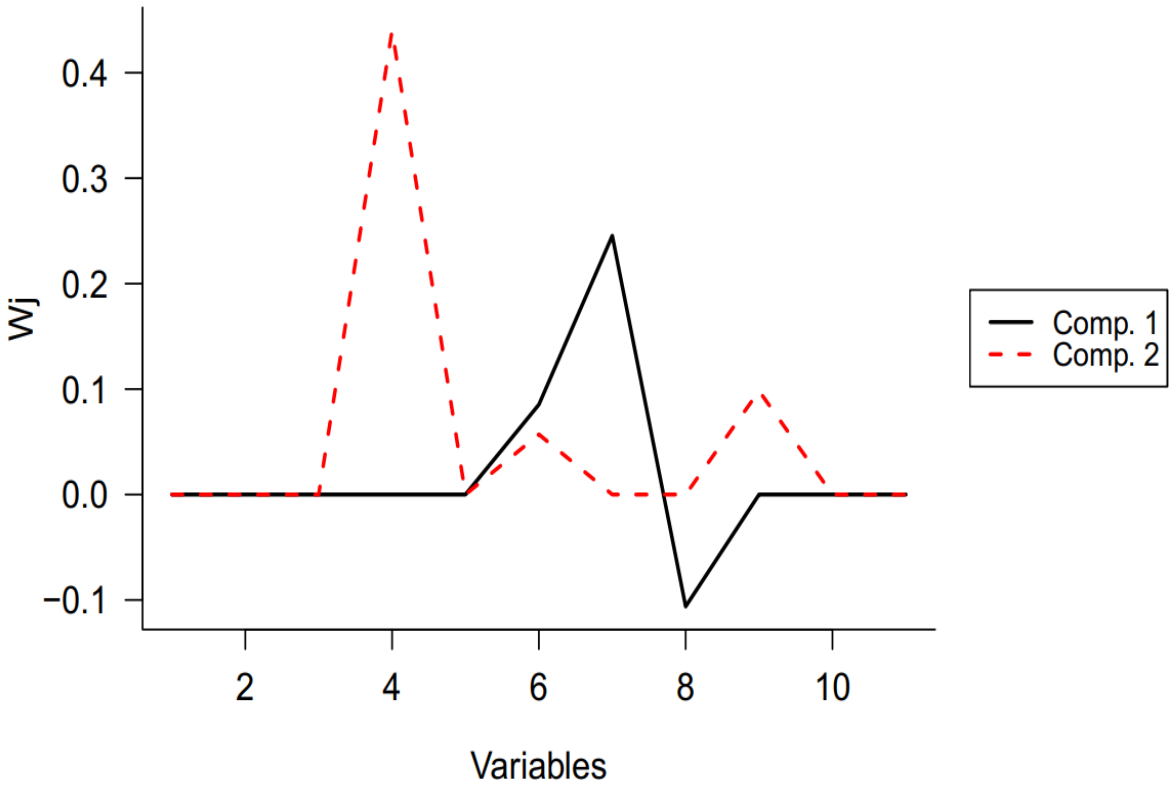


Figure 9: Plot of the second mode

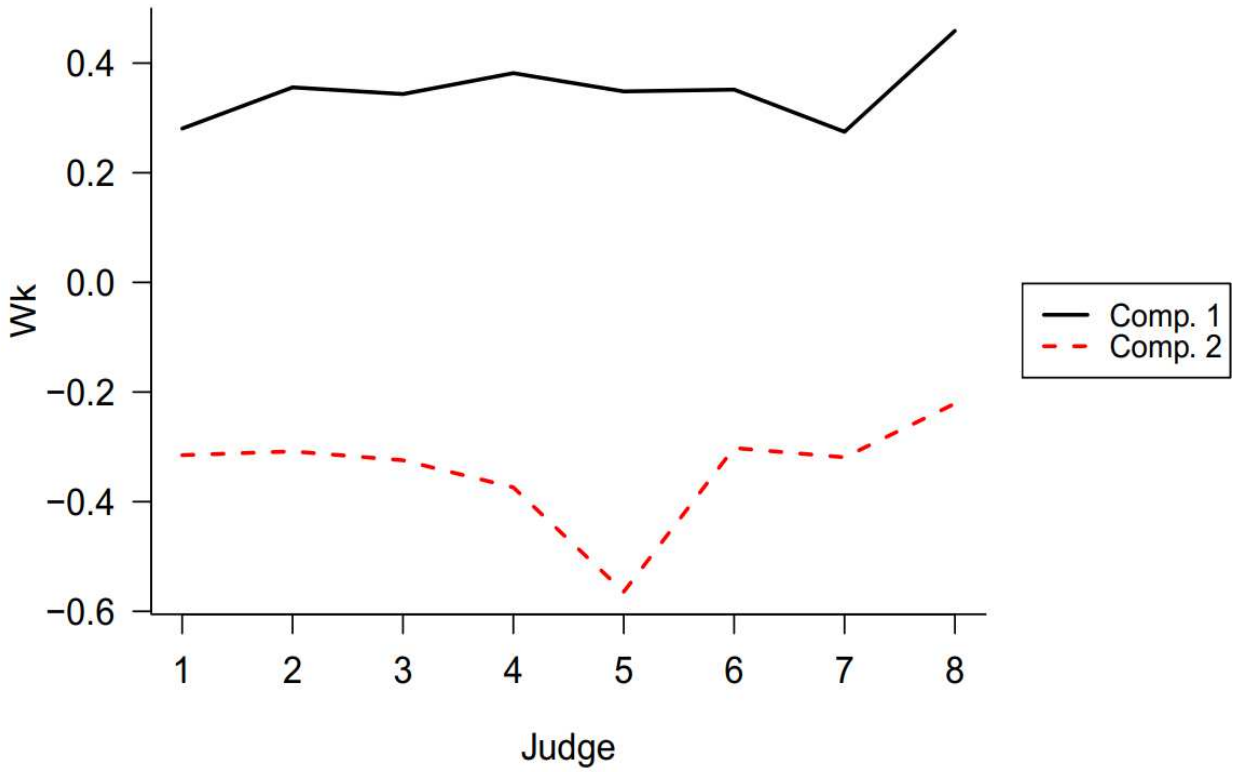


Figure 10: Plot of the third mode

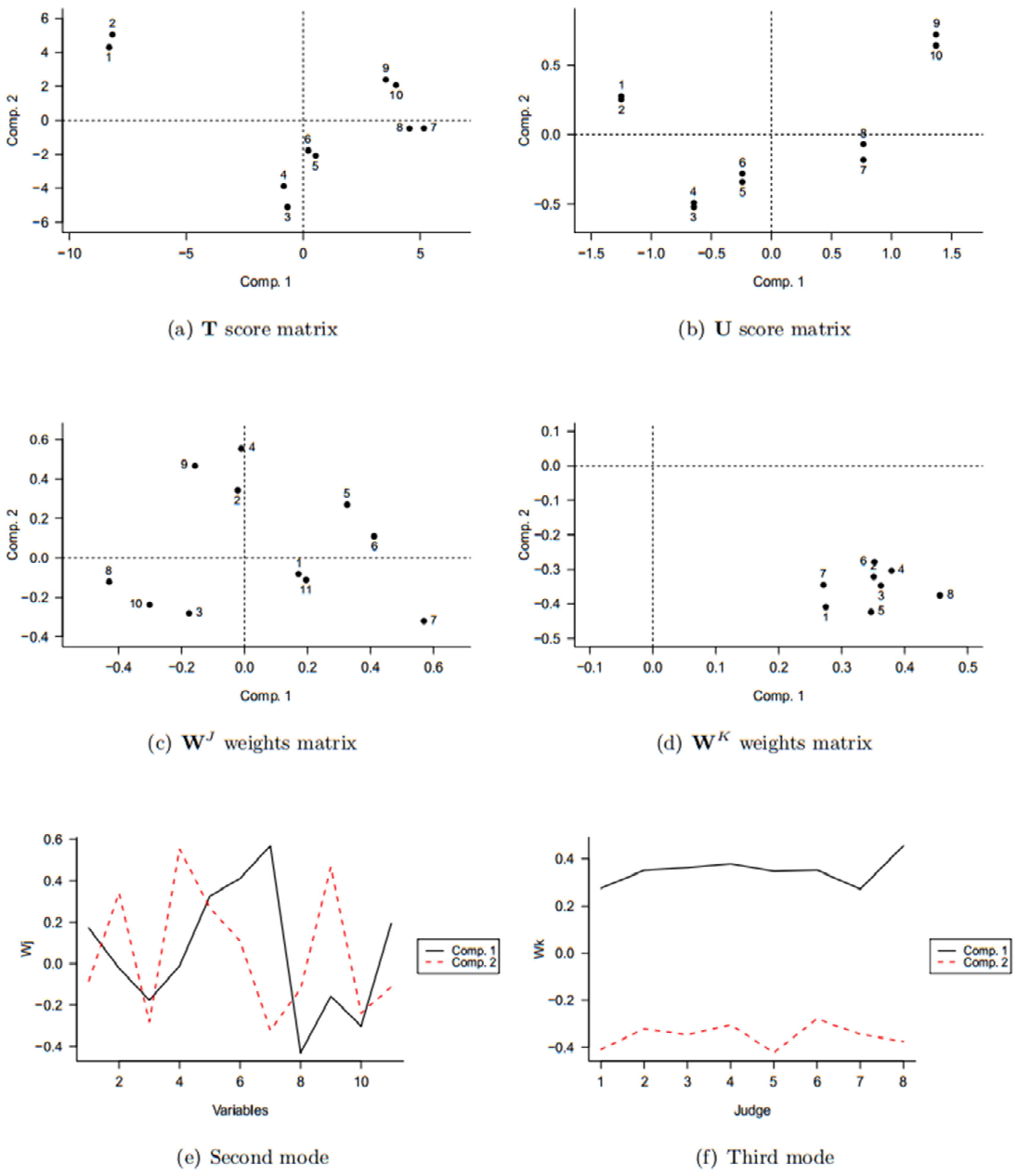


Figure 11: Plots of the standard *N*-PLS model