ddsPLS: A Package to Deal with Multiblock Supervised Problems with Missing Samples in High Dimension

by Hadrien Lorenzo, Jérôme Saracco and Rodolphe Thiébaut

Abstract The `ddsPLS` method considers regression and classification problems in the context of multiblock structured covariate data sets taking into account missing samples. The response outcome can be multidimensional and constitutes the response block. Multiblock covariate data sets are constituted by heterogeneous types of data (e.g. generated by various bioassays) and/or temporal realisations of the same covariates. The `ddsPLS` approach only deals with a specific case of missing values, denoted as missing samples, which means that some individuals have missing values for all the covariates in a given block. The `ddsPLS` method performs imputation of missing data, covariate selection and prediction of the response. The method is based on singular value decompositions of soft-thresholded covariance blocks via a three-steps procedure. The tuning parameters of the `ddsPLS` method can be optimized through an available cross-validation procedure. The **ddsPLS** package is illustrated on a real data set.

Introduction

Analyzing high dimensional data where the number of predictors is much higher than the number of statistical units (e.g. individuals) is challenging in mathematical and practical point of views. Regularization solutions, such as the regularization model introduced by Tikhonov and Arsenin and translated by Willoughby (1979), show theoretical and also practical solutions to the high dimension problem. To introduce sparsity in the models, Tibshirani (1996) has developed a regularization method named Lasso, close in form to the Tikhonov solution. Putting to 0 the coefficient associated with unimportant covariates allows the selection of the most relevant covariates and thus improve the interpretability of the model.

The Lasso method has been adapted to high correlation structured data sets thanks to the Elastic-Net model by Tibshirani (1996). It has also been adapted to grouped data sets, i.e. when the covariates can be divided in *a priori* meaningful groups leading to the so-called group-Lasso and sparse group-Lasso methods (see Bakin, 1999; Simon et al., 2013). However, those methods only consider an unidimensional response variable.

Multiple response predictions can be performed with PLS (Partial Least Squares) which is introduced by Wold et al. (2001). PLS methods work with covariance structures using singular value decompositions, denoted as SVD in the following. Their flexibility allows to develop multiblock covariate typed methods such as the multiblock PLS algorithm, denoted as MBPLS, and developped by Wangen and Kowalski (1989). Then, using classical regularization approaches, a sparse version of the PLS has been proposed by Lê Cao et al. (2008) using Lasso and by Liquet et al. (2015) using group Lasso. Multiblock methods with regularization and covariate selection have been generalized through RGCCA and SGCCA (see Tenenhaus and Tenenhaus, 2011; Tenenhaus et al., 2014) and gather canonical correlation analyses and PLS methods in a regularized general framework. The package **RGCCA** implements different solutions associated with that context.

Missing values occur for numerous reasons. Here, we consider the simplest case referred as *MCAR* (Missing Completely At Random), as detailed by Rubin (1976). In that context, no association exists between the missing probability of a realization and any measured or unmeasured covariate. An usual approach to deal with missing data in such context is to remove units/individuals with missing information. Even if not biased, this complete case analysis shrinks the statistical power and must not be used such as shown in Josse et al. (2009). Obviously, the issue is even more pronounced when the number of samples is low. Another approach is to complete missing data by imputation. The model for imputation could be very simple such as the imputation of the same fixed value for all missing observations based on the mean or the median of observed values. This leads to hard modifications of the covariance structures (Schafer and Graham, 2002) and should therefore be avoided. The alternative is to use the information carried by the observed covariates to drive the imputation of missing data. Since the imputation itself modifies the covariance structure, it is reasonable to consider iterative methods, close to the *EM* (*Estimation-Maximization*) algorithm described by Dempster et al. (1977). Those *EM-typed* methods give valid results in the context of imputation as detailed in Mazumder et al. (2010) for the SVD of very large matrices with sparse structure and a large proportion of missing values. The R package **softImpute** (Hastie and Mazumder, 2015) allows to use that method. Stekhoven and

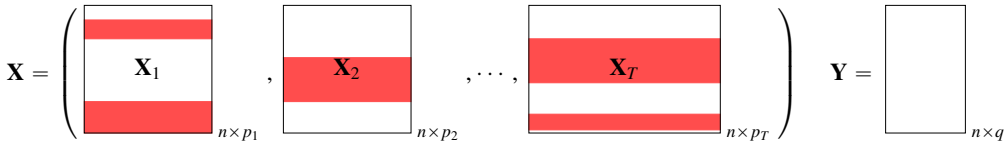


Figure 1: A multiblock data set, of T blocks, where missing values are represented by red areas. n is the number of samples, q is the number of variables in the response matrix and $(p_t)_{t\in[\![1,T]\!]}$ are the number of covariates in each covariate block.

Bühlmann (2011) have developed an *EM-typed* algorithm for random forest objects implemented in the **missForest** package. For unsupervised analyses, Josse and Husson (2012) have proposed a regularized PCA (Principal Component Analysis) which imputes missing values through an *EM* approach. The **missMDA** (see Josse and Husson, 2016) package allows to use that method to multiblock data sets, through the `imputeMFA()` function. The task view *MissingData* gives more examples of packages doing imputation.

The *CMTF* (*Collective Matrix Tensor Factorization*) problem (see for example Acar et al., 2011, 2013) has been designed to deal with heterogeneous problems where a collection of matrices sharing the same structure (which is called a tensor) and an objective matrix ($\mathbf{Y}$ in our case) share a common description of the same individuals. The *CMTF-WOPT* algorithm (*CMTF-Weighted Optimal Collective Matrix Tensor Factorization*) is a modification of *CMTF* which allows to deal with missing values in the collection of covariate matrices by ignoring missing values in the data structure estimation.

The **ddsPLS** package (*data-driven sparse Partial Least Squares*) proposes an *EM-typed* algorithm dedicated to multiblock supervised for classification or multivariate regression taking into account missing samples in the covariate part and also providing covariate selection in order to predict the response part. Figure 1 shows the context of application of the method for a T covariate blocks measured on the same n samples where each block $t \in [\![1, T]\!]$ describes each sample with $p_t \in \mathbb{N}^\star$ covariates. The response block $\mathbf{Y}$ contains $q \in \mathbb{N}^\star$ response variables measured on the same n samples. The `ddsPLS` approach only deals with a specific case of missing values (which often occurs in health data problems), denoted as missing samples, which means that some individuals have missing values for all the covariates in a given block. In Figure 1, red areas correspond to missing samples.

That package has been used through three applications presented in Lorenzo et al. (2019a,b); Ellies-Oury et al. (2019).

The `ddsPLS` method needs to fix two regularization parameters (the number R of components to retain, and the maximum number L_0 of covariates to consider in prediction). Cross-validation can be used to determine those parameters from the available data and a function has been implemented to perform such optimization.

The **ddsPLS** package is based on three S3 classes which are

- the class `"mddsPLS"` provides regression or classification pieces of information of a prediction model (including imputation),
- the class `"perf_mddsPLS"` gathers cross-validation results,
- the third class `"MddsPLS_core"` is never directly built by the user but in each object of the class `"mddsPLS"`, an attribute is an object of the class `"MddsPLS_core"`.

Their associated S3 methods are detailed in Table 1.

In the following, the `ddsPLS` method is briefly presented. Then, the main functions associated with `"mddsPLS"` (estimation/imputation, prediction) and `"perf_mddsPLS"` (cross-validation to tune both of the parameters of the model) classes are described and illustrated on a real data set in a regression context. No example of classification analysis is provided but the last section provides key indications to perform this type of analyses.

Brief description of the ddsPLS method

The `ddsPLS` method is able to impute missing samples (in covariate blocks) in a supervised context (regression or classification), to estimate the underlying model and to make predictions. A full description of the `ddsPLS` method is available in Lorenzo et al. (2019b).

The `ddsPLS` method only needs two parameters to be tuned by the user:

Object	Type	Accessible	Description
`"MddsPLS_core"`	S3 class	No	The core function of the `mddsPLS` class.
`"mddsPLS"`	S3 class	Yes	Main class.
`plot.mddsPLS()`	S3 method	Yes	The plot method
`predict.mddsPLS()`	S3 method	Yes	The predict method.
`summary.mddsPLS()`	S3 method	Yes	The summary method.
`"perf_mddsPLS"`	S3 class	Yes	Function to estimate cross -validation performances.
`plot.perf_mddsPLS()`	S3 method	Yes	The plot method
`summary.perf_mddsPLS()`	S3 method	Yes	The summary method.

Table 1: Main functions of the **ddsPLS** R package

- the number of components $R \in [\![1, min(n, p, q)]\!]$, equivalent to the number of components in PLS or PCA,
- the regularization parameter which can be controlled through one of the following two parameters, depending on user's choice:
  - $\lambda \in [0, 1]$ – `lambda` in the software – which corresponds to the minimum value of the correlation between a covariate and a response variable to take into account in the model,
  - $L_0 \in [1, p]$ – `L0` in the package – which represents the maximum number of covariates to be include in the model.

Those parameters (R, λ) or (R, L_0) can be optimally determined by cross-validation through the `"perf_mddsPLS"` class.

For given values of these two parameters, the class `"mddsPLS"` builds the corresponding model (including imputation of missing samples), and this model can then be used for predictions.

Notations

Let us consider a multiblock data set built on T blocks $(\mathbf{X}_1, \cdots, \mathbf{X}_T)$ with the additional response block, $\mathbf{Y}$, that supervises the analysis. Given any block $t \in [\![1, T]\!]$ and a number R of built components, let us define

- $\mathbf{M}_t = S_\lambda\left(\frac{\mathbf{Y}^T\mathbf{X}}{n-1}\right) \in \mathbb{R}^{q \times p_t}$, the soft-thresholded covariance matrix between the response block $\mathbf{Y}$ and the covariate block $\mathbf{X}_t$, where $S_\lambda : \quad x \to \text{sign}(x)\left(|x| - \lambda\right)_+$ is the soft-thresholding function.
- $\lambda \in [0, 1]$, the regularization coefficient, which can also be parameterized through the L_0 parameter. L_0 corresponds to the number of columns of

$$S_\lambda\left(\frac{\mathbf{Y}^T\left[\mathbf{X}_1, \cdots, \mathbf{X}_T\right]}{n-1}\right)$$

  that must be put to 0 with the lowest value of λ (the smallest degree of regularization).
- $\mathbf{U}_t = \mathbf{SVD}_R(\mathbf{M}_t) \in \mathbb{R}^{p_t \times R}$ the *weight* matrix where "$\to \mathbf{SVD}_R(.)$" gives the R right singular vectors associated with the R largest singular values.
- $\mathbf{Z} = [\mathbf{M}_1\mathbf{U}_1, \cdots, \mathbf{M}_T\mathbf{U}_T] \in \mathbb{R}^{q \times R}$, the concatenation of the projected soft-thresholded covariance matrices.
- $\underline{\boldsymbol{\beta}} = \mathbf{SVD}_R(\mathbf{Z}) \in \mathbb{R}^{RT \times R}$, the *super-weights* per block per axis concatenated per row. Defining also $\underline{\boldsymbol{\beta}} = \left[\boldsymbol{\beta}_1^T, \cdots, \boldsymbol{\beta}_T^T\right]^T$, where $\boldsymbol{\beta}_t \in \mathbb{R}^{R \times R}$.
- $\mathbf{U}_t^\star = \mathbf{U}_t\boldsymbol{\beta}_t \in \mathbb{R}^{p_t \times R}$, the *scaled super-weights* per block.
- $\mathbf{T}_t = \mathbf{X}_t\mathbf{U}_t \in \mathbb{R}^{n \times R}$, the *scores* per block per axis.
- $\mathbf{T}^\star = \sum_{t=1}^T \mathbf{X}_t\mathbf{U}_t^\star \in \mathbb{R}^{n \times R}$, the *super-scores*.
- $\mathbf{V}^\star = \text{norm}_2(\mathbf{Z}\underline{\boldsymbol{\beta}}) \in \mathbb{R}^{q \times R}$, the *weights* of the response, where the function "$\to \text{norm}_2(.)$" returns the columns normalized to a $\mathcal{L}_2$-norm equal to 1 if the corresponding column is non null and returns the null-column otherwise.
- $\mathbf{S}^\star = \mathbf{Y}\mathbf{V}^\star \in \mathbb{R}^{n \times R}$, the *super-scores* of the response.
- $\mathcal{B} = (\mathbf{B}_t \in \mathbb{R}^{p_t \times q})_{t \in [\![1,T]\!]}$ the list of regression matrices per block.

The imputation algorithm

For a given couple of tuning parameters (R, λ), the algorithm denoted as *Koh-Lanta* is an *EM-typed* algorithm applying the following three-steps procedure to the complete data set

- STEP 1: solve independently the T mono-block `ddsPLS` problems based on $\mathbf{X}_t$ and $\mathbf{Y}$. This allows to calculate, $(\mathbf{U}_t)_{t\in[\![1,T]\!]}$ and $\mathbf{T}$.
- STEP 2: combine information from the T blocks. This allows to calculate $\underline{\boldsymbol{\beta}}$, $\forall t \in [\![1, T]\!]$ $\mathbf{U}_t^\star$, $\mathbf{V}^\star$, $\mathbf{T}^\star$ and $\mathbf{S}^\star$.
- STEP 3: predict $\mathbf{Y}$ using a regression model based on the T blocks. This allows to calculate, $\forall t \in [\![1, T]\!]$, $\mathbf{B}_t = \mathbf{U}_t^\star \mathbf{B}_0 \mathbf{V}^{\star T}$, where $\mathbf{B}_0 = \left(\mathbf{T}^{\star T}\mathbf{T}^\star\right)^+ \mathbf{T}^{\star T}\mathbf{S}^\star$ is the linear regression matrix of $\mathbf{S}^\star$ on $\mathbf{T}^\star$ estimated through minimization of *OLS* (*Ordinary Least Squares*, see Kenney and Keeping, 1962, for example) and "$\rightarrow (.)^+$" denotes the Moore-Penrose pseudo-inverse operator (see Planitz, 1979).
  The objective regression model is
$$\mathbf{Y} \approx \sum_{t=1}^{T} \mathbf{X}_t \mathbf{B}_t.$$

For a given couple of tuning parameters (R, λ), the *Koh-Lanta* algorithm allows to impute the missing samples using the above algorithm in an iterative way, alternating model building and missing data prediction. Its specificity is to deal with missing values in the train, in a first part, and in the test data sets, in a second part. More precisely :

- *The Tribe Stage*, the train data set imputation:
  1. For each block $\mathbf{X}_t$, with missing values, predict the missing values using a mono-block `ddsPLS` model taking $\mathbf{Y}$ as covariates and $\mathbf{X}_t$ as response. Note that non missing samples are used for training the underlying model.
  2. Build the multiblock `ddsPLS` model with covariate blocks $\{\mathbf{X}_1, \cdots, \mathbf{X}_T\}$ and response block $\mathbf{Y}$. Keep in memory the variables selected in each block.
  3. Re-do step 1 with only the selected variables of step 2 as response.
  4. Iterate steps 1 and 2 until stabilization of the super-scores.

  The model built in step 2 during the last iteration is then used in the following test data set imputation.
- *The Reunification Stage*, the test data set imputation :
  For each individual i, if there is at least one missing sample, split the train data set in 2 multiblock data sets. The first one, denoted as $B_i^{(1)}$, corresponds to the blocks where i has missing values and the second one, denoted as $B_i^{(2)}$, to the non missing blocks. The algorithm builds a prediction model using scores of $B_i^{(2)}$ as covariates and the concatenation of the blocks $B_i^{(1)}$ as response matrix. The missing values are then calculated according to that model and $B_i^{(2)}$ data can be projected onto the scores thanks to the train model.

In the following, the **ddsPLS** package formalism is illustrated on a toy example (based on a real data).

Build a model of prediction

The `"mddsPLS"` class allows to build prediction models (including imputation and variable selection). The `mode` argument allows to choose between regression or classification. The arguments of that class are detailed in Table 2.

Illustration on a toy example

To illustrate the use of the package, the `liverToxicity` data set (see Bushel et al., 2007) is used ($n = 64$, $p = 3116$ covariates, $q = 10$ response variables). The covariate part has been arbitrarly split in two blocks denoted as `Block_1` and `Block_2`. Moreover, missing samples have been generated as follows: the first five samples of `Block_1` have been deleted and the next five samples of `Block_2` have also been deleted. This has been done with the following code:

Argument	Description
`Xs`	A matrix (covariate block), if there is only one block. A list of matrices, if there is more than one block, of n rows each, the number of individuals. Some rows can be missing. The different matrices (blocks) can have different numbers of columns. The length of `Xs` is denoted by `T` and corresponds to the number of blocks.
`Y`	A matrix of n rows or a vector of length n detailing the response matrix. No missing values are allowed in that matrix.
`lambda`	A real $\in [0,1]$. This is a regularization parameter to select variables. The closer to 1, the less variables are selected. Fix this parameter or `L0`.
`L0`	A non null integer which is the largest number of covariates that can be selected in all the T covariate blocks. Fix this parameter or `lambda`.
`R`	A strictly positive integer corresponding to the number of components to build in the model.
`mode`	A character chain. Possibilities are '`reg`', '`lda`' or '`logit`', which correspond respectively to regression problem, linear discriminant analysis and logistic regression (through functions `glm()` and `lda()`, respectively, from the package **MASS**). Default is '`reg`'. Note that, for classification, '`logit`' does not allow to deal with more than two classes.
`keep_imp_mod`	Logical. Whether or not to save imputation `"mddsPLS"` models which are created in the *Koh-Lanta tribe stage* imputation step. Default to `FALSE`. Be careful because each of the model savings can be memory demanding.
`reg_imp_model`	Logical. Whether or not to regularize the imputation models. `TRUE` by default.
`errMin_imput`	Positive real. Convergence threshold in the *Tribe Stage* of the *Koh-Lanta* algorithm. Default is $1e-9$.
`maxIter_imput`	Positive integer. Maximum number of iterations in the *Tribe Stage* of the *Koh-Lanta* algorithm. Default is 5.
`verbose`	Logical. If `TRUE`, the convergence progress of the *Koh-Lanta* algorithm is reported. Default is `FALSE`.
`NZV`	Float. The floating value above which the weights are set to 0.
`getVariances`	Logical. Whether or not to compute explained variances and RV coefficients of the model. Useful for interpretation but time consuming. Default is `TRUE`.

Table 2: Arguments of the class `"mddsPLS"`

```
library(ddsPLS)
data("liverToxicity")
X <- scale(liverToxicity$gene)
Xs <- list(Block_1 = X[, 1:1910], Block_2 = X[, -(1:1910)])
Xs[[1]][1:5, ] = Xs[[2]][6:10, ] <- NA
Y <- scale(liverToxicity$clinic)
```

Let us first illustrate the main function `mddsPLS()`. For easy readability of the outputs, let us first consider $R = 3$ and $L_0 = 10$. These two parameters will be tuned in the last part via cross-validation with the specific function `perf_mddsPLS()`.

```
model <- mddsPLS(Xs = Xs, Y = Y, L0 = 10, R = 3, mode = 'reg')
```

Some useful outputs

The most important outputs, from the user point of view, are :

- `var_selected`: the list of the weights and super-weights per block and per component. This object is useful for technical reasons. The `plot` method allows helpful visualizations.
- `Xs`: the list of the covariate blocks imputed with the *Koh-Lanta* algorithm.
- `mod`: an object of the class `"MddsPLS_core"`, whose class is described at the end of the paper. It gathers all the quantities calculated in the algorithm.

The summary method

The `mddsPLS.summary()` S3 method provides useful information divided into several parts:

- the main information about the data and the model, just above the title `ddsPLS object description`.
- the description of the quality of association between the scores and the super-scores with the response variables through two different descriptors (explained variances and RV coefficients which are defined below).
- the missing value information,
- and an overview of the selection results of the algorithm, for the `X` and the `Y` parts, per block and per component. The complete list of the selected covariates is one of the three most useful outputs (`var_selected`) of the model.

The two descriptors (explained variances and RV coefficients) are defined as follows.

- The `Explained Variance`, in percent, is based on

$$f : (\mathbf{A}, \mathbf{B}) \longrightarrow \begin{cases} \frac{\left|\left|\mathbf{A}\left(\mathbf{A}^T\mathbf{A}\right)^+\mathbf{A}^T\mathbf{B}\right|\right|_F}{||\mathbf{B}||_F} \times 100 & \text{if} \quad \mathbf{B} \neq 0 \\ 0 & \text{else} \end{cases},$$

  where "$. \rightarrow ||.||_F$" is the Frobenius norm.

- The `RV coefficient` generalizes the correlation notion to matrices, see Robert and Escoufier (1976), can be computed using

$$g : (\mathbf{A}, \mathbf{B}) \longrightarrow \begin{cases} \frac{\mathrm{Tr}\left(\mathbf{A}\mathbf{A}^T\mathbf{B}\mathbf{B}^T\right)}{\mathrm{Tr}(\mathbf{A}\mathbf{A}^T)\mathrm{Tr}(\mathbf{B}\mathbf{B}^T)} \times 100 & \text{if} \quad \mathbf{A} \neq 0 \text{ and } \mathbf{B} \neq 0 \\ 0 & \text{else.} \end{cases}.$$

Those two functions, f and g, are used in the `summary` method in 4 different ways corresponding to 4 different association aspects of the built scores with the response variables. In the following list, the calculations of each of those 4 association descriptors are described.

- *Total Y with all the Super Components*: this is one single value corresponding to the overall association of the final super components and the response matrix.

$$\begin{array}{ll} \texttt{Explained Variance :} & f\left(norm_2\left(\mathbf{T}^\star\right), norm_2\left(\mathbf{Y}\right)\right) \\ \texttt{RV coefficient :} & g\left(norm_2\left(\mathbf{T}^\star\right), norm_2\left(\mathbf{Y}\right)\right) \end{array}$$

- *Total Y variance explained by each Super Component*: this is a vector whose length is equal to the number of components in the model. It allows to evaluate the association of each super component separately on the response matrix.

$$\forall r \in [\![1, R]\!], \quad \begin{array}{ll} \texttt{Explained Variance :} & f\left(norm_2\left(\mathbf{T}^{\star(r)}\right), norm_2\left(\mathbf{Y}\right)\right) \\ \texttt{RV coefficient :} & g\left(norm_2\left(\mathbf{T}^{\star(r)}\right), norm_2\left(\mathbf{Y}\right)\right) \end{array}$$

  where $\mathbf{A}^{(r)}$ denotes the rth column of the matrix $\mathbf{A}$.

- *Marginal Y variable variance explained by each Super Component*: this is a matrix with q rows and R columns that shows the association between each super component and each response variable.

$$\forall (r, j) \in [\![1, R]\!] \times [\![1, q]\!], \quad \begin{array}{ll} \texttt{Explained Variance :} & f\left(norm_2\left(\mathbf{T}^{\star(r)}\right), norm_2\left(\mathbf{Y}^{(j)}\right)\right) \\ \texttt{RV coefficient :} & g\left(norm_2\left(\mathbf{T}^{\star(r)}\right), norm_2\left(\mathbf{Y}^{(j)}\right)\right) \end{array}$$

- *Total Y variance explained by each component of each block*: this is a matrix with T rows and R columns corresponding to the association between each covariate block and each of its corresponding components.

$$\forall (r, t) \in [\![1, R]\!] \times [\![1, T]\!], \quad \begin{array}{ll} \texttt{Explained Variance :} & f\left(norm_2\left(\mathbf{T}_t^{(r)}\right), norm_2\left(\mathbf{Y}\right)\right) \\ \texttt{RV coefficient :} & g\left(norm_2\left(\mathbf{T}_t^{(r)}\right), norm_2\left(\mathbf{Y}\right)\right) \end{array}$$

The corresponding output of the `summary` is provided below.

```
summary(model)
============================================================
           ddsPLS object description
============================================================

Number of blocks: 2
Number of dimensions: 3
Regularization coefficient: 10
Number of individuals: 64
Number of variables in Y part: 10
Model built in mode regression
Maximum number of iterations in the imputation process: 50
Algorithm of imputation has converged


     Variance Explained (%)
-----------------------------------------
Total Y variance explained by all the Super Components
[1] 67
Total Y variance explained by each Super Component
[1] 60 57 54

Marginal Y variable variance explained by each Super Component
          Super Comp. 1 Super Comp. 2 Super Comp. 3
BUN.mg.dL.          69.0     66.0     54.0
Creat.mg.dL.         15.0     18.0      9.2
TP.g.dL.            2.4      1.1     15.0
ALB.g.dL.           16.0     11.0      7.5
ALT.IU.L.           96.0     88.0     84.0
SDH.IU.L.           11.0     21.0     24.0
AST.IU.L.           95.0     85.0     83.0
ALP.IU.L.           38.0     42.0     33.0
TBA.umol.L.          84.0     83.0     90.0
Cholesterol.mg.dL.     66.0     60.0     46.0

Total Y variance explained by each component of each block
    Comp. 1 Comp. 2 Comp. 3
Block_1   59   58    0
Block_2   61   54   21


          RV coefficient
-----------------------------------------------------------
Total Y with all the Super Components
[1] 0.33

Total Y with each Super Component
[1] 0.37 0.32 0.29

Each Y variable with each Super Component
          Super Comp. 1 Super Comp. 2 Super Comp. 3
BUN.mg.dL.        0.47000    0.44000    0.3000
Creat.mg.dL.       0.02100    0.03300    0.0084
TP.g.dL.          0.00058    0.00013    0.0220
ALB.g.dL.         0.02400    0.01100    0.0056
ALT.IU.L.         0.93000    0.77000    0.7100
SDH.IU.L.         0.01200    0.04300    0.0590
AST.IU.L.         0.91000    0.73000    0.6900
ALP.IU.L.         0.14000    0.17000    0.1100
TBA.umol.L.        0.71000    0.68000    0.8100
Cholesterol.mg.dL.    0.44000    0.36000    0.2100

Total Y with each component of each block
    Comp. 1 Comp. 2 Comp. 3
```

```
Block_1  0.35  0.34  0.000
Block_2  0.37  0.29  0.042


     Missing value information
-----------------------------------------

                Block_1 Block_2  Total
Number of covariates        1910.00 1206.00 3116.00
Number of missing samples      5.00  5.00  10.00
Proportion of missing samples (%)  7.81  7.81  7.81


       mddsPLS results
-----------------------------------------

At most 10 covariate(s) can be selected in the X part


 ---- For each block of X, are selected
    Super Comp. 1 Super Comp. 2 Super Comp. 3
Block_1       4       4       0
Block_2       5       5       1
 ---- For the Y block, are selected
    @ (2,2,1) variable(s)


         Thank s for using me
------------------------------------------------------------
                     Hadrien Lorenzo
               hadrien.lorenzo.2015@gmail.com
============================================================
```

In the following, the different visualizations available via the `plot` function are described.

Visualization of the weights

The `weights`, $\mathbf{U}_t, t = 1 \ldots, T$, computed in the first step of the algorithm, denote the importance of the marginal interaction between the selected covariates of each covariate block and the response block.

```
plot(model, vizu = 'weights', mar_left = 3, variance = 'RV')
```

Figure 2 shows these weights. One can observe that:

- the third component of `Block_1` is empty, i.e. no covariate has been selected on this component;
- the other components of *Block_1* or *Block_2* are all non null;
- the second component of *Block_2* is built with only one covariate and has a *RV coefficient* equal to 0.29 with the response matrix, which is important for a single covariate;
- all the other components are built with four or five covariates.

Note that only the study of the *scaled super-weights*, $\boldsymbol{U}_t^\star, t = 1, \ldots, T$, would allow to compare the importance of the different covariates from one block to another.

Visualization of the scaled super-weights

The *scaled super-weights*, $\boldsymbol{U}_t^\star, t = 1 \ldots, T$, computed in the second step of the algorithm, denote the importance of each covariate predicting the response block in the first column as shown in Figures 3, 4 and 5. These three figures are respectively obtained with the following code lines:

```
plot(model, vizu = 'weights', super = T, mar_left = 3, variance = 'RV',
     addY = T, pos_legend = NULL, mar_bottom = 3.5, comp = 1)
plot(model, vizu = 'weights', super = T, mar_left = 3, variance = 'RV',
     addY = T, pos_legend = NULL, mar_bottom = 3.5, comp = 2)
plot(model, vizu = 'weights', super = T, mar_left = 3, variance = 'RV',
     addY = T, pos_legend = NULL, mar_bottom = 3.5, comp = 3)
```

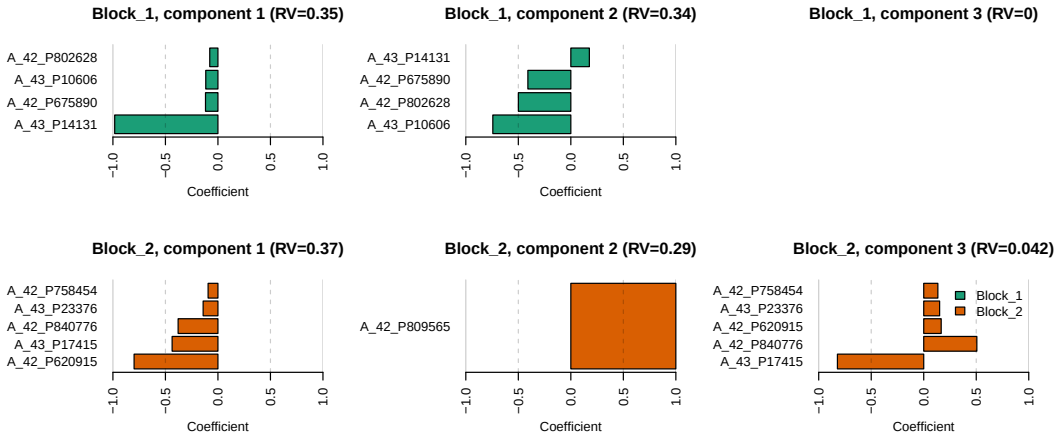


Figure 2: Visualization of the *weights*

One can note that

- the second column denotes the variance shared (according to the *RV coefficient* in that case) between the built component and each of the variables of the response block, the variables which are not selected are plotted in light blue while the selected variables are in dark blue;
- the first two components – Figures 3 and 4 – mainly explain the response variables *ALT.IU.L* and *AST.IU.L* and the third component – Figures 5 – explain the response variable *TBA.umol.L* preferentially, using the single covariate selected on component 2 of *Block_2* already described in Figure 2;
- these three response variables are the most described by the current model and the associated covariates are ranked by increasing importance in the first column of those Figures 3, 4 and 5, by component.

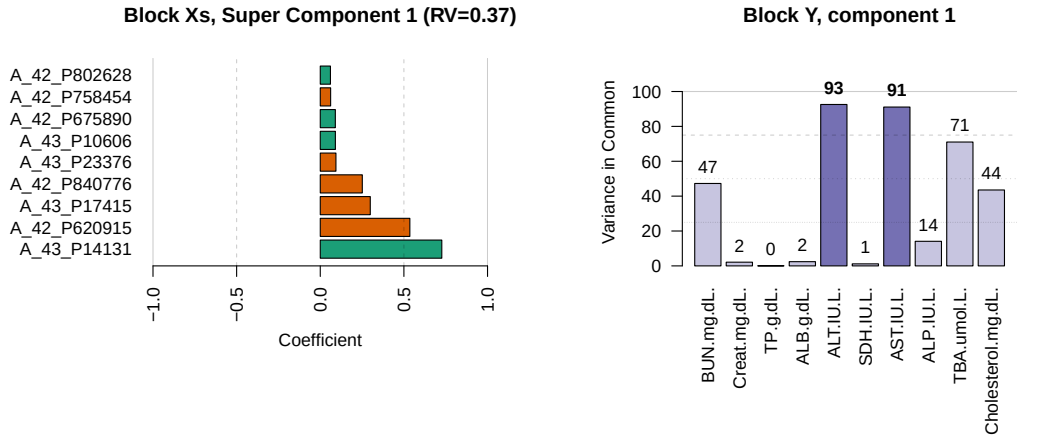


Figure 3: Visualization of the *scaled super-weights* of the 1^{st} component

Heatmaps to visualize individual behaviors along single super-component

The package also allows to draw heatmaps mixing covariates and response variables selected per components. For example, Figure 6 shows the results for the first and the third super components generated by the following code lines:

```
plot(model, vizu = 'heatmap', comp = 1, pos_legend = 'topright',
     margins_heatmap = c(1.5, 11))
plot(model, vizu = 'heatmap', comp = 1, pos_legend = 'topright',
     margins_heatmap = c(1.5, 11))
```

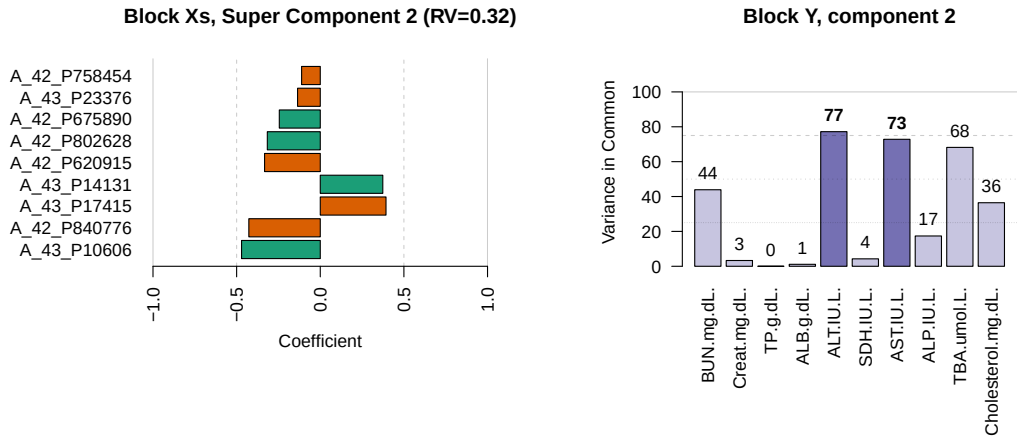


Figure 4: Visualization of the *scaled super-weights* of the 2^{nd} component

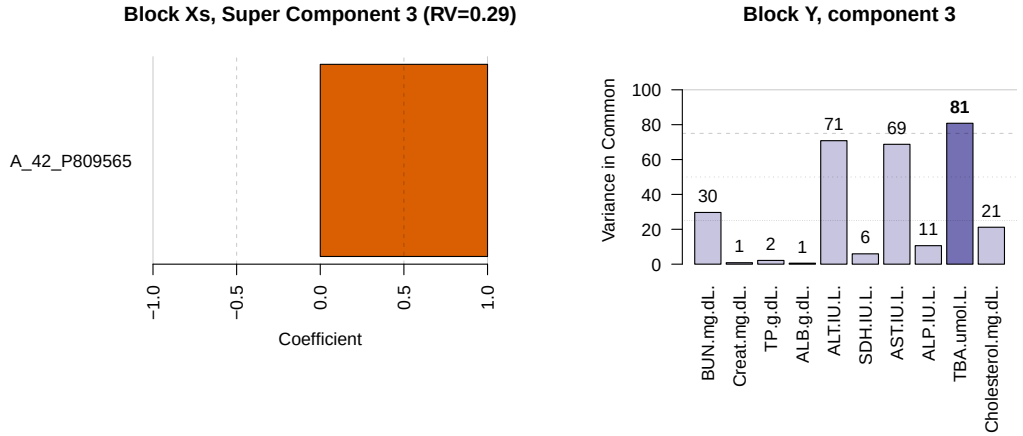


Figure 5: Visualization of the *scaled super-weights* of the 3^{rd} component

In these plots, dendrograms have been performed using hierarchical classification based on Ward distance (see Ward Jr, 1963, for example). In Figure 6, one can observe that

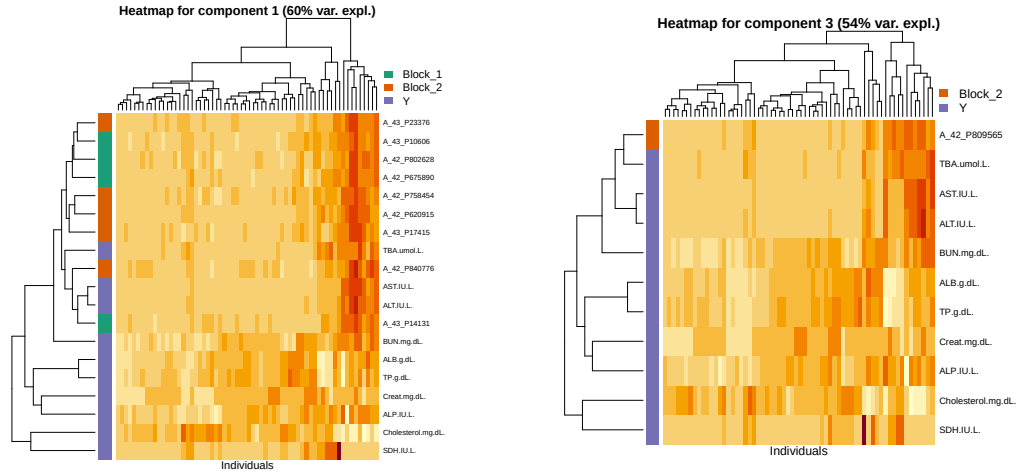
- each heatmap exhibits that individuals are strongly divided in two groups (dendrogram on the top of the graph);
- for each component, the two groups of individuals are unbalanced. For example, for the first component, the group of 9 individuals on the right of the heatmap are represented with dark colors, which is associated with high expression. On Figure 6b, 13 individuals seem to show high expression, at the far right of the figure, and correlated values.

Correlograms per super-component

Figure 7 represents the correlation matrices grouping covariates selected on components 1 and 3 respectively obtained with the code lines

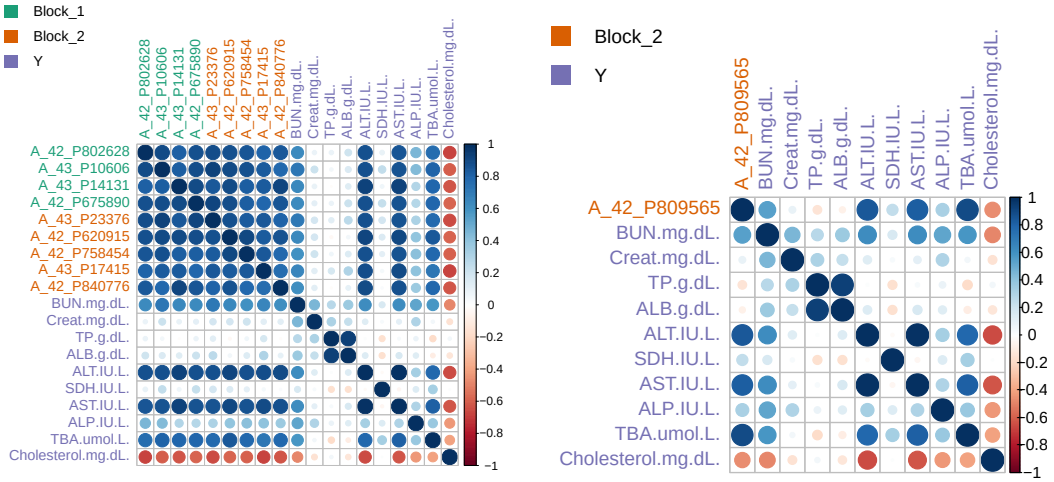
```
plot(model, vizu = 'correlogram', comp = 1)
plot(model, vizu = 'correlogram', comp = 3)
```

Contrary to heatmaps previously presented, the covariates are not ordered thanks to the Ward distance. This may be an interesting way to present the already discussed points. The value of the correlation coefficients can be plotted by setting the *values_corr* parameter to `TRUE` in the command showed just above. One can observe in Figure 7 that the variable *Cholesterol.mg.dl.* is negatively correlated with other response variables but also with all the covariates.



(a) 1^{st} component **(b)** 3^{rd} component

Figure 6: Heatmaps of the 1^{st} and 3^{rd} components mixing response and covariate selected per components and ordered with associated dendrogram. High expressions are represented with dark colors.



(a) 1^{st} Component **(b)** 3^{rd} Component

Figure 7: Correlograms of the 1^{st} and 3^{rd} components mixing response and covariates.

Predict values from a new sample

Based on an existing `mddsPLS` model, the `mddsPLS.predict()` S3 method allows to predict, from a new sample `X`,

- the response values associated with the new `X` sample,
- the imputed new `X` sample (if there are missing values).

Parameter `type` allows to choose between one of those two prediction types: 'y' or 'x' respectively allows the user to predict the response values and the imputed values of the concerned data set. The case 'both' allows to get both of the results in a single list object. In practice this can be done with the code lines:

```
y_predict <- predict(model, newdata = Xs, type = 'y')
x_predict <- predict(model, newdata = Xs, type = 'x')
both_predict <- predict(model, newdata = Xs, type = 'both')
```

Perform cross-validation

The aim of the "perf_mddsPLS" class is to determine optimal values for the tuning parameters, (R, λ) or (R, L_0), via cross-validation. The arguments of that class are detailed in the Table 3. For sake of clarity, arguments for which description are filled with "_" have been already detailed in Table 2.

Argument	Description
Xs	_
Y	_
lambda_min	A real in [0,1]. The minimum value of λ considered. Default is 0.
lambda_max	A real in [0,1]. The maximum value of λ considered. Default is NULL, interpreted as the largest correlation between X and Y.
n_lambda	A strictly positive integer corresponding to the number of λ values to consider in cross-validation, the λ values are equidistributed between lambda_min and lambda_max. Default is 1.
lambdas	A vector of λ values (in [0,1]) to consider in cross-validation. Default is NULL, when that parameter is not taken into account.
R	_
reg_imp_model	Logical. Whether or not to regularize the imputation models. Initialized to TRUE.
L0s	A vector of non null positive integers corresponding to the L_0 values to consider in cross-validation. Default is NULL and is then not taken into account.
kfolds	Character or integer. If equals to 'loo' then a leave-one-out cross-validation is started. No other characters are understood. Any strictly positive integer gives the number of folds to make in the cross-validation process.
mode	_
fold_fixed	Vector of length n. Each element corresponds to the number of the corresponding fold (see details below). If NULL then that argument is not considered. Default to NULL.
maxIter_imput	_
errMin_imput	_
NCORES	Integer. The number of cores to use (see details below). Default is 1.
NZV	_
plot_result	Logical. Whether or not to plot the result. Initialized to TRUE. The reg_error argument of the plot.perf_mddsPLS() function is left to its default value.
legend_label	Logical. Whether or not to add the legend names to the plot. Initialized to TRUE.

Table 3: Argumets of the class "perf_mddsPLS"

The NCORES allows to select the number of *cpu* that should be used by the software. This is bounded by the maximum number of *cpu* that are present on the machine and protected in that way.

The user must choose between λ and L_0 to make the regularization/selection. The L0s parameter is a vector of L_0 values to consider in the cross-validation process. Otherwise, for the λ parameters, the user can whether use

- n_lambda values of λ to consider in the cross-validation process. These values are taken equidistributed in the range corresponding to no covariate to remove on the left and no covariate to add on the right. lambda_min and/or lambda_max can be used to move those two previous automatic bounds.
- or lambdas, a vector of λ values to consider in the cross-validation process.

The number of folds in the cross-validation process is fixed thanks to the parameter kfolds, when that parameter is equal to the number of individuals or if it is equal to 'loo', then leave-one-out cross-validation is performed, otherwise k-fold cross-validation is performed.

Sometimes a more complex case can be encountered when dependant samples are considered for instance. In that specific context it might be interesting or even necessary to keep some samples together. The vector parameter fold_fixed of size n must then be used. If the considered data-set is build on nine samples and the first three must be kept together, as the next three and the last three ones, then fold_fixed should be equal to:

```
fold_fixed <- c(1, 1, 1, 2, 2, 2, 3, 3, 3)
```

Start cross-validation

The following example allows to select, for $R = 2$ components (value fixed by the user), the best value for L_0 using leave-one-out cross-validation. Only six different values of L_0 have been tested, deploying the calculations on three cores.

```
cv <- perf_mddsPLS(Xs = Xs, Y = Y, L0s = c(1, 3, 5, 10, 25, 50),
                   R = 2, mode = 'reg', kfolds = 'loo', NCORES = 3,
                   plot_result = FALSE)
```

The summary method

The part of the output denoted as `Cross-Validation results`allows to access to the proportion of models which have converged against the total number of models built (third column) and also to the main time statistics in terms of computation time (fourth column) for given R and L_0 parameters (first and second columns respectively).

```
summary(cv, plot_res_cv = FALSE)
=====================================================
   Cross-Validation ddsPLS object description
=====================================================

Number of blocks: 2
Number of individuals: 64
Number of variables in Y part: 10
Model built in mode regression


  Missing value information
---------------------------------

                 Block_1 Block_2  Total
Number of variables        1910.00 1206.00 3116.00
Number of missing samples      5.00  5.00  10.00
Proportion of missing samples (%)  7.81  7.81  7.81


   Cross-Validation results
---------------------------------

 R L0 Nb.of.convergences.VS.nb.of.fold Mean.AND.sd.time.of.computation
1 2 1         64/64      0.19(0.025)
2 2 3         64/64      0.21(0.023)
3 2 5         64/64      0.22(0.021)
4 2 10         64/64      0.21(0.02)
5 2 25         64/64      0.22(0.022)
6 2 50         64/64      0.22(0.019)


         Thank s for using me
-----------------------------------------------------
                   Hadrien Lorenzo
            hadrien.lorenzo.2015@gmail.com
=====================================================
```

In that example, all the models have converged and each model is build in around $0.2s$.

The plot method

The plot method allows to plot, for a given value for R, the error curves in the regression case (the lower the better) and the accuracy curves in the classification case (so the higher the better). In the case of multivariate regression, it is important to perform the same normalization on response variables as

to get comparable *MSEP* (*Mean Squared Error in Prediction*) curves. As a reminder, *MSEP* is computed on the estimation $\hat{\mathbf{Y}}$ of $\mathbf{Y}$ such as

$$MSEP\left(\mathbf{Y}, \hat{\mathbf{Y}}\right) = \text{diag}\left(\left(\mathbf{Y} - \hat{\mathbf{Y}}\right)^T \left(\mathbf{Y} - \hat{\mathbf{Y}}\right)\right) / n,$$

where ". $\rightarrow$ diag(.)" gives the diagonal of the square matrix argument. And so, $MSEP(\mathbf{Y}, \hat{\mathbf{Y}})$ is filled with q elements corresponding to each of the q *MSEP*s of each predicted variable of $\mathbf{Y}$.

Remark. It is also possible to check the *MPE* (*Mean Prediction Error*), the sum of the absolute errors, using the argument `reg_error` and putting it to 'MPE'. The default value is 'MSEP' and allows to appreciate the *MSEP* error.

As mentionned before, each of the response variables has been standardized (zero mean and unit variance) such as all error curves would be comparable. Hence, the error would be comparable to 1 above which the model does not perform better than mean prediction, at the limit. The following line code

```
plot(cv, legend_names = colnames(Y), pos_legend = 'right', plot_mean = T)
```

provides Figure 8 which shows the corresponding results. The three well predicted response variables are the ones already discussed in the model building part. The other variables have a *MSEP* close to 1 which means that the different leave-one-out model would not perform better than mean prediction. Two vertical dotted lines represent the value for which the mean of the *MSEP* curves is minimum and the minimum value of the q *MSEP* curves. Both of those minima can be good candidates but it belongs to the analyst to choose the most suitable parameter, taking into account that also the number of components must be chosen (and this procedure must be repeated for various values of R). In the next paragraph, a way to determine simultaneously, thanks to cross-validation,the parameters L_0 and R are provided.

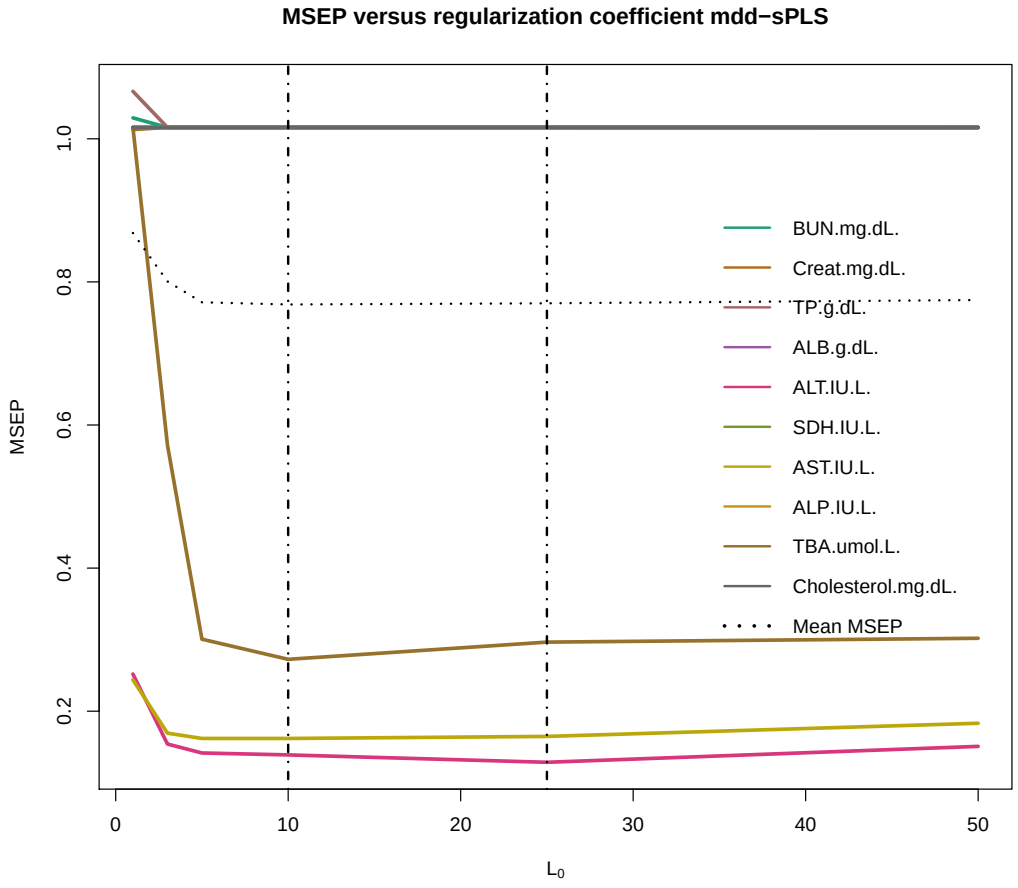


Figure 8: Cross-validation leave-one-out curve errors for the ten to be predicted variables

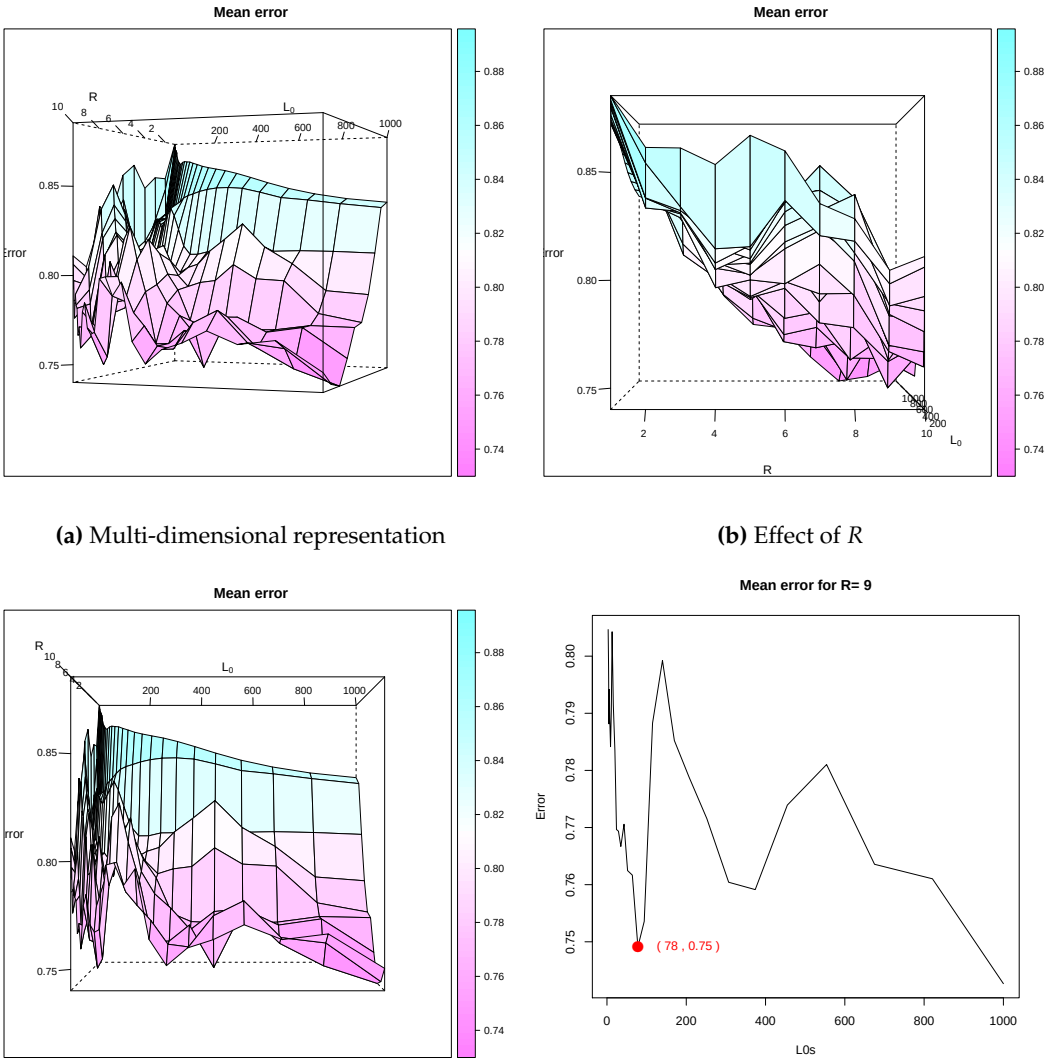
Cross-validation to fix the all parameters

Both of the tuning parameters (R, L_0) or (R, λ) of the method can be fixed thanks to cross-validation. The following code shows how to perform this analysis for (R, L_0) using the toy example. Since this is multivariate regression, it has been chosen to consider the mean of the 10 *MSEP* of leave-one-out cross-validation error as a common criterion. No function has yet been implemented to visualize that error but the following script gives an idea of a feasible solution.

```
L0s <- unique(round(seq(1, 1000, length.out = 30)))
Rs <- 1:10
mat_means <- matrix(NA, length(L0s), length(Rs))
CVs <- list()
for(r in Rs){
  CVs[[r]] <- perf_mddsPLS(Xs = Xs, Y = Y, L0s = L0s,
                           R = r, mode = 'reg', kfolds = 'loo',
                           NCORES = 7, plot_result = FALSE)
  mat_means[, r] <- rowMeans(CVs[[r]]$RMSEP[, -c(1:2)])
}
data <- expand.grid(Rs, L0s)
data <- cbind(data, as.vector(t(mat_means)))
data.f <- data.frame(data)
names(data.f) <- c('L0', 'R', 'Error')
lattice::wireframe(Error ~ L0*R, data = data.f,
                   ylab = expression(L[0]), xlab = 'R',
                   main = 'Mean error',
                   drape = TRUE,
                   colorkey = TRUE,
                   screen = list(x = -90, y = -110),
                   scales = list( arrows = FALSE)
)
lattice::wireframe(Error ~ L0*R, data = data.f,
                   ylab = expression(L[0]), xlab = 'R',
                   main = 'Mean error',
                   drape = TRUE,
                   colorkey = TRUE,
                   screen = list(x = -90),
                   scales = list( arrows = FALSE)
)
lattice::wireframe(Error ~ L0*R, data = data.f,
                   ylab = expression(L[0]), xlab = 'R',
                   main = 'Mean error',
                   drape = TRUE,
                   colorkey = TRUE,
                   screen = list(y = -90, z = -90),
                   scales = list( arrows = FALSE)
)
r <- 9
Err <- rowMeans(CVs[[r]]$RMSEP[, -c(1:2)])
pos_min <- which.min(Err[which(L0s<200)])
L0_min <- L0s[pos_min]
Err_min <- Err[pos_min]
plot(L0s, Err, ylab = 'Error', main = paste('Mean error for R = ', r),
     type = 'l')
points(L0_min, Err_min, pch = 16, col = 'red', cex = 2)
text(L0_min, Err_min,
     labels = paste('(', round(L0_min, 2), ', ',
round(Err_min, 2), ')'), col = 'red', adj = -0.3)
```

Figures 9a, 9b and 9c show the results through 3-dimensional plots of the mean errors along the 10 response variables. It seems that $R = 9$ and $L_0 \approx 78$ (red point in Figure 9d) allow to get an interesting point in terms of prediction error and sparsity, discussions should be conducted with domain specialists in front of those curves to select the best model.

Since the tuning parameters are now determined, the methods detailed before in the manuscript can be used to build the model and should help the user to learn more about the data.



(a) Multi-dimensional representation

(b) Effect of R

(c) Effect of L_0

(d) Mean error for $R = 9$, red point corresponds to the chosen minimum

Figure 9: Plot of the means of the 10 response variables MSEP's for cross-validation leave-one-out analysis.

How to implement the classification case?

Until now, the paper mainly focuses on the regression case. Classification case can also be taken into account, following Barker and Rayens (2003). It is quite transparent for the user, changing the argument `mode` in functions `mddsPLS()` and `perf_mddsPLS()` to one of

- '`lda`', for *linear discriminant analysis*,
- '`logit`', for *logistic regression* (it only works in two-classes classification problems).

Recall also that the cross-validation is monitored through the *good classification rate* (and not the *error rate* as in regression case) which implies to look for maxima on the curves (and not minima as in regression case).

The hidden class of the package

The user cannot build directly an object of this class because it is protected according to different aspects corresponding to whether there are missing values and whether L_0 or λ is used as a parameter. This class is then used internally in the *Koh-Lanta* process. Any object of this class would be an attribute of the `"mddsPLS"` class. The attributes of that class are given in Table 4.

Attribute	Symbol	Description
`u`	$(\mathbf{U}_t)_{t\in[\![1,T]\!]}$	A list of length `T`. Each element is a $p_t \times R$ matrix.
`u_t_super`	$(\mathbf{U}^\star_t)_{t\in[\![1,T]\!]}$	A list of length `T`. Each element is a $p_t \times R$ matrix.
`v`	$\mathbf{V}^\star$	A $q \times R$ matrix : the weights for the `Y` part.
`ts`		A list of length R. Each element is a $n \times T$ matrix : the scores per component.
`( t, s)`	$(\mathbf{T}^\star, \mathbf{S}^\star)$	Two $n \times R$ matrices, super-scores of the `X` and `Y` parts.
`( t_ort,s_ort)`		Two $n \times R$ matrices, final scores of the `X` and `Y` part. They correspond to `PLS` scores of `(t,s)` scores and so $\texttt{t\_ort}^T$ `s_ort` is diagonal, `t_ort`, respectively `s_ort`, carries the same information as `t`, respectively `s`.
`B`	$\mathcal{B}$	A list of length `T`, for the number of blocks. Each element is a $p_t \times q$ matrix.
`( mu_x_s,sd_x_s)`		Two lists of length `T`, for the number of blocks. Each element is a p_t vector : the mean and standard deviation covariates per block.
(mu_y,sd_y)		Two vectors of length q : the mean and the standard deviation variables for `Y` part.
`R`	R	Given as an input.
`q`	q	A non negative integer.
`Ms`	$(\mathbf{M}_t)_{t\in[\![1,T]\!]}$	A list of length `T`, for the number of blocks.
`lambda`	λ	The regularization coefficient.

Table 4: Attributes of the class `mddsPLS_core`

Let be supposed that a model `Model` from the `"mddsPLS"` class has been created, then one can get to the corresponding object of the `"MddsPLS_core"` class such as

```
Model <- mddsPLS(Xs = Xs, Y = Y, L0 = 10, R = 3, mode = 'reg')
Model_MddsPLS_core_object <- Model$mod
Out <- lapply(Model_MddsPLS_core_object, function(x){
  out <- dim(x)
  if(is.null(out)){
    out <- length(x)
  }else{
    out <- paste(out, sep = '', collapse = 'x')
  }
  out
 })
```

The printed command shows the dimension(s) of the attributes of the considered object of the `"MddsPLS_core"` class.

```
print(do.call(rbind, Out))
     [,1]
u     "2"
u_t_super "2"
V_super  "10x2"
ts    "2"
beta_comb "4x2"
T_super  "64x2"
S_super  "64x2"
t_ort   "64x2"
s_ort   "64x2"
B     "2"
mu_x_s  "2"
sd_x_s  "2"
mu_y   "10"
sd_y   "10"
R     "1"
q     "1"
Ms    "2"
lambda  "1"
```

Conclusion

The package **ddsPLS** allows to solve linear multivariate regression or classification problems in the context of heterogeneous multiblock data sets for high dimensional data.

The **ddsPLS** package is simple to use and provides dimension reduction and variable selection through a computational method based on SVD of correlation matrix soft-thresholding. Computation time has been reduced thanks to C++ functions using the **Rcpp** package. A cross-validation function using parallel computations has also been included in that package thanks to the **doParallel**, **parallel** and **foreach** packages. Different S3 methods allow to visualize model specificities (through barplots, heatmaps or correlogram) and cross-validation performances.

Classification case analyses are performed thanks to `lda()` or `logit()` functions (from the **MASS** package), those functions are based on the `ddsPLS` objects built on the regression multivariate models predicting the dummy matrix generated from the class response.

Acknowledgments

Hadrien Lorenzo is supported by a 2016 Inria-Inserm thesis grant *Médecine Numérique* (for *Digital Medicine*).

Included data sets

Two data sets are included in the package and correspond to

- a regression problem, through the `liverToxicity` data set. This data set contains the expression measure of 3116 genes and 10 clinical measurements for $n = 64$ subjects (rats) that were exposed to non-toxic, moderately toxic or severely toxic doses of acetaminophen in a controlled experiment (see Bushel et al., 2007).
- a classification problem. The data set `penicilliumYES` has 36 rows and 3754 columns. The covariates are 1st order statistics from multi-spectral images of three species of Penicillium fungi: Melanoconidium, Polonicum, and Venetum (see Clemmensen et al., 2011).

Bibliography

E. Acar, T. G. Kolda, and D. M. Dunlavy. All-at-once Optimization for Coupled Matrix and Tensor Factorizations. *arXiv preprint arXiv:1105.3422*, 2011. [p2]

E. Acar, M. A. Rasmussen, F. Savorani, T. Næs, and R. Bro. Understanding data fusion within the framework of coupled matrix and tensor factorizations. *Chemometrics and Intelligent Laboratory Systems*, 129:53–63, 2013. URL https://doi.org/10.1016/j.chemolab.2013.06.006. [p2]

S. Bakin. *Adaptive regression and model selection in data mining problems*. PhD thesis, School of Mathematical Sciences, Australian National University, 1999. URL https://doi.org/10.25911/5d78db4c25dbb. [p1]

M. Barker and W. Rayens. Partial least squares for discrimination. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 17(3):166–173, 2003. URL https://doi.org/10.1002/cem.785. [p17]

P. R. Bushel, R. D. Wolfinger, and G. Gibson. Simultaneous clustering of gene expression data with clinical chemistry and pathological evaluations reveals phenotypic prototypes. *BMC Systems Biology*, 1(1):15, 2007. URL https://doi.org/10.1186/1752-0509-1-15. [p4, 18]

L. Clemmensen, T. Hastie, D. Witten, and B. Ersbøll. Sparse discriminant analysis. *Technometrics*, 53(4): 406–413, 2011. URL https://doi.org/10.1198/TECH.2011.08118. [p18]

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977. [p1]

M.-P. Ellies-Oury, H. Lorenzo, C. Denoyelle, J. Saracco, and B. Picard. An original methodology for the selection of biomarkers of tenderness in five different muscles. *Foods*, 8(6):206, 2019. URL https://doi.org/10.3390/foods8060206. [p2]

T. Hastie and R. Mazumder. softImpute: Matrix Completion via Iterative Soft-Thresholded SVD. *R package version*, 1, 2015. [p1]

J. Josse and F. Husson. Handling missing values in exploratory multivariate data analysis methods. *Journal de la Société Française de Statistique*, 153(2):79–99, 2012. [p2]

J. Josse and F. Husson. missMDA: A Package for Handling Missing Values in Multivariate Data Analysis. *Journal of Statistical Software*, 70(1):1–31, 2016. URL https://doi.org/10.18637/jss.v070.i01. [p2]

J. Josse, F. Husson, and J. Pagès. Gestion des données manquantes en analyse en composantes principales. *Journal de la Société Française de Statistique*, 150(2):28–51, 2009. [p1]

J. F. Kenney and E. Keeping. Linear regression and correlation. *Mathematics of statistics*, 1:252–285, 1962. [p4]

K.-A. Lê Cao, D. Rossouw, C. Robert-Granié, and P. Besse. A Sparse PLS for Variable Selection when Integrating Omics data. *Statistical applications in genetics and molecular biology*, 7(1), 2008. URL https://doi.org/10.2202/1544-6115.1390. [p1]

B. Liquet, P. L. de Micheaux, B. P. Hejblum, and R. Thiébaut. Group and sparse group partial least square approaches applied in genomics context. *Bioinformatics*, 32(1):35–42, 2015. URL https://doi.org/10.1093/bioinformatics/btv535. [p1]

H. Lorenzo, R. Misbah, J. Odeber, P.-E. Morange, J. Saracco, D.-A. Trégouët, and R. Thiébaut. High-dimensional multi-block analysis of factors associated with thrombin generation potential. In *2019 IEEE 32nd International Symposium on Computer-Based Medical Systems (CBMS)*, pages 453–458. IEEE, 2019a. URL https://doi.org/10.1109/CBMS.2019.00094. [p2]

H. Lorenzo, J. Saracco, and R. Thiébaut. Supervised learning for multi-block incomplete data. *arXiv preprint arXiv:1901.04380*, 2019b. [p2]

R. Mazumder, T. Hastie, and R. Tibshirani. Spectral regularization algorithms for learning large incomplete matrices. *Journal of machine learning research*, 11(Aug):2287–2322, 2010. [p1]

M. Planitz. 3. Inconsistent systems of linear equations. *The Mathematical Gazette*, 63(425):181–185, 1979. URL https://doi.org/10.2307/3617890. [p4]

P. Robert and Y. Escoufier. A Unifying Tool for Linear Multivariate Statistical Methods: The RV-Coefficient. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 25(3):257–265, 1976. URL https://doi.org/10.2307/2347233. [p6]

D. B. Rubin. Inference and missing data. *Biometrika*, 63(3):581–592, 1976. URL https://doi.org/10.2307/2335739. [p1]

J. L. Schafer and J. W. Graham. Missing data: our view of the state of the art. *Psychological methods*, 7 (2):147, 2002. URL http://doi.org/10.1037/1082-989X.7.2.147. [p1]

N. Simon, J. Friedman, T. Hastie, and R. Tibshirani. A Sparse-Group Lasso. *Journal of Computational and Graphical Statistics*, 22(2):231–245, 2013. URL https://doi.org/10.1080/10618600.2012.681250. [p1]

D. J. Stekhoven and P. Bühlmann. MissForest-—non-parametric missing value imputation for mixed-type data. *Bioinformatics*, 28(1):112–118, 2011. URL https://doi.org/10.1093/bioinformatics/btr597. [p1]

A. Tenenhaus and M. Tenenhaus. Regularized generalized canonical correlation analysis. *Psychometrika*, 76(2):257, 2011. URL https://doi.org/10.1007/s11336-011-9206-8. [p1]

A. Tenenhaus, C. Philippe, V. Guillemot, K.-A. Le Cao, J. Grill, and V. Frouin. Variable selection for generalized canonical correlation analysis. *Biostatistics*, 15(3):569–583, 2014. URL https://doi.org/10.1093/biostatistics/kxu001. [p1]

R. Tibshirani. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996. URL https://doi.org/10.1111/j.2517-6161.1996.tb02080.x. [p1]

L. Wangen and B. Kowalski. A multiblock partial least squares algorithm for investigating complex chemical systems. *Journal of chemometrics*, 3(1):3–20, 1989. URL https://doi.org/10.1002/cem.1180030104. [p1]

J. H. Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963. URL https://doi.org/10.1080/01621459.1963.10500845. [p10]

R. A. Willoughby. Solutions of ill-posed problems (an tikhonov and vy arsenin). *SIAM Review*, 21(2): 266, 1979. URL https://doi.org/10.1137/1021044. [p1]

S. Wold, M. Sjöström, and L. Eriksson. PLS-regression: a basic tool of chemometrics. *Chemometrics and intelligent laboratory systems*, 58(2):109–130, 2001. URL https://doi.org/10.1016/S0169-7439(01)00155-1. [p1]

Hadrien Lorenzo
ISTM INRIA BSO
Inserm-U1219 BPH
ISPED, Bordeaux University
146 rue Léo Saignat
33000, Bordeaux, France
hadrien.lorenzo@u-bordeaux.fr

Jérême Saracco
CQFD INRIA BSO
IMB, UMR 5251 CNRS
ENSC - Bordeaux INP
109 avenue Roul
33400, Talence, France
jerome.saracco@ensc.fr

Rodolphe Thiébaut
SISTM INRIA BSO
INSERM-U1219 BPH
ISPED, Bordeaux University
146 rue Léo Saignat
33000, Bordeaux, France
rodolphe.thiebaut@u-bordeaux.fr