mixOmics vignette

Kim-Anh L Cao, Sebastien Dejean, Al J Abadi

July 25, 2019

Contents

Preface

This document outlines some of the utilities of *mixOmics* package. If you run into any issues reproducing these results, please let us know by creating an issue here. Also, feel free to share your experience/suggestions in mixOmics Discourse forum.

Chapter 1

Introduction

`mixOmics` is an R toolkit dedicated to the exploration and integration of biological data sets with a specific focus on variable selection. The package currently includes nineteen multivariate methodologies, mostly developed by the `mixOmics` team (see some of our references in 1.2.3). Originally, all methods were designed for omics data, however, their application is not limited to biological data only. Other applications where integration is required can be considered, but mostly for the case where the predictor variables are continuous (see also 1.1).

In `mixOmics`, a strong focus is given to graphical representation to better translate and understand the relationships between the different data types and visualize the correlation structure at both sample and variable levels.

1.1 Input data

Note the data pre-processing requirements before analysing data with `mixOmics`:

- **Types of data**. Different types of biological data can be explored and integrated with `mixOmics`. Our methods can handle molecular features measured on a continuous scale (e.g. microarray, mass spectrometry-based proteomics and metabolomics) or sequenced-based count data (RNA-seq, 16S, shotgun metagenomics) that become 'continuous' data after pre-processing and normalisation.
- **Normalisation**. The package does not handle normalisation as it is platform-specific and we cover a too wide variety of data! Prior to the analysis, we assume the data sets have been normalised using appropriate normalisation methods and pre-processed when applicable.
- **Prefiltering**. While `mixOmics` methods can handle large data sets (several tens of thousands of predictors), we recommend pre-filtering the data

to less than 10K predictor variables per data set, for example by using Median Absolute Deviation (Teng et al., 2016) for RNA-seq data, by removing consistently low counts in microbiome data sets (Lê Cao et al., 2016) or by removing near-zero variance predictors. Such step aims to lessen the computational time during the parameter tuning process.

- **Data format**. Our methods use matrix decomposition techniques. Therefore, the numeric data matrix or data frames have n observations or samples in rows and p predictors or variables (e.g. genes, proteins, OTUs) in columns.
- **Covariates**. In the current version of `mixOmics`, covariates that may confound the analysis are not included in the methods. We recommend correcting for those covariates beforehand using appropriate univariate or multivariate methods for batch effect removal. Contact us for more details as we are currently working on this aspect.

1.2 Methods

1.2.1 Some background knowledge

We list here the main methodological or theoretical concepts you need to know to be able to efficiently apply `mixOmics`:

- **Individuals, observations or samples**: the experimental units on which information are collected, e.g. patients, cell lines, cells, faecal samples etc.
- **Variables, predictors**: read-out measured on each sample, e.g. gene (expression), protein or OTU (abundance), weight etc.
- **Variance**: measures the spread of one variable. In our methods, we estimate the variance of components rather that variable read-outs. A high variance indicates that the data points are very spread out from the mean, and from one another (scattered).
- **Covariance**: measures the strength of the relationship between two variables, i.e. whether they co-vary. A high covariance value indicates a strong relationship, e.g. weight and height in individuals frequently vary roughly in the same way; roughly, the heaviest are the tallest. A covariance value has no lower or upper bound.
- **Correlation**: a standardized version of the covariance that is bounded by -1 and 1.
- **Linear combination**: variables are combined by multiplying each of them by a coefficient and adding the results. A linear combination of height and weight could be $2 * \text{weight} - 1.5 * \text{height}$ with the coefficients 2 and -1.5 assigned with weight and height respectively.

- **Component**: an artificial variable built from a linear combination of the observed variables in a given data set. Variable coefficients are optimally defined based on some statistical criterion. For example in Principal Component Analysis, the coefficients of a (principal) component are defined so as to maximise the variance of the component.

- **Loadings**: variable coefficients used to define a component.

- **Sample plot**: representation of the samples projected in a small space spanned (defined) by the components. Samples coordinates are determined by their components values or scores.

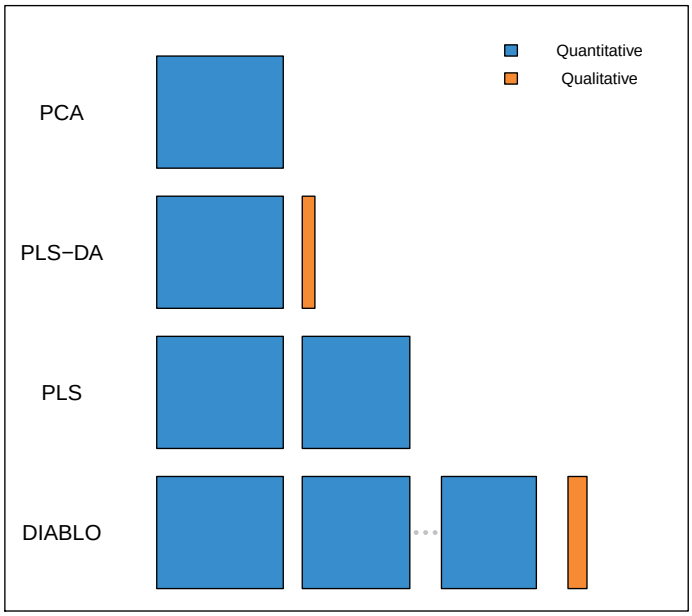
- **Correlation circle plot**: representation of the variables in a space spanned by the components. Each variable coordinate is defined as the correlation between the original variable value and each component. A correlation circle plot enables to visualise the correlation between variables - negative or positive correlation, defined by the cosine angle between the centre of the circle and each variable point) and the contribution of each variable to each component - defined by the absolute value of the coordinate on each component. For this interpretation, data need to be centred and scaled (by default in most of our methods except PCA). For more details on this insightful graphic, see Figure 1 in (González et al., 2012).

- **Unsupervised analysis**: the method does not take into account any known sample groups and the analysis is exploratory. Examples of unsupervised methods covered in this vignette are Principal Component Analysis (PCA, Chapter 3), Projection to Latent Structures (PLS, Chapter 5), and also Canonical Correlation Analysis (CCA, not covered here).

- **Supervised analysis**: the method includes a vector indicating the class membership of each sample. The aim is to discriminate sample groups and perform sample class prediction. Examples of supervised methods covered in this vignette are PLS Discriminant Analysis (PLS-DA, Chapter 4), DIABLO (Chapter 6) and also MINT (not covered here (Rohart et al., 2017b)).

1.2.2 Overview

Here is an overview of the most widely used methods in `mixOmics` that will be further detailed in this vignette, with the exception of rCCA and MINT. We depict them along with the type of data set they can handle.



1.2.3 Key publications

The methods implemented in `mixOmics` are described in detail in the following publications. A more extensive list can be found at this link.

- **Overview and recent integrative methods**: Rohart F., Gautier, B, Singh, A, Le Cao, K. A. mixOmics: an R package for 'omics feature selection and multiple data integration. *PLoS Comput Biol* 13(11): e1005752.
- **Graphical outputs for integrative methods**: (González et al., 2012) Gonzalez I., Le Cao K.-A., Davis, M.D. and Dejean S. (2012) Insightful graphical outputs to explore relationships between two omics data sets. *BioData Mining* 5:19.
- **DIABLO**: Singh A, Gautier B, Shannon C, Vacher M, Rohart F, Tebbutt S, K-A. Le Cao. DIABLO - multi-omics data integration for biomarker discovery.
- **sparse PLS**: Le Cao K.-A., Martin P.G.P, Robert-Granie C. and Besse, P. (2009) Sparse Canonical Methods for Biological Data Integration: application to a cross-platform study. *BMC Bioinformatics*, 10:34.
- **sparse PLS-DA**: Le Cao K.-A., Boitard S. and Besse P. (2011) Sparse PLS Discriminant Analysis: biologically relevant feature selection and graphical displays for multiclass problems. *BMC Bioinformatics*, 22:253.

Framework		Function name	Sparse	Prediction
Single 'omics	unsupervised	pca	-	-
		ipca	-	-
		sipca	✓	-
		spca	✓	-
	supervised	plsda	-	✓
		splsda	✓	✓
N-integration	unsupervised (2 'omics)	rcca	-	-
		pls	-	✓
		spls	✓	✓
	unsupervised	wrapper.rgcca	-	-
		wrapper.sgcca	✓	✓
		block.pls	-	✓
		block.spls	✓	✓
	supervised (DIABLO)	block.plsda	-	✓
		block.splsda	✓	✓
P-integration (MINT)	unsupervised	mint.pls	-	✓
		mint.spls	✓	✓
	supervised	mint.plsda	-	✓
		mint.splsda	✓	✓

Figure 1.1: List of methods in mixOmics, sparse indicates methods that perform variable selection

- **Multilevel approach for repeated measurements**: Liquet B, Le Cao K-A, Hocini H, Thiebaut R (2012). A novel approach for biomarker selection and the integration of repeated measures experiments from two assays. *BMC Bioinformatics*, 13:325
- **sPLS-DA for microbiome data**: Le Cao K-A*, Costello ME *, Lakis VA , Bartolo F, Chua XY, Brazeilles R and Rondeau P. (2016) MixMC: Multivariate insights into Microbial Communities.PLoS ONE 11(8): e0160169

1.3 Outline of this Vignette

- **Chapter 2** details some practical aspects to get started
- **Chapter 3**: Principal Components Analysis (PCA)
- **Chapter 4**: Projection to Latent Structure - Discriminant Analysis (PLS-DA)
- **Chapter 5**: Projection to Latent Structures (PLS)
- **Chapter 6**: Integrative analysis for multiple data sets (DIABLO)

	functions	PCA	rCCA	PLS	sPLS	PLS-DA	sPLS-DA	DIABLO	MINT
function call		pca	rcc	pls	spls	plsda	splsda	block.splsda	mint.splsda
parameters		ncomp	ncomp lambda1 lambda2	ncomp mode	ncomp mode keepX	ncomp	ncomp keepX	design ncomp keepX	ncomp keepX
performance	tune, plot.tune	✓		✓	✓		✓	✓	✓
	perf, plot.perf			✓	✓	✓	✓	✓	✓
	auroc					✓	✓	✓	✓
sample plot	plotIndiv	✓	✓	✓	✓	✓	✓	✓	✓
	plotArrow		✓	✓	✓	✓	✓	✓	✓
	plotDiablo							✓	
variable plot	plotVar	✓	✓	✓	✓	✓	✓	✓	✓
	plotLoadings			✓	✓	✓	✓	✓	✓
	circosPlot							✓	
	cim	✓	✓	✓	✓	✓	✓	✓	✓
	network		✓	✓	✓	✓	✓	✓	✓
variable list	selectVar	✓		✓	✓	✓	✓	✓	✓

Figure 1.2: Main functions and parameters of each method

Each methods chapter has the following outline:

1. Type of biological question to be answered
2. Brief description of an illustrative data set
3. Principle of the method
4. Quick start of the method with the main functions and arguments
5. To go further: customized plots, additional graphical outputs, and tuning parameters
6. FAQ

1.4 Other methods not covered in this vignette

Other methods not covered in this document are described on our website and the following references:

- regularised Canonical Correlation Analysis, see the **Methods** and **Case study** tabs, and (González et al., 2008) that describes CCA for large data sets.
- Microbiome (16S, shotgun metagenomics) data analysis, see also (Lê Cao et al., 2016) and kernel integration for microbiome data. The latter is

in collaboration with Drs J Mariette and Nathalie Villa-Vialaneix (INRA Toulouse, France), an example is provided for the Tara ocean metagenomics and environmental data, see also (Mariette and Villa-Vialaneix, 2017).

- MINT or P-integration to integrate independently generated transcriptomics data sets. An example in stem cells studies, see also (Rohart et al., 2017b).

Chapter 2

Let's get started

2.1 Installation

First, download the latest `mixOmics` version from Bioconductor:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
 BiocManager::install("mixOmics")
```

Alternatively, you can install the latest GitHub version of the package:

```
BiocManager::install("mixOmicsTeam/mixOmics")
```

The `mixOmics` package should directly import the following packages: `igraph`, `rgl`, `ellipse`, `corpcor`, `RColorBrewer`, `plyr`, `parallel`, `dplyr`, `tidyr`, `reshape2`, `methods`, `matrixStats`, `rARPACK`, `gridExtra`. **For Apple mac users:** if you are unable to install the imported package `rgl`, you will need to install the XQuartz software first.

2.2 Load the package

```
library(mixOmics)
```

Check that there is no error when loading the package, especially for the `rgl` library (see above).

2.3 Upload data

The examples we give in this vignette use data that are already part of the package. To upload your own data, check first that your working directory is

set, then read your data from a `.txt` or `.csv` format, either by using **File > Import Dataset** in RStudio or via one of these command lines:

```
# from csv file
data <- read.csv("your_data.csv", row.names = 1, header = TRUE)

# from txt file
data <- read.table("your_data.txt", header = TRUE)
```

For more details about the arguments used to modify those functions, type `?read.csv` or `?read.table` in the R console.

2.4 Quick start in `mixOmics`

Each analysis should follow this workflow:

1. Run the method
2. Graphical representation of the samples
3. Graphical representation of the variables

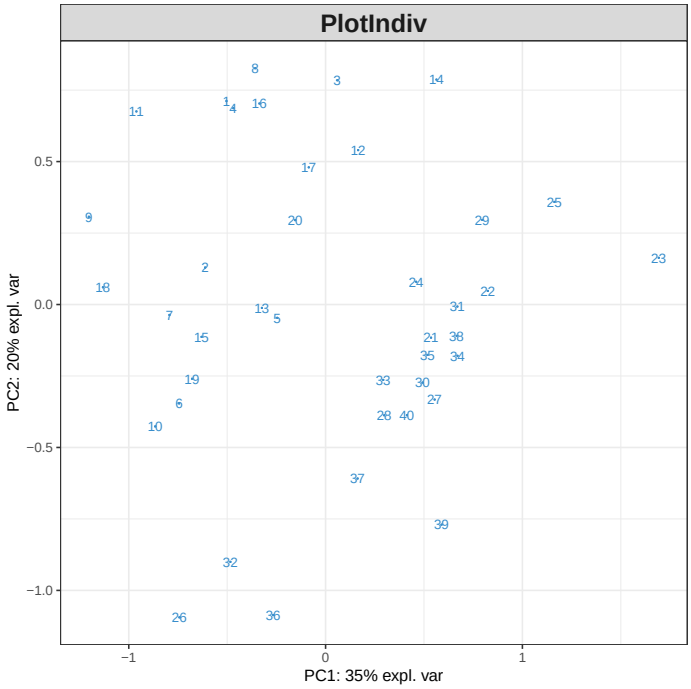
Then use your critical thinking and additional functions and visual tools to make sense of your data! (some of which are listed in 1.2.2) and will be described in the next Chapters.

For instance, for Principal Components Analysis, we first load the data:

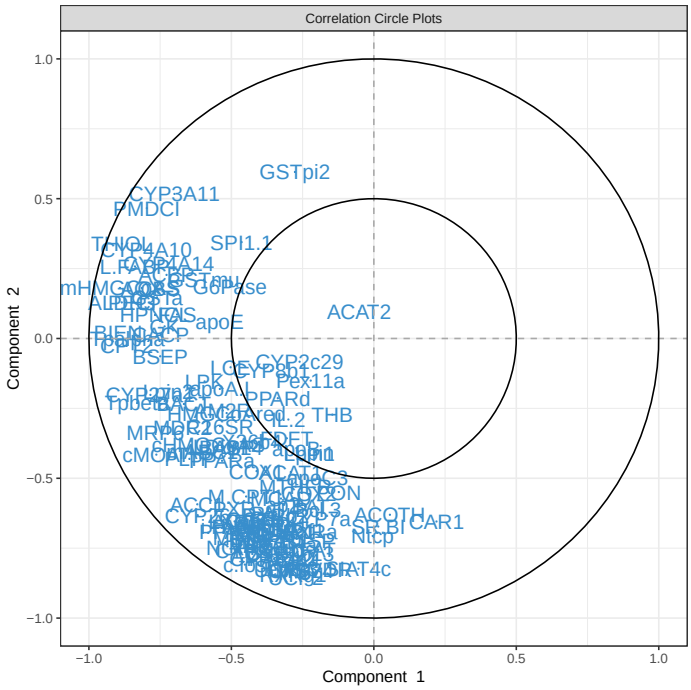
```
data(nutrimouse)
X <- nutrimouse$gene
```

Then use the following steps:

```
MyResult.pca <- pca(X)  # 1 Run the method
plotIndiv(MyResult.pca) # 2 Plot the samples
```



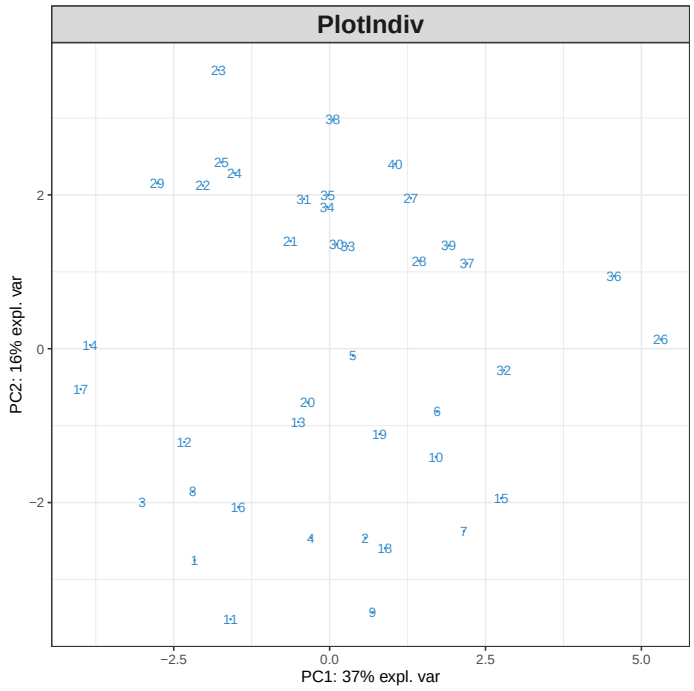

```
plotVar(MyResult.pca)    # 3 Plot the variables
```



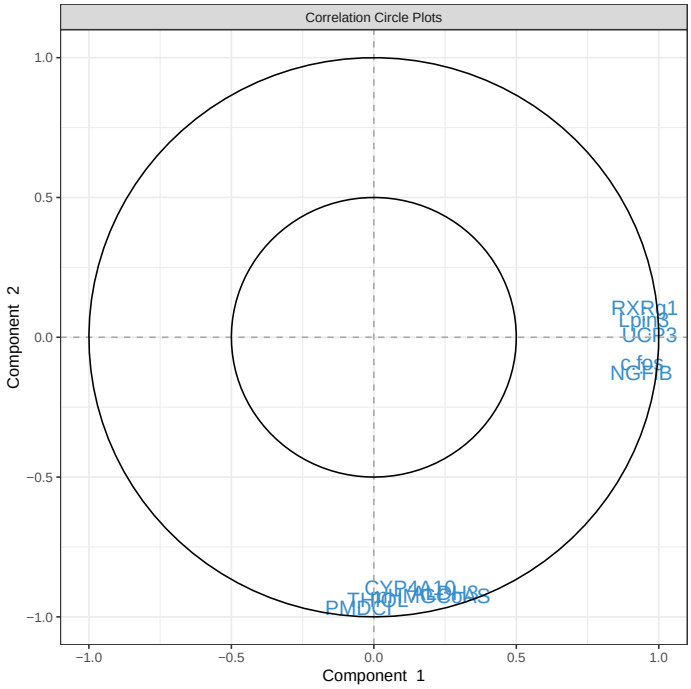
This is only a first quick-start, there will be many avenues you can take to deepen your exploratory and integrative analyses. The package proposes several methods to perform variable, or feature selection to identify the relevant information from rather large omics data sets. The sparse methods are listed in the Table in 1.2.2.

Following our example here, sparse PCA can be applied to select the top 5 variables contributing to each of the two components in PCA. The user specifies the number of variables to selected on each component, for example, here 5 variables are selected on each of the first two components (`keepX=c(5,5)`):

```
MyResult.spca <- spca(X, keepX=c(5,5)) # 1 Run the method
plotIndiv(MyResult.spca)               # 2 Plot the samples
```




```
plotVar(MyResult.spca)                 # 3 Plot the variables
```



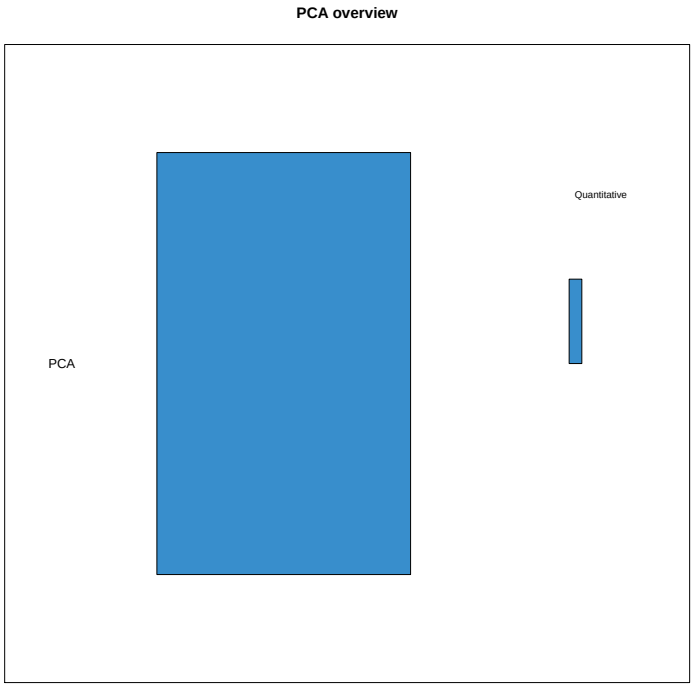
You can see know that we have considerably reduced the number of genes in the `plotVar` correlation circle plot.

Do not stop here! We are not done yet. You can enhance your analyses with the following:

- Have a look at our manual and each of the functions and their examples, e.g. `?pca`, `?plotIndiv`, `?sPCA`, ...
- Run the examples from the help file using the `example` function: `example(pca)`, `example(plotIndiv)`, ...
- Have a look at our website that features many tutorials and case studies,
- Keep reading this vignette, this is *just the beginning!*

Chapter 3

Principal Component Analysis (PCA)



3.1 Biological question

I would like to identify the major sources of variation in my data and identify

whether such sources of variation correspond to biological conditions or experimental bias. I would like to visualise trends or patterns between samples, whether they 'naturally' cluster according to known biological conditions.

3.2 The `liver.toxicity` study

The `liver.toxicity` is a list in the package that contains:

- `gene`: a data frame with 64 rows and 3116 columns, corresponding to the expression levels of 3,116 genes measured on 64 rats.
- `clinic`: a data frame with 64 rows and 10 columns, corresponding to the measurements of 10 clinical variables on the same 64 rats.
- `treatment`: data frame with 64 rows and 4 columns, indicating the treatment information of the 64 rats, such as doses of acetaminophen and times of necropsy.
- `gene.ID`: a data frame with 3116 rows and 2 columns, indicating geneBank IDs of the annotated genes.

More details are available at `?liver.toxicity`.

To illustrate PCA, we focus on the expression levels of the genes in the data frame `liver.toxicity$gene`. Some of the terms mentioned below are listed in 1.2.1.

3.3 Principle of PCA

The aim of PCA (Jolliffe, 2005) is to reduce the dimensionality of the data whilst retaining as much information as possible. 'Information' is referred here as *variance*. The idea is to create uncorrelated artificial variables called *principal components* (PCs) that combine in a linear manner the original (possibly correlated) variables (e.g. genes, metabolites, etc.).

Dimension reduction is achieved by projecting the data into space spanned by the principal components (PC). In practice, it means that each sample is assigned a score on each new PC dimension - this score is calculated as a linear combination of the original variables to which a weight is applied. The weights of each of the original variables are stored in the so-called *loading vectors* associated to each PC. The dimension of the data is reduced by projecting the data into the smaller subspace spanned by the PCs, while capturing the largest sources of variation between samples.

The principal components are obtained so that their variance is maximised. To that end, we calculate the eigenvectors/eigenvalues of the variance-covariance matrix, often via singular value decomposition when the number of variables is very large. The data are usually centred (`center = TRUE`), and sometimes

scaled (`scale = TRUE`) in the method. The latter is especially advised in the case where the variance is not homogeneous across variables.

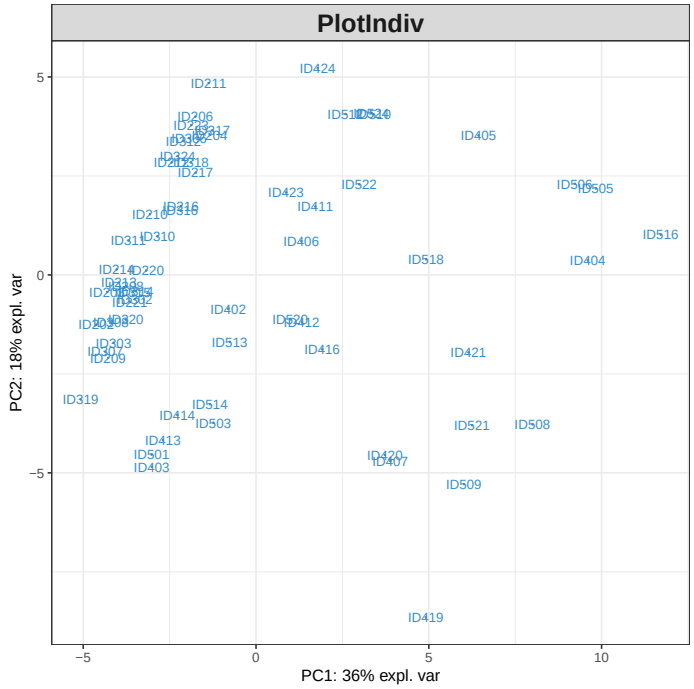
The first PC is defined as the linear combination of the original variables that explains the greatest amount of variation. The second PC is then defined as the linear combination of the original variables that accounts for the greatest amount of the remaining variation subject of being orthogonal (uncorrelated) to the first component. Subsequent components are defined likewise for the other PCA dimensions. The user must, therefore, report how much information is explained by the first PCs as these are used to graphically represent the PCA outputs.

3.4 Load the data

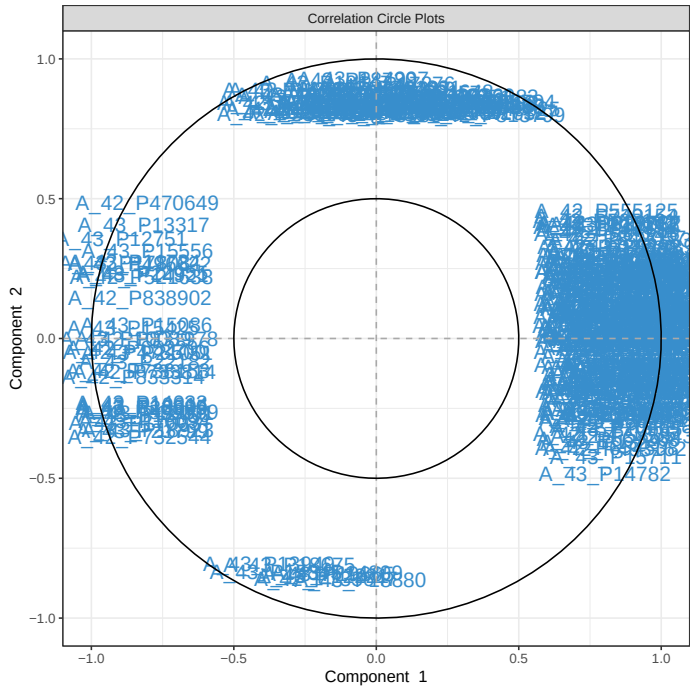
We first load the data from the package. See 2.2 to upload your own data.

```
library(mixOmics)
data(liver.toxicity)
X <- liver.toxicity$gene
```

```
MyResult.pca <- pca(X)      # 1 Run the method
plotIndiv(MyResult.pca)     # 2 Plot the samples
```




```
plotVar(MyResult.pca, cutoff = 0.8)       # 3 Plot the variables
```



If you were to run `pca` with this minimal code, you would be using the following default values:

- `ncomp =2`: the first two principal components are calculated and are used for graphical outputs;
- `center = TRUE`: data are centred (mean = 0)
- `scale = FALSE`: data are not scaled. If `scale = TRUE` standardizes each variable (variance = 1).

Other arguments can also be chosen, see `?pca`.

This example was shown in Chapter 2.4. The two plots are not extremely meaningful as specific sample patterns should be further investigated and the variable correlation circle plot contains too many variables to be easily interpreted. Let's improve those graphics as shown below to improve interpretation.

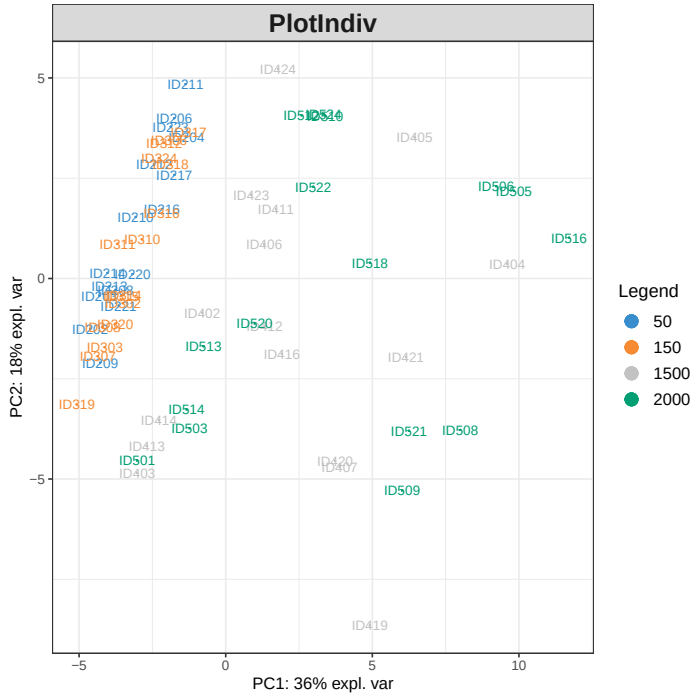
3.6 To go further

3.6.1 Customize plots

Plots can be customized using numerous options in `plotIndiv` and `plotVar`. For instance, even if PCA does not take into account any information regarding the known group membership of each sample, we can include such information on the sample plot to visualize any 'natural' cluster that may correspond to biological conditions.

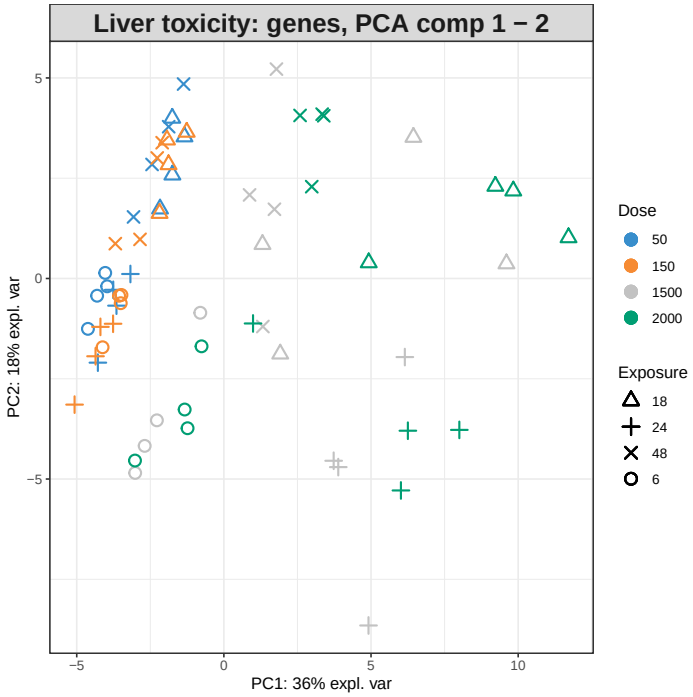
Here is an example where we include the sample groups information with the argument `group`:

```
plotIndiv(MyResult.pca, group = liver.toxicity$treatment$Dose.Group,
          legend = TRUE)
```



Additionally, two factors can be displayed using both colours (argument `group`) and symbols (argument `pch`). For example here we display both Dose and Time of exposure and improve the title and legend:

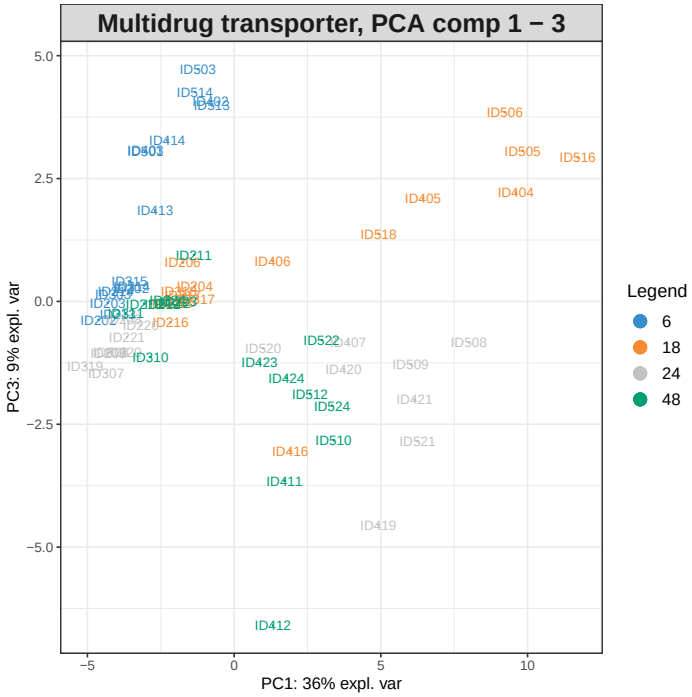
```
plotIndiv(MyResult.pca, ind.names = FALSE,
          group = liver.toxicity$treatment$Dose.Group,
          pch = as.factor(liver.toxicity$treatment$Time.Group),
          legend = TRUE, title = 'Liver toxicity: genes, PCA comp 1 - 2',
          legend.title = 'Dose', legend.title.pch = 'Exposure')
```



By including information related to the dose of acetaminophen and time of exposure enables us to see a cluster of low dose samples (blue and orange, top left at 50 and 100mg respectively), whereas samples with high doses (1500 and 2000mg in grey and green respectively) are more scattered, but highlight an exposure effect.

To display the results on other components, we can change the `comp` argument provided we have requested enough components to be calculated. Here is our second PCA with 3 components:

```
MyResult.pca2 <- pca(X, ncomp = 3)
plotIndiv(MyResult.pca2, comp = c(1,3), legend = TRUE,
          group = liver.toxicity$treatment$Time.Group,
          title = 'Multidrug transporter, PCA comp 1 - 3')
```

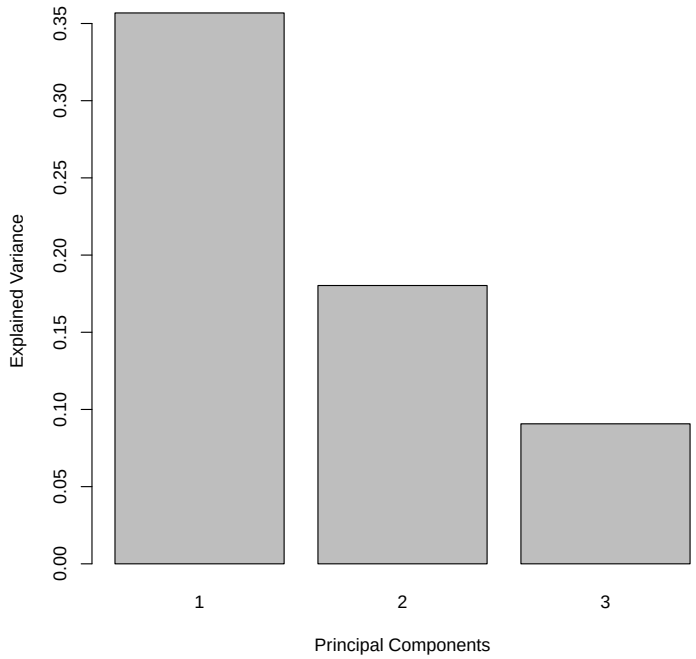


Here, the 3rd component on the y-axis clearly highlights a time of exposure effect.

3.6.2 Amount of variance explained and choice of a number of components

The amount of variance explained can be extracted with the following: a screeplot or the actual numerical proportions of explained variance, and cumulative proportion.

```
plot(MyResult.pca2)
```




```
MyResult.pca2
```

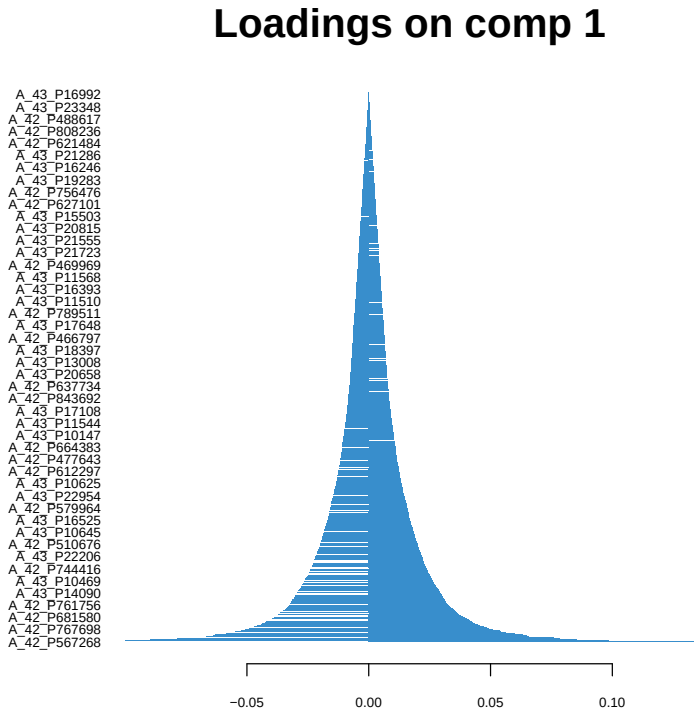
```
## Eigenvalues for the first 3 principal components, see object$sdev^2:
##       PC1       PC2       PC3
## 17.971416  9.079234  4.567709
##
## Proportion of explained variance for the first 3 principal components, see object$explained_va
##        PC1        PC2        PC3
## 0.35684128 0.18027769 0.09069665
##
## Cumulative proportion explained variance for the first 3 principal components, see object$cum.
##       PC1       PC2       PC3
## 0.3568413 0.5371190 0.6278156
##
##  Other available components:
##  --------------------
##  loading vectors: see object$rotation
```

There are no clear guidelines on how many components should be included in PCA: it is data-dependent and level of noise dependent. We often look at the 'elbow' on the screeplot above as an indicator that the addition of PCs does not drastically contribute to explain the remaining variance.

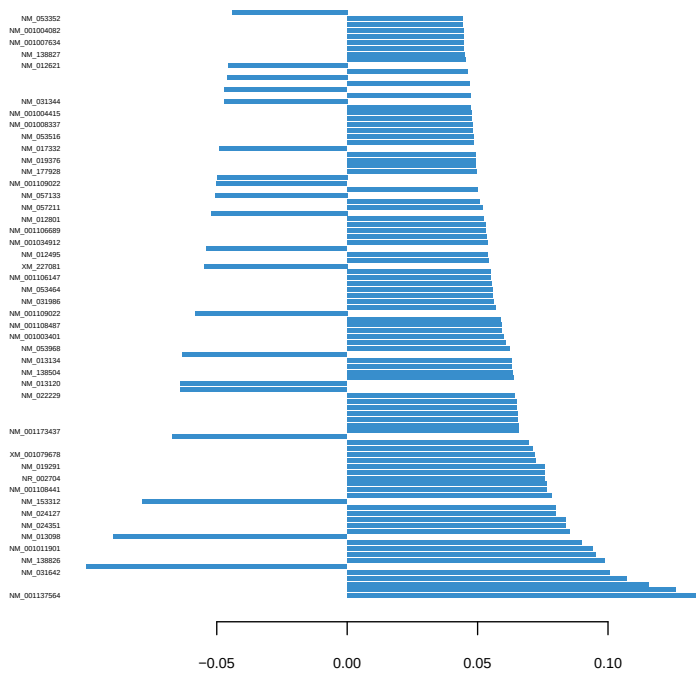
3.6.3 Other useful plots

We can also have a look at the variable coefficients in each component with the loading vectors. The loading weights are represented in decreasing order from bottom to top in `plotLoadings`. Their absolute value indicates the importance of each variable to define each PC, as represented by the length of each bar. See `?plotLoadings` to change the arguments.

```
# a minimal example
plotLoadings(MyResult.pca)
```




```
# a customized example to only show the top 100 genes
# and their gene name
plotLoadings(MyResult.pca, ndisplay = 100,
             name.var = liver.toxicity$gene.ID[, "geneBank"],
             size.name = rel(0.3))
```



Such representation will be more informative once we select a few variables in the next section 3.7.

Plots can also be interactively displayed in 3 dimensions using the option `style="3d"`. We use the `rgl` package for this (the interative figure is only interatice in the `html` vignette).

```
plotIndiv(MyResult.pca2,
          group = liver.toxicity$treatment$Dose.Group, style="3d",
          legend = TRUE, title = 'Liver toxicity: genes, PCA comp 1 - 2 - 3')
```

You must enable Javascript to view this page properly.

3.7 Variable selection with sparse PCA

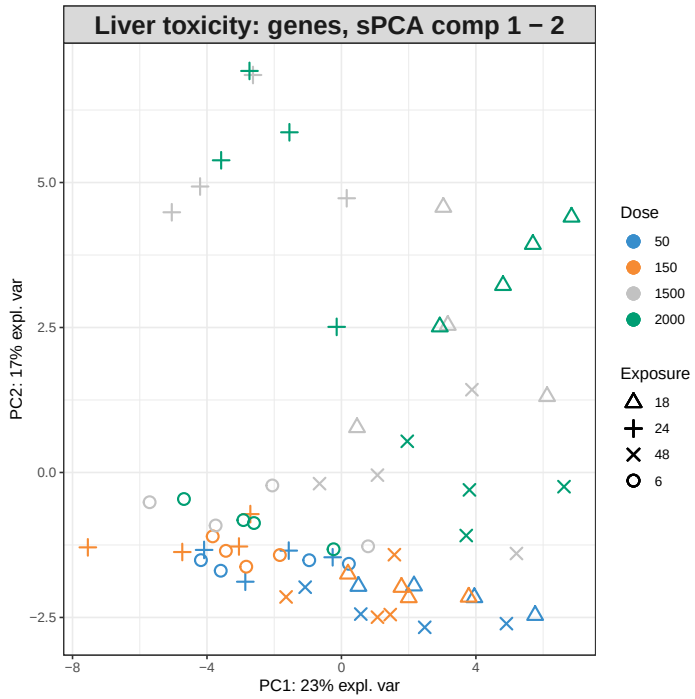
3.7.1 Biological question

I would like to apply PCA but also be able to identify the key variables that contribute to the explanation of most variance in the data set.

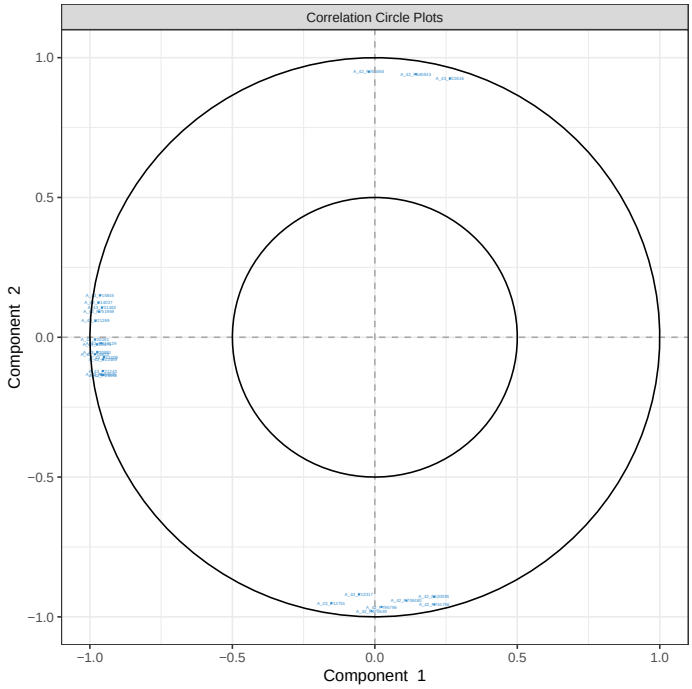
Variable selection can be performed using the sparse version of PCA implemented in `spca` (Shen and Huang, 2008). The user needs to provide the number of variables to select on each PC. Here, for example, we ask to select the top

15 genes contributing to the definition of PC1, the top 10 genes contributing to PC2 and the top 5 genes for PC3 (`keepX=c(15,10,5)`).

```
MyResult.spca <- spca(X, ncomp = 3, keepX = c(15,10,5))                  # 1 Run the me
plotIndiv(MyResult.spca, group = liver.toxicity$treatment$Dose.Group,    # 2 Plot the s
          pch = as.factor(liver.toxicity$treatment$Time.Group),
          legend = TRUE, title = 'Liver toxicity: genes, sPCA comp 1 - 2',
          legend.title = 'Dose', legend.title.pch = 'Exposure')
```




```
plotVar(MyResult.spca, cex = 1)                                          # 3 Plot the va
```




```
# cex is used to reduce the size of the labels on the plot
```

Selected variables can be identified on each component with the `selectVar` function. Here the coefficient values are extracted, but there are other outputs as well, see `?selectVar`:

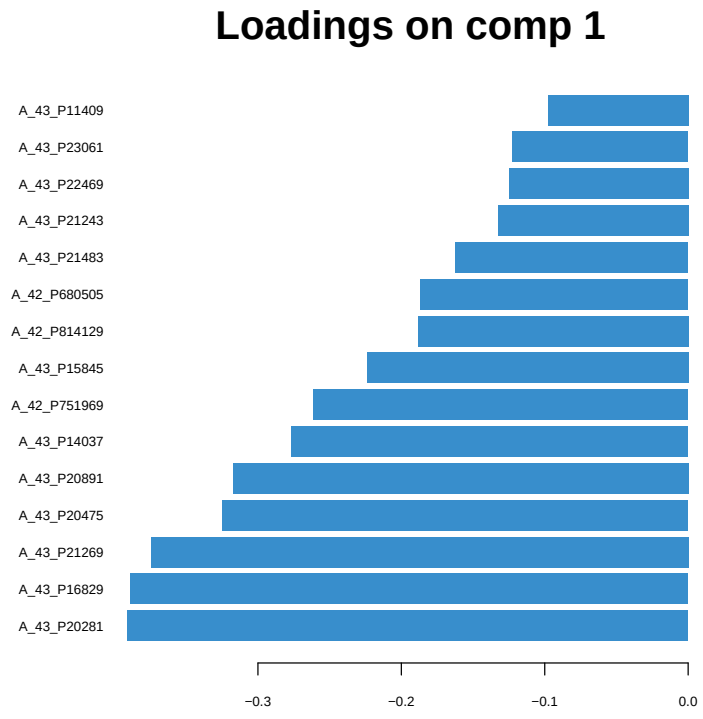
```
selectVar(MyResult.spca, comp = 1)$value
```

```
##                value.var
## A_43_P20281  -0.39077443
## A_43_P16829  -0.38898291
## A_43_P21269  -0.37452039
## A_43_P20475  -0.32482960
## A_43_P20891  -0.31740002
## A_43_P14037  -0.27681845
## A_42_P751969 -0.26140533
## A_43_P15845  -0.22392912
## A_42_P814129 -0.18838954
## A_42_P680505 -0.18672610
## A_43_P21483  -0.16202222
## A_43_P21243  -0.13259471
## A_43_P22469  -0.12493156
## A_43_P23061  -0.12255308
## A_43_P11409  -0.09768656
```

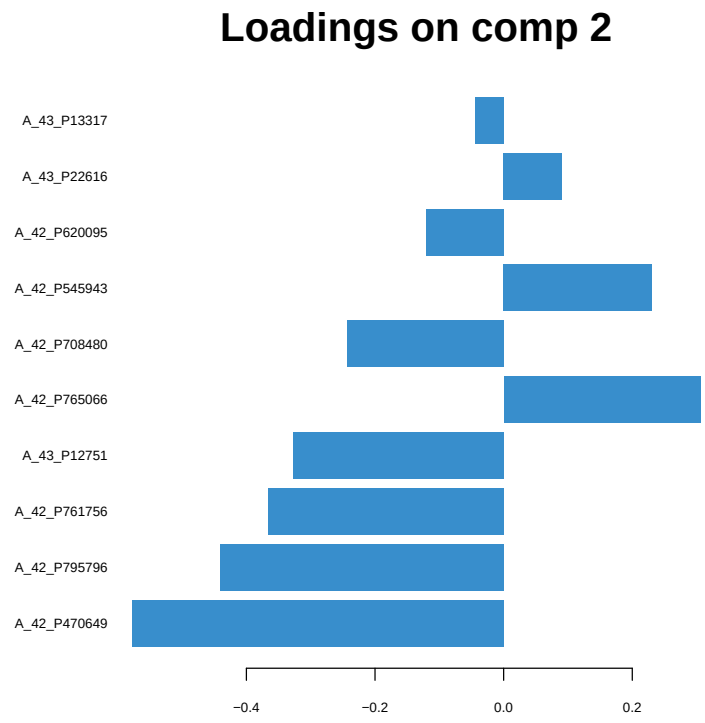
Those values correspond to the loading weights that are used to define each component. A large absolute value indicates the importance of the variable in this PC. Selected variables are ranked from the most important (top) to the least important.

We can complement this output with `plotLoadings`. We can see here that all coefficients are negative.

```
plotLoadings(MyResult.spca, comp=1)
```



If we look at the 2nd component, we can see a mix of positive and negative weights (also see in the `plotVar`), those correspond to variables that oppose the low and high doses (see from the `plotIndiv`):



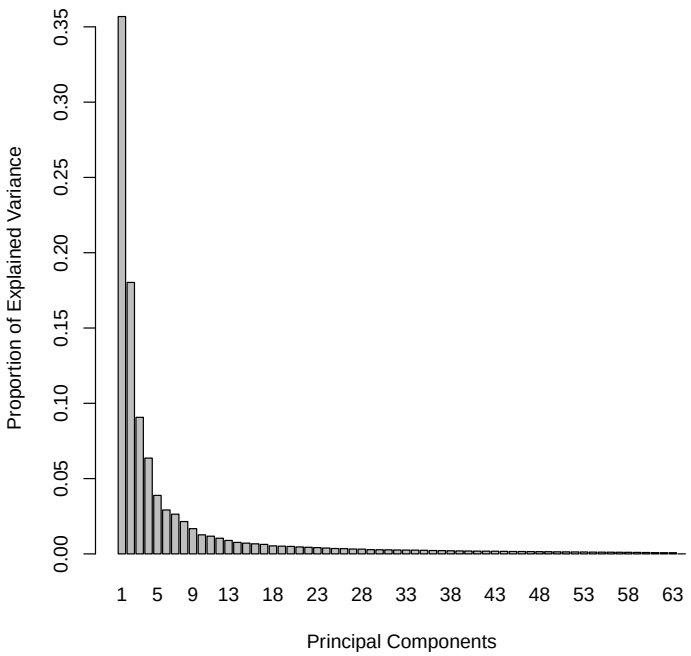
3.8 Tuning parameters

For this set of methods, two parameters need to be chosen:

- The number of components to retain,
- The number of variables to select on each component for sparse PCA.

The function `tune.pca` calculates the percentage of variance explained for each component, up to the minimum between the number of rows, or column in the data set. The 'optimal' number of components can be identified if an elbow appears on the screeplot. In the example below the cut-off is not very clear, we could choose 2 components.

```
tune.pca(X)
```



Regarding the number of variables to select in sparse PCA, there is not a clear criterion at this stage. As PCA is an exploration method, we prefer to set arbitrary thresholds that will pinpoint the key variables to focus on during the interpretation stage.

3.9 Additional resources

Additional examples are provided in `example(pca)` and in our case studies on our website in the **Methods** and **Case studies** sections.

Additional reading in (Shen and Huang, 2008).

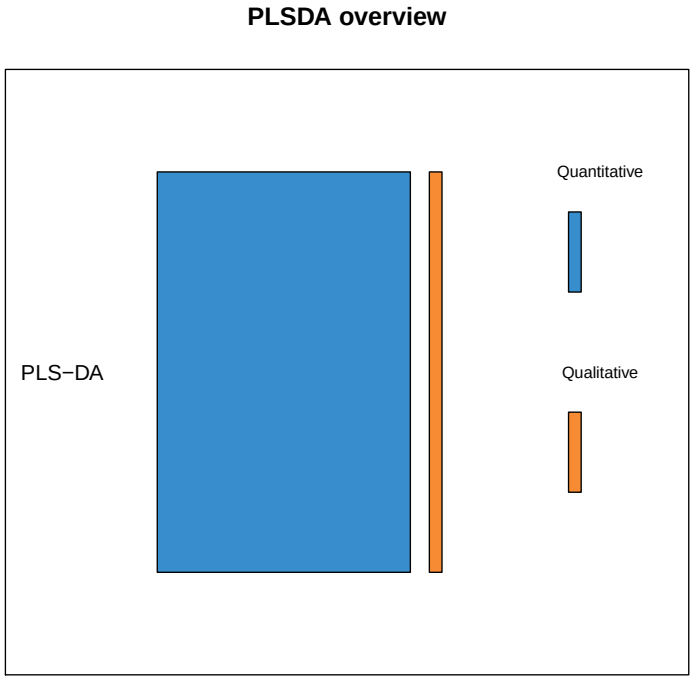
3.10 FAQ

- Should I scale my data before running PCA? (`scale = TRUE` in `pca`)
  - Without scaling: a variable with high variance will solely drive the first principal component
  - With scaling: one noisy variable with low variability will be assigned the same variance as other meaningful variables
- Can I perform PCA with missing values?
  - NIPALS (Non-linear Iterative PArtial Least Squares, implemented in mixOmics) can impute missing values but must be built on many components. The proportion of NAs should not exceed 20% of the total data.

- When should I apply a multilevel approach in PCA? (`multilevel` argument in `PCA`)
  - When the unique individuals are measured more than once (repeated measures)
  - When the individual variation is less than treatment or time variation. This means that samples from each unique individual will tend to cluster rather than the treatments.
  - When a multilevel vs no multilevel seems to visually make a difference on a PCA plot
  - More details in this case study
- When should I apply a CLR transformation in PCA? (`logratio = 'CLR'` argument in `PCA`)
  - When data are compositional, i.e. expressed as relative proportions. This is usually the case with microbiome studies as a result of preprocessing and normalisation, see more details here and in our case studies in the same tab.

Chapter 4

PLS - Discriminant Analysis (PLS-DA)



4.1 Biological question

I am analysing a single data set (e.g. transcriptomics data) and I would like to classify my samples into known groups and predict the class of new samples.

In addition, I am interested in identifying the key variables that drive such discrimination.

4.2 The `srbct` study

The data are directly available in a processed and normalised format from the package. The Small Round Blue Cell Tumours (SRBCT) dataset from (Khan et al., 2001) includes the expression levels of 2,308 genes measured on 63 samples. The samples are classified into four classes as follows: 8 Burkitt Lymphoma (BL), 23 Ewing Sarcoma (EWS), 12 neuroblastoma (NB), and 20 rhabdomyosarcoma (RMS).

The srbct dataset contains the following:

$gene: a data frame with 63 rows and 2308 columns. The expression levels of 2,308 genes in 63 subjects.

$class: a class vector containing the class tumour of each individual (4 classes in total).

$gene.name: a data frame with 2,308 rows and 2 columns containing further information on the genes.

More details can be found in `?srbct`.

To illustrate PLS-DA, we will analyse the gene expression levels of `srbct$gene` to discriminate the 4 groups of tumours.

4.3 Principle of sparse PLS-DA

Although Partial Least Squares was not originally designed for classification and discrimination problems, it has often been used for that purpose (Nguyen and Rocke, 2002; Tan et al., 2004). The response matrix `Y` is qualitative and is internally recoded as a dummy block matrix that records the membership of each observation, i.e. each of the response categories are coded via an indicator variable (see (Rohart et al., 2017a) Suppl. Information S1 for an illustration). The PLS regression (now PLS-DA) is then run as if Y was a continuous matrix. This PLS classification trick works well in practice, as demonstrated in many references (Barker and Rayens, 2003; Nguyen and Rocke, 2002; Boulesteix and Strimmer, 2007; Chung and Keles, 2010).

Sparse PLS-DA (Lê Cao et al., 2011) performs variable selection and classification in a one-step procedure. sPLS-DA is a special case of sparse PLS described later in 5, where ℓ_1 penalization is applied on the loading vectors associated to the X data set.

4.4 Inputs and outputs

We use the following data input matrices: `X` is a $n \times p$ data matrix, `Y` is a factor vector of length n that indicates the class of each sample, and Y^* is the associated dummy matrix $(n \times K)$ with n the number of samples (individuals), p the number of variables and K the number of classes. PLS-DA main outputs are:

- A **set of components**, also called latent variables. There are as many components as the chosen *dimension* of the PLS-DA model.
- A **set of loading vectors**, which are coefficients assigned to each variable to define each component. These coefficients indicate the importance of each variable in PLS-DA. Importantly, each loading vector is associated to a particular component. Loading vectors are obtained so that the covariance between a linear combination of the variables from X (the X-component) and the factor of interest Y (the Y^*-component) is maximised.
- A **list of selected variables** from `X` and associated to each component if sPLS-DA is applied.

4.5 Set up the data

We first load the data from the package. See 2.2 to upload your own data.

We will mainly focus on sparse PLS-DA that is more suited for large biological data sets where the aim is to identify molecular signatures, as well as classifying samples. We first set up the data as `X` expression matrix and `Y` as a factor indicating the class membership of each sample. We also check that the dimensions are correct and match:

```
library(mixOmics)
data(srbct)
X <- srbct$gene
Y <- srbct$class
summary(Y) ## class summary
```

```
## EWS  BL  NB RMS
##  23   8  12  20
```

```
dim(X) ## number of samples and features
```

```
## [1]   63 2308
```

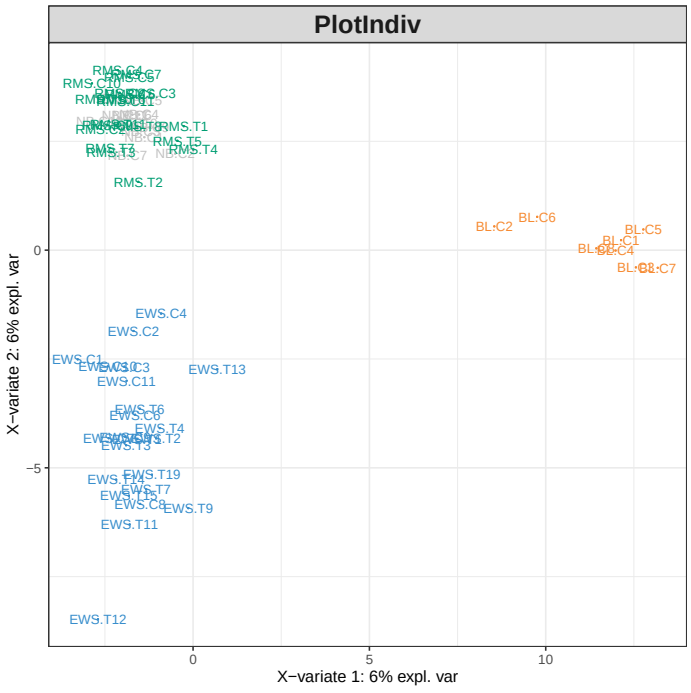
```
length(Y) ## length of class memebrship factor = number of samples
```

```
## [1] 63
```

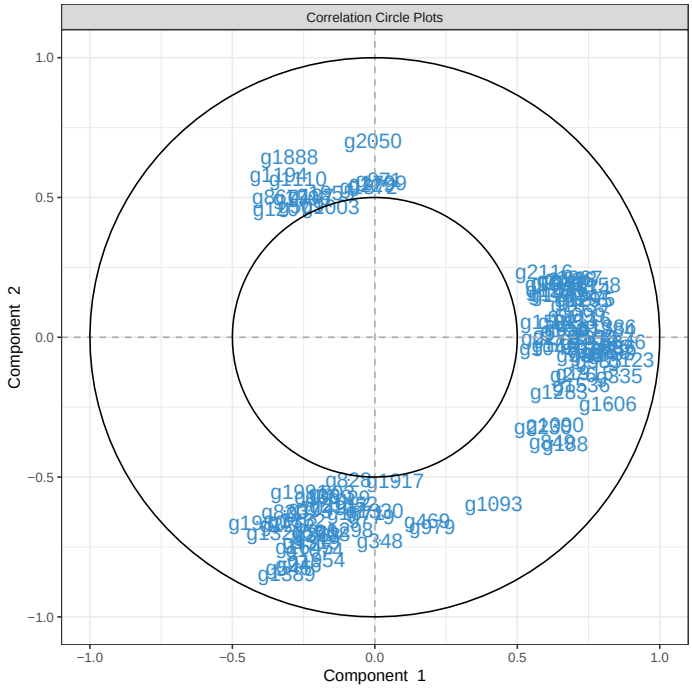
4.6 Quick start

For a quick start, we arbitrarily set the number of variables to select to 50 on each of the 3 components of PLS-DA (see section 4.7.5 for tuning these values).

```
MyResult.splsda <- splsda(X, Y, keepX = c(50,50)) # 1 Run the method
plotIndiv(MyResult.splsda)                        # 2 Plot the samples (coloured by
```




```
plotVar(MyResult.splsda)                          # 3 Plot the variables
```




```
selectVar(MyResult.splsda, comp=1)$name            # Selected variables on component 1
```

As PLS-DA is a supervised method, the sample plot automatically displays the group membership of each sample. We can observe clear discrimination between the BL samples and the others on the first component (x-axis), and EWS vs the others on the second component (y-axis). Remember that this discrimination spanned by the first two PLS-DA components is obtained based on a subset of 100 variables (50 selected on each component).

From the `plotIndiv` the axis labels indicate the amount of variation explained per component. Note that the interpretation of this amount is *not* the same as in PCA. In PLS-DA, the aim is to maximise the covariance between `X` and `Y`, not only the variance of `X` as it is the case in PCA!

If you were to run `splsda` with this minimal code, you would be using the following default values:

- `ncomp = 2`: the first two PLS components are calculated and are used for graphical outputs;
- `scale = TRUE`: data are scaled (variance = 1, strongly advised here);
- `mode = "regression"`: by default, a PLS regression mode should be used.

PLS-DA without variable selection can be performed as:

```
MyResult.plsda <- plsda(X,Y) # 1 Run the method
plotIndiv(MyResult.plsda)    # 2 Plot the samples
```

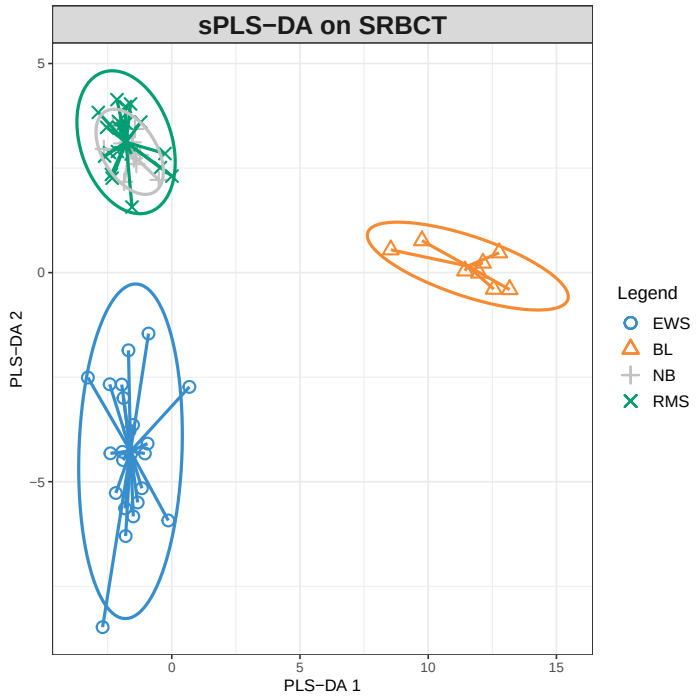
```
plotVar(MyResult.plsda, cutoff = 0.7)      # 3 Plot the variables
```

4.7 To go further

4.7.1 Customize the sample plots

The sample plots can be improved in various ways. First, if the names of the samples are not meaningful at this stage, they can be replaced by symbols (`ind.names=TRUE`). Confidence ellipses can be plotted for each sample (`ellipse = TRUE`, confidence level set to 95% by default, see the argument `ellipse.level`), Additionally, a star plot displays arrows from each group centroid towards each individual sample (`star = TRUE`). A 3D plot is also available, see `plotIndiv` for more details.

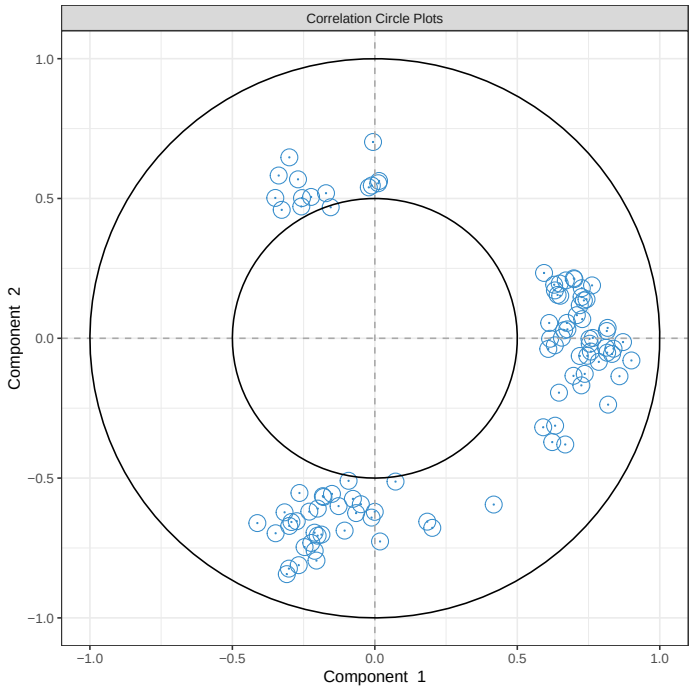
```
plotIndiv(MyResult.splsda, ind.names = FALSE, legend=TRUE,
          ellipse = TRUE, star = TRUE, title = 'sPLS-DA on SRBCT',
          X.label = 'PLS-DA 1', Y.label = 'PLS-DA 2')
```



4.7.2 Customize variable plots

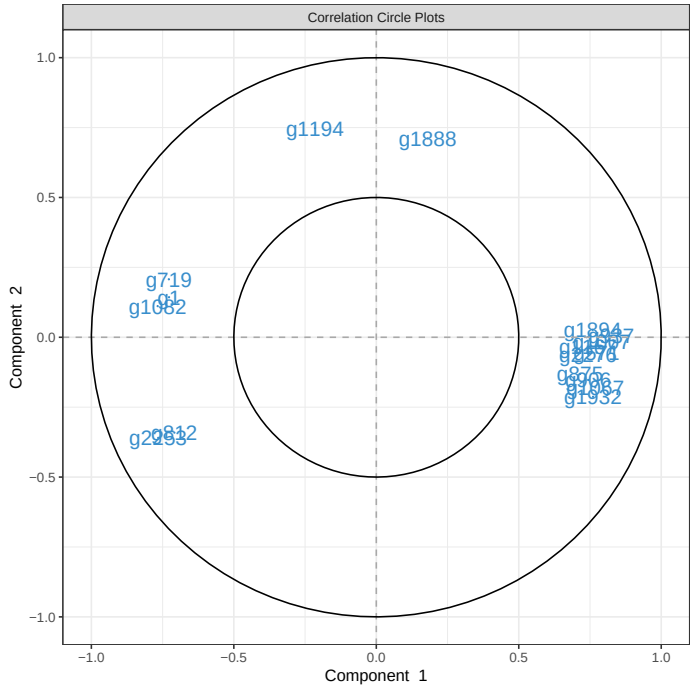
The name of the variables can be set to FALSE (`var.names=FALSE`):

```
plotVar(MyResult.splsda, var.names=FALSE)
```



In addition, if we had used the non-sparse version of PLS-DA, a cut-off can be set to display only the variables that mostly contribute to the definition of each component. These variables should be located towards the circle of radius 1, far from the centre.

```
plotVar(MyResult.plsda, cutoff=0.7)
```



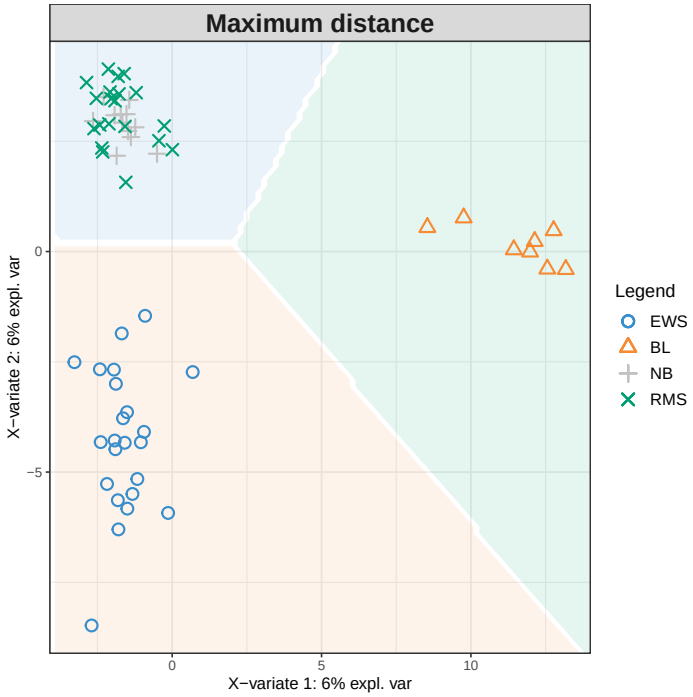
In this particular case, no variable selection was performed. Only the display was altered to show a subset of variables.

4.7.3 Other useful plots

4.7.3.1 Background prediction

A 'prediction' background can be added to the sample plot by calculating a background surface first, before overlaying the sample plot. See `?background.predict` for more details. More details about prediction, prediction distances can be found in (Rohart et al., 2017a) in the Suppl. Information.

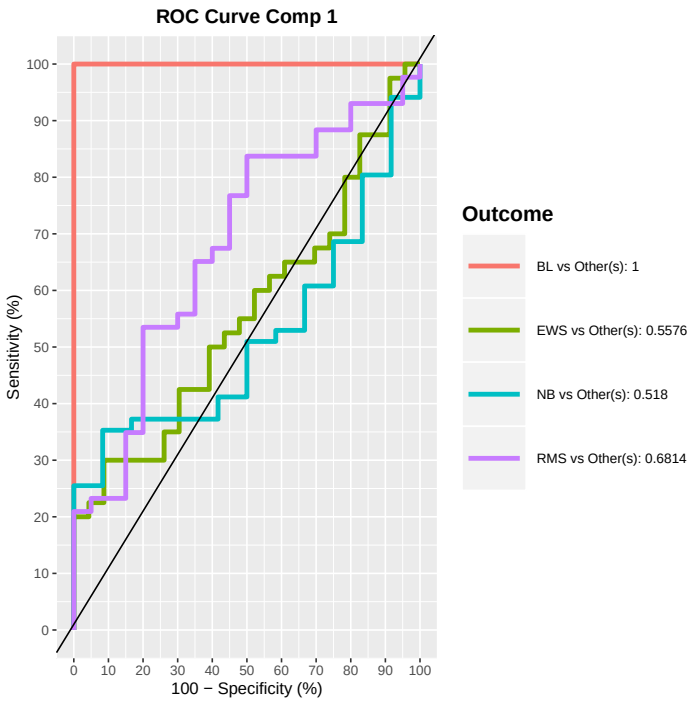
```
background <- background.predict(MyResult.splsda, comp.predicted=2,
                                 dist = "max.dist")
plotIndiv(MyResult.splsda, comp = 1:2, group = srbct$class,
          ind.names = FALSE, title = "Maximum distance",
          legend = TRUE,  background = background)
```



4.7.3.2 ROC

As PLS-DA acts as a classifier, we can plot a ROC Curve to complement the sPLS-DA classification performance results detailed in 4.7.5. The AUC is calculated from training cross-validation sets and averaged. Note however that ROC and AUC criteria may not be particularly insightful, or may not be in full agreement with the PLSDA performance, as the prediction threshold in PLS-DA is based on specified distance as described in (Rohart et al., 2017a).

```
auc.plsda <- auroc(MyResult.splsda)
```



4.7.4 Variable selection outputs

First, note that the number of variables to select on each component does not need to be identical on each component, for example:

```
MyResult.splsda2 <- splsda(X,Y, ncomp=3, keepX=c(15,10,5))
```

Selected variables are listed in the `selectVar` function:

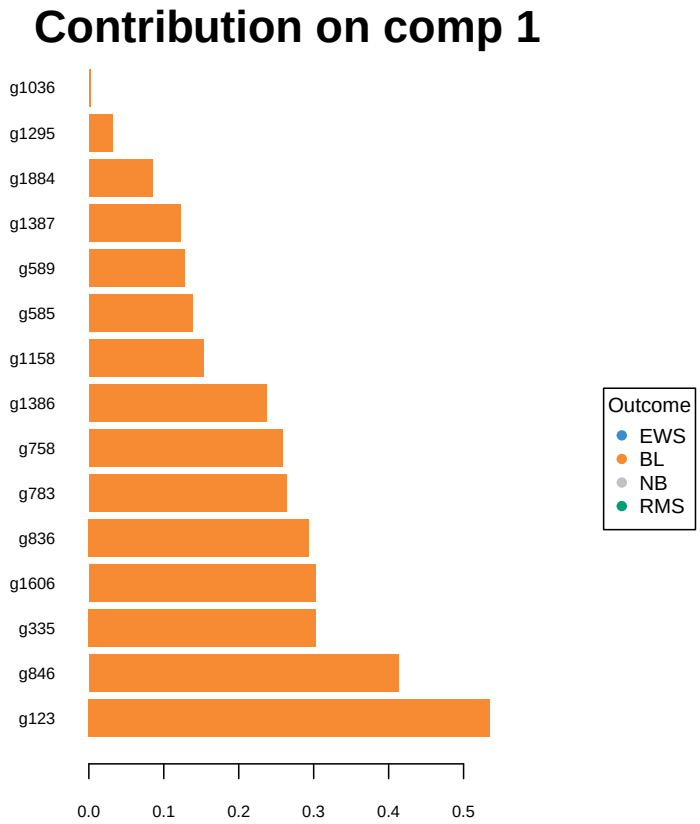
```
selectVar(MyResult.splsda2, comp=1)$value
```

```
##        value.var
## g123  0.53516982
## g846  0.41271455
## g335  0.30309695
## g1606 0.30194141
## g836  0.29365241
## g783  0.26329876
## g758  0.25826903
## g1386 0.23702577
## g1158 0.15283961
## g585  0.13838913
## g589  0.12738682
## g1387 0.12202390
```

```
## g1884 0.08458869
## g1295 0.03150351
## g1036 0.00224886
```

and can be visualised in `plotLoadings` with the arguments `contrib = 'max'` that is going to assign to each variable bar the sample group colour for which the mean (`method = 'mean'`) is maximum. See `example(plotLoadings)` for other options (e.g. min, median)

```
plotLoadings(MyResult.splsda2, contrib = 'max', method = 'mean')
```



Interestingly from this plot, we can see that all selected variables on component 1 are highly expressed in the BL (orange) class. Setting `contrib = 'min'` would highlight that those variables are lowly expressed in the NB grey class, which makes sense when we look at the sample plot.

Since 4 classes are being discriminated here, samples plots in 3d may help interpretation (available in the `html` vignette only):

```
plotIndiv(MyResult.splsda2, style="3d")
```

You must enable Javascript to view this page properly.

4.7.5 Tuning parameters and numerical outputs

For this set of methods, three parameters need to be chosen:

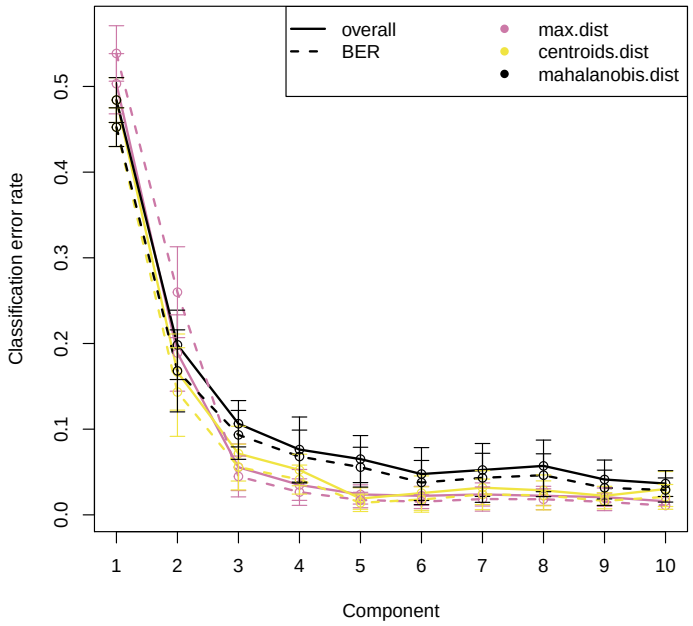
1 - The number of components to retain `ncomp`. The rule of thumb is usually $K - 1$ where K is the number of classes, but it is worth testing a few extra components.

2 - The number of variables `keepX` to select on each component for sparse PLS-DA,

3 - The prediction distance to evaluate the classification and prediction performance of PLS-DA.

For **item 1**, the `perf` evaluates the performance of PLS-DA for a large number of components, using repeated k-fold cross-validation. For example here we use 3-fold CV repeated 10 times (note that we advise to use at least 50 repeats, and choose the number of folds that are appropriate for the sample size of the data set):

```
MyResult.plsda2 <- plsda(X,Y, ncomp=10)
set.seed(30) # for reproducbility in this vignette, otherwise increase nrepeat
MyPerf.plsda <- perf(MyResult.plsda2, validation = "Mfold", folds = 3,
                  progressBar = FALSE, nrepeat = 10) # we suggest nrepeat = 50

plot(MyPerf.plsda, col = color.mixo(5:7), sd = TRUE, legend.position = "horizontal")
```



The plot outputs the classification error rate, or *Balanced* classification error rate when the number of samples per group is unbalanced, the standard deviation

according to three prediction distances. Here we can see that for the BER and the maximum distance, the best performance (i.e. low error rate) seems to be achieved for `ncomp = 3`.

In addition (**item 3** for PLS-DA), the numerical outputs listed here can be reported as performance measures:

```
MyPerf.plsda
```

```
##
## Call:
##  perf.mixo_plsda(object = MyResult.plsda2, validation = "Mfold", folds = 3, nrepeat = 10, prog
##
##  Main numerical outputs:
##  --------------------
##  Error rate (overall or BER) for each component and for each distance: see object$error.rate
##  Error rate per class, for each component and for each distance: see object$error.rate.class
##  Prediction values for each component: see object$predict
##  Classification of each sample, for each component and for each distance: see object$class
##  AUC values: see object$auc if auc = TRUE
##
##  Visualisation Functions:
##  --------------------
##  plot
```

Regarding **item 2**, we now use `tune.splsda` to assess the optimal number of variables to select on each component. We first set up a grid of `keepX` values that will be assessed on each component, one component at a time. Similar to above we run 3-fold CV repeated 10 times with a maximum distance prediction defined as above.

```
list.keepX <- c(5:10,  seq(20, 100, 10))
list.keepX # to output the grid of values tested
```

```
##  [1]   5   6   7   8   9  10  20  30  40  50  60  70  80  90 100
```

```
set.seed(30) # for reproducbility in this vignette, otherwise increase nrepeat
tune.splsda.srbct <- tune.splsda(X, Y, ncomp = 3, # we suggest to push ncomp a bit more, e.g. 4
                                 validation = 'Mfold',
                                 folds = 3, dist = 'max.dist', progressBar = FALSE,
                                 measure = "BER", test.keepX = list.keepX,
                                 nrepeat = 10)   # we suggest nrepeat = 50
```

We can then extract the classification error rate averaged across all folds and repeats for each tested `keepX` value, the optimal number of components (see `?tune.splsda` for more details), the optimal number of variables to select per component which is summarised in a plot where the diamond indicated the optimal `keepX` value:

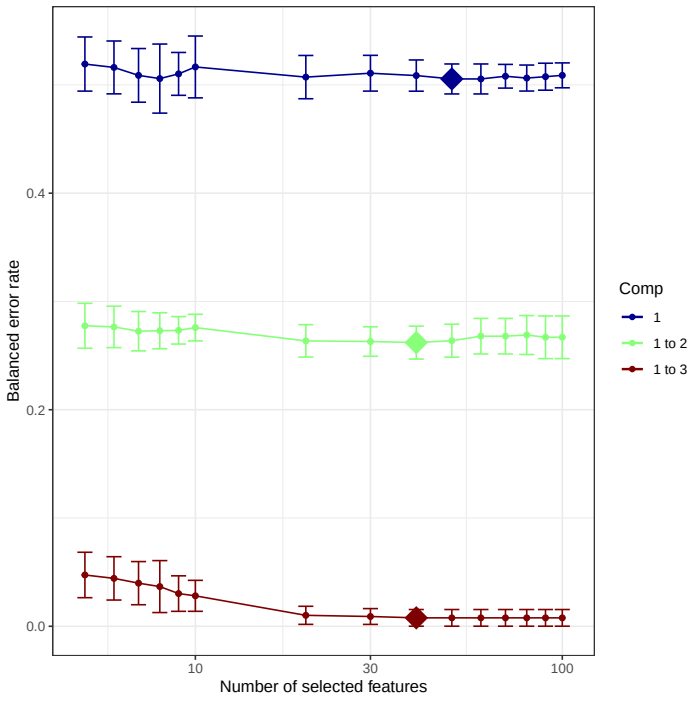
```
error <- tune.splsda.srbct$error.rate
ncomp <- tune.splsda.srbct$choice.ncomp$ncomp # optimal number of components based on
ncomp
```

```
## [1] 3
```

```
select.keepX <- tune.splsda.srbct$choice.keepX[1:ncomp]  # optimal number of variables
select.keepX
```

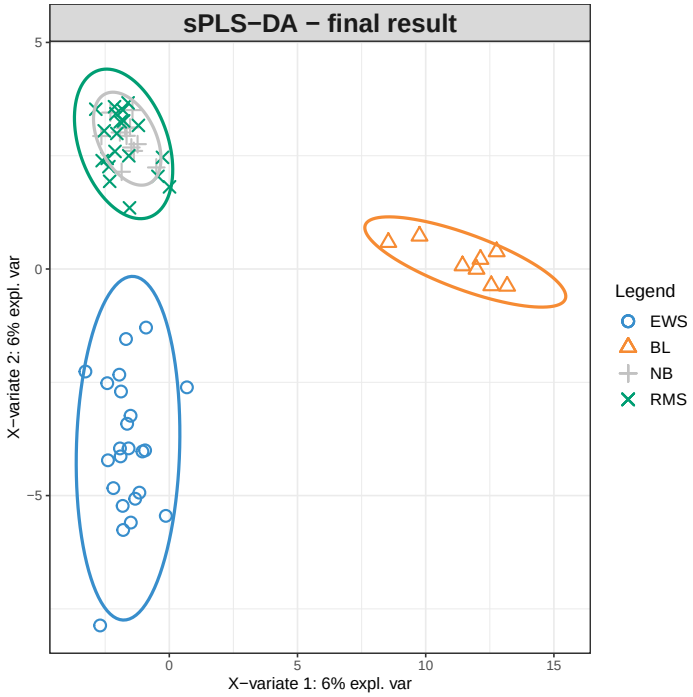
```
## comp1 comp2 comp3
##    50    40    40
```

```
plot(tune.splsda.srbct, col = color.jet(ncomp))
```



Based on those tuning results, we can run our final and tuned sPLS-DA model:

```
MyResult.splsda.final <- splsda(X, Y, ncomp = ncomp, keepX = select.keepX)

plotIndiv(MyResult.splsda.final, ind.names = FALSE, legend=TRUE,
          ellipse = TRUE, title="sPLS-DA - final result")
```



Additionally, we can run `perf` for the final performance of the sPLS-DA model. Also, note that `perf` will output `features` that lists the frequency of selection of the variables across the different folds and different repeats. This is a useful output to assess the confidence of your final variable selection, see a more detailed example here.

4.8 Additional resources

Additional examples are provided in `example(splsda)` and in our case studies on our website in the **Methods** and **Case studies** sections, and in particular here. Also have a look at (Lê Cao et al., 2011)

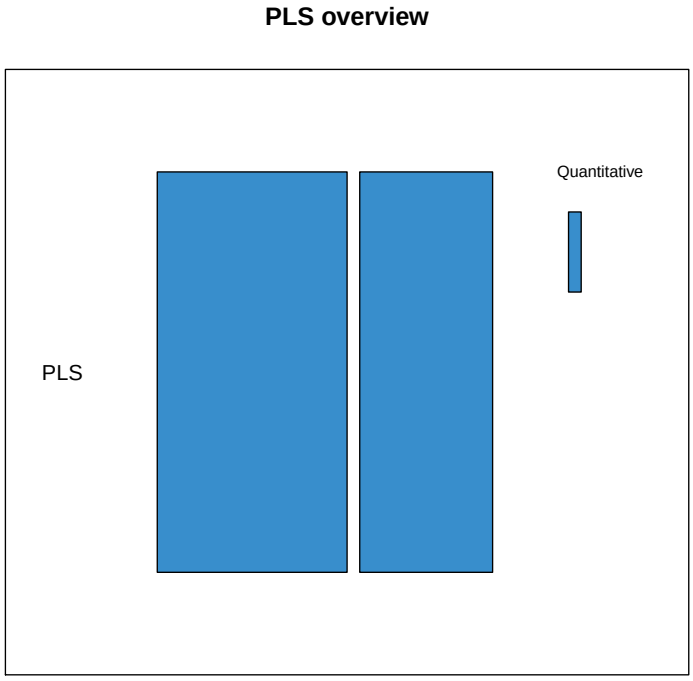
4.9 FAQ

- Can I discriminate more than two groups of samples (multiclass classification)?
  - Yes, this is one of the advantages of PLS-DA, see this example above
- Can I have a hierarchy between two factors (e.g. diet nested into genotype)?
  - Unfortunately no, sparse PLS-DA only allows to discriminate all groups at once (i.e. 4 x 2 groups when there are 4 diets and 2 genotypes)

- Can I have missing values in my data?
  - Yes in the X data set, but you won't be able to do any prediction (i.e. `tune, perf, predict`)
  - No in the Y factor

Chapter 5

Projection to Latent Structure (PLS)



5.1 Biological question

I would like to integrate two data sets measured on the same samples by extracting correlated information, or by highlighing commonalities between data sets.

5.2 The `nutrimouse` study

The `nutrimouse` study contains the expression levels of genes potentially involved in nutritional problems and the concentrations of hepatic fatty acids for forty mice. The data sets come from a nutrigenomic study in the mouse from our collaborator (Martin et al., 2007), in which the effects of five regimens with contrasted fatty acid compositions on liver lipids and hepatic gene expression in mice were considered. Two sets of variables were measured on 40 mice:

- `gene`: the expression levels of 120 genes measured in liver cells, selected among (among about 30,000) as potentially relevant in the context of the nutrition study. These expressions come from a nylon microarray with radioactive labelling.
- `lipid`: concentration (in percentage) of 21 hepatic fatty acids measured by gas chromatography.
- `diet`: a 5-level factor. Oils used for experimental diets preparation were corn and colza oils (50/50) for a reference diet (REF), hydrogenated coconut oil for a saturated fatty acid diet (COC), sunflower oil for an Omega6 fatty acid-rich diet (SUN), linseed oil for an Omega3-rich diet (LIN) and corn/colza/enriched fish oils for the FISH diet (43/43/14).
- `genotype` 2-levels factor indicating either wild-type (WT) and PPARα -/- (PPAR).

More details can be found in `?nutrimouse`.

To illustrate sparse PLS, we will integrate the gene expression levels (`gene`) with the concentrations of hepatic fatty acids (`lipid`).

5.3 Principle of PLS

Partial Least Squares (PLS) regression (Wold, 1966; Wold et al., 2001) is a multivariate methodology which relates (*integrates*) two data matrices `X` (e.g. transcriptomics) and `Y` (e.g. lipids). PLS goes beyond traditional multiple regression by modelling the structure of both matrices. Unlike traditional multiple regression models, it is not limited to uncorrelated variables. One of the many advantages of PLS is that it can handle many noisy, collinear (correlated) and missing variables and can also simultaneously model several response variables in `Y`.

PLS is a multivariate projection-based method that can address different types of integration problems. Its flexibility is the reason why it is the backbone of most methods in `mixOmics`. PLS is computationally very efficient when the number of variables $p + q >> n$ the number of samples. It performs successive local regressions that avoid computational issues due to the inversion of large singular covariance matrices. Unlike PCA which maximizes the variance of components from a single data set, PLS maximizes the covariance between

components from two data sets. The mathematical concepts of covariance and correlation are similar, but the covariance is an unbounded measure and covariance has a unit measure (see 1.2.1). In PLS, a linear combination of variables is called *latent variables* or *latent components*. The weight vectors used to calculate the linear combinations are called the *loading vectors*. Latent variables and loading vectors are thus associated and come in pairs from each of the two data sets being integrated.

5.4 Principle of sparse PLS

Even though PLS is highly efficient in a high dimensional context, the interpretability of PLS needed to be improved. sPLS has been recently developed by our team to perform simultaneous variable selection in both data sets `X` and `Y` data sets, by including LASSO ℓ_1 penalizations in PLS on each pair of loading vectors (Lê Cao et al., 2008).

5.5 Inputs and outputs

We consider the data input matrices: `X` is a $n \times p$ data matrix and `Y` a $n \times q$ data matrix, where n the number of samples (individuals), p and q are the number of variables in each data set. PLS main outputs are:

- A **set of components**, also called latent variables associated to each data set. There are as many components as the chosen dimension of the PLS.
- A **set of loading vectors**, which are coefficients assigned to each variable to define each component. These coefficients indicate the importance of each variable in PLS. Importantly, each loading vector is associated to a particular component. Loading vectors are obtained so that the covariance between a linear combination of the variables from `X` (the X-component) and from `Y` (the Y-component) is maximised.
- A **list of selected variables** from both `X` and `Y` and associated to each component if sPLS is applied.

5.6 Set up the data

We first set up the data as `X` expression matrix and `Y` as the lipid abundance matrix. We also check that the dimensions are correct and match:

```
library(mixOmics)
data(nutrimouse)
X <- nutrimouse$gene
Y <- nutrimouse$lipid
dim(X); dim(Y)
```

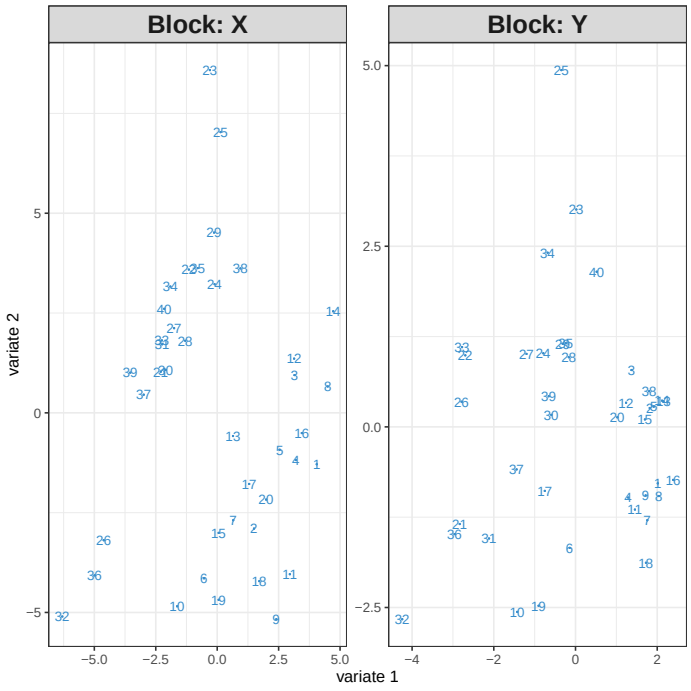
```
## [1]  40 120
```

```
## [1] 40 21
```

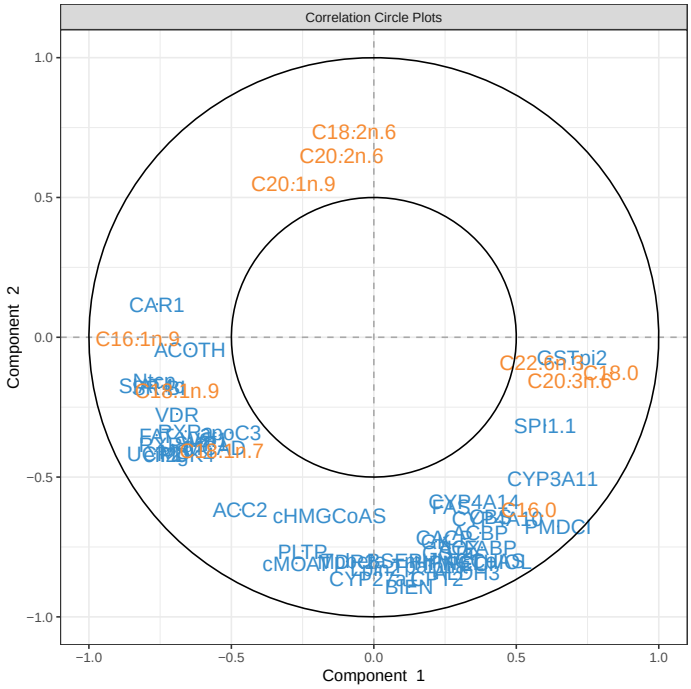
5.7 Quick start

We will mainly focus on sparse PLS for large biological data sets where variable selection can help the interpretation of the results. See `?pls` for a model with no variable selection. Here we arbitrarily set the number of variables to select to 50 on each of the 2 components of PLS (see section 5.8.5 for tuning these values).

```
MyResult.spls <- spls(X,Y, keepX = c(25, 25), keepY = c(5,5))
plotIndiv(MyResult.spls)      ## sample plot
```




```
plotVar(MyResult.spls)        ## variable plot
```



If you were to run `spls` with this minimal code, you would be using the following default values:

- `ncomp = 2`: the first two PLS components are calculated and are used for graphical outputs;
- `scale = TRUE`: data are scaled (variance = 1, strongly advised here);
- `mode = "regression"`: by default, a PLS regression mode should be used (see 5.8.6 for more details).

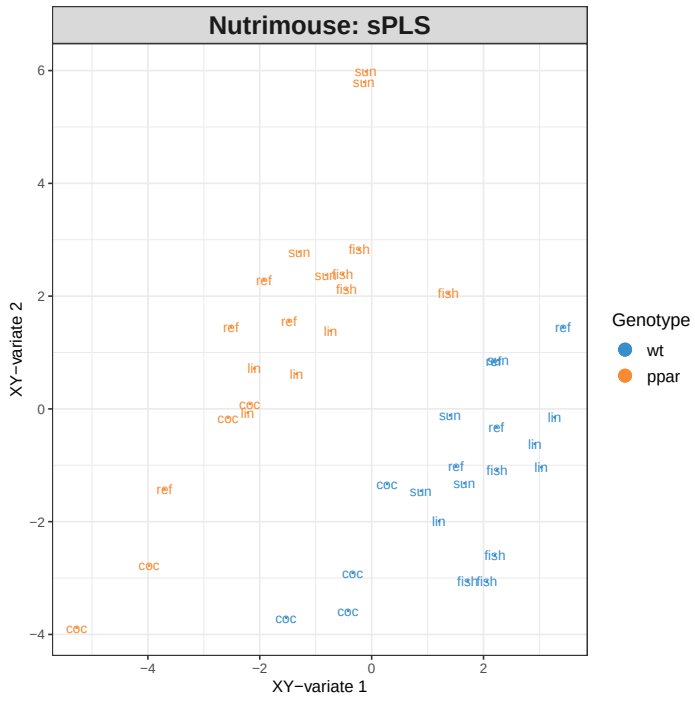
Because PLS generates a pair of components, each associated to each data set, the function `plotIndiv` produces 2 plots that represent the same samples projected in either space spanned by the X-components or the Y-components. A single plot can also be displayed, see section 5.8.1.

5.8 To go further

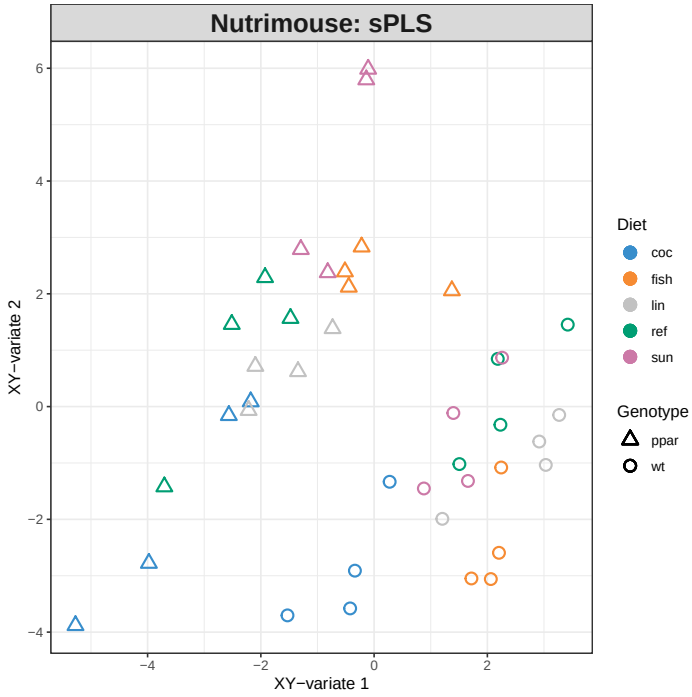
5.8.1 Customize sample plots

Some of the sample plot additional arguments were described in 4.7.1. In addition, you can choose the representation space to be either the components from the `X`-data set, the `Y`- data set, or an average between both components `rep.space = 'XY-variate'`. See more examples in `examples(plotIndiv)` and on our website. Here are two examples with colours indicating genotype or diet:

```
plotIndiv(MyResult.spls, group = nutrimouse$genotype,
          rep.space = "XY-variate", legend = TRUE,
          legend.title = 'Genotype',
          ind.names = nutrimouse$diet,
          title = 'Nutrimouse: sPLS')
```



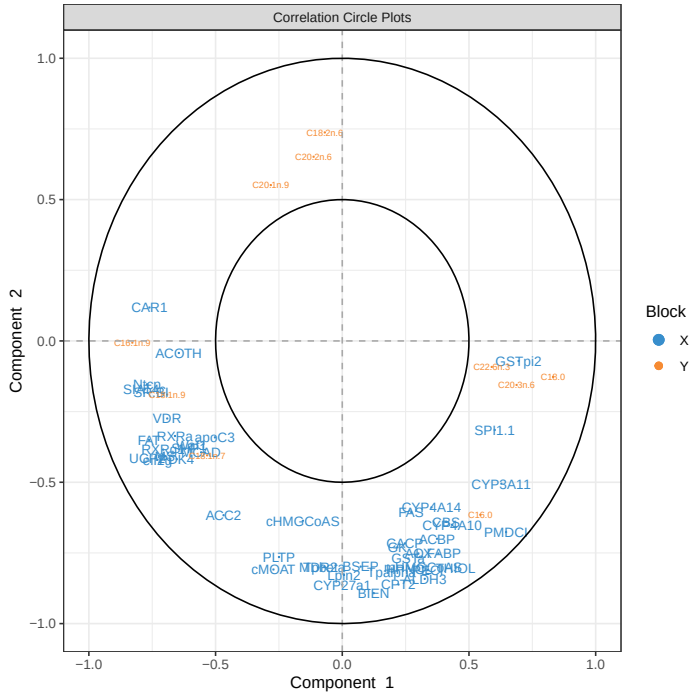

```
plotIndiv(MyResult.spls, group=nutrimouse$diet,
          pch = nutrimouse$genotype,
          rep.space = "XY-variate",  legend = TRUE,
          legend.title = 'Diet', legend.title.pch = 'Genotype',
          ind.names = FALSE,
          title = 'Nutrimouse: sPLS')
```



5.8.2 Customize variable plots

See (`example(plotVar)`) for more examples. Here we change the size of the labels. By default the colours are assigned to each type of variable.

```
plotVar(MyResult.spls, cex=c(3,2), legend = TRUE)
```



The coordinates of the variables can also be saved as follows:

```
coordinates <- plotVar(MyResult.spls, plot = FALSE)
```

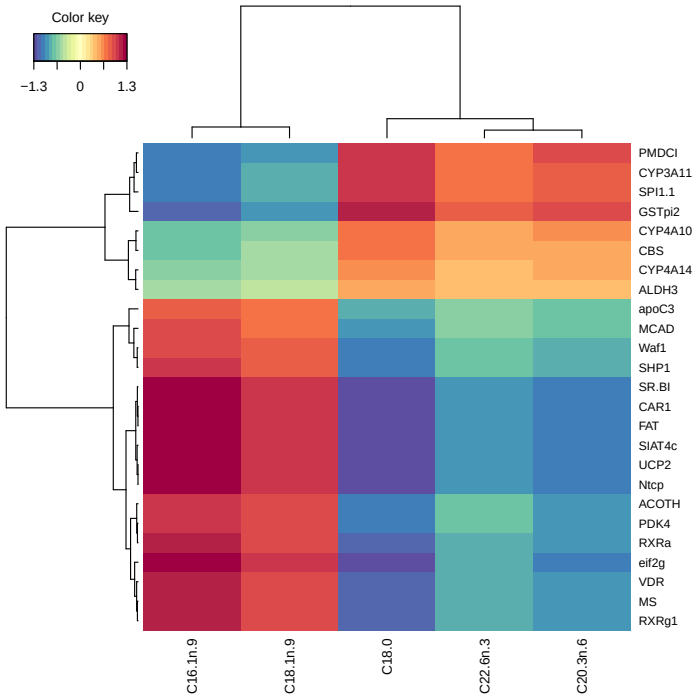
5.8.3 Other useful plots for data integration

We extended other types of plots, based on clustered image maps and relevance networks to ease the interpretation of the relationships between two types of variables. A similarity matrix is calculated from the outputs of PLS and represented with those graphics, see (González et al., 2012) for more details, and our website

5.8.3.1 Clustered Image Maps

A clustered image map can be produced using the `cim` function. You may experience figures margin issues in RStudio. Best is to either use `X11()` or save the plot as an external file. For example to show the correlation structure between the X and Y variables selected on component 1:

```
X11()
cim(MyResult.spls, comp = 1)
```




```
## or save it
cim(MyResult.spls, comp = 1, save = 'jpeg', name.save = 'PLScim')
```

5.8.3.2 Relevance networks

Using the same similarity matrix input in CIM, we can also represent relevance bipartite networks. Those networks only represent edges between one type of variables from `X` and the other type of variable, from `Y`. Whilst we use sPLS to narrow down to a few key correlated variables, our `keepX` and `keepY` values might still be very high for this kind of output. A cut-off can be set based on the correlation coefficient between the different types of variables.

Other arguments such as `interactive = TRUE` enables a scrollbar to change the cut-off value interactively, see other options in `?network`. Additionally, the graph object can be saved to be input into Cytoscape for improved visualisation.

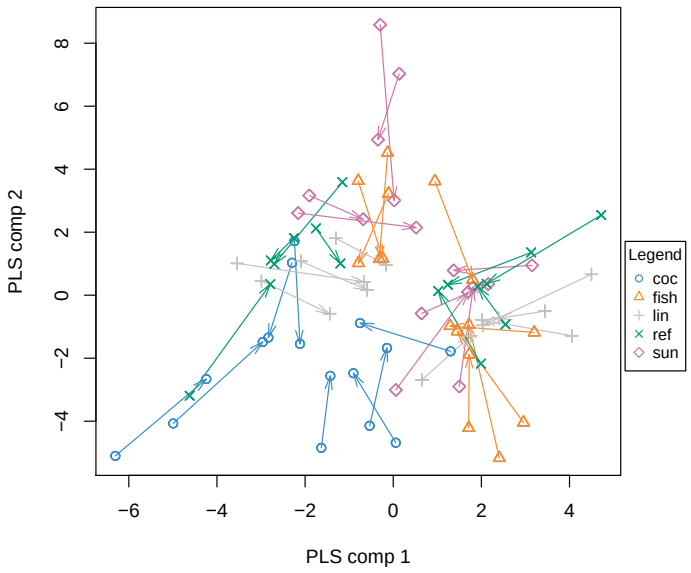
```
X11()
network(MyResult.spls, comp = 1)
```

```
## or save it
network(MyResult.spls, comp = 1, cutoff = 0.6, save = 'jpeg', name.save = 'PLSnetwork')
# save as graph object for cytoscape
myNetwork <- network(MyResult.spls, comp = 1)$gR
```

5.8.3.3 Arrow plots

Instead of projecting the samples into the combined XY representation space, as shown in 5.8.1, we can overlap the X- and Y- representation plots. One arrow joins the same sample from the X- space to the Y- space. Short arrows indicate a good agreement found by the PLS between both data sets.

```
plotArrow(MyResult.spls,group=nutrimouse$diet, legend = TRUE,
          X.label = 'PLS comp 1', Y.label = 'PLS comp 2')
```



5.8.4 Variable selection outputs

The selected variables can be extracted using the `selectVar` function for further analysis.

```
MySelectedVariables <- selectVar(MyResult.spls, comp = 1)
MySelectedVariables$X$name # Selected genes on component 1
```

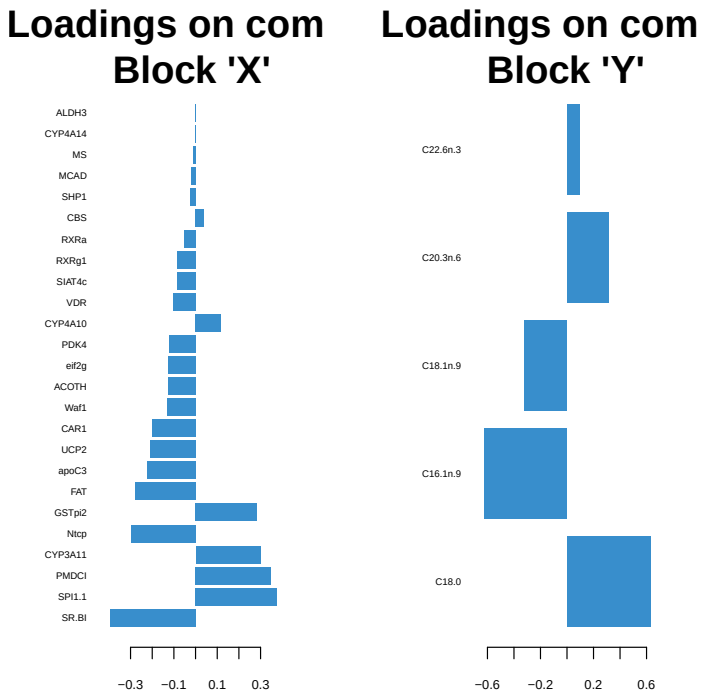
```
##  [1] "SR.BI"   "SPI1.1"  "PMDCI"   "CYP3A11" "Ntcp"    "GSTpi2"  "FAT"
##  [8] "apoC3"   "UCP2"    "CAR1"    "Waf1"    "ACOTH"   "eif2g"   "PDK4"
## [15] "CYP4A10" "VDR"     "SIAT4c"  "RXRg1"   "RXRa"    "CBS"     "SHP1"
## [22] "MCAD"    "MS"      "CYP4A14" "ALDH3"
```

```
MySelectedVariables$Y$name # Selected lipids on component 1
```

```
## [1] "C18.0"    "C16.1n.9" "C18.1n.9" "C20.3n.6" "C22.6n.3"
```

The loading plots help visualise the coefficients assigned to each selected variable on each component:

```
plotLoadings(MyResult.spls, comp = 1, size.name = rel(0.5))
```



5.8.5 Tuning parameters and numerical outputs

For PLS and sPLS, two types of parameters need to be chosen:

1 - The number of components to retain `ncomp`,

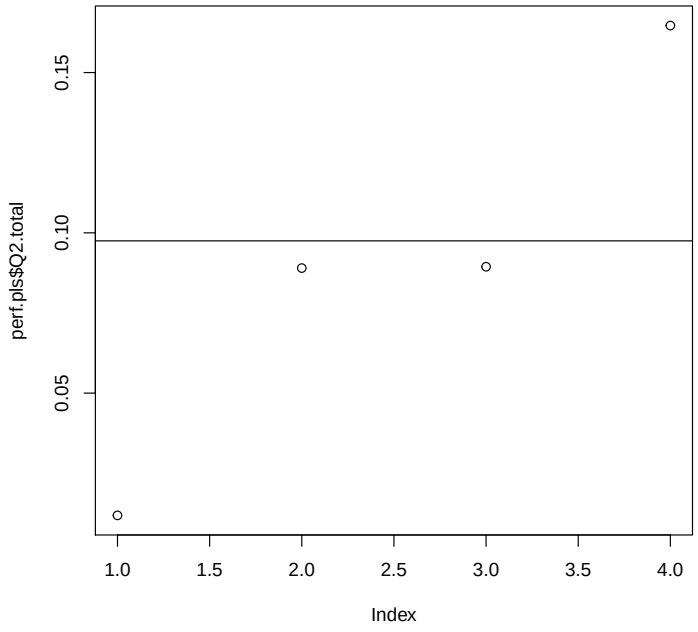
2 - The number of variables to select on each component and on each data set `keepX` and `keepY` for sparse PLS.

For **item 1** we use the `perf` function and repeated k-fold cross-validation to calculate the Q^2 criterion used in the SIMCA-P software (Umetri, 1996). The rule of thumbs is that a PLS component should be included in the model if its value is ≤ 0.0975. Here we use 3-fold CV repeated 10 times (note that we advise using at least 50 repeats, and choose the number of folds that are appropriate for the sample size of the data set).

We run a PLS model with a sufficient number of components first, then run `perf` on the object.

```
MyResult.pls <- pls(X,Y, ncomp = 4)
set.seed(30) # for reproducbility in this vignette, otherwise increase nrepeat
perf.pls <- perf(MyResult.pls, validation = "Mfold", folds = 5,
                 progressBar = FALSE, nrepeat = 10)
```

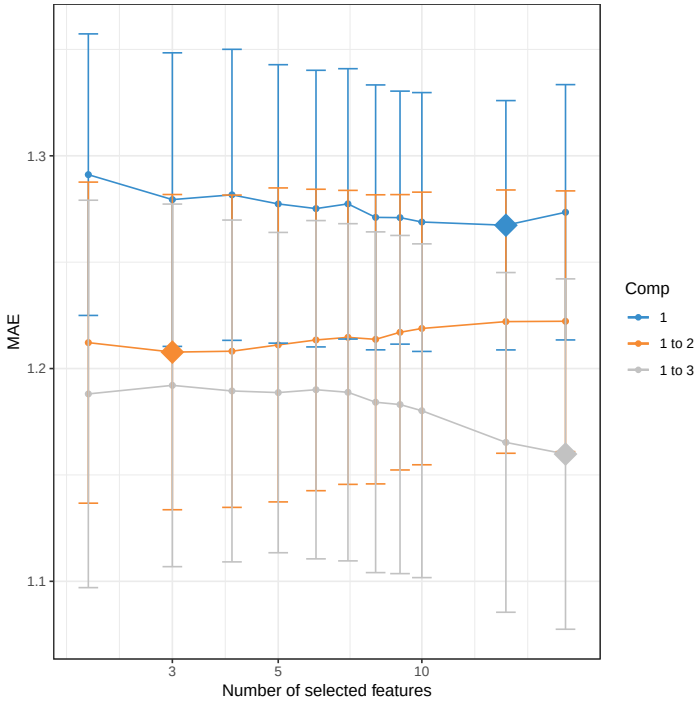
```
plot(perf.pls$Q2.total)
abline(h = 0.0975)
```



This example seems to indicate that up to 3 components could be enough. In a small $p + q$ setting we generally observe a Q^2 that decreases, but that is not the case here as $n << p + q$.

Item 2 can be quite difficult to tune. Here is a minimal example where we only tune `keepX` based on the Mean Absolute Value. Other measures proposed are Mean Square Error, Bias and R2 (see `?tune.spls`):

```
list.keepX <- c(2:10, 15, 20)
# tuning based on MAE
set.seed(30) # for reproducbility in this vignette, otherwise increase nrepeat
tune.spls.MAE <- tune.spls(X, Y, ncomp = 3,
                           test.keepX = list.keepX,
                           validation = "Mfold", folds = 5,
                           nrepeat = 10, progressBar = FALSE,
                           measure = 'MAE')
plot(tune.spls.MAE, legend.position = 'topright')
```



Based on the lowest MAE obtained on each component, the optimal number of variables to select in the `X` data set, including all variables in the `Y` data set would be:

```
tune.spls.MAE$choice.keepX
```

```
## comp1 comp2 comp3
##    15     3    20
```

Tuning `keepX` and `keepY` conjointly is still work in progress. What we advise in the meantime is either to adopt an arbitrary approach by setting those parameters arbitrarily, depending on the biological question, or tuning one parameter then the other.

5.8.6 PLS modes

You may have noticed the `mode` argument in PLS. We can calculate the residual matrices at each PLS iteration differently. Note: this is for **advanced** users.

5.8.6.1 Regression mode

The PLS regression mode models *uni-directional* (or 'causal') relationship between two data sets. The `Y` matrix is deflated with respect to the information extracted/modelled from the local regression on X. Here the goal is to predict `Y` from `X` (`Y` and `X` play an *asymmetric role*). Consequently, the latent variables

computed to predict Y from X are different from those computed to predict X from Y. More details about the model can be found in[Appendix (Lê Cao et al., 2008).

PLS regression mode, also called PLS2, is commonly applied for the analysis of biological data (Boulesteix and Strimmer, 2005; Bylesjö et al., 2007) due to the biological assumptions or the biological dogma. In general, the number of variables in `Y` to predict are fewer in number than the predictors in `X`.

5.8.6.2 Canonical mode

Similar to a Canonical Correlation Analysis (CCA) framework, this mode is used to model a *bi-directional* (or *symmetrical*) relationship between the two data sets. The `Y` matrix is deflated with respect to the information extracted or modelled from the local regression on `Y`. Here `X` and `Y` play a symmetric role and the goal is similar to CCA. More details about the model can be found in (Lê Cao et al., 2009).

PLS canonical mode is not well known (yet), but is applicable when there is no *a priori* relationship between the two data sets, or in place of CCA but when variable selection is required in large data sets. In (Lê Cao et al., 2009), we compared the measures of the same biological samples on different types of microarrays, cDNA and Affymetrix arrays, to highlight complementary information at the transcripts levels. Note however that for this mode we do not provide any tuning function.

5.8.6.3 Other modes

The 'invariant' mode performs a redundancy analysis, where the `Y` matrix is not deflated. The 'classic' mode is similar to a regression mode. It gives identical results for the variates and loadings associated to the `X` data set, but differences for the loadings vectors associated to the Y data set (different normalisations are used). Classic mode is the PLS2 model as defined by (Tenenhaus, 1998), Chap 9.

5.8.6.4 Difference between PLS modes

For the first PLS dimension, all PLS modes will output the same results in terms of latent variables and loading vectors. After the first dimension, these vectors will differ, as the matrices are deflated differently.

5.9 Additional resources

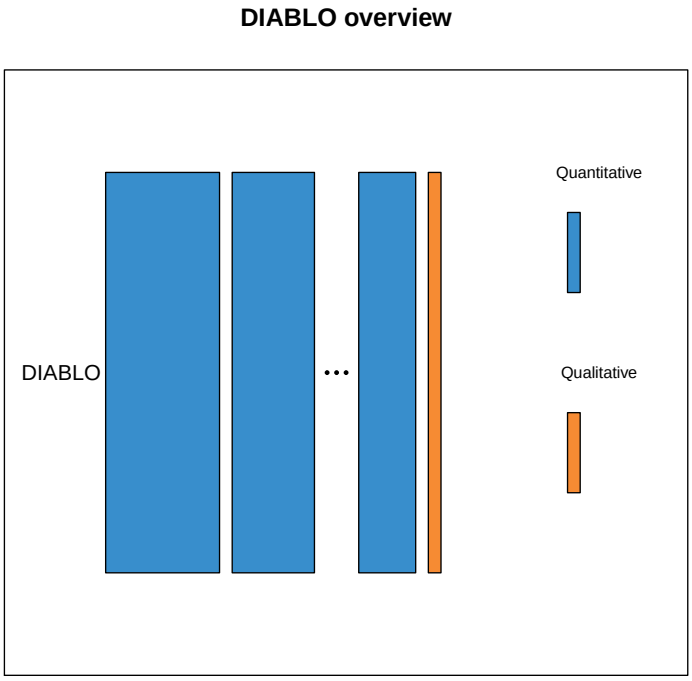
Additional examples are provided in `example(spls)` and in our case studies on our website in the **Methods** and **Case studies** sections, see also (Lê Cao et al., 2008, 2009).

5.10 FAQ

- Can PLS handle missing values?
  - Yes it can, but only for the learning / training analysis. Prediction with `perf` or `tune` is not possible with missing values.
- Can PLS deal with more than 2 data sets?
  - sPLS can only deal with 2 data sets but see `DIABLO` (Chapter 6) for multi-block analyses
- What are the differences between sPLS and Canonical Correlation Analysis (CCA, see `?rcca` in mixOmics)?
  - CCA maximises the correlation between components; PLS maximises the covariance
  - Both methods give similar results if the components are scaled, but the underlying algorithms are different:
    - CCA calculates all component at once, there is no deflation
    - PLS has different deflation mode
  - sparse PLS selects variables, CCA cannot perform variable selection
- Can I perform PLS with more variables than observations?
  - Yes, and sparse PLS is particularly useful to identify sets of variables that play a role in explaining the relationship between two data sets.
- Can I perform PLS with 2 data sets that are highly unbalanced (thousands of variables in one data set and less than 10 in the other?
  - Yes! Even if you performed sPLS to select variables in one data set (or both), you can still control the number of variables selected with `keepX`.

Chapter 6

Multi-block Discriminant Analysis with DIABLO



DIABLO is our new framework in `mixOmics` that extends PLS for multiple data sets integration and PLS-discriminant analysis. The acronyms stands for **D**ata **I**ntegration **A**nalysis for **B**iomarker discovery using a **L**atent c**O**mponents (Singh et al., 2017).

6.1 Biological question

I would like to identify a highly correlated multi-omics signature discriminating known groups of samples.

6.2 The `breast.TCGA` study

Human breast cancer is a heterogeneous disease in terms of molecular alterations, cellular composition, and clinical outcome. Breast tumours can be classified into several subtypes, according to levels of mRNA expression [Sorlie et al. (2001). Here we consider a subset of data generated by The Cancer Genome Atlas Network (Cancer Genome Atlas Network et al., 2012). For the package, data were normalised and drastically prefiltered for illustrative purposes but DIABLO can handle larger data sets, see (Rohart et al., 2017a) Table 2. The data were divided into a training set with a subset of 150 samples from the mRNA, miRNA and proteomics data, and a test set including 70 samples, but only with mRNA and miRNA data (proteomics missing). The aim of this integrative analysis is to identify a highly correlated multi-omics signature discriminating the breast cancer subtypes Basal, Her2 and LumA.

The `breast.TCGA` is a list containing training and test sets of omics data `data.train` and `data.test` which both include:

- `miRNA`: a data frame with 150 (70) rows and 184 columns in the training (test) data set for the miRNA expression levels.
- `mRNA`: a data frame with 150 (70) rows and 520 columns in the training (test) data set for the mRNA expression levels.
- `protein`: a data frame with 150 rows and 142 columns in the training data set only for the protein abundance.
- `subtype`: a factor indicating the breast cancer subtypes in the training (length of 150) and test (length of 70) sets.

More details can be found in `?breast.TCGA`.

To illustrate DIABLO, we will integrate the expression levels of miRNA, mRNA and the abundance of proteins while discriminating the subtypes of breast cancer, then predict the subtypes of the test samples in the test set.

6.3 Principle of DIABLO

The core DIABLO method extends Generalised Canonical Correlation Analysis (Tenenhaus and Tenenhaus, 2011), which contrary to what its name suggests, generalises PLS for multiple matching datasets, and the sparse sGCCA method (Tenenhaus et al., 2014). Starting from the R package RGCCA, we extended these methods for different types of analyses, including unsupervised

N-integration (`block.pls`, `block.spls`) and supervised analyses (`block.plsda`, `block.splsda`).

The aim of **N-integration** with our sparse methods is to identify correlated (or co-expressed) variables measured on heterogeneous data sets which also explain the categorical outcome of interest (supervised analysis). The multiple data integration task is not trivial, as the analysis can be strongly affected by the variation between manufacturers or omics technological platforms despite being measured on the same biological samples. Before you embark on data integration, we strongly suggest individual or paired analyses with sPLS-DA and PLS to first understand the major sources of variation in each data set and to guide the integration process.

More methodological details can be found in (Singh et al., 2017).

6.4 Inputs and outputs

We consider as input a list `X` of data frames with n rows (the number of samples) and a different number of variations in each data frame. `Y` is a factor vector of length n that indicates the class of each sample. Internally and similar to PLS-DA in Chapter 4 it will be coded as a dummy matrix.

DIABLO main outputs are:

- A **set of components**, also called latent variables associated to each data set. There are as many components as the chosen dimension of DIABLO.
- A **set of loading vectors**, which are coefficients assigned to each variable to define each component. These coefficients indicate the importance of each variable in DIABLO. Importantly, each loading vector is associated to a particular component. Loading vectors are obtained so that the covariance between a linear combination of the variables from `X` (the X-component) and from `Y` (the Y-component) is maximised.
- A **list of selected variables** from each data set and associated to each component if sparse DIABLO is applied.

6.5 Set up the data

We first set up the input data as a list of data frames `X` expression matrix and `Y` as a factor indicating the class membership of each sample. Each data frame in `X` should be **named consistently** to match with the `keepX` parameter.

We check that the dimensions are correct and match. We then set up arbitrarily the number of variables `keepX` that we wish to select in each data set and each component.

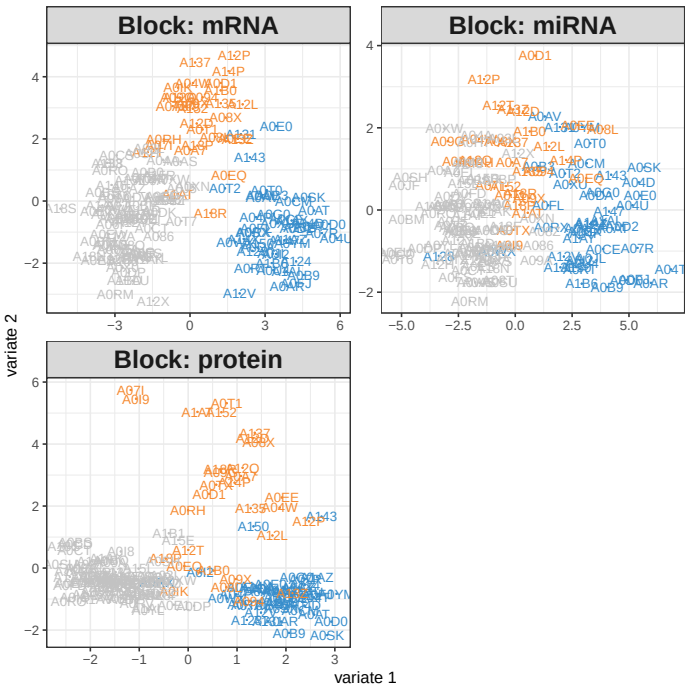
```
library(mixOmics)
data(breast.TCGA)
# extract training data and name each data frame
X <- list(mRNA = breast.TCGA$data.train$mrna,
          miRNA = breast.TCGA$data.train$mirna,
          protein = breast.TCGA$data.train$protein)
Y <- breast.TCGA$data.train$subtype
summary(Y)
```

```
## Basal  Her2  LumA
##    45    30    75
```

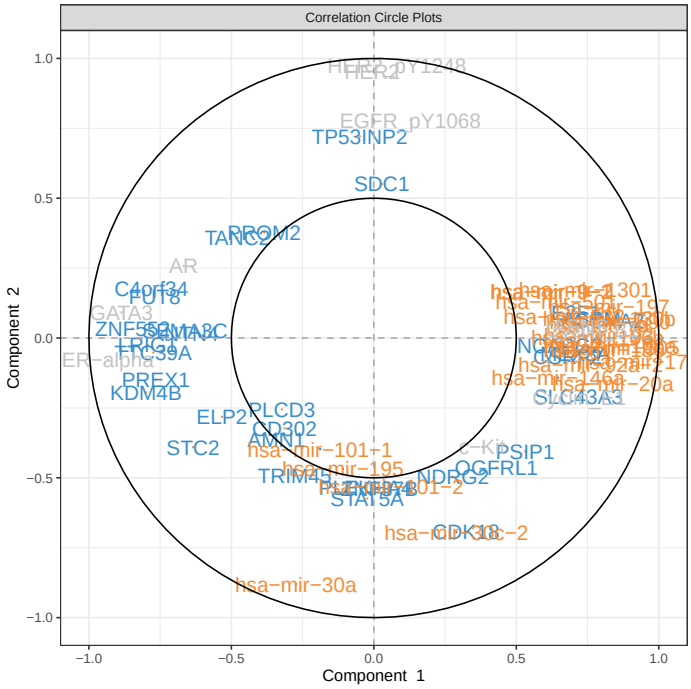
```
list.keepX <- list(mRNA = c(16, 17), miRNA = c(18,5), protein = c(5, 5))
```

6.6 Quick start

```
MyResult.diablo <- block.splsda(X, Y, keepX=list.keepX)
plotIndiv(MyResult.diablo) ## sample plot
```



```
plotVar(MyResult.diablo) ## variable plot
```



Similar to PLS (Chapter 5), DIABLO generates a pair of components, each associated to each data set. This is why we can visualise here 3 sample plots. As DIABLO is a supervised method, samples are represented with different colours depending on their known class.

The variable plot suggests some correlation structure between proteins, mRNA and miRNA. We will further customize these plots in Sections 6.7.1 and 6.7.2.

If you were to run `block.splsda` with this minimal code, you would be using the following default values:

- `ncomp = 2`: the first two PLS components are calculated and are used for graphical outputs;
- `scale = TRUE`: data are scaled (variance = 1, strongly advised here for data integration);
- `mode = "regression"`: by default a PLS regression mode should be used (see Section 5.8.6 for more details) .

We focused here on the sparse version as would like to identify a minimal multi-omics signature, however, the non-sparse version could also be run with `block.plsda`:

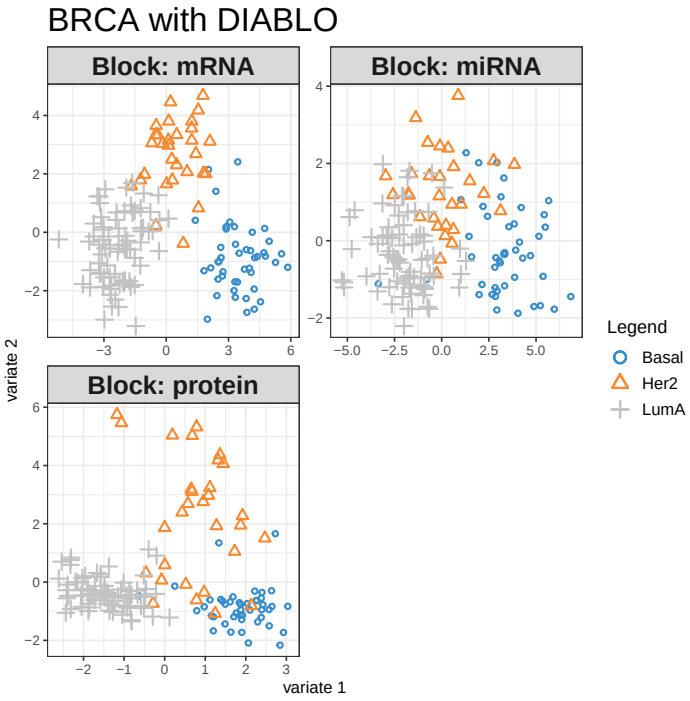
```
MyResult.diablo2 <- block.plsda(X, Y)
```

6.7 To go further

6.7.1 Customize sample plots

Here is an example of an improved plot, see also Section 4.7.1 for additional sources of inspiration.

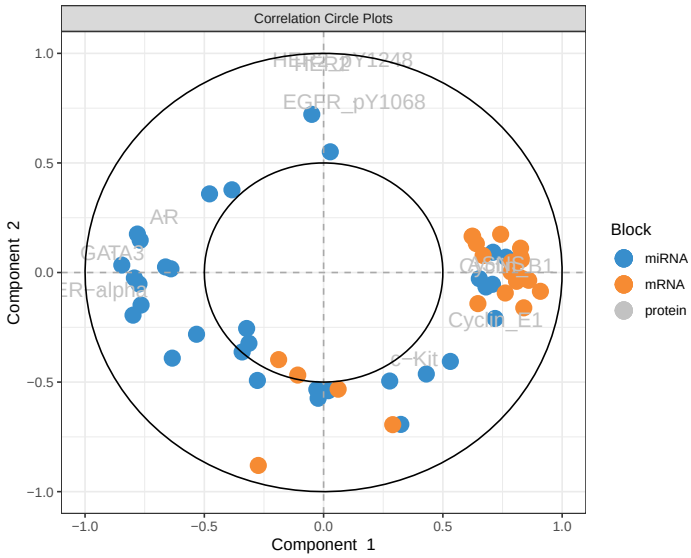
```
plotIndiv(MyResult.diablo,
          ind.names = FALSE,
          legend=TRUE, cex=c(1,2,3),
          title = 'BRCA with DIABLO')
```



6.7.2 Customize variable plots

Labels can be omitted in some data sets to improve readability. For example her we only show the name of the proteins:

```
plotVar(MyResult.diablo, var.names = c(FALSE, FALSE, TRUE),
        legend=TRUE, pch=c(16,16,1))
```



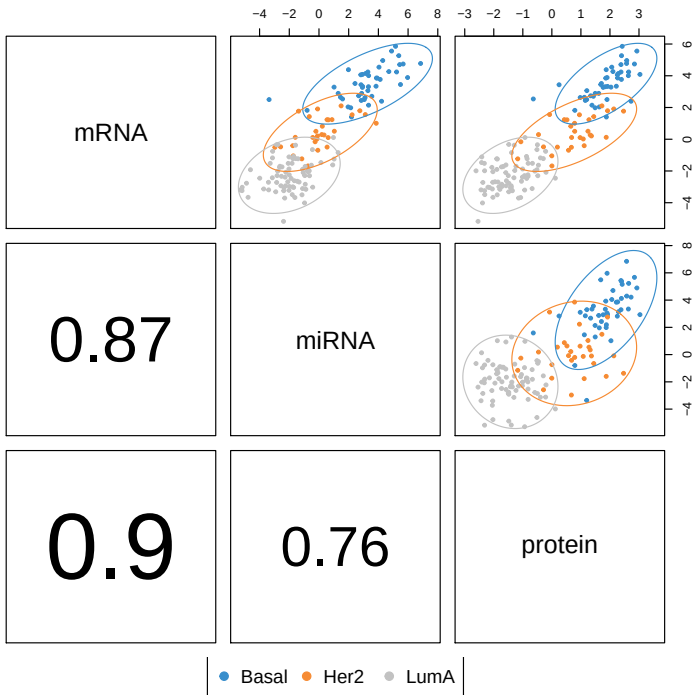
6.7.3 Other useful plots for data integration

Several plots were added for the DIABLO framework.

6.7.3.1 `plotDiablo`

A global overview of the correlation structure at the component level can be represented with the `plotDiablo` function. It plots the components across the different data sets for a given dimension. Colours indicate the class of each sample.

```
plotDiablo(MyResult.diablo, ncomp = 1)
```

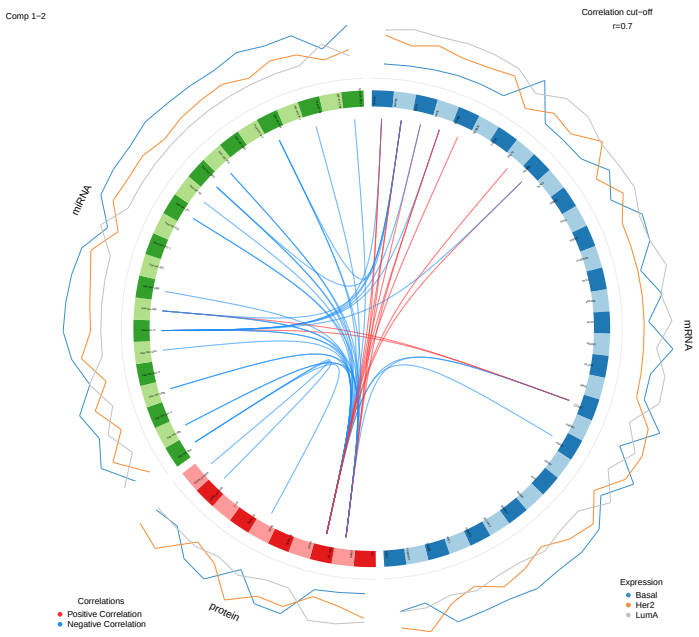


Here, we can see that a strong correlation is extracted by DIABLO between the mRNA and protein data sets. Other dimensions can be plotted with the argument `comp`.

6.7.3.2 `circosPlot`

The circos plot represents the correlations between variables of different types, represented on the side quadrants. Several display options are possible, to show within and between connexions between blocks, expression levels of each variable according to each class (argument `line = TRUE`). The circos plot is built based on a similarity matrix, which was extended to the case of multiple data sets from (González et al., 2012). A `cutoff` argument can be included to visualise correlation coefficients above this threshold in the multi-omics signature.

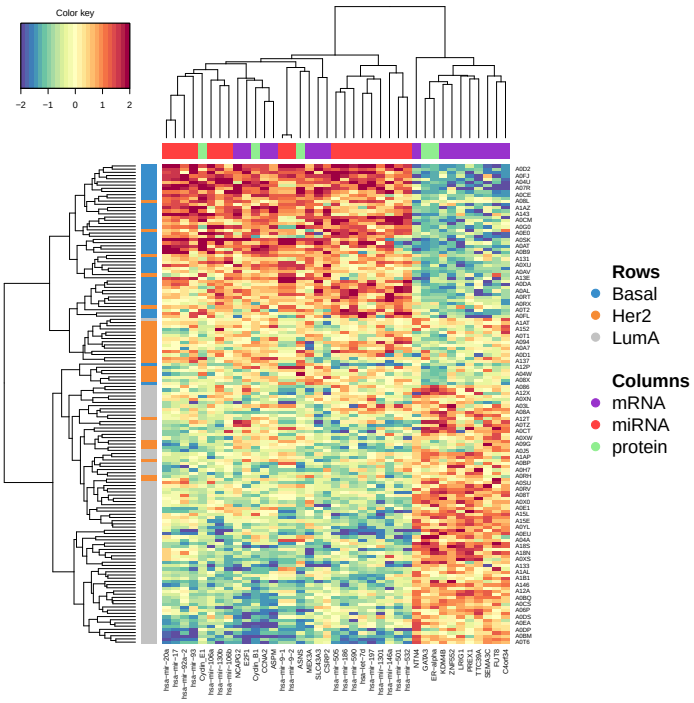
```
circosPlot(MyResult.diablo, cutoff=0.7)
```



6.7.3.3 `cimDiablo`

The `cimDiablo` function is a clustered image map specifically implemented to represent the multi-omics molecular signature expression for each sample. It is very similar to a classic hierarchical clustering:

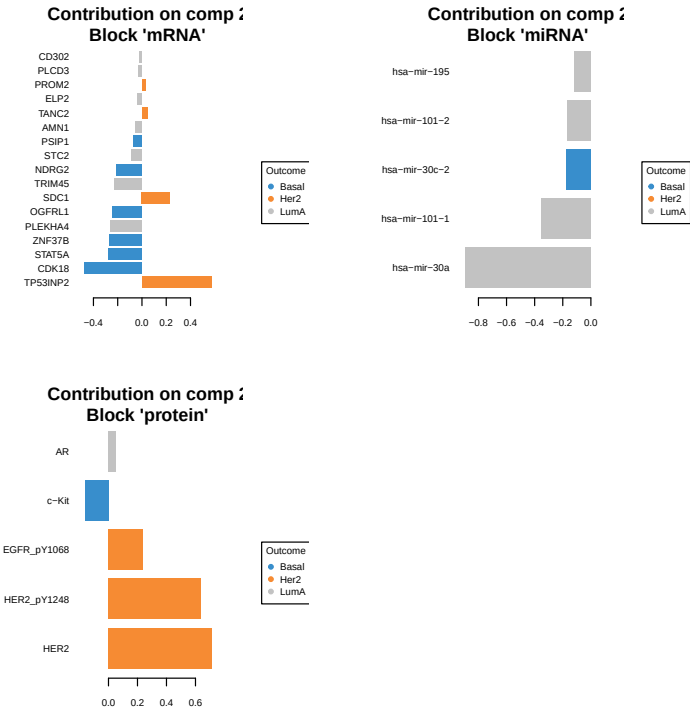
```
# minimal example with margins improved:
# cimDiablo(MyResult.diablo, margin=c(8,20))
# extended example:
cimDiablo(MyResult.diablo, color.blocks = c('darkorchid', 'brown1', 'lightgreen'), comp = 1, marg
```



6.7.3.4 plotLoadings

The `plotLoadings` function visualises the loading weights of each selected variables on each component (default is `comp = 1`) and each data set. The color indicates the class in which the variable has the maximum level of expression (`contrib = "max"`) or minimum (`contrib ="min"`), on average (`method="mean"`) or using the median (`method ="median"`). We only show the last plot here:

```
#plotLoadings(MyResult.diablo, contrib = "max")
plotLoadings(MyResult.diablo, comp = 2, contrib = "max")
```



6.7.3.5 Relevance networks

Another visualisation of the correlation between the different types of variables is the relevance network, which is also built on the similarity matrix (González et al., 2012). Each colour represents a type of variable. A threshold can also be set using the argument `cutoff`.

See also Section 5.8.3.2 to save the graph and the different options, or `?network`.

```
network(MyResult.diablo, blocks = c(1,2,3),
        color.node = c('darkorchid', 'brown1', 'lightgreen'),
        cutoff = 0.6, save = 'jpeg', name.save = 'DIABLOnetwork')
```

6.8 Numerical outputs

6.8.1 Classification performance

Similar to what is described in Section 4.7.5 we use repeated cross-validation with `perf` to assess the prediction of the model. For this complex classification problems, often a centroid distance is suitable, see details in (Rohart et al., 2017a) Suppl. Material S1.

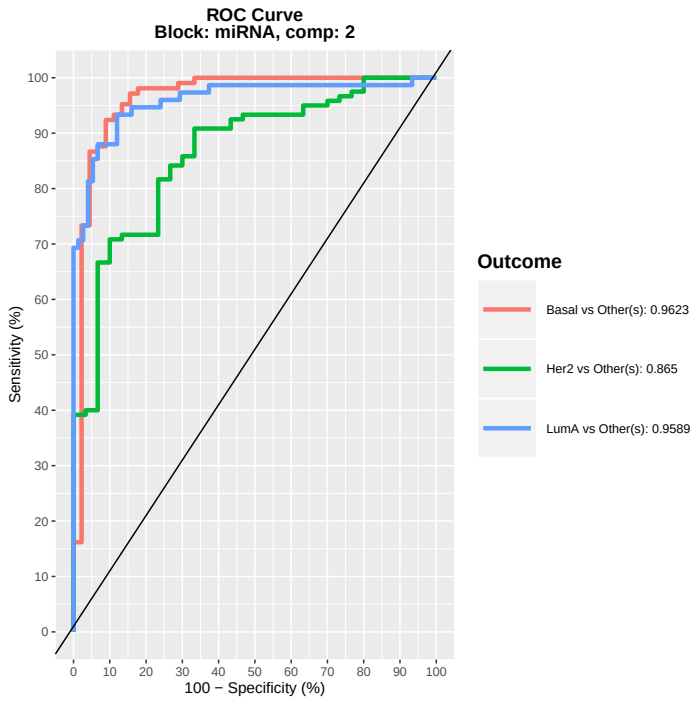
```
set.seed(123) # for reproducibility in this vignette
MyPerf.diablo <- perf(MyResult.diablo, validation = 'Mfold', folds = 5,
                  nrepeat = 10,
                  dist = 'centroids.dist')

#MyPerf.diablo  # lists the different outputs

# Performance with Majority vote
#MyPerf.diablo$MajorityVote.error.rate
```

6.8.2 AUC

An AUC plot per block is plotted using the function `auroc` see (Rohart et al., 2017a) for the interpretation of such output as the ROC and AUC criteria are not particularly insightful in relation to the performance evaluation of our methods, but can complement the statistical analysis.

Here we evaluate the AUC for the model that includes 2 components in the miRNA data set.

```
Myauc.diablo <- auroc(MyResult.diablo, roc.block = "miRNA", roc.comp = 2)
```



6.8.2.1 Prediction on an external test set

The `predict` function predicts the class of samples from a test set. In our specific case, one data set is missing in the test set but the method can still be applied. Make sure the names of the blocks correspond exactly.

```
# prepare test set data: here one block (proteins) is missing
X.test <- list(mRNA = breast.TCGA$data.test$mrna,
                       miRNA = breast.TCGA$data.test$mirna)

Mypredict.diablo <- predict(MyResult.diablo, newdata = X.test)
# the warning message will inform us that one block is missing
#Mypredict.diablo # list the different outputs
```

The confusion table compares the real subtypes with the predicted subtypes for a 2 component model, for the distance of interest:

```
confusion.mat <- get.confusion_matrix(
                truth = breast.TCGA$data.test$subtype,
                predicted = Mypredict.diablo$MajorityVote$centroids.dist[,2])
kable(confusion.mat)
```

	predicted.as.Basal	predicted.as.Her2	predicted.as.LumA	predicted.as.NA
Basal	15	1	0	5
Her2	0	11	0	3
LumA	0	0	27	8

```
get.BER(confusion.mat)
```

```
## [1] 0.2428571
```

6.8.3 Tuning parameters

For DIABLO, the parameters to tune are:

1 - The design matrix `design` indicates, which data sets or blocks should be connected to maximise the covariance between components, and to which extent. A compromise needs to be achieved between maximising the correlation between data sets (design value between 0.5 and 1) and maximising the discrimination with the outcome `Y` (design value between 0 and 0.5), see (Singh et al., 2017) for more details.

2 - The number of components to retain `ncomp`. The rule of thumb is usually $K - 1$ where K is the number of classes, but it is worth testing a few extra components.

3 - The number of variables to select on each component and on each data set in the list `keepX`.

For **item 1**, by default all data sets are linked as follows:

```
MyResult.diablo$design
```

```
##         mRNA miRNA protein Y
## mRNA       0     1       1 1
## miRNA      1     0       1 1
## protein    1     1       0 1
## Y          1     1       1 0
```

The `design` can be changed as follows. By default each data set will be linked to the `Y` outcome.

```
MyDesign <- matrix(c(0, 0.1, 0.3,
                     0.1, 0, 0.9,
                     0.3, 0.9, 0),
                   byrow=TRUE,
                   ncol = length(X), nrow = length(X),
                  dimnames = list(names(X), names(X)))
MyDesign
```

```
##         mRNA miRNA protein
## mRNA     0.0   0.1     0.3
## miRNA    0.1   0.0     0.9
## protein  0.3   0.9     0.0
```

```
MyResult.diablo.design <- block.splsda(X, Y, keepX=list.keepX, design=MyDesign)
```

Items 2 and 3 can be tuned using repeated cross-validation, as we described in Chapter 4. A detailed tutorial is provided on our website in the different DIABLO tabs.

6.9 Additional resources

Additional examples are provided in `example(block.splsda)` and in our DIABLO tab in http://www.mixomics.org. Also, have a look at (Singh et al., 2017)

6.10 FAQ

- When performing a multi-block analysis, how do I choose my design?
  - We recommend first relying on some prior biological knowledge you may have on the relationship you expect to see between data sets. Conduct a few trials on a non-sparse version `block.plsda`, look at the classification performance with `perf` and `plotDiablo` before you can decide on your final design.
- I have a small number of samples (n < 10), should I still tune `keepX`?

  - It is probably not worth it. Try with a few `keepX` values and look at the graphical outputs so see if they make sense. With a small n you can adopt an exploratory approach that does not require a performance assessment.
- During `tune` or `perf` the code broke down (`system computationally singular`).
  - Check that the M value for your M-fold is not too high compared to n (you want $n/M > 6 - 8$ as rule of thumb). Try leave-one-out instead with `validation = 'loo'` and make sure `ncomp` is not too large as you are running on empty matrices!
- My tuning step indicated the selection of only 1 miRNA…
  - Choose a grid of `keepX` values that starts at a higher value (e.g. 5). The algorithm found an optimum with only one variable, either because it is highly discriminatory or because the data are noisy, but it does not stop you from trying for more.
- My Y is continuous, what can I do?
  - You can perform a multi-omics regression with `block.spls`. We have not found a way yet to tune the results so you will need to adopt an exploratory approach or back yourself up with downstream analyses once you have identified a list of highly correlated features.

Chapter 7

Session Information

```
sessionInfo()
```

```
## R version 3.6.0 (2019-04-26)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Mojave 10.14.4
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_AU.UTF-8/en_AU.UTF-8/en_AU.UTF-8/C/en_AU.UTF-8/en_AU.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] magrittr_1.5  Alstuff_0.1.0
##
## loaded via a namespace (and not attached):
##  [1] Biobase_2.44.0       viridis_0.5.1        tidyr_0.8.3
##  [4] bit64_0.9-7          jsonlite_1.6         viridisLite_0.3.0
##  [7] topGO_2.36.0         R.utils_2.9.0-9000   assertthat_0.2.1
## [10] stats4_3.6.0         blob_1.2.0           xlsxjars_0.6.1
## [13] yaml_2.2.0           pillar_1.4.2         RSQLite_2.1.1
## [16] backports_1.1.4      lattice_0.20-38      glue_1.3.1
## [19] downloader_0.4       digest_0.6.20        RColorBrewer_1.1-2
## [22] colorspace_1.4-1     htmltools_0.3.6      R.oo_1.22.0
## [25] plyr_1.8.4           XML_3.98-1.20        pkgconfig_2.0.2
```

```
## [28] SparseM_1.77          bookdown_0.12         DiagrammeR_1.0.1
## [31] purrr_0.3.2           GO.db_3.8.2           scales_1.0.0
## [34] brew_1.0-6            tibble_2.1.3          IRanges_2.18.1
## [37] ggplot2_3.2.0         influenceR_0.1.0      BiocGenerics_0.30.0
## [40] lazyeval_0.2.2        rgexf_0.15.3          crayon_1.3.4
## [43] memoise_1.1.0         evaluate_0.14         data.tree_0.7.8
## [46] R.methodsS3_1.7.1     graph_1.62.0          Rook_1.1-1
## [49] tools_3.6.0           hms_0.5.0             matrixStats_0.54.0
## [52] stringr_1.4.0         xlsx_0.6.1            S4Vectors_0.22.0
## [55] munsell_0.5.0         AnnotationDbi_1.46.0  compiler_3.6.0
## [58] rlang_0.4.0           grid_3.6.0            rstudioapi_0.10
## [61] htmlwidgets_1.3       visNetwork_2.0.7      igraph_1.2.4.1
## [64] rmarkdown_1.14        gtable_0.3.0          DBI_1.0.0
## [67] R6_2.4.0              gridExtra_2.3         knitr_1.23
## [70] dplyr_0.8.3           bit_1.1-14            zeallot_0.1.0
## [73] readr_1.3.1           rJava_0.9-11          stringi_1.4.3
## [76] parallel_3.6.0        Rcpp_1.0.1            vctrs_0.2.0
## [79] tidyselect_0.2.5      xfun_0.8
```

Bibliography

Barker, M. and Rayens, W. (2003). Partial least squares for discrimination. *Journal of chemometrics*, 17(3):166–173.

Boulesteix, A. and Strimmer, K. (2005). Predicting transcription factor activities from combined analysis of microarray and chip data: a partial least squares approach. *Theor Biol Med Model*, 2(23).

Boulesteix, A. and Strimmer, K. (2007). Partial least squares: a versatile tool for the analysis of high-dimensional genomic data. *Briefings in Bioinformatics*, 8(1):32.

Bylesjö, M., Eriksson, D., Kusano, M., Moritz, T., and Trygg, J. (2007). Data integration in plant biology: the o2pls method for combined modeling of transcript and metabolite data. *The Plant Journal*, 52:1181–1191.

Cancer Genome Atlas Network et al. (2012). Comprehensive molecular portraits of human breast tumours. *Nature*, 490(7418):61–70.

Chung, D. and Keles, S. (2010). Sparse Partial Least Squares Classification for High Dimensional Data. *Statistical Applications in Genetics and Molecular Biology*, 9(1):17.

González, I., Déjean, S., Martin, P. G., and Baccini, A. (2008). CCA: An R package to extend canonical correlation analysis. *Journal of Statistical Software*, 23(12):1–14.

González, I., Lê Cao, K.-A., Davis, M. J., Déjean, S., et al. (2012). Visualising associations between paired 'omics' data sets. *BioData mining*, 5(1):19.

Jolliffe, I. (2005). *Principal component analysis*. Wiley Online Library.

Khan, J., Wei, J. S., Ringner, M., Saal, L. H., Ladanyi, M., Westermann, F., Berthold, F., Schwab, M., Antonescu, C. R., Peterson, C., et al. (2001). Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks. *Nature medicine*, 7(6):673–679.

Lê Cao, K., Rossouw, D., Robert-Granié, C., Besse, P., et al. (2008). A sparse PLS for variable selection when integrating omics data. *Statistical applications in genetics and molecular biology*, 7:Article–35.

Lê Cao, K.-A., Boitard, S., and Besse, P. (2011). Sparse PLS Discriminant Analysis: biologically relevant feature selection and graphical displays for multiclass problems. *BMC bioinformatics*, 12(1):253.

Lê Cao, K.-A., Costello, M.-E., Chua, X.-Y., Brazeilles, R., and Rondeau, P. (2016). Mixmc: Multivariate insights into microbial communities. *PloS one*, 11(8):e0160169.

Lê Cao, K.-A., Martin, P. G., Robert-Granié, C., and Besse, P. (2009). Sparse canonical methods for biological data integration: application to a cross-platform study. *BMC bioinformatics*, 10(1):34.

Mariette, J. and Villa-Vialaneix, N. (2017). Unsupervised multiple kernel learning for heterogeneous data integration. *Bioinformatics*, 34(6):1009–1015.

Martin, P., Guillou, H., Lasserre, F., Déjean, S., Lan, A., Pascussi, J.-M., San Cristobal, M., Legrand, P., Besse, P., and Pineau, T. (2007). Novel aspects of PPARalpha-mediated regulation of lipid and xenobiotic metabolism revealed through a multrigenomic study. *Hepatology*, 54:767–777.

Nguyen, D. and Rocke, D. (2002). Tumor classification by partial least squares using microarray gene expression data. *Bioinformatics*, 18(1):39.

Rohart, F., Gautier, B., Singh, A., and Lê Cao, K.-A. (2017a). mixomics: an r package for 'omics feature selection and multiple data integration. *PLoS Computational Biology*, 13(11).

Rohart, F., Matigian, N., Eslami, A., S, B., and Lê Cao, K.-A. (2017b). Mint: A multivariate integrative method to identify reproducible molecular signatures across independent experiments and platforms. *BMC bioinformatics*, 18(1):128.

Shen, H. and Huang, J. Z. (2008). Sparse principal component analysis via regularized low rank matrix approximation. *Journal of multivariate analysis*, 99(6):1015–1034.

Singh, A., Gautier, B., Shannon, C., Rohart, F., Vacher, M., S, T., and Lê Cao, K.-A. (2017). Diablo: identifying key molecular drivers from multi-omic assays, an integrative approach.

Sorlie, T., Perou, C. M., Tibshirani, R., Aas, T., Geisler, S., Johnsen, H., Hastie, T., Eisen, M. B., Van De Rijn, M., Jeffrey, S. S., et al. (2001). Gene expression patterns of breast carcinomas distinguish tumor subclasses with clinical implications. *Proceedings of the National Academy of Sciences*, 98(19):10869–10874.

Tan, Y., Shi, L., Tong, W., Gene Hwang, G., and Wang, C. (2004). Multi-class tumor classification by discriminant partial least squares using microarray gene expression data and assessment of classification models. *Computational Biology and Chemistry*, 28(3):235–243.

Tenenhaus, A., Philippe, C., Guillemot, V., Le Cao, K.-A., Grill, J., and Frouin, V. (2014). Variable selection for generalized canonical correlation analysis. *Biostatistics*, 15(3):569–583.

Tenenhaus, A. and Tenenhaus, M. (2011). Regularized generalized canonical correlation analysis. *Psychometrika*, 76(2):257–284.

Tenenhaus, M. (1998). *La régression PLS: théorie et pratique*. Editions Technip.

Teng, M., Love, M. I., Davis, C. A., Djebali, S., Dobin, A., Graveley, B. R., Li, S., Mason, C. E., Olson, S., Pervouchine, D., et al. (2016). A benchmark for rna-seq quantification pipelines. *Genome biology*, 17(1):74.

Umetri, A. (1996). SIMCA-P for windows, Graphical Software for Multivariate Process Modeling. *Umea, Sweden*.

Wold, H. (1966). *Estimation of principal components and related models by iterative least squares*. New York: Academic Press.

Wold, S., Sjöström, M., and Eriksson, L. (2001). Pls-regression: a basic tool of chemometrics. *Chemometrics and intelligent laboratory systems*, 58(2):109–130.