Introduction to PCA and PLS

Benoit Liquet*

*benoit.liquet-weiland@mq.edu.au

20 May 2022

Abstract

This document presents briefly the Principal Component Analyis (PCA) and the Partial Least Square method (PLS). These popular reduction methods are presented through different case studies using different R packages. Sparse versions which enable variable selection are also presented.

Contents

1 Introduction

1.1 Data and Goals

Goal of Integrative Analysis:

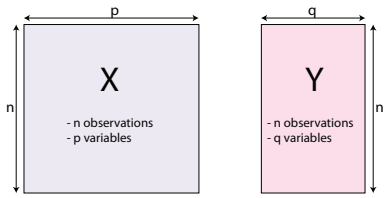
Wikipedia. **Data integration** "involves **combining data** residing in different sources and providing users with a unified view of these data. This process becomes significant in a variety of situations, which include both commercial and **scientific**".

System Biology. **Integrative Analysis:** Analysis of heterogeneous types of data from inter-platform technologies.

Goal. Combine multiple types of data:

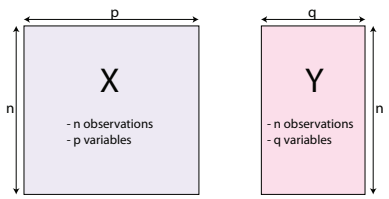
- Contribute to a better understanding of biological mechanism.
- Have the potential to improve the diagnosis and treatments of complex diseases.

1.2 Example: Data definition



- "Omics." $\mathbf{Y}$ matrix: gene expression, $\mathbf{X}$ matrix: SNP (single nucleotide polymorphism). Many others such as proteomic, metabolomic data.
- "neuroimaging". $\mathbf{Y}$ matrix: behavioral variables, $\mathbf{X}$ matrix: brain activity (e.g., EEG, fMRI, NIRS)
- "neuroimaging genetics." $\mathbf{Y}$ matrix: fMRI (Fusion of functional magnetic resonance imaging), $\mathbf{X}$ matrix: SNP
- "Ecology/Environment." $\mathbf{Y}$ matrix: Water quality variables , $\mathbf{X}$ matrix: Landscape variables

1.3 Questions



Some possible questions:

- How to investigate the relation of these two blocks of Data?

- How to dentify which of the p variables in X (OMICs data) are associated with the outcome Y (disease status or correlated continuous marker)
- Integrative Omics Analysis: Can we identify a subset of correlated genes and proteins?

1.3.1 Constraints

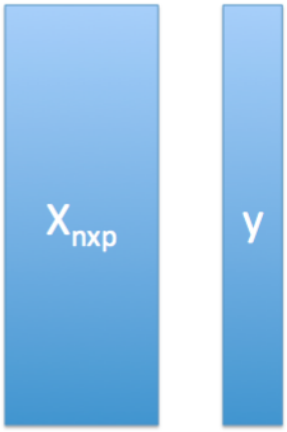
- The $n < p$ situation:
  - More predictors than observations

    $\Rightarrow$ numerically intractable statistical inferences
  - Data from multiple source.

1.4 Low-Dimensional Versus High-Dimensional

- The data set that you used to analyse in traditional statistics course is low-dimensional: $n >> p$
- Lots of the data sets coming out of modern biological techniques are high-dimensional: $n \simeq p$ or $n << p$.
- This poses statistical challenges! For example Linear model no longer applies.

1.4.1 Low Dimensional



1.4.2 High Dimensional

1.5 Dimension Reduction

Dimension Reduction methods is a popular approach to solve the high dimensional situation ($p >> n$).

Dimension reduction techniques summarize X into a lower dimension matrix.

1.5.1 Why Dimension Reduction?

- Some features may be irrelevant
- We want to visualize high dimensional data
- High dimensional data often have high degrees of redundancy (correlation among features).

Dimension Reduction enables us to:

- Map the data into a new low-dimensional space, where important characteristics of the data are preserved.
- The new space often gives a (linear or non-linear) transformation of the original data.
- Visualization and analysis (clustering/prediction/...) is then performed in the new space.

1.6 Supervised Learning or Unsupervised Leraning

1.6.1 Supervised Learning

1.6.2 Unsupervised Learning

- no response variable
- In biological applications, often $p >> n$.

1.6.3 Unsupervised Learning Examples

- Do genes/samples in microarray expression data form interesting groups?
- Can individuals' SNP profiles be used to learn about their ethnic/racial backgrounds?
- Can we find cancer subtypes based on gene/metabolic expression patterns?
- What's the best way to visualize a high dimensional genomic data?
- How to find "interesting patterns" in the data?

2 Principal Component Analysis

Principal Component Analysis, or PCA, is a well-known and widely used technique applicable to a wide variety of applications such as dimensionality reduction, data compression, feature extraction, and visualization. The basic idea is to project a dataset from many correlated coordinates onto fewer uncorrelated coordinates called principal components while still retaining most of the variability present in the data (Jolliffe 2005).

As an example, principal component analysis is commonly performed to account for population stratification in genome wide association study.

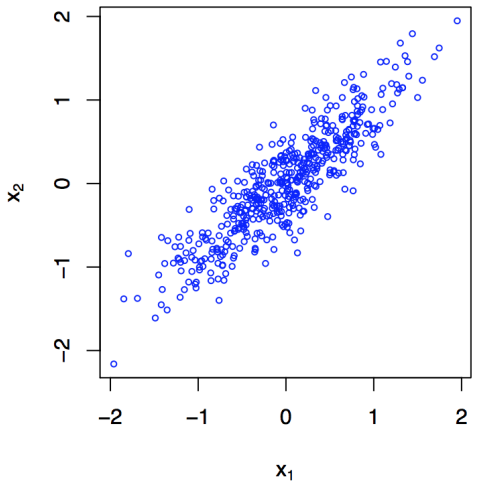
2.1 Definition: Principal Component Analysis (PCA)

- Data: n observations living in a p-dimensional space.
- Not all p dimensions are equally useful, especially when $p >> n$.
- Many are either completely redundant (correlated features) or uninformative (noise features).
- Need low-dimensional representation of the variables that captures most of the "information" in the data.

- To maximize the information retained, we need to minimize the redundancy, and to do this, we look for low-dimensional representations that capture most of the variation in the data.

2.2 PCA: The main Idea

What is a good 1-dim representation of the data?



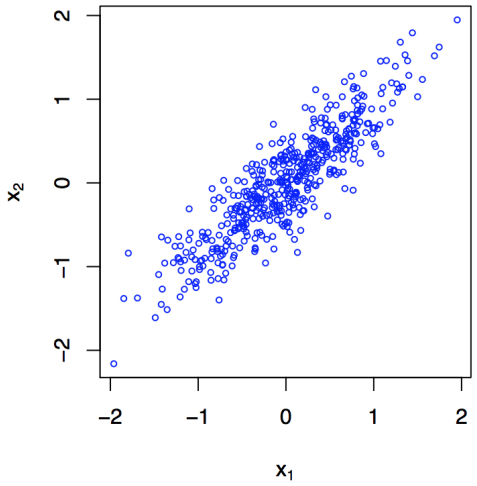
Use a linear combination of the variables; i.e. a weighted average of the variables.

$$\boldsymbol{c}_1 = w_1\boldsymbol{x}_1 + w_2\boldsymbol{x}_2$$

What is a good choice for the weights w_1 and w_2?

2.3 The Criterion for Principal Components

In PCA, we try to find the direction with maximum variance

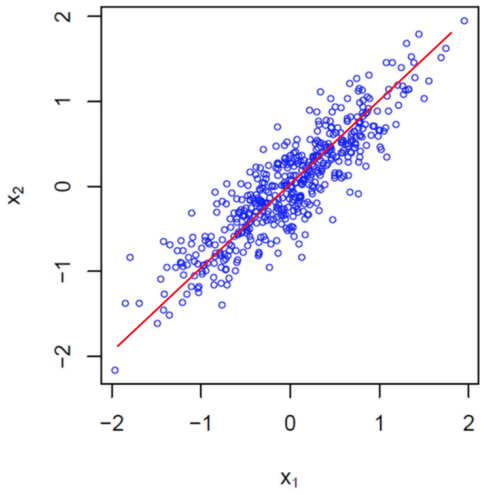


Formally, we seek the vector of weights $\boldsymbol{v} = (w_1, w_2)^T$ using the criterion

$$\max_{||\boldsymbol{v}||=1} Var(X\boldsymbol{v})$$

2.4 The 1-Dimensional PCA Solution

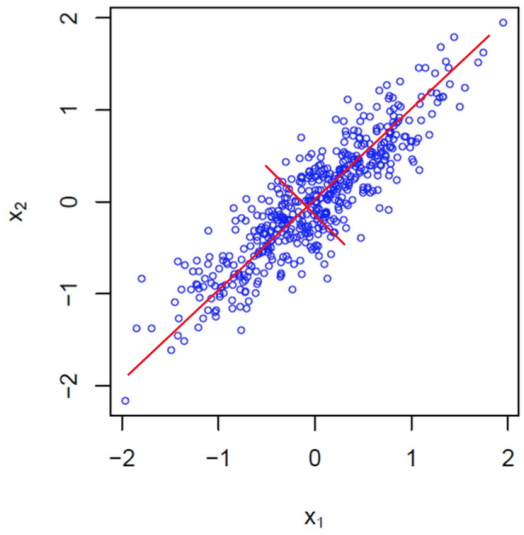
The interesting direction according to the PCA criterion is the one that captures the majority of the variance in the data.



- But, what if we need another direction.
- A systematic way to find additional principal components (PC's), is to choose subsequent linear combinations orthogonal/perpendicular to previous ones.
- This means that we want to choose v to be orthogonal to w, but to explain the majority of variability in the data.
- In the case of 2-dimensional data, there is only one choice!! This is always the case for the last PC.
- For $p > 2$, there are many orthogonal vectors to choose from, and we need to find the one that explains the maximum variation in the data, and is orthogonal to the first one.

2.5 The Full PCA Solution for 2 Dimensions

What is a good 1-dim representation of the data?



2.6 PCA: Summary

Seek the best directions in the data that account for most of the variability

→ principal components: artificial variables that are linear combinations of the original variables:

$$\begin{array}{ccc} \boldsymbol{c} = & \boldsymbol{X} & \boldsymbol{v} \\ (n) & (n \times p) & (p) \end{array}$$

- $\boldsymbol{c}$ is a linear combination of the elements of $\boldsymbol{X}$ having maximal variance
- $\boldsymbol{v}$ is called the associated loading vector
- For example, we get the first principal component with max. variance

  $$\boldsymbol{c}_1 = w_1\boldsymbol{x}_1 + \ldots + w_p\boldsymbol{x}_p$$

  where $\boldsymbol{v}_1 = (w_1, \ldots, w_p)^T$.
- PCA $\Longrightarrow$ finds units vectors $v_1, \ldots, v_r$ that maximise the variance Xv under the constraint that v_{i+1} is orthogonal to $v_1, \ldots, v_i$
- Reduction dimension:
  p variables $\Longrightarrow$ r new variables $Xv_1, \ldots, Xv_r$ called Principal Components

The new PCs form a vectorial subspace of dimension $< p$

- Project the data on these new axes.

  → approximate representation of the data points in a lower dimensional space
- The first few PCs account for most of the variation in all the original variables

2.6.1 Cons:

- Interpretation difficult with very large number of variables
- Unsupervised approach

2.7 Objective function for PCA

Objective function:

$$\max_{||\boldsymbol{v_h}||=1} var(X_h \boldsymbol{v_h}), \qquad h = 1 \dots H$$

Several ways of solving PCA:

1. Eigenvalue problem: $S\boldsymbol{v} = \lambda \boldsymbol{v}$; $\boldsymbol{c} = X\boldsymbol{v}$

   $S =$ variance covariance matrix or correlation matrix if X is scaled

2. Singular Value Decomposition (SVD):

   $$X = UDV^T$$

   where the columns of $U(n \times r)$ and V $(p \times r)$ are orthonormal; $D = diag[d_1, \dots, d_r]$ such that $d_1 \geq d_2 \geq \dots \geq d_r > 0$ are the square root of the eigenvalue of $X^T X$.

2.7.1 Solution based on SVD Decomposition

- The column of $C = UD$ are the principal component;
- The columns of V are the corresponding loading.
- The variance of a principal component $\boldsymbol{c}_1$ is equal to its associated eigenvalue λ_1.
- The obtained eigenvalues λ_j are decreasing.

2.8 Singular Value Decomposition (SVD)

Singular Value Decomposition, or SVD, is a computational method often employed to calculate principal components for a dataset. Using SVD to perform PCA is efficient and numerically robust. (see https://intoli.com/blog/pca-and-svd/).

Let a matrix $M : p \times q$ of rank r:

$$M = U\Delta V^T = \sum_{l=1}^{r} \delta_l u_l v_l^T,$$

- $U = (u_l) : p \times p$ and $V = (v_l) : q \times q$ are two orthogonal matrices which contain the normalised left (resp. right) singular vectors
- $\Delta = \text{diag}(\delta_1, \dots, \delta_r, 0, \dots, 0)$: the ordered singular values $\delta_1 \geq \delta_2 \geq \dots \geq \delta_r > 0$.

2.9 Relation of PCA and SVD

Let the data matrix X be of $n \times p$ size, where n is the number of samples and p is the number of variables. Let us assume that it is centered, i.e. column means have been subtracted and are now equal to zero.

Then the $p \times p$ covariance matrix C is given by $C = X^T X/(n-1)$. It is a symmetric matrix and so it can be diagonalized:

$$C = VLV^T,$$

where V is a matrix of eigenvectors (each column is an eigenvector) and L is a diagonal matrix with eigenvalues λ_i in the decreasing order on the diagonal.

The eigenvectors are called principal axes or principal directions of the data.

Projections of the data on the principal axes are called principal components, also known as PC scores; these can be seen as new, transformed, variables.

The j-th principal component is given by j-th column of XV. The coordinates of the i-th data point in the new PC space are given by the i-th row of XV.

if we now perform singular value decomposition of X, we obtain a decomposition

$$X = U\Delta V^T,$$

From here one can easily see that

$$C = V\Delta^T U^T U\Delta V^T/(n-1) = V\frac{\Delta^2}{n-1}V^T,$$

meaning that right singular vectors V are principal directions and that singular values are related to the eigenvalues of covariance matrix via $\lambda_i = s_i^2/(n-1)$.

Principal components are given by $XV = U\Delta V^T V = U\Delta$.

2.10 Illustration PCA: GWAS

The dataset comes from the R package `bigstatr` (Privé et al. 2018). It is a subset of a Genome Wide Asssociation study.

The variables are Single Nucleotide Polymorphism (SNPs) coded 0,1 amd 2 represented the number of rare allele at different locus (https://ghr.nlm.nih.gov/primer/genomicresearch/snp).

Principal Components Analysis (PCA) is a popular tool that has been used to infer population structure in genetic data for several decades (see Patterson, Price, and Reich (2006))

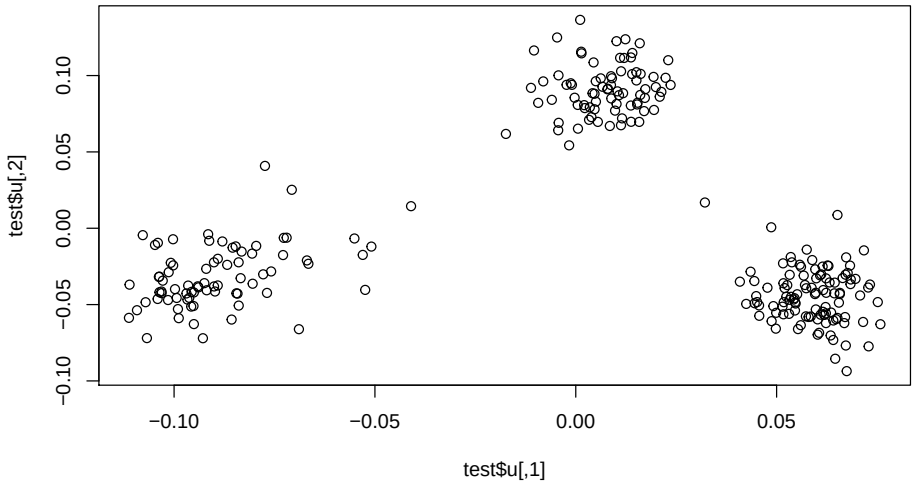
```
library(bigstatsr)
set.seed(1)

X <- big_attachExtdata()
n <- nrow(X)
X[1:2,1:3]
##      [,1] [,2] [,3]
## [1,]    2    2    0
## [2,]    1    2    1
dim(X)
## [1]  517 4542
```

We only use only half of the data

```
ind <- sort(sample(n, n/2))
test <- big_SVD(X, fun.scaling = big_scale(), ind.row = ind)
str(test)
## List of 5
##  $ d     : num [1:10] 178.5 114.5 91 87.1 86.3 ...
##  $ u     : num [1:258, 1:10] -0.1092 -0.0928 -0.0806 -0.0796 -0.1028 ...
##  $ v     : num [1:4542, 1:10] 0.00607 0.00739 0.02921 -0.01283 0.01473 ...
##  $ center: num [1:4542] 1.34 1.63 1.51 1.64 1.09 ...
##  $ scale : num [1:4542] 0.665 0.551 0.631 0.55 0.708 ...
##  - attr(*, "class")= chr "big_SVD"
```

we project the sample on a lower dimensional space to see any structure

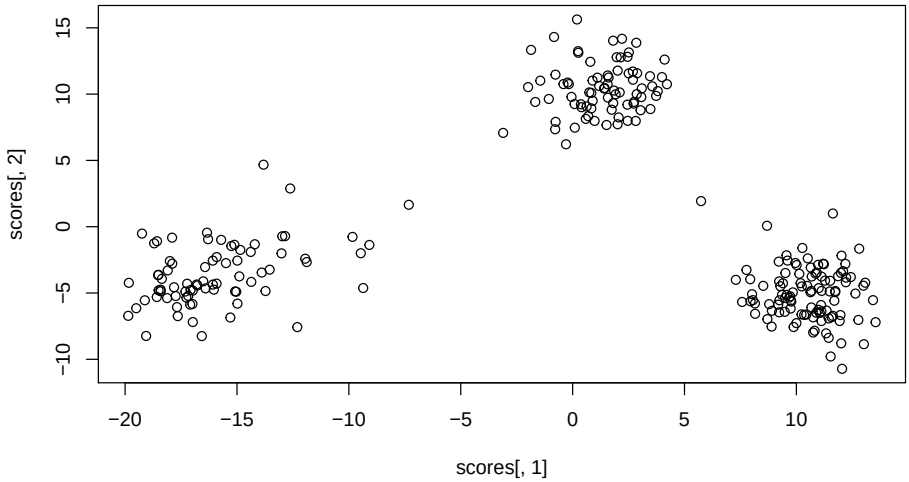
```
plot(test$u)
```




```
class(test)
## [1] "big_SVD"
```

- A more realistic projection based on the scores

```
scores <- test$u %*% diag(test$d)
plot(scores[,2]~scores[,1])
```

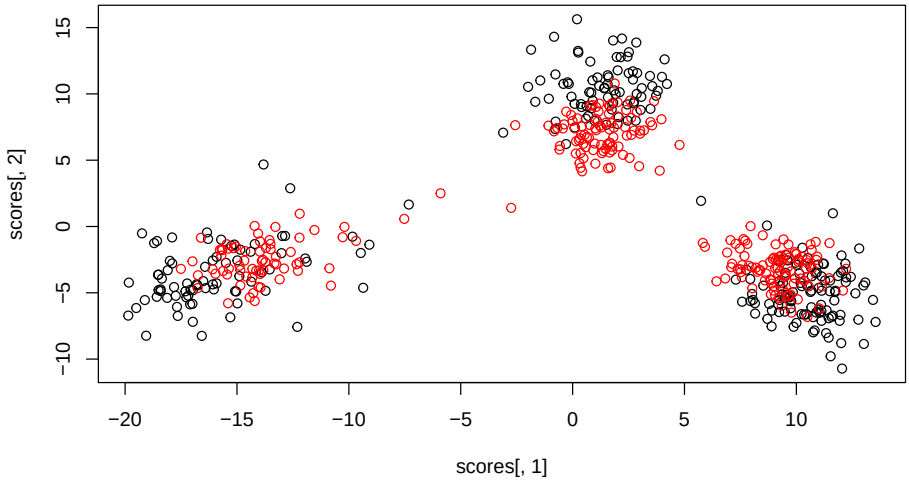


using the classical pca from R

```
pca <- prcomp(X[ind, ], center = TRUE, scale. = TRUE)
# same scaling
all.equal(test$center, pca$center)
## [1] TRUE
all.equal(test$scale,  pca$scale)
## [1] TRUE
```

projecting on new data

```
# projecting on new data
ind2 <- setdiff(rows_along(X), ind)
scores.test2 <- predict(test, X, ind.row = ind2)
plot(scores[,2]~scores[,1])
points(scores.test2[,2]~scores.test2[,1],col="red")
```



Using top PCs as covariates corrects for stratification in GWAS. For example, for case control studies one can use the following model to detect variants which are related to a disease:

$$\text{logit}(P(Y=1|SNP_j, PC_1, \ldots, PC_{20})) = \beta_0 + \beta_1 SNP_j + \gamma_1 PC_1 + \ldots + \gamma_{20} PC_{20}$$

2.11 SVD as a Compression/Dimension Reduction Tool

We start by reading an image and we perform SVDs on this image.

```
if (!"jpeg" %in% installed.packages()) install.packages("jpeg")
# Read image file into an array with three channels (Red-Green-Blue, RGB)
liquet <- jpeg::readJPEG("../../../WORKSHOP-ACEMS-2019/DATA_WORKSHOP/Lecture3/liquet.jpeg")
r <- liquet[, , 1] ; g <- liquet[, , 2] ; b <- liquet[, , 3]
# Performs full SVD of each channel
liquet.r.svd <- svd(r) ; liquet.g.svd <- svd(g) ; liquet.b.svd <- svd(b)
rgb.svds <- list(liquet.r.svd, liquet.g.svd, liquet.b.svd)
```

These two functions will be needed to display an image stored in an RGB array:

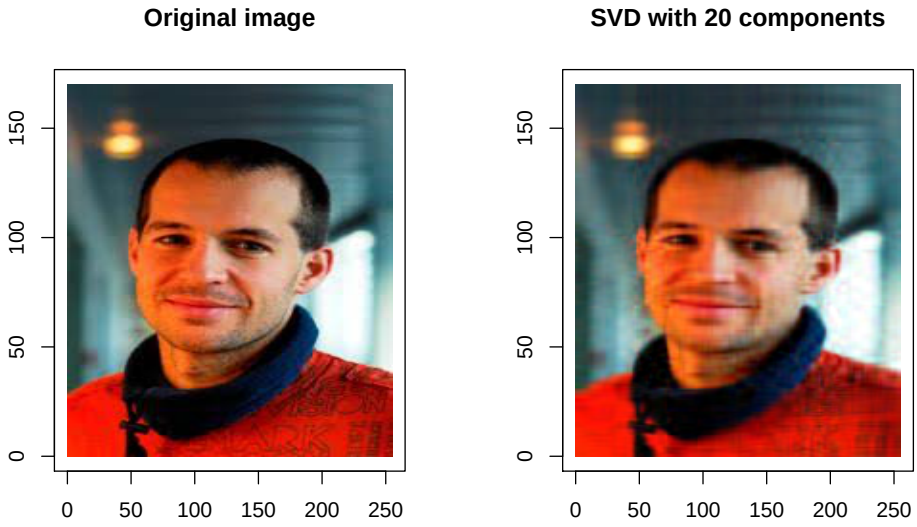
```
# Function to display an image stored in an RGB array

plot.image <- function(pic, main = "") {
 h <- dim(pic)[1] ; w <- dim(pic)[2]
 plot(x = c(0, h), y = c(0, w), type = "n", xlab = "", ylab = "", main = main)
 rasterImage(pic, 0, 0, h, w)
}
```

Function to compress an image via SVD of each channel

```


compress.image <- function(rgb.svds, nb.comp) {
 # nb.comp (number of components) should be less than min(dim(img[,,1])),
 # i.e., 170 here
 svd.lower.dim <- lapply(rgb.svds, function(i) list(d = i$d[1:nb.comp],
                                                    u = i$u[, 1:nb.comp],
                                                    v = i$v[, 1:nb.comp]))
 img <- sapply(svd.lower.dim, function(i) {
    img.compressed <- i$u %*% diag(i$d) %*% t(i$v)
  }, simplify = 'array')
 img[img < 0] <- 0
 img[img > 1] <- 1
 return(list(img = img, svd.reduced = svd.lower.dim))
}
```

Let's plot side-by-side the original and compressed images now.

```
par(mfrow = c(1, 2))
plot.image(liquet, "Original image")
p <- 20 ; plot.image(compress.image(rgb.svds, p)$img,
                     paste("SVD with", p, "components"))
```



As you can see, with 20 components (over 170 maximum), we can still recognize Benoit!

How much compression did we achieve with 20 components?

```
object.size(rgb.svds) # Original image
## 1740920 bytes
object.size(compress.image(rgb.svds, p)$svd.reduced) # Compressed image
## 207320 bytes
```

2.12 Case Study: The `liver.toxicity` study

I would like to thanks mixOmics team (I contributed in the past) to their great work on Multivariate Analysis tools. Great materials are provided in the website http://mixomics.org. This case study has been well and the data are availaible in the `mixOmics` R packages

First, you have to install the R package `mixOmics` (Rohart et al. 2017) which is now availaible on Bioconductor

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("mixOmics", version = "3.8")
```

```
library(mixOmics)
```

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`.

2.12.1 Load the data

We first load the data from the package.

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

2.12.2 Quick start

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

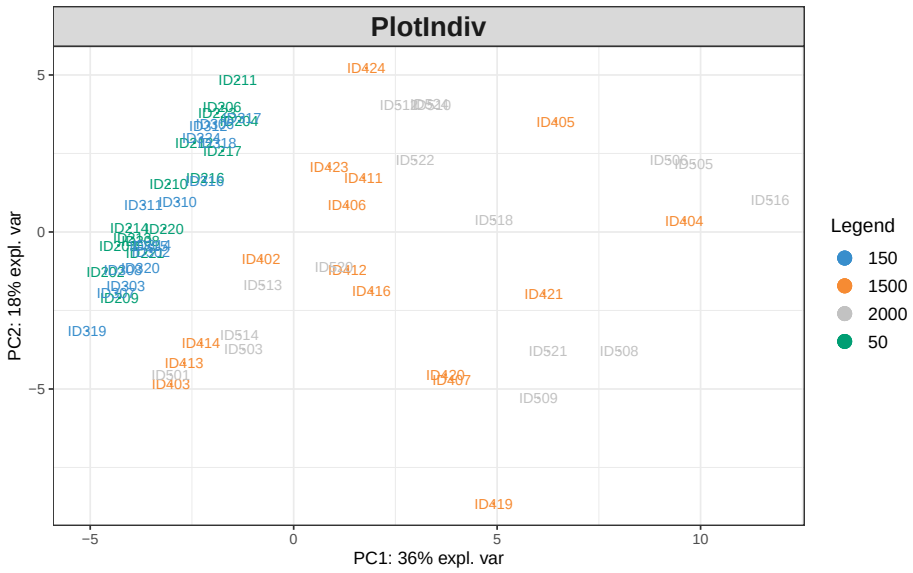
The two plots are not extremely meaningful as specific sample patterns should be further investigated and the variable correlation circle plot contains too many variables to be easily interpreted. Let's improve those graphics as shown below to improve interpretation.

2.12.3 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 corresponds to biological conditions.

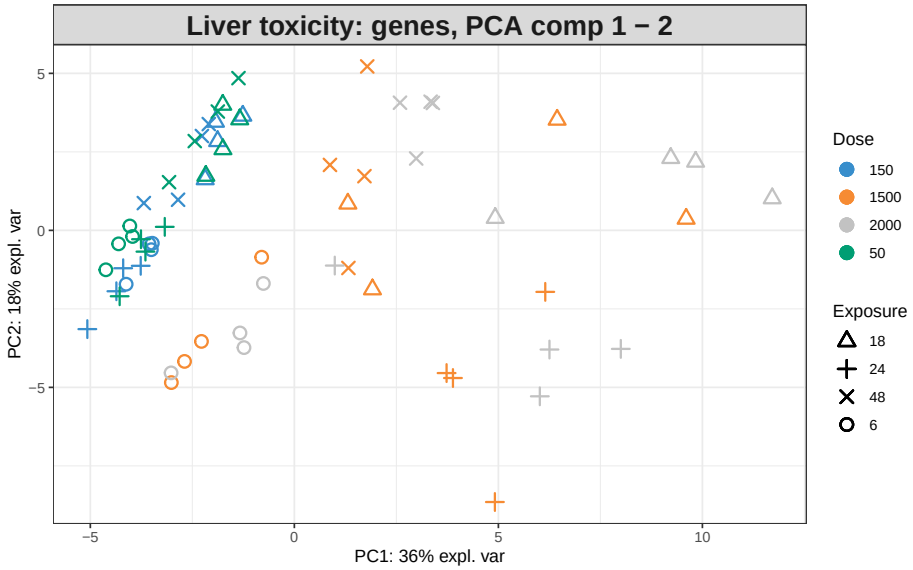
Here is an example where we include the sample groups information with the argument `group`:

```
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:

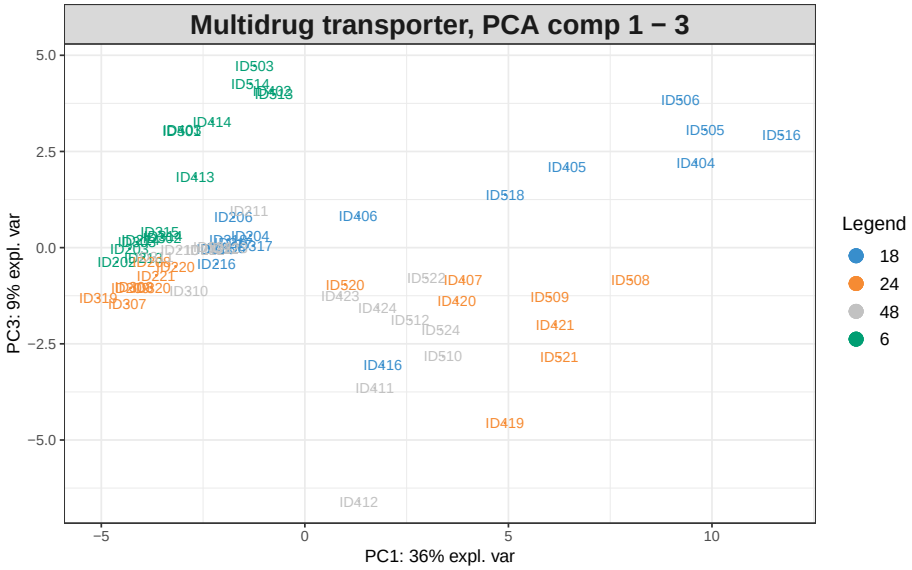
```
plotIndiv(MyResult.pca, ind.names = FALSE,
          group = liver.toxicity$treatment$Dose.Group,
          pch = as.factor(liver.toxicity$treatment$Time.Group),
          legend = TRUE, title = 'Liver toxicity: genes, PCA comp 1 - 2',
          legend.title = 'Dose', legend.title.pch = 'Exposure')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



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')
```

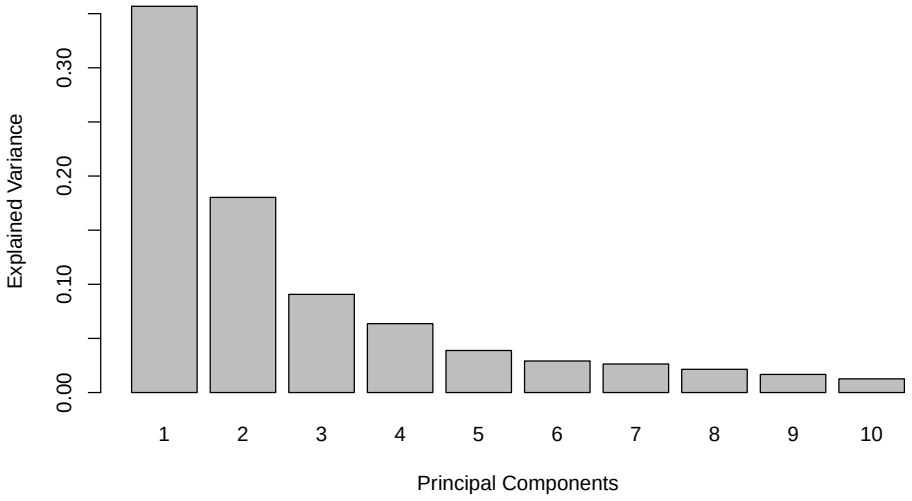


Here, the 3rd component on the y-axis clearly highlights a time of exposure effect.

2.12.4 Amount of variance explained and choice of 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.

```
MyResult.pca3 <- pca(X, ncomp = 10)
plot(MyResult.pca3)
```



```
MyResult.pca3
## Eigenvalues for the first 10 principal components, see object$sdev^2:
##        PC1        PC2        PC3        PC4        PC5        PC6        PC7
## 17.9714164  9.0792340  4.5677094  3.2043829  1.9567988  1.4686086  1.3281206
##        PC8        PC9       PC10
##  1.0820554  0.8434155  0.6373565
##
## Proportion of explained variance for the first 10 principal components, see object$explained_variance:
##        PC1        PC2        PC3        PC4        PC5        PC6        PC7
## 0.35684128 0.18027769 0.09069665 0.06362638 0.03885429 0.02916076 0.02637122
##        PC8        PC9       PC10
## 0.02148534 0.01674690 0.01265538
##
## Cumulative proportion explained variance for the first 10 principal components, see object$cum.var:
##       PC1       PC2       PC3       PC4       PC5       PC6       PC7       PC8
## 0.3568413 0.5371190 0.6278156 0.6914420 0.7302963 0.7594570 0.7858283 0.8073136
##       PC9      PC10
## 0.8240605 0.8367159
##
##  Other available components:
##  --------------------
##  loading vectors: see object$rotation
```

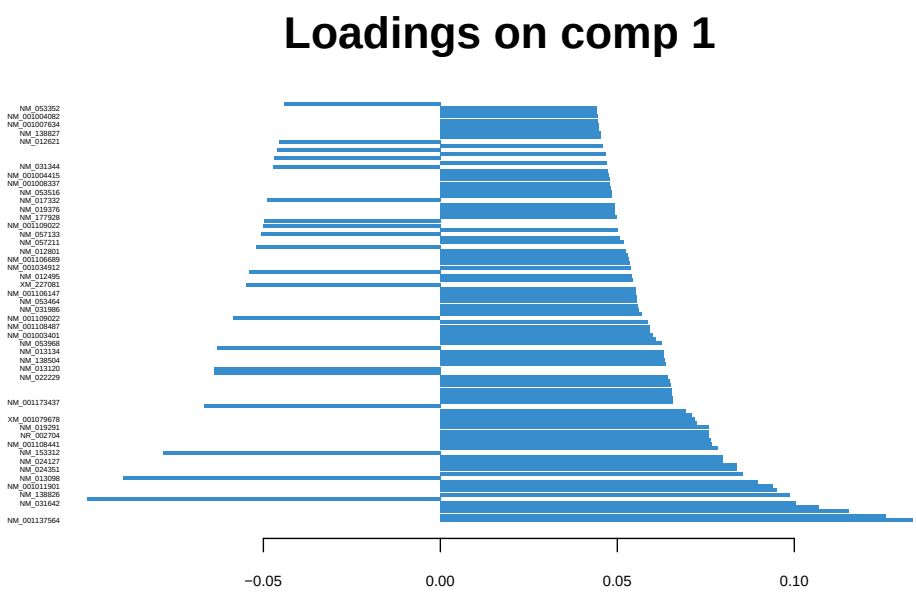
There are no clear guidelines on how many components should be included in PCA: it is data dependent and level of noise dependent. We often look at the 'elbow' on the screeplot above as an indicator that the addition of PCs does not drastically contribute to explain the remainder variance.

2.12.5 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))
```



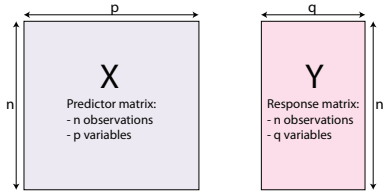
Such representation will be more informative once we select a few variables using an extension of PCA called Sparse PCA.

Plots can also be displayed in 3 dimensions using the option `style="3d"`, and interactively (we use the `rgl` package for this).

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

3 Partial Least Square (PLS)

PLS is a family of multivariate statistical techniques based on dimension reduction developed by S. Wold and H. Wold (1966, 1983). It can be seen as a supervised version of PCA.



3.1 Modelling Aims

Partial Least Square is also a Dimension reduction method with the two following aims:

- Symmetric relationship: analyse the shared information.
- Asymmetric relationship: X = predictors, and Y = response.

When Y is only one column (univariate case), PLS can be summarized as

- Dimension reduction method: p dimension space $\Rightarrow$ K dimension space ($K << p$)
- PLS looks the best components the most correlated to the response variable

- The PLS components are linear combinations of the variables

$$C^k = u_1 \times SNP_1 + u_2 \times SNP_2 + \ldots + u_p \times SNP_p$$

- It is a supervised approach

In more general case (Y multivariate $q > 1$):

- PLS finds pairs of latent (score) vectors $\xi = \mathbf{Xu}$, $\omega = \mathbf{Yv}$

$$\xi = u_1 \times \text{gene}_1 + u_2 \times \text{gene}_2 + \cdots + u_p \times \text{gene}_p$$
$$\omega = v_1 \times \text{pheno}_1 + v_2 \times \text{pheno}_2 + \cdots + v_p \times \text{pheno}_q$$

- Symmetric relationship. Analyse the shared information.
- Asymmetric relationship. There is a set of response and predictor variables that can be used for prediction.

3.2 Objective function:

$$\max_{||\boldsymbol{u}_h||=1, ||\boldsymbol{v}_h||=1} cov(X_h \boldsymbol{u}_h, Y_h \boldsymbol{v}_h) \qquad h = 1 \ldots H$$

Principle:

- Iterative procedure $\mapsto$ orthogonal component (latent variable $\boldsymbol{\xi}_h = X_h \boldsymbol{u}_h$).
- successive local regressions on the latent variables.
- X and Y are successively deflated.

3.3 Univariate case $Y \in \Re^n$

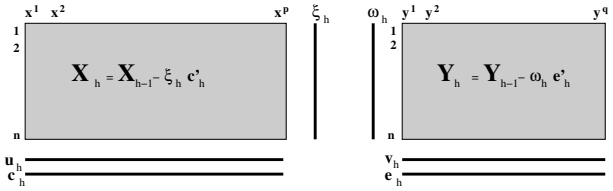
Univariate case $Y \in \Re^n$: step 1 :$\max_{||\boldsymbol{u}||=1} cov(X\boldsymbol{u}, Y)$

$$\begin{aligned} cov(X\boldsymbol{u}, Y) &= (X\boldsymbol{u})^T Y \\ &= < \boldsymbol{u}, X^T Y >= ||X^T Y|| cos(\boldsymbol{u}, X^T Y) \end{aligned}$$

$\hookrightarrow \boldsymbol{u} = \frac{X^T Y}{||X^T Y||}$

- Step 2: find a new linear combination no correlated to $\xi = X\boldsymbol{u}$ which explain the residuals $Y - d\xi$ where d is the regression of Y on $\boldsymbol{\xi} = X\boldsymbol{u}$
- Deflated step: $Y \leftarrow Y - d\xi$ and $X \leftarrow X - \boldsymbol{\xi}\boldsymbol{c}^T$ where $\boldsymbol{c}$ is the regression of X on $\boldsymbol{\xi} = X\boldsymbol{u}$;
- the columns of the new X are orthogonal (no correlated) to $X\boldsymbol{u}$.

3.4 Partial Least Squares: regression mode (multivariate case: Y $(n \times q)$)



For each iteration h, $h = 1..H$, decompose X and Y into:

1. Loadings vectors $\mathbf{u}_h$ and $\mathbf{v}_h$, p- and q- dimensional vectors
2. Latent variables $\boldsymbol{\xi}_h$ and $\boldsymbol{\omega}_h$, n-dimensional vectors
3. Regression of X_{h-1} and Y_{h-1} on $\boldsymbol{\xi}_h$ reg. coeff. $\mathbf{c}_h$ and $\mathbf{e}_h$
4. Residual matrices: deflation step of X_{h-1} and Y_{h-1}

3.5 Algorithm: regression mode

Objective function:

$$\max_{||\boldsymbol{u}_h||=1, ||\boldsymbol{v}_h||=1} cov(X_h\boldsymbol{u}_h, Y_h\boldsymbol{v}_h) \qquad h = 1 \ldots H$$

Start: set w to the first column of Y

1. $\boldsymbol{u} = \frac{X^T w}{w^T w}$, scale $\boldsymbol{u}$ to one. $\boldsymbol{u}$ is the loading vector associated to X
2. $\xi = X\boldsymbol{u}$ is the latent variable associated to X
3. $\boldsymbol{v} = \frac{Y^T \xi}{\xi^T \xi}$, scale $\boldsymbol{v}$ to one. $\boldsymbol{v}$ is the loading vector associated to Y
4. $w = Y\boldsymbol{v}$ is the latent variable associated to Y.
5. If convergence then 6 else 1
6. $\boldsymbol{c} = \frac{X^T \xi}{\xi^T \xi}$, $\boldsymbol{e} = \frac{Y^T \xi}{\xi^T \xi}$ are the partial regression coefficients from the regression of X (Y) onto ξ.
7. Deflation step: Compute the residual matrices $X \leftarrow X - \xi\boldsymbol{c}^T$ and $Y \leftarrow Y - \xi\boldsymbol{e}^T$

3.5.1 PLS family

PLS = Partial Least Squares or Projection to Latent Structures

Four main methods coexist in the literature:

(i) Partial Least Squares Correlation (PLSC) also called PLS-SVD;

(ii) PLS in mode A (PLS-W2A, for Wold's Two-Block, Mode A PLS);

(iii) PLS in mode B (PLS-W2B) also called Canonical Correlation Analysis (CCA);

(iv) Partial Least Squares Regression (PLSR, or PLS2).

- (i),(ii) and (iii) are symmetric while (iv) is asymmetric.
- Different objective functions to optimise.
- Good news: all use the singular value decomposition (SVD).

3.6 PLS connected to Singular Value Decomposition (SVD)

3.6.1 Reminder

Let a matrix $M : p \times q$ of rank r:

$$M = U\Delta V^T = \sum_{l=1}^{r} \delta_l u_l v_l^T,$$

- $U = (u_l) : p \times p$ and $V = (v_l) : q \times q$ are two orthogonal matrices which contain the normalised left (resp. right) singular vectors
- $\Delta = \mathrm{diag}(\delta_1, \ldots, \delta_r, 0, \ldots, 0)$: the ordered singular values $\delta_1 \geq \delta_2 \geq \cdots \geq \delta_r > 0$.

3.6.2 Connexion between SVD and maximum covariance

We were able to describe the optimization problem of the **four** PLS methods as:

$$(u^*, v^*) = \underset{\|u\|_2 = \|v\|_2 = 1}{\mathrm{argmax}} \; Cov(X_{h-1}u, Y_{h-1}v), \qquad h = 1, \ldots, H$$

Matrices X_h and Y_h are obtained recursively from X_{h-1} and Y_{h-1}.

The four methods differ by the deflation process, chosen so that the above scores or weight vectors satisfy given constraints.

The solution at step h is obtained by computing **only the first** triplet (δ_1, u_1, v_1) of singular elements of the SVD of $M_{h-1} = X_{h-1}^T Y_{h-1}$:

$$(u^*, v^*) = (u_1, v_1)$$

3.7 PLS in practice: 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, 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 PLS, we will integrate the gene expression levels (`gene`) with the concentrations of hepatic fatty acids (`lipid`).

3.7.1 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
```

3.7.2 Quick start

```
MyResult.pls <- pls(X,Y, ncomp=10)
plotIndiv(MyResult.pls)
plotVar(MyResult.pls)
```

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

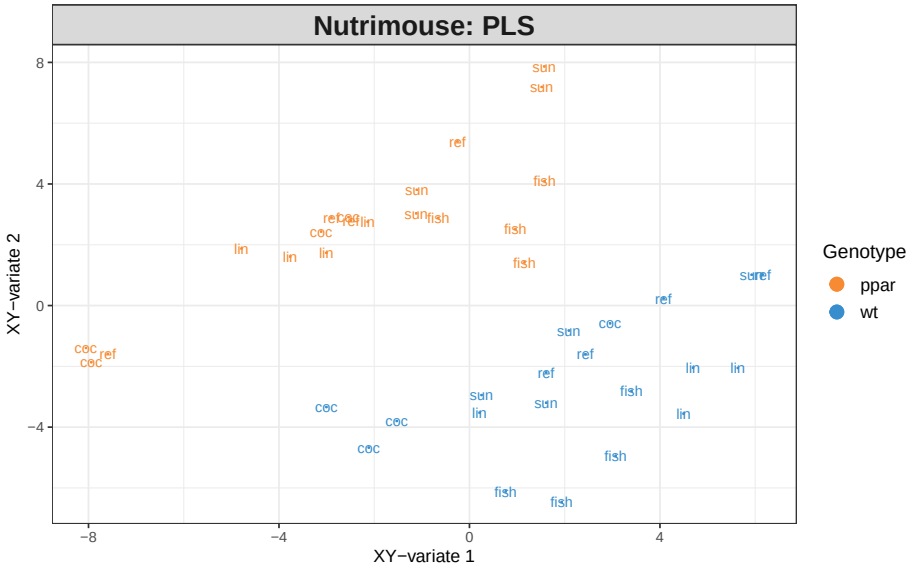
- `ncomp = 2`: the first two PLS components are calculated and are used for graphical outputs;
- `scale = TRUE`: data are scaled (variance = 1, strongly advised here);
- `mode = "regression"`: by default a PLS regression mode should be used (see **??** for more details) .

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 the space spanned by the X-components, or the Y-components.

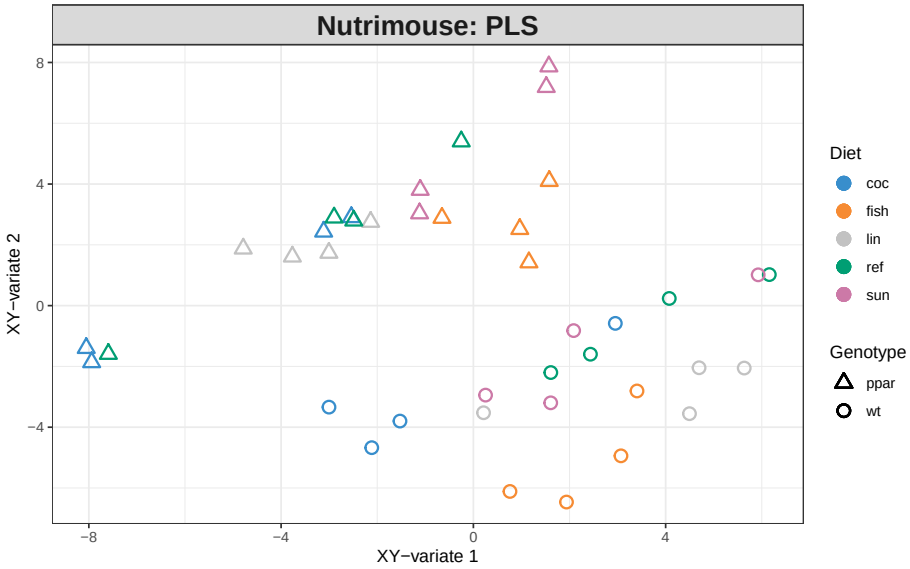
3.7.3 Customize sample plots

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 website. Here are two examples with colours indicating genotype or diet:

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



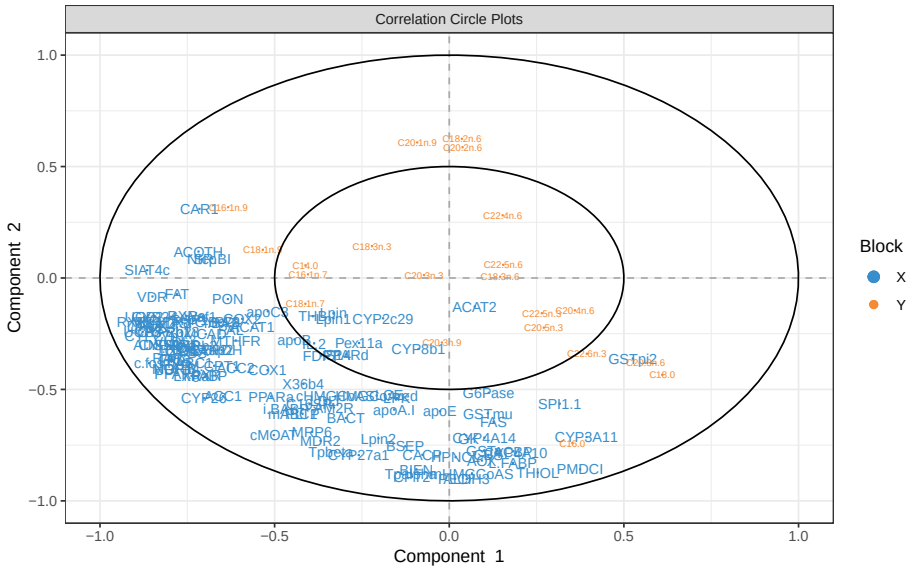

```
plotIndiv(MyResult.pls, group=nutrimouse$diet,
          pch = nutrimouse$genotype,
          rep.space = "XY-variate",  legend = TRUE,
          legend.title = 'Diet', legend.title.pch = 'Genotype',
          ind.names = FALSE,
          title = 'Nutrimouse: PLS')
```



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

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

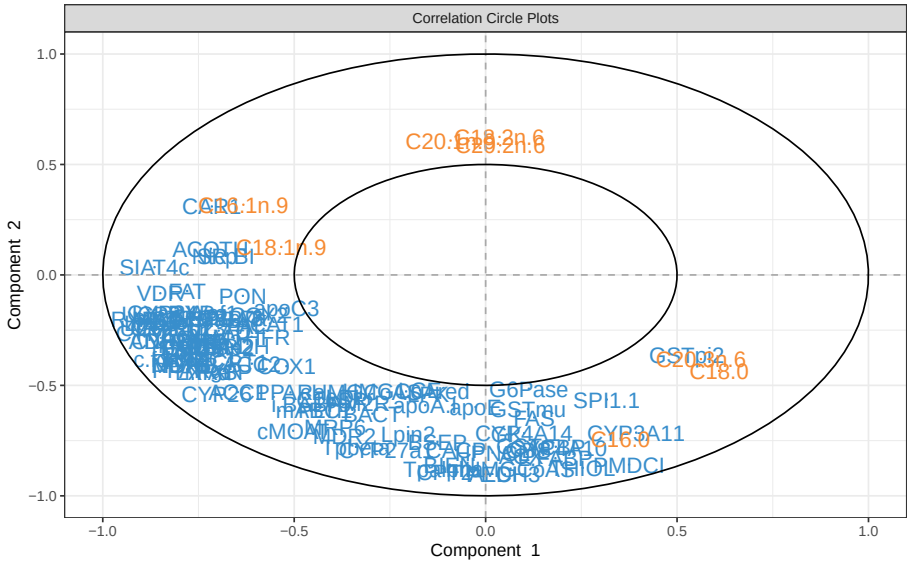



```
coordinates <- plotVar(MyResult.pls, plot = FALSE)
```

In this example, the figure is difficult to interpret and we would prefer to use a sparse vesrion of PLS to selecet the most important variable.

A cut-off can be set to display only the variables that mostly contribute to the definition of each component. Those variables should be located towards the circle of radius 1, far from the centre.

```
plotVar(MyResult.pls, cutoff=0.5)
```

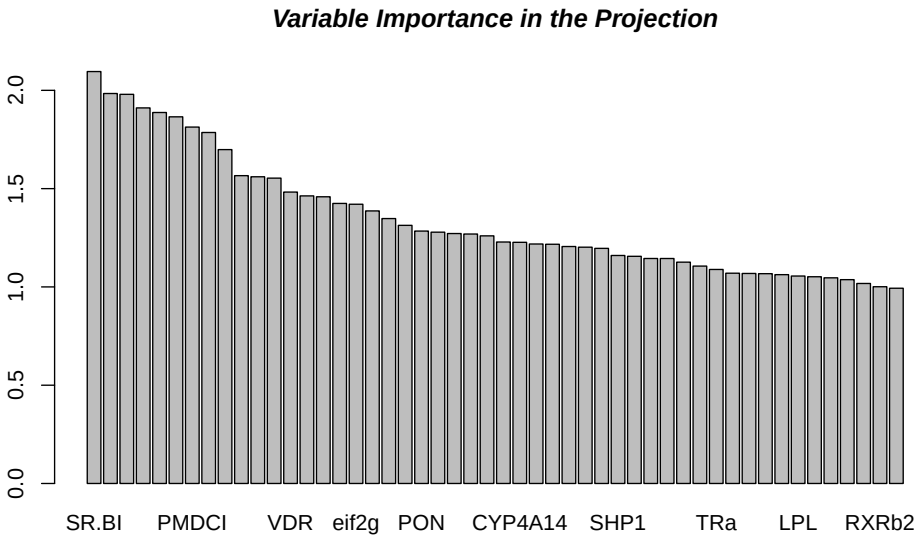


In this particular case, no variable selection was performed. Only the display was altered to show a subset of variables.

3.7.5 Variable Importance in the Projection (VIP)

Variable importance in projection (VIP) coefficients reflect the relative importance of each X variable for each X variate in the prediction model. VIP coefficients thus represent the importance of each X variable in fitting both the X- and Y-variates, since the Y-variates are predicted from the X-variates.

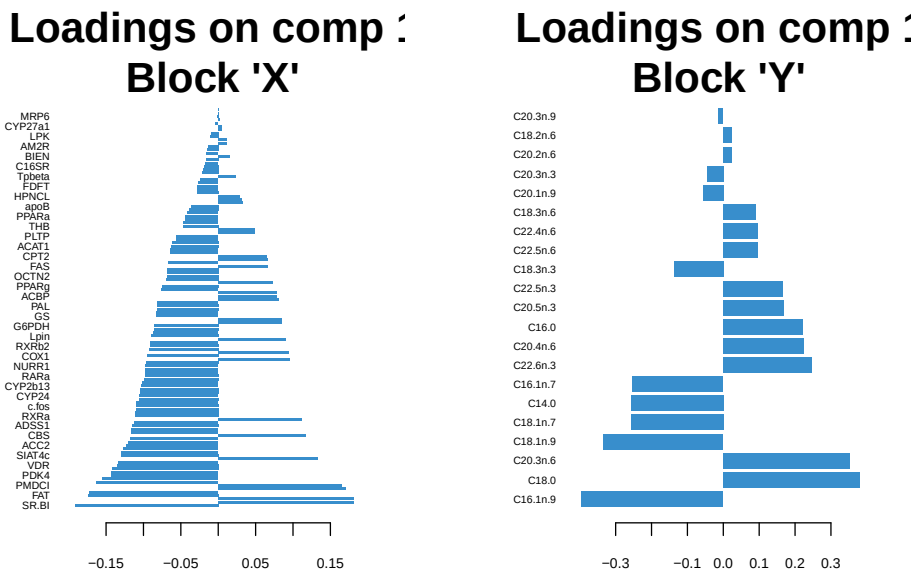
```
my.vip <- sort(vip(MyResult.pls)[,1],decreasing = TRUE)
barplot(my.vip[1:50],
beside = FALSE,
ylim = c(0, max(my.vip)), legend = rownames(my.vip)[1:50],
main = "Variable Importance in the Projection", font.main = 4)
```



3.7.6 Loading plots

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

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



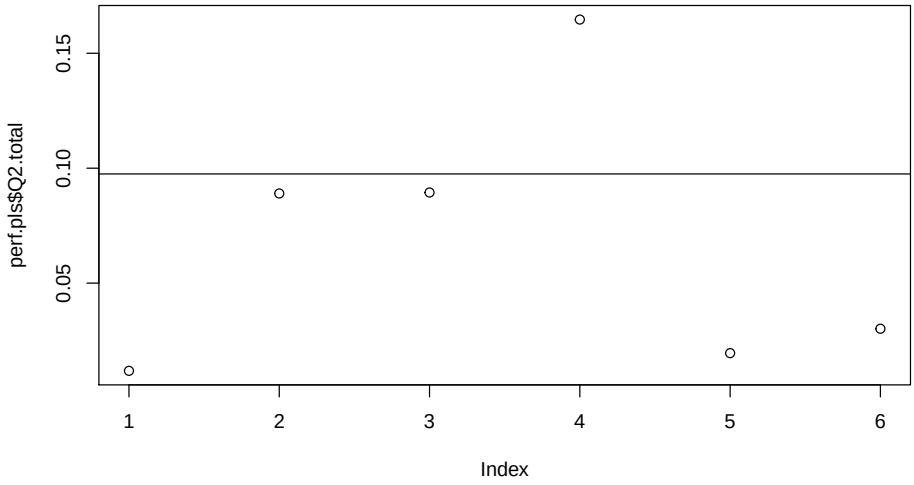
3.8 Tuning parameters and numerical outputs

For PLS, we have to choose the number of components to retain `ncomp`.

We use the `perf` function and repeated k-fold cross-validation to calculate the Q^2 criterion used in the SIMCA-P software. The rule of thumbs is that a PLS component should be included in the model if its value is ≤ 0.0975. Here we use 5-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).

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

```
MyResult.pls <- pls(X,Y, ncomp = 6)
set.seed(30) # for reproducbility
perf.pls <- perf(MyResult.pls, validation = "Mfold", folds = 5,
                 progressBar = FALSE, nrepeat = 10)
plot(perf.pls$Q2.total)
abline(h = 0.0975)
```

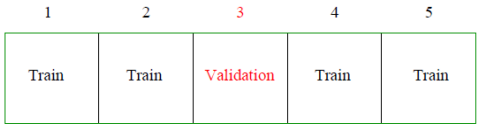


This example seems to indicate that up to 3 components could be enough. In a small $p+q$ setting we generally observe a Q^2 that decreases, but that is not the case here as $n << p+q$.

3.8.1 Reminder of Cross-Validation

- One idea is to split the data set into two fractions, then use one portion to fit the model and the other to evaluate how well the estimated modelpredicted the observations in the second portion.
- The problem with this solution is that we rarely have so much data that we can freely part with half of it solely for the purpose of choosing tuning parameters.
- To finesse this problem, cross-validation splits the data into K folds, fits the data on $K-1$ of the folds, and evaluates risk on the fold that was left out.

3.8.2 K-fold cross validation

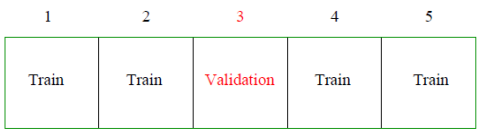


Let $k : 1, \ldots, N \to 1, \ldots, K$ the function indicating the partition to which observation i is allocated:

$$CV = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}^{-k(i)}(\boldsymbol{X}_i))^2$$

where $\hat{f}^{-k(i)}$ is the prediction of the subject i based on a model fitted with the $k(i)$th part of the data removed.

3.8.3 M-K-fold cross validation



Let $k : 1, \ldots, N \to 1, \ldots, K$ the function indicating the partition to which observation i is allocated:

$$CV = \frac{1}{M} \sum \left(\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{f}^{-k(i)}(\boldsymbol{X}_i))^2 \right)$$

where $\hat{f}^{-k(i)}$ is the prediction of the subject i based on a model fitted with the $k(i)$th part of the data removed.

3.8.4 Cross-validation: Leave One Out (loo)

- $\widehat{Y}_i$ is the prediction of the i-th observation obtained with the model fitted on all the observation.
- $\widehat{Y}_i^{(-i)}$ is the prediction of the i-th observation obtained with the model fitted on all the observation except the i-th observation.

The cross-validation approach compares predictions $\widehat{Y}_i^{(-i)}$ to observations Y_i.

3.8.5 Choice of the number of latent variables H using Q_H^2

Determine $\widehat{H}$ by cross-validation

For each $H = 1 \ldots n$:

1. Evaluate $\widehat{Y}_i^H$ and $\widehat{Y}_i^{H(-i)}$
2. Evaluate *Residual Sum of Squares* : $RSS_H = \sum_{i=1}^{n} \left(Y_i - \widehat{Y}_i^H \right)^2$
3. Evaluate *PRediction Error Sum of Squares* : $PRESS_H = \sum_{i=1}^{n} \left(Y_i - \widehat{Y}_i^{H(-i)} \right)^2$
4. Evaluate $Q_H^2 = 1 - \frac{PRESS_H}{RSS_{H-1}}$

4 Sparse Version of PCA and PLS

Both PCA and PLS approaches enable to perform dimension reduction by constructing H latent variables which are linear combination of all variables:

$$C_k = u_k^1 \times \boldsymbol{X}_1 + u_k^2 \times \boldsymbol{X}_2 + \ldots + u_k^p \times \boldsymbol{X}_p, \quad k = 1, \ldots, H$$

PCA and PLS do not provide a direct variable selection method.

4.1 Sparse Version

- sparse model select the relevant predictors
- Some coefficients u_k^l are equal to 0

$$C_k = u_k^1 \times \boldsymbol{X}_1 + \underbrace{u_k^2}_{=0} \times \boldsymbol{X}_2 + \underbrace{u_k^3}_{=0} \times \boldsymbol{X}_3 + \ldots + u_k^p \times \boldsymbol{X}_p$$

- Both sparse PCA and sparse PLS components are linear combinations of the selected variables

→ use SVD and low rank approximation to include penalization on the loading vector.

4.2 Intuition of sparse PCA and sparse PLS

4.2.1 SVD properties

Eckart-Young (1936) states that the (truncated) SVD of a given matrix M (of rank r) provides the best reconstitution (in a least squares sense) of M by a matrix with a lower rank k:

$$\min_{A \text{ of rank } k} \|M - A\|_F^2 = \left\| M - \sum_{\ell=1}^{k} \delta_\ell u_\ell v_\ell^T \right\|_F^2 = \sum_{\ell=k+1}^{r} \delta_\ell^2.$$

If the minimum is searched for matrices A of rank 1, which are under the form $\tilde{u}\tilde{v}^T$ where $\tilde{u}$, $\tilde{v}$ are non-zero vectors, we obtain

$$\min_{\tilde{u},\tilde{v}} \left\|M - \tilde{u}\tilde{v}^T\right\|_F^2 = \sum_{\ell=2}^{r} \delta_\ell^2 = \left\|M - \delta_1 u_1 v_1^T\right\|_F^2.$$

Thus, solving

$$\underset{\tilde{u},\tilde{v}}{\operatorname{argmin}} \left\|M_{h-1} - \tilde{u}\tilde{v}^T\right\|_F^2$$

and norming the resulting vectors gives us u_1 and v_1. This is another approach to solve the PLS optimization problem.

4.2.2 Towards sparse PLS

- Shen and Huang (2008) connected the previous optimization problem (in a PCA context) to least square minimisation in regression:

$$\left\|M_{h-1} - \tilde{u}\tilde{v}^T\right\|_F^2 = \left\| \underbrace{\text{vec}(M_{h-1})}_{y} - \underbrace{(I_p \otimes \tilde{u})\tilde{v}}_{X\beta} \right\|_2^2 = \left\| \underbrace{\text{vec}(M_{h-1})}_{y} - \underbrace{(v \otimes I_q)\tilde{u}}_{X\beta} \right\|_2^2.$$

  ↪ Possible to use many existing variable selection techniques using regularization penalties.

We propose iterative **alternating** algorithms to find normed vectors $\tilde{u}/\|\tilde{u}\|$ and $\tilde{v}/\|\tilde{v}\|$ that minimise the following penalised sum-of-squares criterion

$$\left\|M_{h-1} - \tilde{u}\tilde{v}^T\right\|_F^2 + P_\lambda(\tilde{u}, \tilde{v}),$$

for various penalization terms $P_\lambda(\tilde{u}, \tilde{v})$.

↪ We can obtain several sparse versions (in terms of the weights u and v).

4.3 Example sparse PLS

Sparse PLS solves:

$$\min_{\mathbf{u_h},\mathbf{v_h}} ||M_h - \mathbf{u}_h\mathbf{v}_h^T||_F^2 + \lambda_1^h \sum_{i=1}^{P} 2|u_i| + \lambda_2^h \sum_{i=1}^{Q} 2|v_i|, \qquad h = 1 \dots H$$

4.4 Choice of the sparsity: λ_1^h and λ_2^h

- k-fold cross validation or leave-one-out

  $\hookrightarrow$ RMSEP=Root Mean Squared Error Prediction

- For small samples (e.g $n \leq 100$) estimated prediction error might be biased

  $\hookrightarrow$ arbitrary choose the number of non-zero components in each loading vector u_h and v_h.

the biologist will also help choosing these parameters!

4.5 Sparse PLS in action

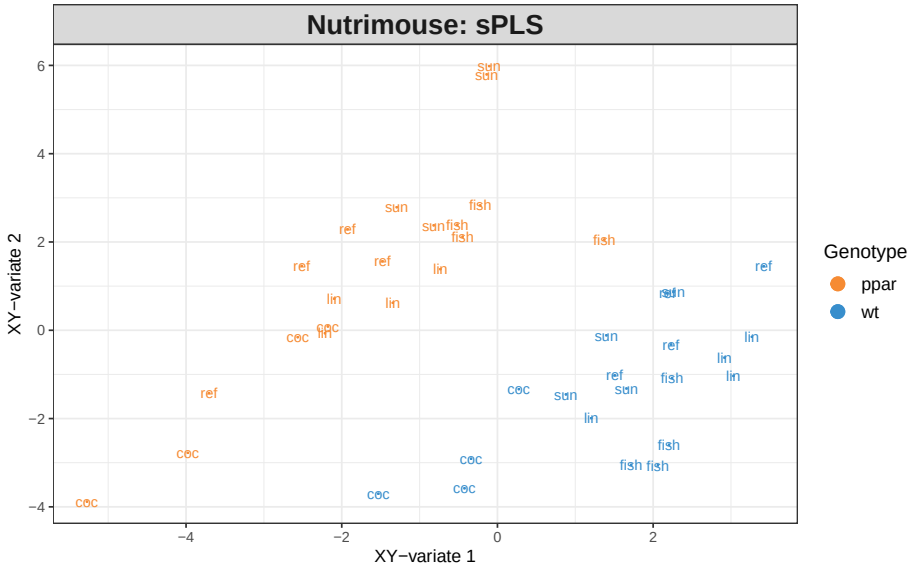
```
MyResult.spls <- spls(X,Y, keepX = c(25, 25), keepY = c(5,5))
plotIndiv(MyResult.spls)
plotVar(MyResult.spls)
```

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

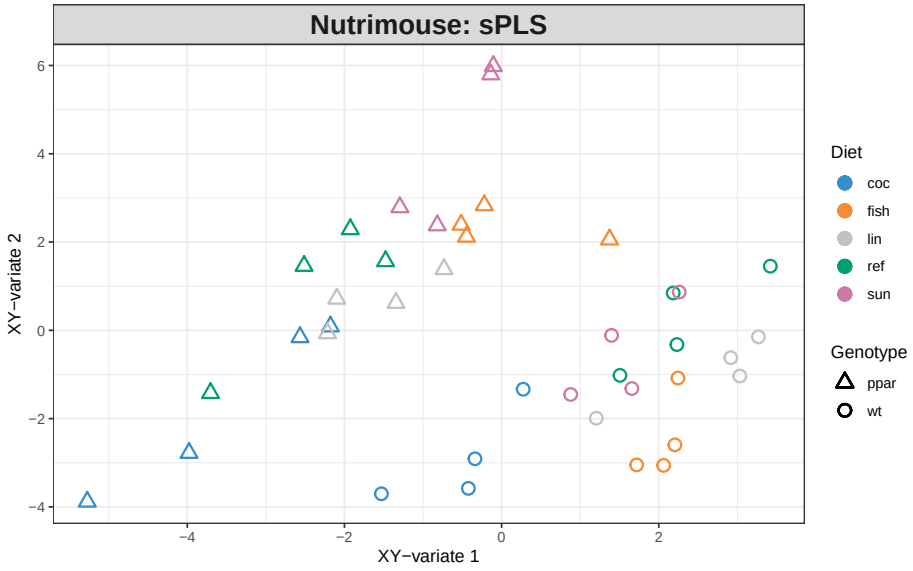
- `ncomp = 2`: the first two PLS components are calculated and are used for graphical outputs;
- `scale = TRUE`: data are scaled (variance = 1, strongly advised here);
- `mode = "regression"`: by default a PLS regression mode should be used

4.5.1 Customize sample plots

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

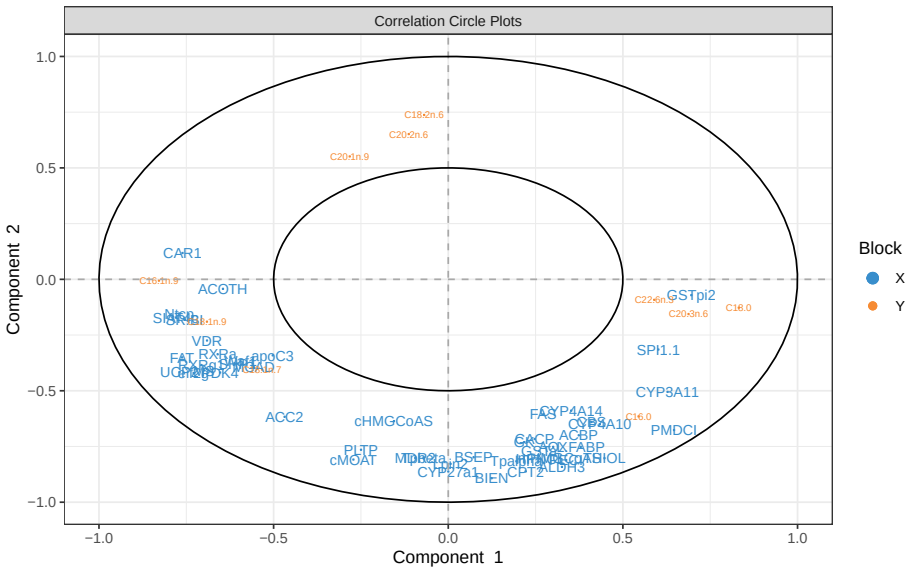



```
plotIndiv(MyResult.spls, group=nutrimouse$diet,
          pch = nutrimouse$genotype,
          rep.space = "XY-variate",  legend = TRUE,
          legend.title = 'Diet', legend.title.pch = 'Genotype',
          ind.names = FALSE,
          title = 'Nutrimouse: sPLS')
```



4.5.2 Customize variable plots

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




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

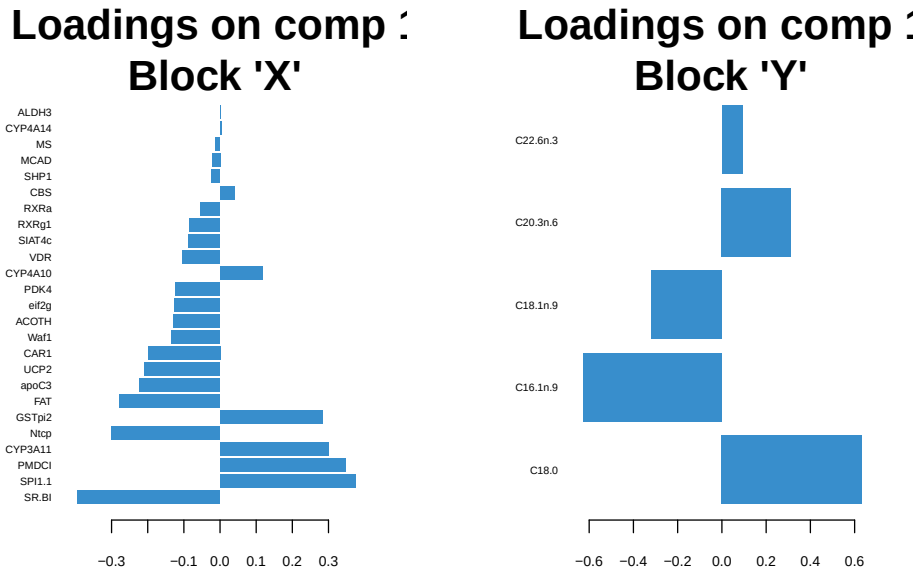
4.5.3 Variable selection outputs

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))
```

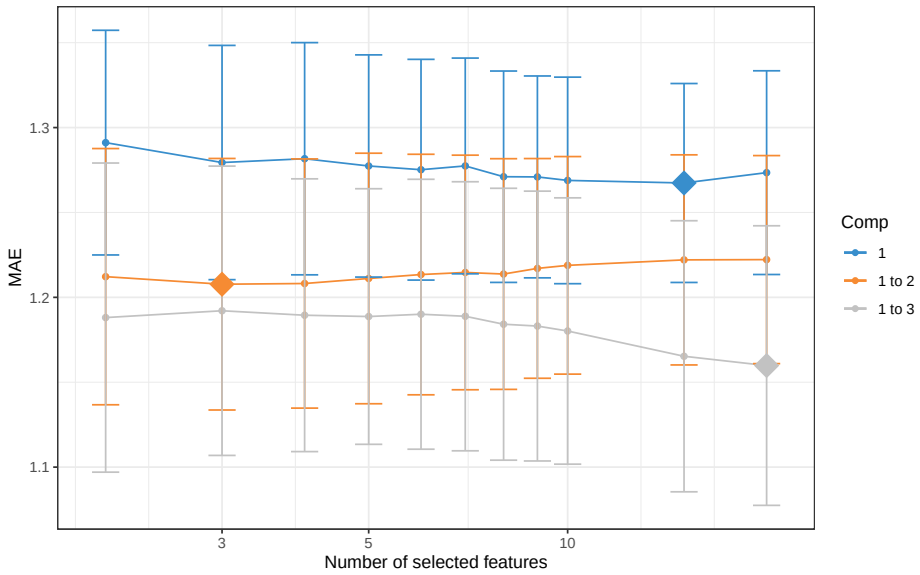


4.5.4 Tuning parameter and numerical outputs

The number of variables to select on each component and on each data set `keepX` and `keepY` have to be chosen.

These tuning parameters 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
```

To Tune `keepX` and `keepY` conjointly, one can tune one parameter then the other.

4.5.5 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)
cim(MyResult.spls, comp = 1, save = 'jpeg', name.save = 'PLScim')
```

4.5.6 Relevance networks

Using the same similarity matrix input in CIM, we can also represent relevance bipartite networks. Those networks only represent edges between on type of variable 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 an improved visualisation.

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

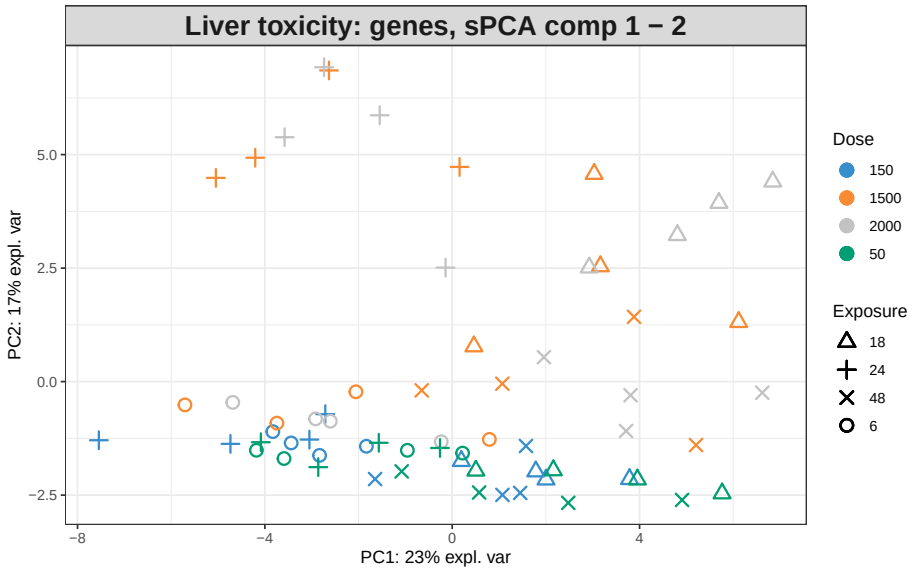
```
network(MyResult.spls, comp = 1, cutoff = 0.6, save = 'jpeg', name.save = 'PLSnetwork')
# save as graph object for cytoscape
myNetwork <- network(MyResult.spls, comp = 1)$gR
```

4.6 Sparse PCA in action

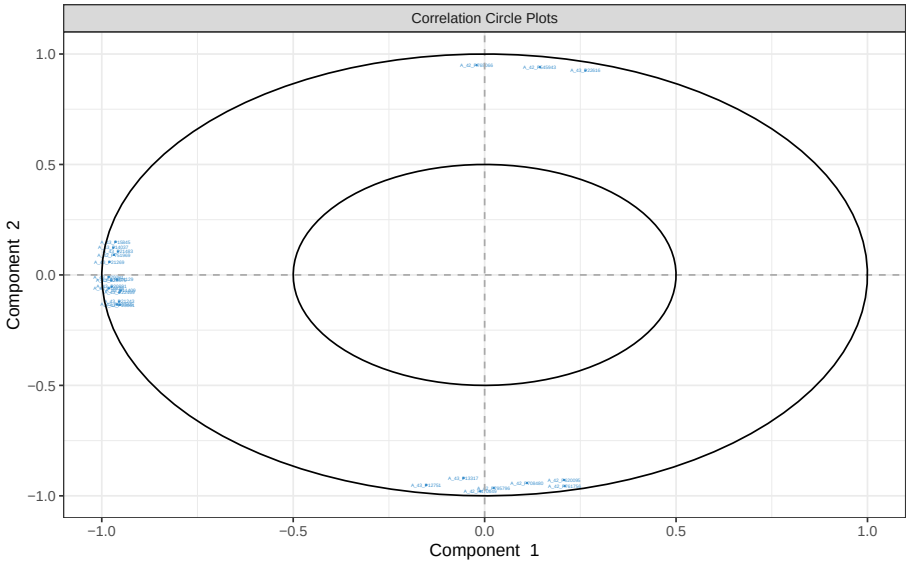
I would like to apply PCA but also be able to identify the key variables that contribute to the explanation of most variance in the data set.

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)`).

```
data(liver.toxicity)
X <- liver.toxicity$gene
MyResult.spca <- mixOmics::spca(X, ncomp = 3, keepX = c(15,10,5))                  # 1 Run the method
plotIndiv(MyResult.spca, group = liver.toxicity$treatment$Dose.Group,   # 2 Plot the samples
          pch = as.factor(liver.toxicity$treatment$Time.Group),
          legend = TRUE, title = 'Liver toxicity: genes, sPCA comp 1 - 2',
          legend.title = 'Dose', legend.title.pch = 'Exposure')
## Warning in shape.input.plotIndiv(object = object, n = n, blocks = blocks, :
## 'ind.names' is set to FALSE as 'pch' overrides it
```




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




```
# 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, see `?selectVar`:

```
selectVar(MyResult.spca, comp = 1)$value
## Error in selectVar(MyResult.spca, comp = 1): object 'MyResult.spca' not found
```

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)
## Error in plotLoadings(MyResult.spca): object 'MyResult.spca' not found
```

If we look at component two, 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):

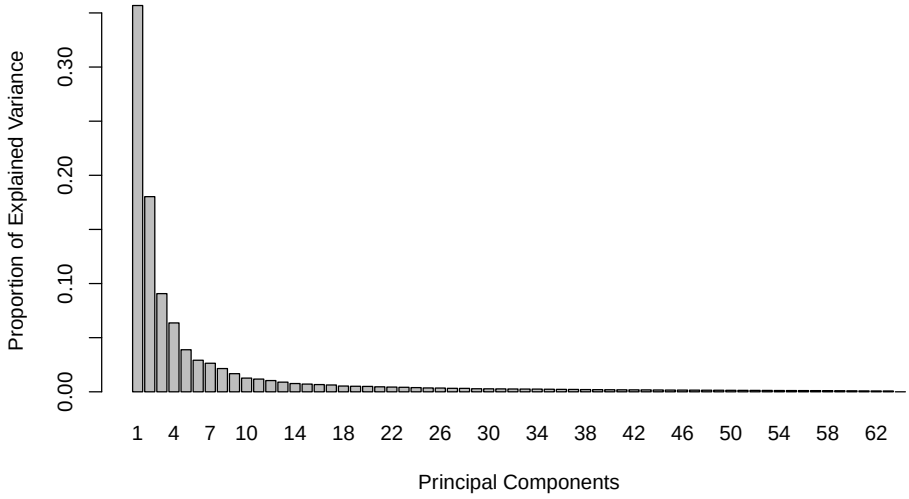
```
selectVar(MyResult.spca, comp=2)$value
## Error in selectVar(MyResult.spca, comp = 2): object 'MyResult.spca' not found
plotLoadings(MyResult.spca, comp = 2)
## Error in plotLoadings(MyResult.spca, comp = 2): object 'MyResult.spca' not found
```

4.7 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.



Regarding the number of variables to select in sparse PCA, there is not 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.

4.8 Other implementation of Sparse PCA

The R package `elasticnet` (Zou and Hastie 2018) provides the `spca` function to perform a sparse PCA model.

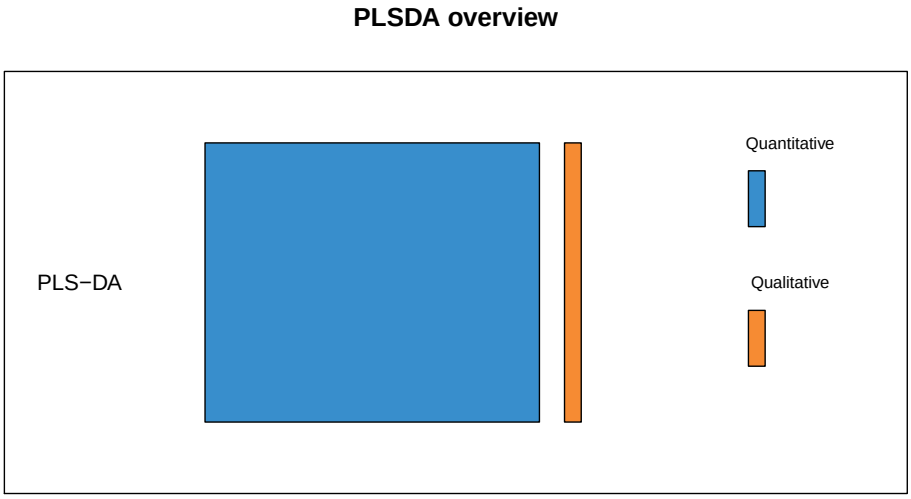
```
library(elasticnet)
```

However, the package does not provide a function to choose the number of variables in each component.

The R package `PMA` ((Witten et al. 2019)) provides a way to tune the number of variables in each component. You can explore the function `SPC.cv` for it.

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("impute", version = "3.8")
library(PMA)
?SPC.cv
```

5 PLS - Discriminant Analysis (PLS-DA) and Sparse PLS-DA



5.1 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.

5.2 The `srbct` study

The data are directly available in a processed and normalised format from the `mixOmics` 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.

5.3 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.

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 performs variable selection and classification in a one step procedure. sPLS-DA is a special case of sparse PLS described previously, where ℓ_1 penalization is applied on the loading vectors associated to the X data set.

We will mainly focus on sparse PLS-DA (see Lê Cao, Boitard, and Besse (2011)) 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)
## EWS  BL  NB RMS
##  23   8  12  20
dim(X); length(Y)
## [1]   63 2308
## [1] 63
```

5.3.1 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.

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

PLS-DA without variable selection can be performed as:

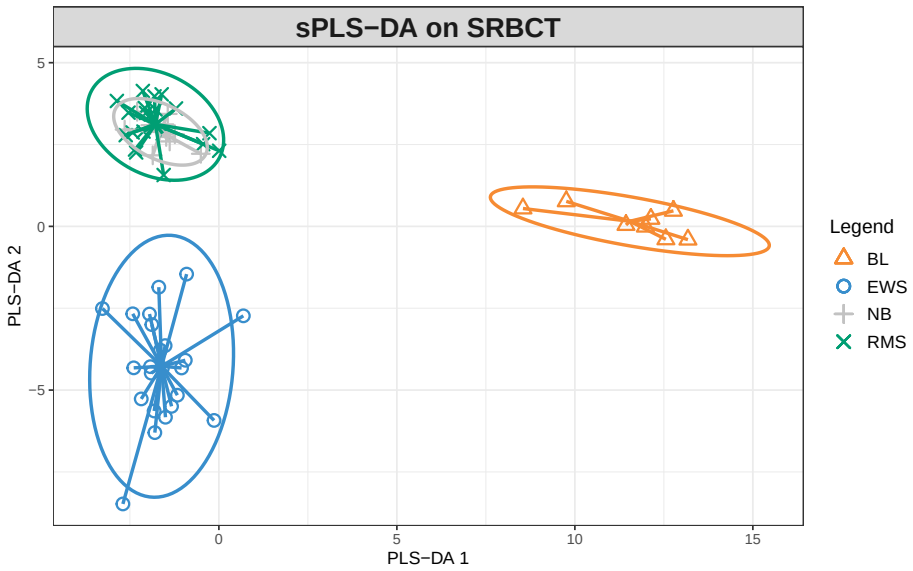
```
MyResult.plsda <- plsda(X,Y) # 1 Run the method
plotIndiv(MyResult.plsda)    # 2 Plot the samples
plotVar(MyResult.plsda)      # 3 Plot the variables
```

5.4 To go further

5.4.1 Customize 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.

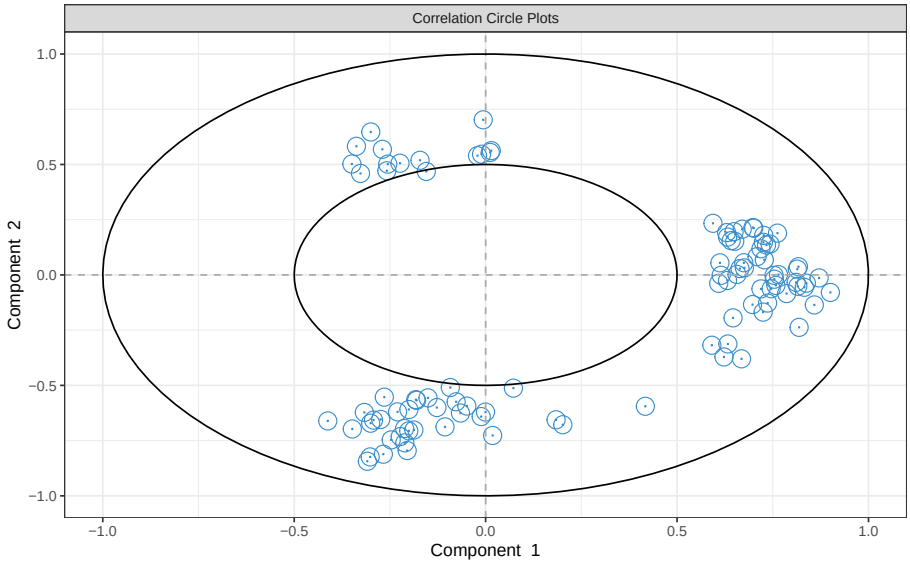
```
plotIndiv(MyResult.splsda, ind.names = FALSE, legend=TRUE,
          ellipse = TRUE, star = TRUE, title = 'sPLS-DA on SRBCT',
          X.label = 'PLS-DA 1', Y.label = 'PLS-DA 2')
```



5.4.2 Customize variable plots

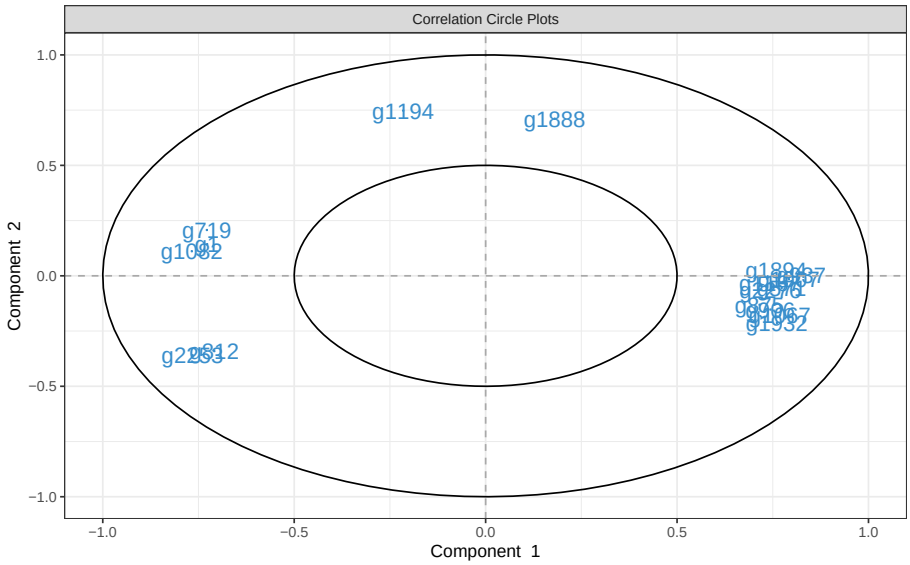
The name of the variables can be set to FALSE (`var.names=FALSE`):

```
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. Those variables should be located towards the circle of radius 1, far from the centre.

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

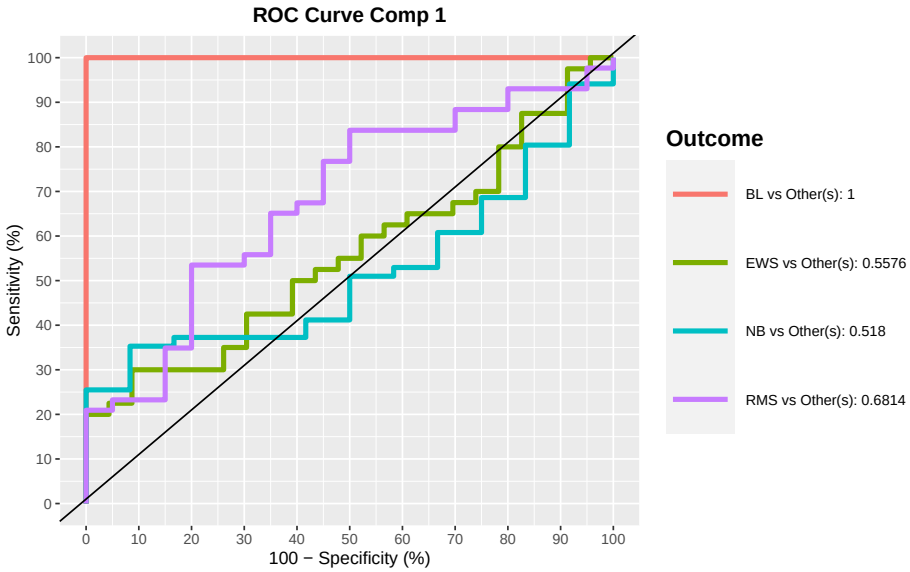


In this particular case, no variable selection was performed. Only the display was altered to show a subset of variables.

5.4.3 Other useful plots

5.4.3.1 ROC As PLS-DA acts as a classifier, we can plot a ROC Curve to complement the sPLS-DA classification performance results detailed latter. The AUC is calculated from training cross-validation sets and averaged.

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



5.4.4 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:

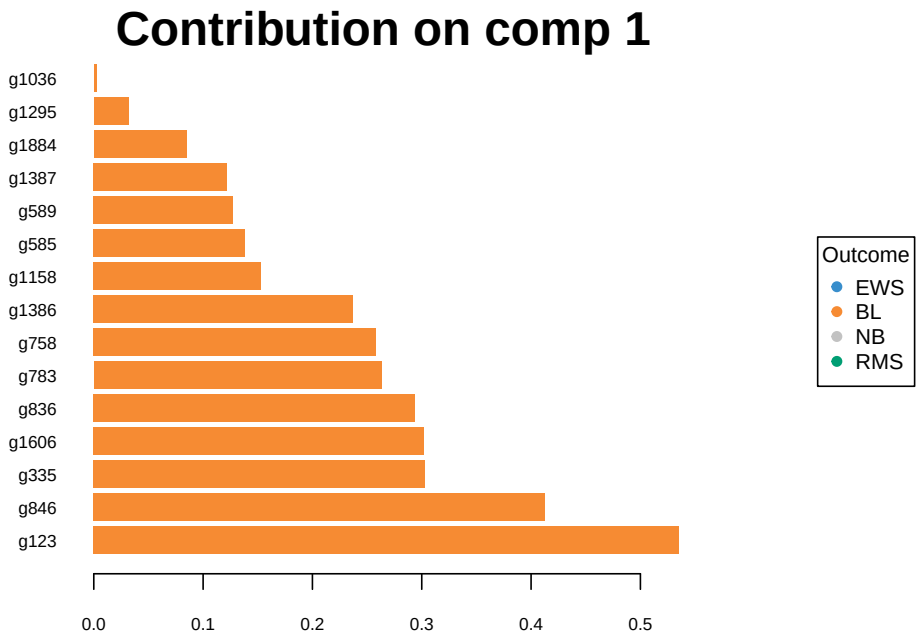
```
MyResult.splsda2 <- splsda(X,Y, ncomp=3, keepX=c(15,10,5))
```

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:

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

5.4.5 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.

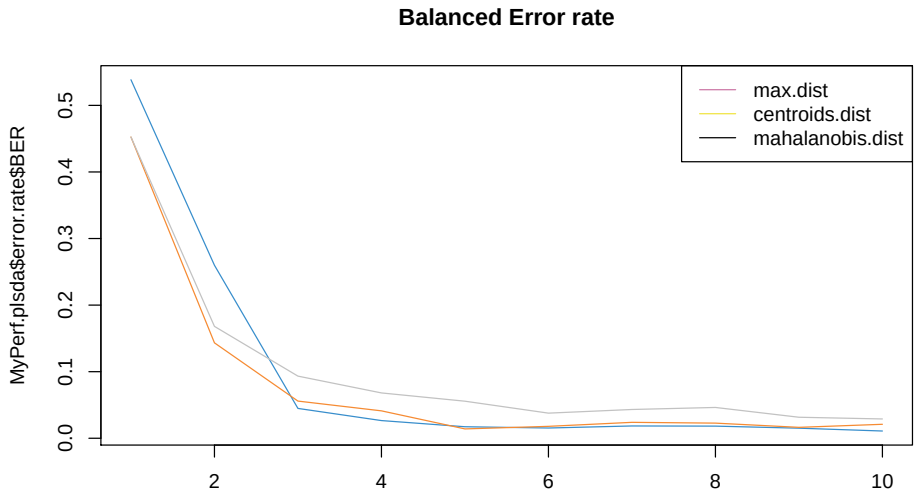
2 - The number of variables `keepX` to select on each component for sparse PLS-DA,

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 reproducibility in this vignette, otherwise increase nrepeat
MyPerf.plsda <- perf(MyResult.plsda2, validation = "Mfold", folds = 3,
                  progressBar = FALSE, nrepeat = 10) # we suggest nrepeat = 50
```

```
# type attributes(MyPerf.plsda) to see the different outputs
# slight bug in the output function currently see the quick fix below
#plot(MyPerf.plsda, col = color.mixo(5:7), sd = TRUE, legend.position = "horizontal")

# quick fix
matplot(MyPerf.plsda$error.rate$BER, type = 'l', lty = 1,
        col = color.mixo(1:3),
        main = 'Balanced Error rate')
legend('topright',
       c('max.dist', 'centroids.dist', 'mahalanobis.dist'),
       lty = 1,
       col = color.mixo(5:7))
```



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, progressBar = FA
##
##  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:
##  --------------------
```

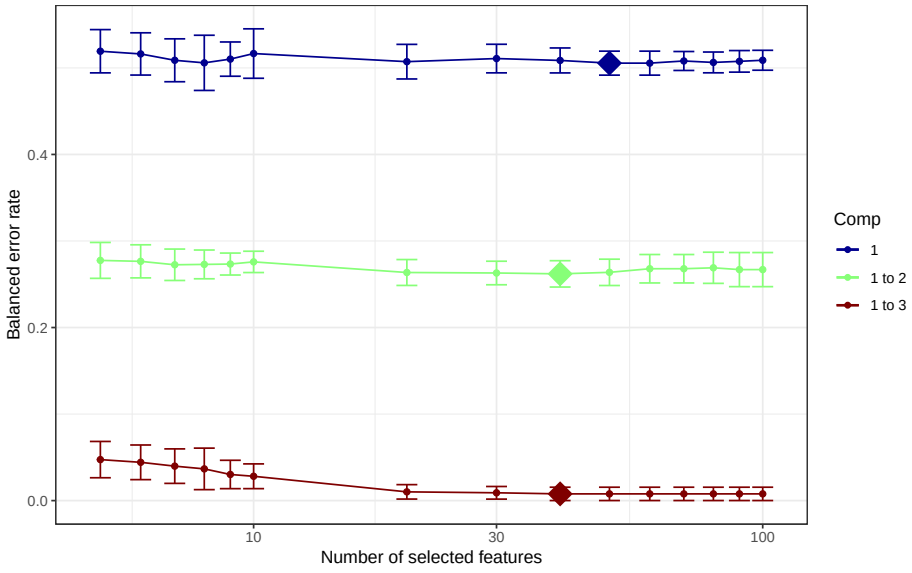
```
##  plot
```

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$ncomp # optimal number of components based on t-tests on the error ra
ncomp
## [1] 3
select.keepX <- tune.splsda.srbct$choice.keepX[1:ncomp]  # optimal number of variables to select
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.

6 Extension of Sparse PLS

The two following extenstions of Sparse PLS have been introduced by Liquet et al. (2015).

6.1 Incorporating Group structures within the data

- Natural example: Categorical variables which is a group of dummies variables in a regression setting.
- Genomics: genes within the same pathway have similar functions and act together in regulating a biological system.

↪ These genes can add up to have a larger effect

↪ can be detected as a group (i.e., at a pathway or gene set/module level).

We consider variables are divided into groups:

- {Example p: SNPs grouped into K genes} {

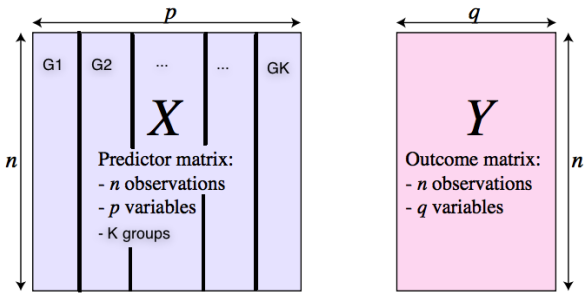
$$\mathbf{X} = [\underbrace{SNP_1, \ldots + SNP_k}_{gene_1} | \underbrace{SNP_{k+1}, SNP_{k+2}, \ldots, SNP_h}_{gene_2} | \ldots | \underbrace{SNP_{l+1}, \ldots, SNP_p}_{gene_K}]$$

  }

- Example p: genes grouped into K pathways/modules ($X_j = \text{gene}_j$)

$$\mathbf{X} = [\underbrace{X_1, X_2, \ldots, X_k}_{M_1} | \underbrace{X_{k+1}, X_{k+2}, \ldots, X_h}_{M_2} | \ldots | \underbrace{X_{l+1}, X_{l+2}, \ldots, X_p}_{M_K}]$$

6.2 Aims in regression setting:



- Select group variables taking into account the data structures; all the variables within a group are selected otherwise none of them are selected
- Combine both sparsity of groups and within each group; only relevant variables within a group are selected

6.3 Sparse Models

To understand the different sparse model, we consider here that genes (variables) are grouped into modules (pathways).

Aim: Select gene expressions.

- sparse PLS

$$\xi = u_1 \times X_1 + 0 \times X_2 + u_3 \times X_3 + \cdots + u_p \times X_p$$

Aim: Select groups of gene expressions.

- group PLS

$$\xi = \underbrace{u_1 \times X_1 + u_2 \times X_2}_{Module\ 1} + \underbrace{0 \times X_3 + 0 \times X_4}_{Module\ 2} + \cdots + \underbrace{u_{p-1} \times X_{p-1} + u_p \times X_p}_{Module\ k}$$

Aim: Select group and within-group gene expressions.

- sparse group PLS

$$\xi = \underbrace{u_1 \times X_1 + 0 \times X_2}_{Module\ 1} + \underbrace{0 \times X_3 + 0 \times X_4}_{Module\ 2} + \cdots + \underbrace{u_{p-1} \times X_{p-1} + u_p \times X_p}_{Module\ k}$$

6.4 Optimisation functions

6.4.1 Sparse PLS: sPLS

Optimisation of the weights

- X-score $\xi = \mathbf{Xu}$, Y-score $\omega = \mathbf{Yv}$

$$\underset{\mathbf{v}_h^\intercal \mathbf{v}_h \leq 1, \mathbf{u}_h^\intercal \mathbf{u}_h \leq 1}{argmax} \quad Cov(\mathbf{Xu}, \mathbf{Yv}) - \lambda_1 \|\mathbf{u}\|_1$$

- Sparse PLS

$$\xi = u_1 \times X_1 + 0 \times X_2 + u_3 \times X_3 + \cdots + u_p \times X_p$$

6.4.2 Sparse group PLS: gPLS

Optimisation of the weights

- X-score $\xi = \mathbf{Xu}$, Y-score $\omega = \mathbf{Yv}$

$$\underset{\mathbf{v}_h^\intercal \mathbf{v}_h \leq 1, \mathbf{u}_h^\intercal \mathbf{u}_h \leq 1}{argmax} \quad Cov(\mathbf{Xu}, \mathbf{Yv}) - \lambda_2 \sum_{k=1}^{K} \|\mathbf{u}^{(k)}\|_2$$

- Group PLS

$$\xi = \underbrace{0 \times X_1 + 0 \times X_2}_{Module\ 1} + \underbrace{0 \times X_3 + 0 \times X_4}_{Module\ 2} + \cdots + \underbrace{u_{p-1} \times X_{p-1} + u_p \times X_p}_{Module\ k}$$

6.4.3 Sparse Group PLS: sgPLS

Optimisation of the weights

- X-score $\xi = \mathbf{Xu}$, Y-score $\omega = \mathbf{Yv}$

$$\underset{\mathbf{v}_h^\intercal \mathbf{v}_h \leq 1, \mathbf{u}_h^\intercal \mathbf{u}_h \leq 1}{argmax} \quad Cov(\mathbf{Xu}, \mathbf{Yv}) - \lambda_1 \|\mathbf{u}\|_1 - \lambda_2 \sum_{k=1}^{K} \|\mathbf{u}^{(k)}\|_2$$

- Sparse Group PLS

$$\xi = \underbrace{u_1 \times X_1 + 0 \times X_2}_{Module\ 1} + \underbrace{0 \times X_3 + 0 \times X_4}_{Module\ 2} + \cdots + \underbrace{u_{p-1} \times X_{p-1} + u_p \times X_p}_{Module\ k}$$

6.5 sparse group subgroup PLS

Taking into account one more layer in the group structure:

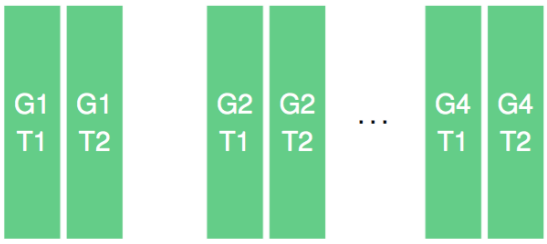
- Example: SNP $\subset$ Gene $\subset$ Pathways
- Longitudinal study

6.5.1 Longitudinal group structures:

This extension to longitudinal group structures has been developped by Sutton, Thiébaut, and Liquet (2018).

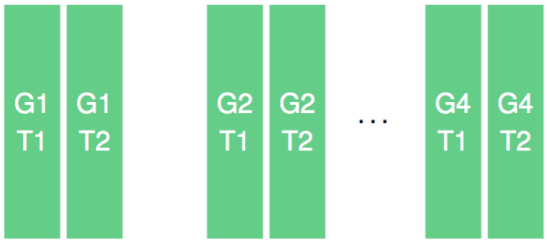
- Time index: genes within the same pathway at the same time index have similar functions in regulating a biological system.2

$$\mathbf{G1} = [\underbrace{\text{gene}_1, \ldots, \text{gene}_k}_{G1T1} \mid \underbrace{\text{gene}_1, \ldots, \text{gene}_k}_{G1T2}]$$



$$\mathbf{X} = [\underbrace{G1T1, G1T2}_{G1} \mid \underbrace{G2T1, G2T2}_{G2} \mid \cdots \mid \underbrace{G4T1, G4T2}_{G4}]$$

6.5.2 Aims:



- Identify important modules at a group level, important times at a subgroup level and single genes at an individual level.

6.5.3 sparse group subgroup PLS: sgsPLS

$$\xi = \overbrace{\overbrace{0 \times X_1}^{Time\ 1} + \overbrace{0 \times X_2}^{Time\ 2}}^{Module\ 1} + \cdots$$
$$+ \overbrace{\overbrace{u_{p-1} \times X_{p-1}}^{Time\ 1} + \overbrace{0 \times X_p}^{Time\ 2}}^{Module\ k}$$

Optimisation of the weights

- X-score $\xi_h = \mathbf{X}_{h-1}\mathbf{u}_h$, Y-score $\omega_h = \mathbf{Y}_{h-1}\mathbf{v}_h$

$$\max_{\mathbf{v}_h,\ \mathbf{u}_h} \ Cov(\mathbf{Xu}, \mathbf{Yv}) - \lambda_1 \sum_{k=1}^{K} \|\mathbf{u}^{(k)}\|_2 - \lambda_2 \sum_{k=1}^{K} \sum_{a=1}^{A_k} \|\mathbf{u}^{(k,a)}\|_2 - \lambda_3 \|\mathbf{u}\|_1$$

such that $\mathbf{v}_h^T \mathbf{v}_h \leq 1$ and $\mathbf{u}_h^T \mathbf{u}_h \leq 1$.

6.6 R Package

sgPLS ~ available on CRAN see Liquet, de Micheaux, and Broc (2017)

```
library(sgPLS)
example("gPLS")
```

sgsPLS Available now on GITHUB https://github.com/matt-sutton/sgspls

```
library(devtools)
install_github("matt-sutton/sgspls")
```

6.7 Big sgPLS

`bigsgPLS` is an R package that provides an implementation of the two block PLS methods. The method makes use of bigmemory and matrix algebra by chunks to deal with datasets too large for R.

A preliminary paper describing the PLS methods and some of the statistical properties is avaliable on ArXiv Pre-prints (Lafaye de Micheaux, Liquet, and Sutton 2017) https://arxiv.org/abs/1702.07066

```
library(devtools)
install_github("matt-sutton/bigsgPLS", host = "https://api.github.com")
```

An example of PLS on the EMNIST dataset is provided here https://github.com/matt-sutton/bigsgPLS/blob/master/Examples/Example-3-PLS.md

References

Barker, Matthew, and William Rayens. 2003. "Partial Least Squares for Discrimination." *Journal of Chemometrics* 17 (3): 166–73.

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

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

Jolliffe, Ian. 2005. *Principal Component Analysis*. Wiley Online Library.

Khan, Javed, Jun S Wei, Markus Ringner, Lao H Saal, Marc Ladanyi, Frank Westermann, Frank Berthold, et al. 2001. "Classification and Diagnostic Prediction of Cancers Using Gene Expression Profiling and Artificial Neural Networks." *Nature Medicine* 7 (6): 673–79.

Lafaye de Micheaux, Pierre, Benoit Liquet, and Matthew Sutton. 2017. "A Unified Parallel Algorithm for Regularized Group Pls Scalable to Big Data." *arXiv Preprint arXiv:1702.07066*.

Lê Cao, Kim-Anh, Simon Boitard, and Philippe Besse. 2011. "Sparse PLS Discriminant Analysis: Biologically Relevant Feature Selection and Graphical Displays for Multiclass Problems." *BMC Bioinformatics* 12 (1): 253.

Liquet, Benoit, Pierre Lafaye de Micheaux, and Camilo Broc. 2017. *SgPLS: Sparse Group Partial Least Square Methods*. https://CRAN.R-project.org/package=sgPLS.

Liquet, Benoit, Pierre Lafaye de Micheaux, Boris P Hejblum, and Rodolphe Thiébaut. 2015. "Group and Sparse Group Partial Least Square Approaches Applied in Genomics Context." *Bioinformatics* 32 (1): 35–42.

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

Patterson, Nick, Alkes L Price, and David Reich. 2006. "Population Structure and Eigenanalysis." *PLoS Genetics* 2 (12): e190.

Privé, Florian, Hugues Aschard, Andrey Ziyatdinov, and Michael GB Blum. 2018. "Efficient Analysis of Large-Scale Genome-Wide Data with Two R Packages: Bigstatsr and Bigsnpr." *Bioinformatics* 34 (16): 2781–7.

Rohart, Florian, Benoit Gautier, Amrit Singh, and Kim-Anh Lê Cao. 2017. "MixOmics: An R Package for 'Omics Feature Selection and Multiple Data Integration." *PLoS Computational Biology* 13 (11): e1005752.

Sutton, Matthew, Rodolphe Thiébaut, and Benoit Liquet. 2018. "Sparse Partial Least Squares with Group and Subgroup Structure." *Statistics in Medicine* 37 (23): 3338–56.

Tan, Y., L. Shi, W. Tong, GT Gene Hwang, and C. Wang. 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): 235–43.

Witten, Daniela, Rob Tibshirani, Sam Gross, and Balasubramanian Narasimhan. 2019. *PMA: Penalized Multivariate Analysis*. https://CRAN.R-project.org/package=PMA.

Zou, Hui, and Trevor Hastie. 2018. *Elasticnet: Elastic-Net for Sparse Estimation and Sparse Pca*. https://CRAN.R-project.org/package=elasticnet.