Sparse Version of PCA and PLS

B. Liquet

Introduction

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.

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

$\rightarrow$ use SVD and low rank approximation to include penalization on the loading vector.

Intuition of sparse PCA and sparse PLS

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 $\widetilde{u}\widetilde{v}^T$ where $\widetilde{u}$, $\widetilde{v}$ are non-zero vectors, we obtain

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

Intuition of sparse PCA and sparse PLS

Thus, solving

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

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

Towards sparse PLS

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

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

  $\hookrightarrow$ Possible to use many existing variable selection techniques using regularization penalties.

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

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

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

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

Example sparse PLS

Sparse PLS solves:

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

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 $n \leqslant 100$) 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!

Sparse PLS in action

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

```
[1]  40 120
```

```
[1] 40 21
```

Sparse PLS in action

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

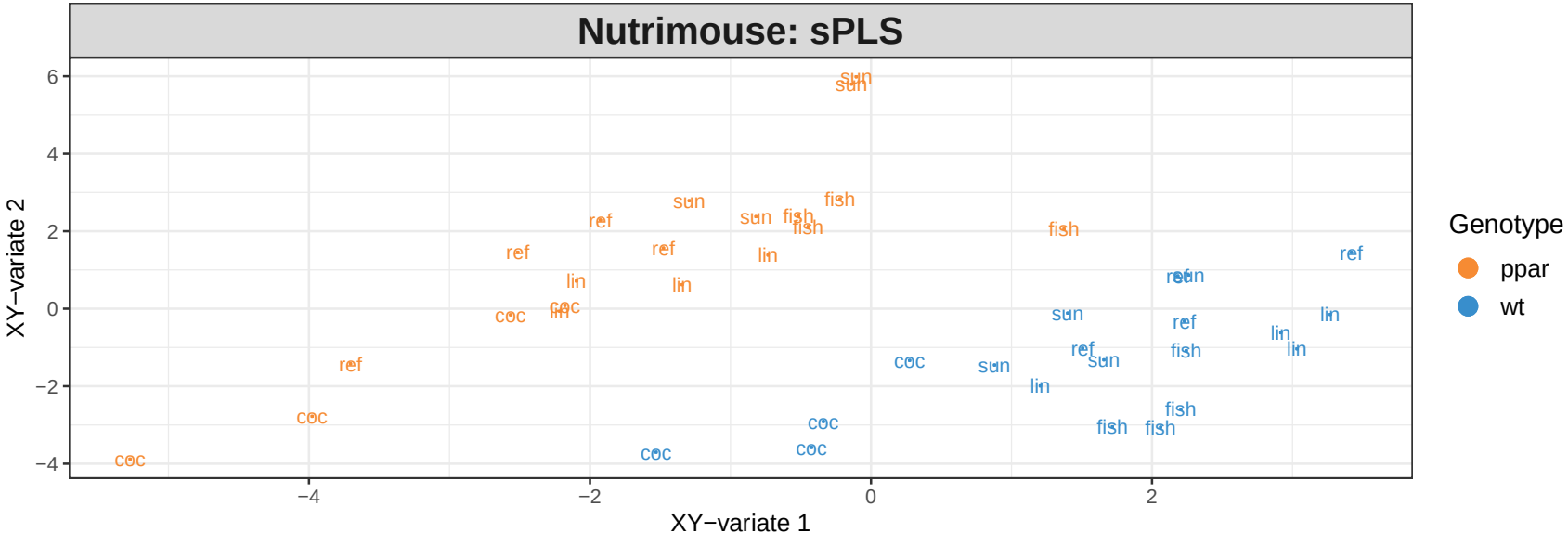
```
plotVar(MyResult.spls)
```

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

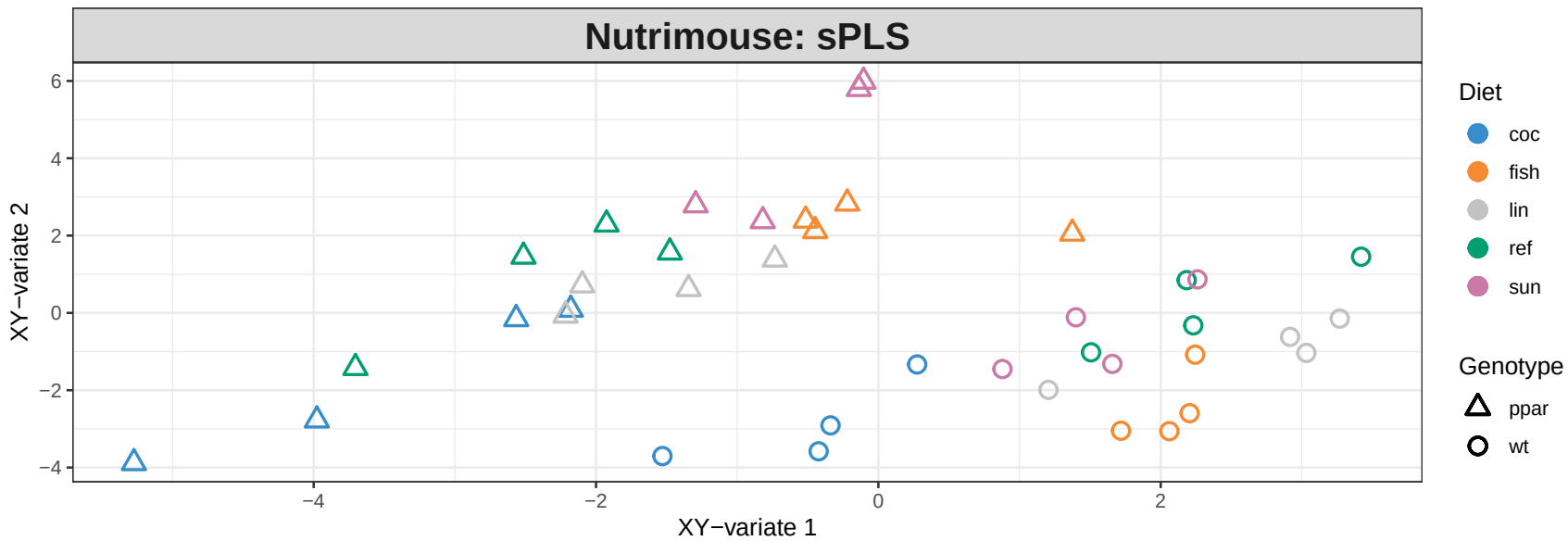
Customize sample plots

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



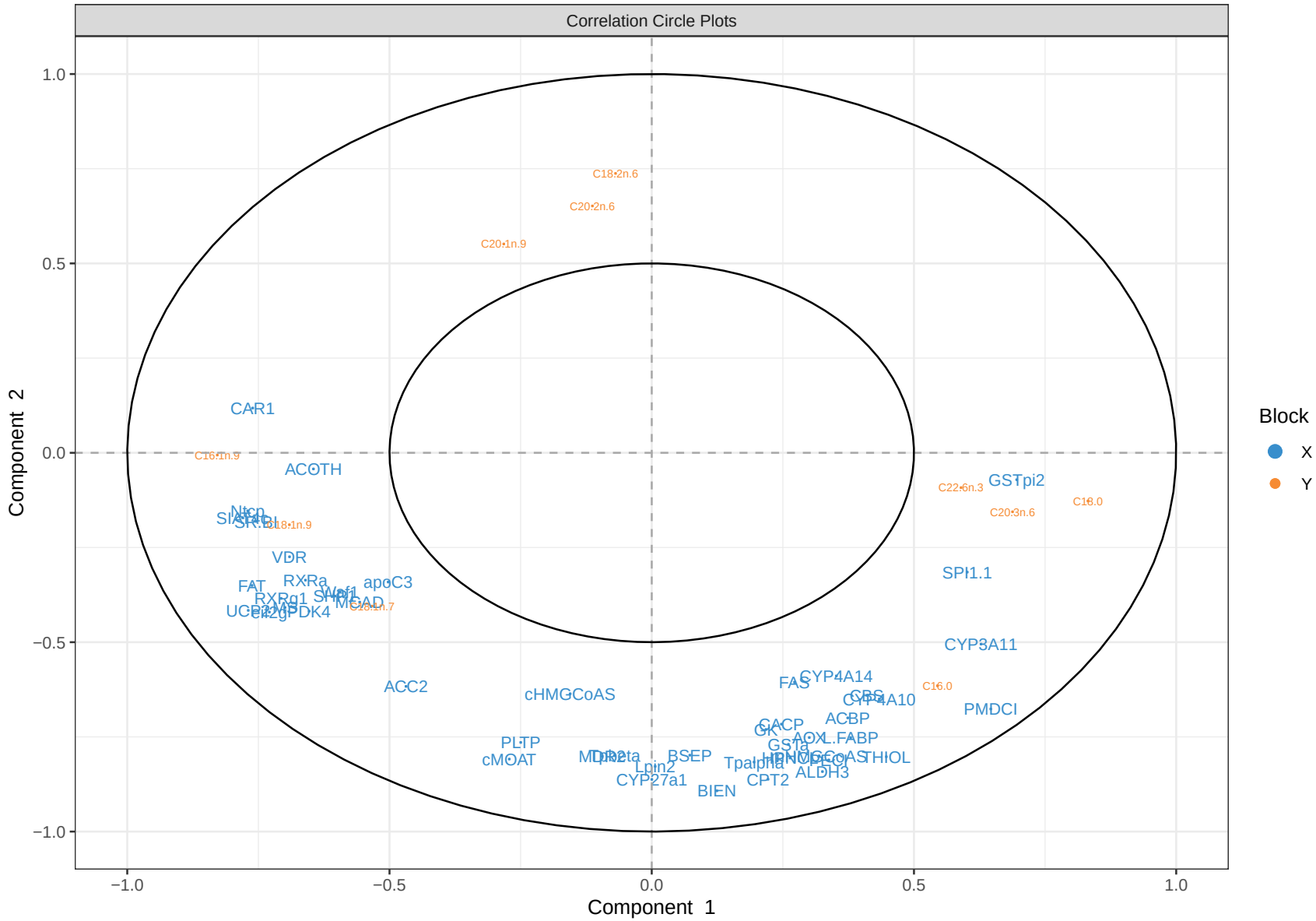
Customize sample plots

```
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')
```



Customize variable plots

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




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

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"
```

Variable selection outputs

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))
```

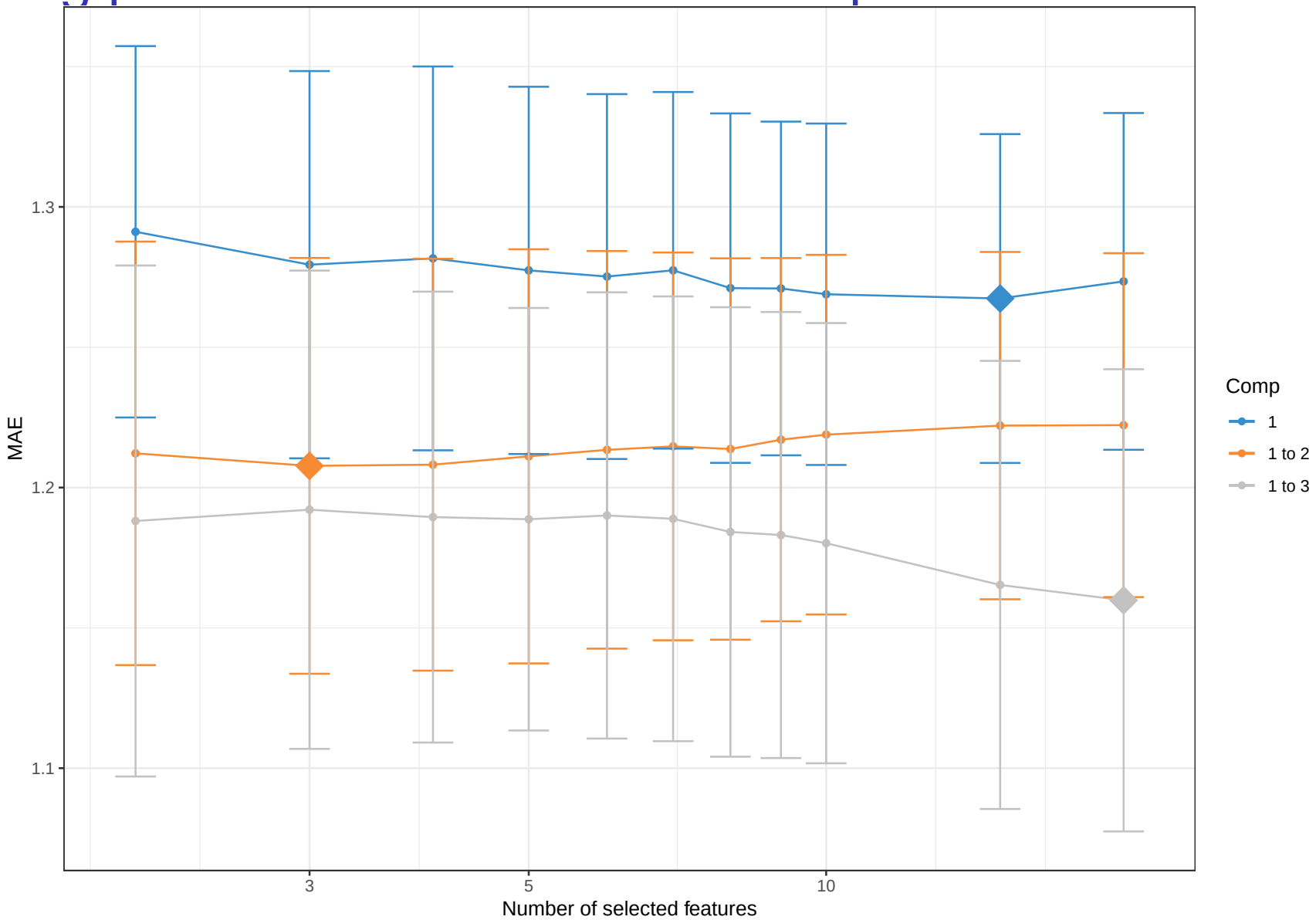
Tuning parameter and numerical outputs

- 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')
```

Tuning parameter and numerical outputs



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:

Tuning parameter

```
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.

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')
```

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.

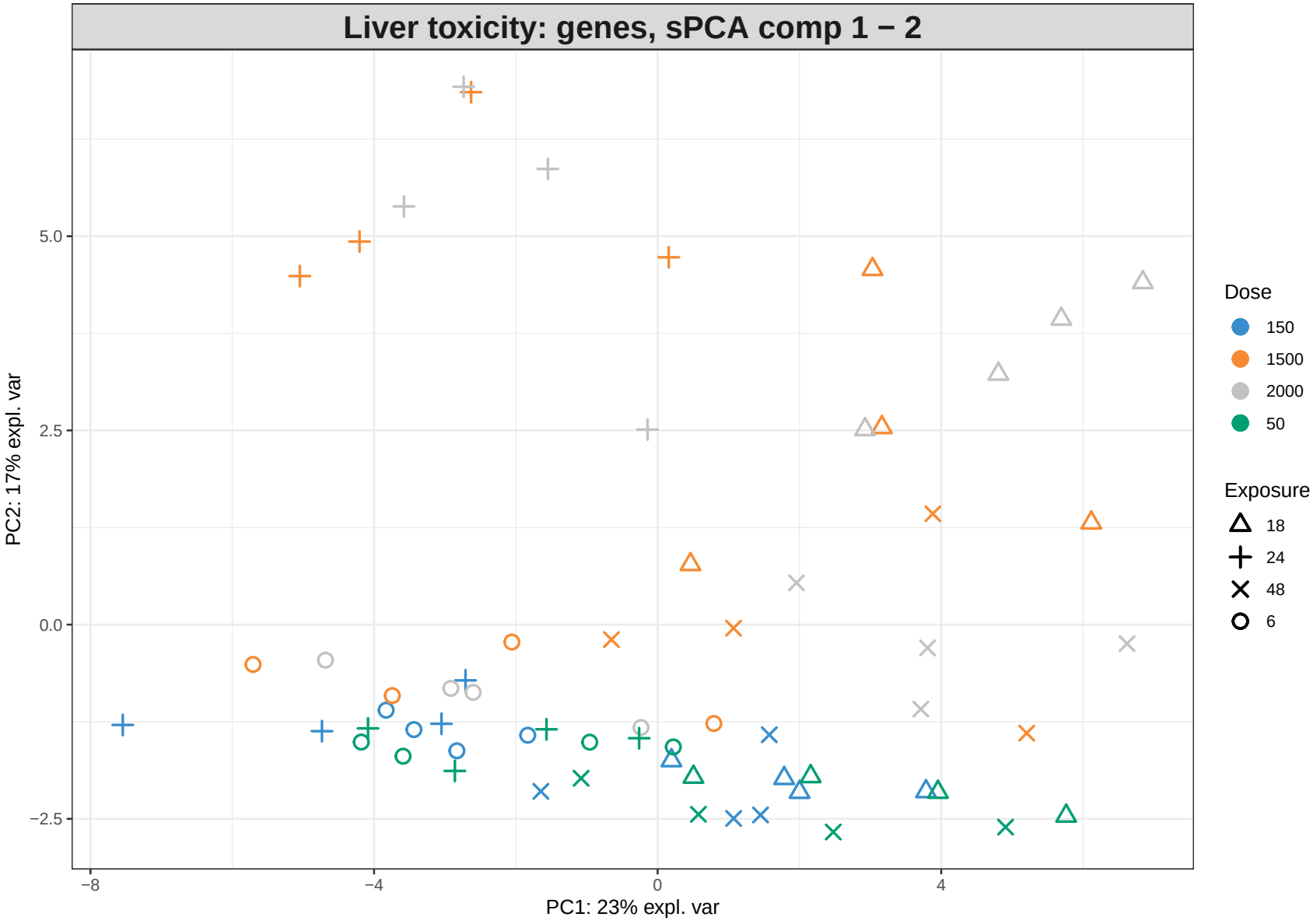
```
X11()
network(MyResult.spls, comp = 1)
network(MyResult.spls, comp = 1, cutoff = 0.6,
        save = 'jpeg', name.save = 'PLSnetwork')
# save as graph object for cytoscape
myNetwork <- network(MyResult.spls, comp = 1)$gR
```

Sparse PCA in action

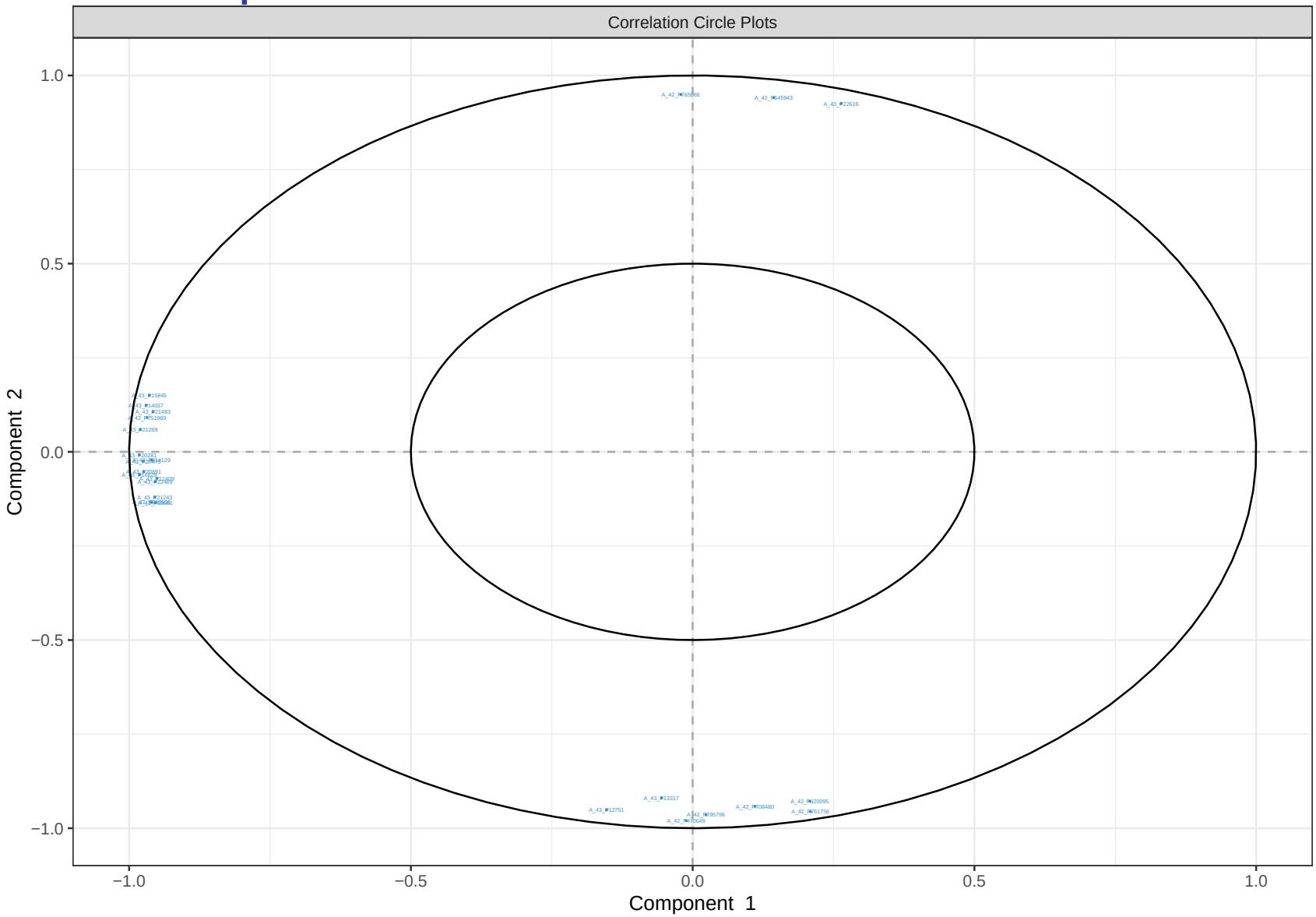
I would like to apply PCA but also be able to identify the key variables that contribute to the explanation of most variance in the data set.

```
data(liver.toxicity)
X <- liver.toxicity$gene
MyResult.spca <- spca(X, ncomp = 3, keepX = c(15,10,5))
plotIndiv(MyResult.spca, group = liver.toxicity$treatment$Dose.Group,
          pch = as.factor(liver.toxicity$treatment$Time.Group),
          legend = TRUE, title = 'Liver toxicity: genes, sPCA comp 1 -
          legend.title = 'Dose', legend.title.pch = 'Exposure')
plotVar(MyResult.spca, cex = 1)
# cex is used to reduce the size of the labels on the plot
```

Plot the samples



Plot the samples

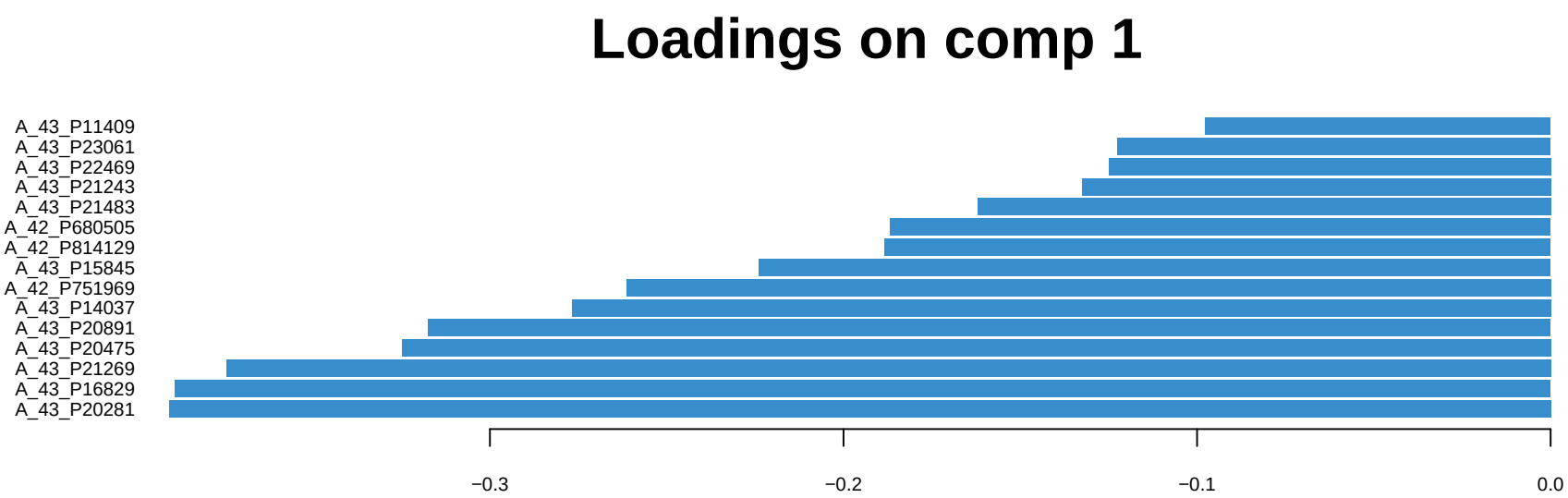


Selected variables can be identified on each component with the `selectVar` function. # Selected variables

Selected variables

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

```
plotLoadings(MyResult.spca)
```



Selected variables

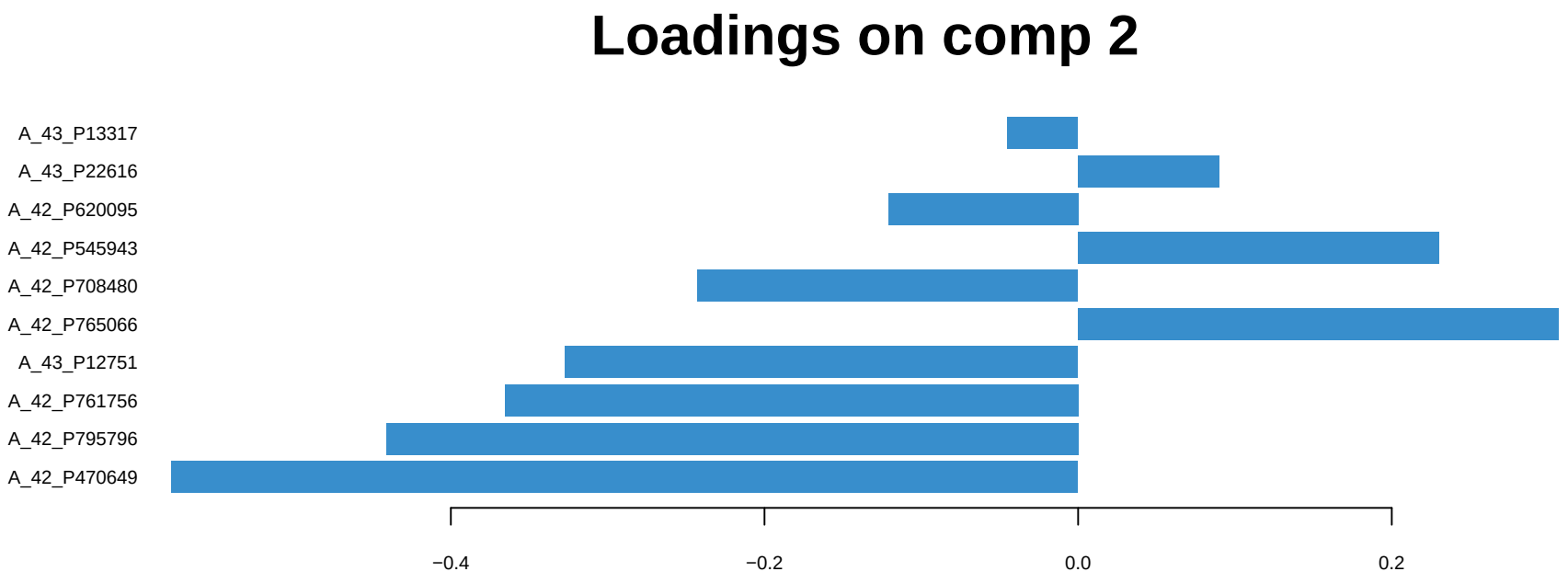
If we look at component two, we can see a mix of positive and negative weights (also see in the `plotVar`), those correspond to variables that oppose the low and high doses (see from the 'plotIndiv):

```
selectVar(MyResult.spca, comp=2)$value
```

```
                value.var
A_42_P470649 -0.57806702
A_42_P795796 -0.44100784
A_42_P761756 -0.36558200
A_43_P12751  -0.32721979
A_42_P765066  0.30628938
A_42_P708480 -0.24273534
A_42_P545943  0.23040165
A_42_P620095 -0.12099536
A_43_P22616   0.09024518
A_43_P13317  -0.04499990
```

Selected variables

```
plotLoadings(MyResult.spca, comp = 2)
```



Tuning parameters

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

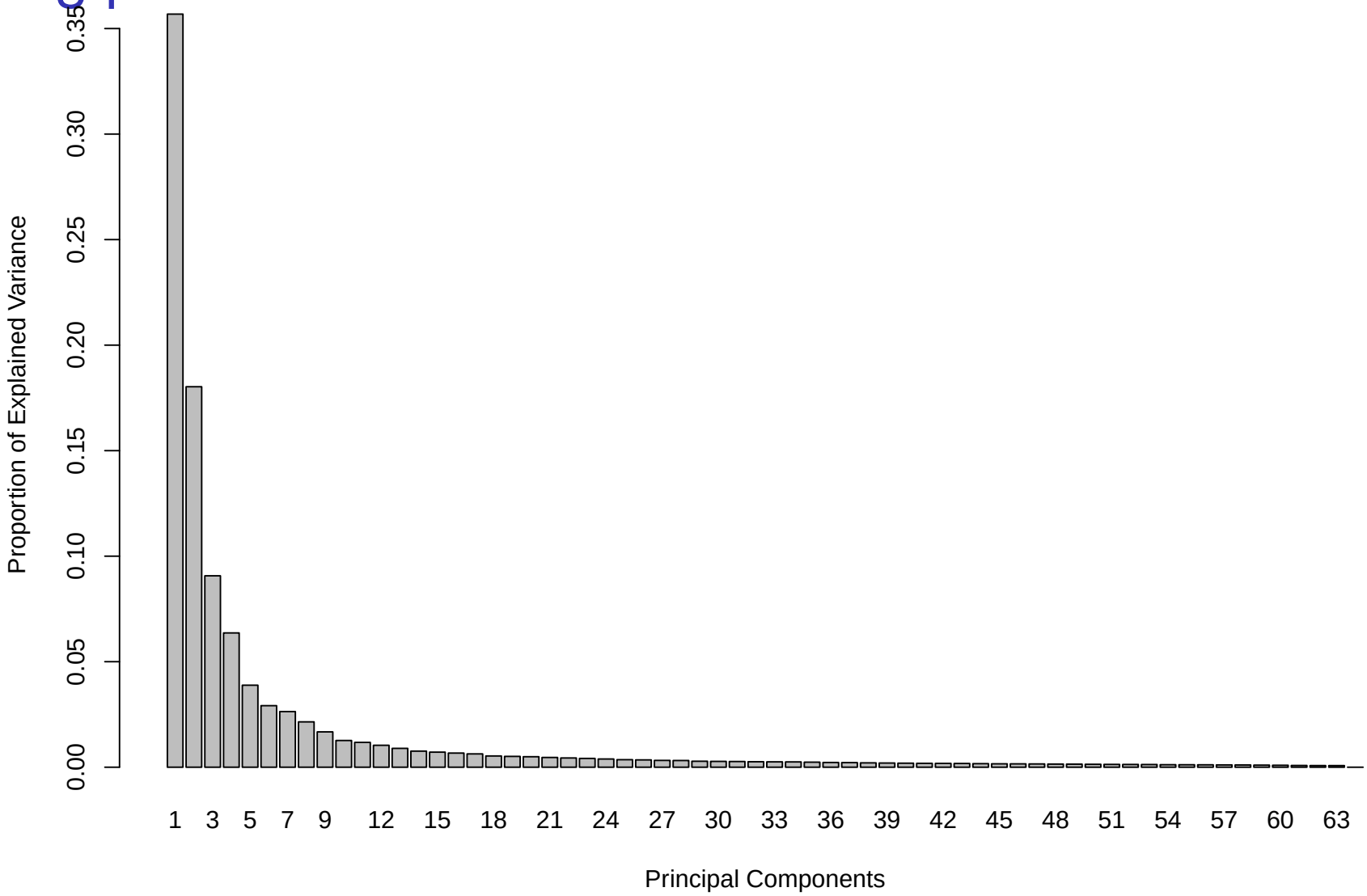
- The number of components to retain,
- The number of variables to select on each component for sparse PCA.

Tuning parameters

- 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)
```

Tuning parameters



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 recommend to set arbitrary thresholds that will pinpoint the key variables to focus on during the interpretation stage.

Other implementation of Sparse PCA

The R package `elasticnet` 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` 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
```

Take Home Message: Sparse PCA and PLS

- Sparse version enables us variable selection

- Tunning parameters could be difficult to calibrate

- Use cross-validation approach for tunning parameters