MOSS: Multi-omic integration via sparse singular value decomposition

Working example of MOSS' main capabilities.

Agustin Gonzalez-Reymundez, Alexander Grueneberg, Filipe Couto, Ana I. Vazquez

April 22, 2020

Contact: gonza@msu.edu

Introduction

Biological processes emerge from the interplay between different molecular factors. High-throughput technologies allow us to study this interplay across different molecular components of the cell (e.g. genome, transcriptome, proteome, etc.) and other *omics* (such as phenome, metabolome, meta-genome, etc.). Here, we present **MOSS**, a R package to perform omic integration dealing with the usual limitations of handling omic data, meaning

a. High variability among samples (e.g. patients, animal litters, plots, meta-genomics samples, etc.).
b. Scarcity of relevant signal among features (e.g. SNPs, genes, proteins, micro RNA's, taxa abundance, etc.).
c. Large number of subjects and features.

Package description

Package MOSS performs omic integration by concatenating several omic blocks, allowing one of them to be a multivariate numeric response $\mathbf{Y}$, or a uni-variate classification response (to allow both unsupervised and supervised omic integration). In general, omic blocks consisting of predictors are concatenated and normalized to create an extended omic matrix $\mathbf{Z}$. To summarize the main sources of variability across subjects and features, package MOSS performs a *generalized* singular value decomposition (gSVD) (basically, a SVD on a function of the cross-product between two matrices ($\mathbf{B}$= f($\mathbf{Z'Y}$)). Different functions $\mathbf{B}$ would allow application of different multi-variate techniques to identify the main axes of (co)variation across omics blocks. Examples of this are:

- $\mathbf{B} = \mathbf{Z'Z}$, in principal components analysis (PCA).
- $\mathbf{B} = \mathbf{Z'Y}$ in partial least squares (PLS).
- $\mathbf{B} = (\mathbf{Z'Z})^{+}\mathbf{Z'Y}$ in reduced rank regression (RRR).

By imposing the assumption of each omic block being conditionally independent from the rest (allowing marginal dependency), each multivariate technique can be extended to a 'multi-block' approach, where the contribution of each omic block to the total (co)variance can be addressed. Moreover, when response Y is a character column matrix, with classes or categories by subject, each multivariate technique can be extended to perform linear discriminant analysis.

Once results of the gSVD are obtained, package MOSS seeks for optimal *degrees of sparcity* (*dg*) by training sparse SVD (sSVD) models over a grid of possible *dg* values. This sSVD is performed using the algorithm of Shen and Huang (2008), extended to include Elastic Net type of regularization. To select an appropriate *dg* value, the package uses an ad-hoc method, based on the one presented in Shen & Huang (2008, see reference) and generalized for tuning two sparsity degrees parameters (for both, subjects and features levels).This is

done by exploring the proportion of explained variance (PEV) at each one of a grid of possible values. Drastic and/or steep changes in the PEV trajectory across *dg* values are used to automatically select an *optimal* one (see help for the function ssvdEN_sol_path). Finally, omic blocks too big to be handled in memory, can be turned into Filed-backed Big Matrix (FBM) objects. Then, MOSS calls package **bigstatsr** (Piove, 2020) to obtain **B** and the first round of SVD.

The following codes illustrate how to install MOSS, as well as some possible applications and capabilities . For more details and examples, please refer to the extended vignette. FOr an application pan-cancer omic integration, please refer to (Gonzalez-Reymundez 2020).

Installing and loading MOSS.

The code below illustrates how to install and load MOSS.

```
devtools::document("~/../Dropbox/MOSS")
#> Updating MOSS documentation
#> Warning: Version of roxygen2 last used with this package is 7.1.0. You only have
#> version 7.0.2
#> Loading MOSS
```

Examples and syntax

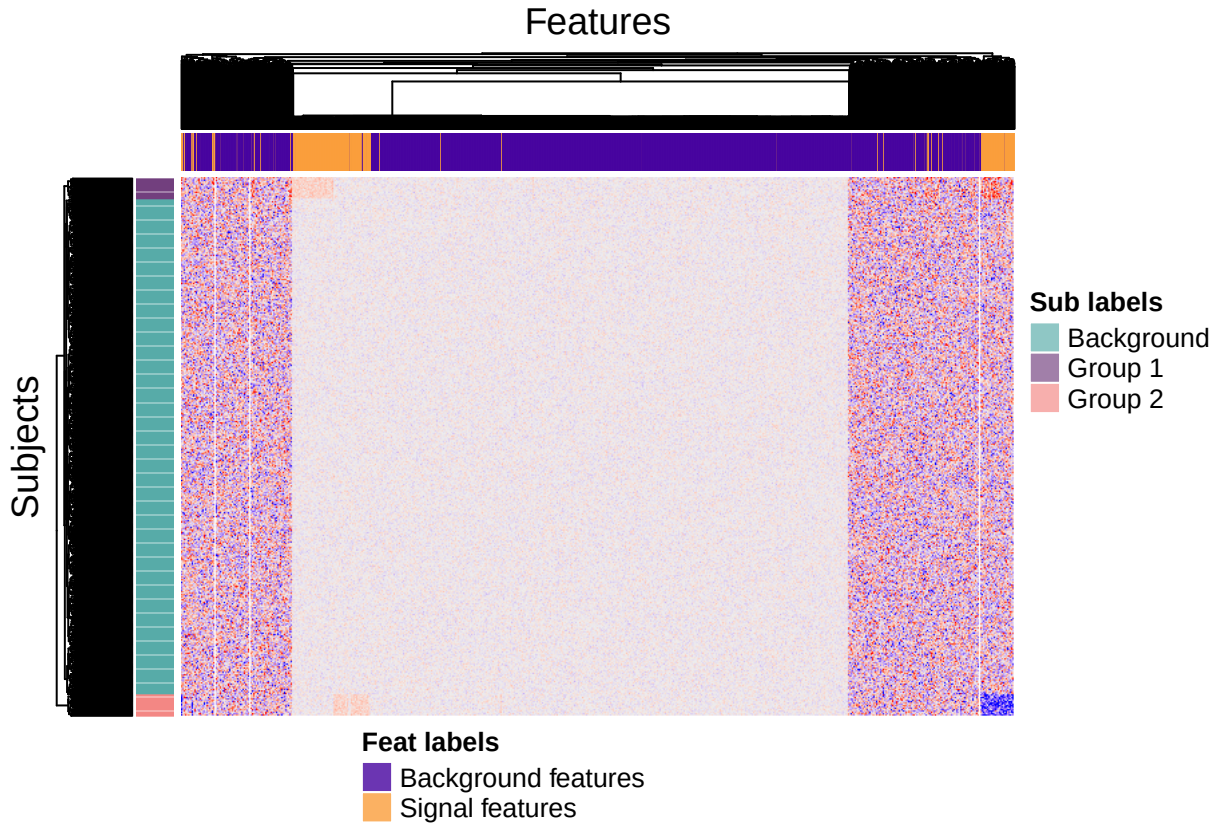
The examples below were produced using the results of a very straightforward simulation of two _**regulatory modules** (i.e. groups of features affecting specific groups of subjects -(Li et al., 2012))) across four data blocks (see **help(sim_data)** for a description of the simulation procedure). The following code shows how to load these data and check its classes and dimensions.

```
data(sim_data)
#Extracting simulated omic blocks.
sim_blocks <- sim_data$sim_blocks

#Extracting subjects and features labels.
lab.sub <- sim_data$labels$lab.sub
lab.feat <- sim_data$labels$lab.feat
```

The object **sim_blocks** is a list with four omic blocks, the vectors **lab.sub** and **lab.feat** indicate what subjects and features representing the simulated regulatory modules. We will use these two vectors specifying the position of *signal* subjects and features in order to show **MOSS**' performance at detecting the right sets of subjects and features. The fourth data block within **sim_blocks** represents a categorical variable (see examples of its use in the context of omic integration via linear discriminant analysis; also see **help(moss)**). The following code allows a nice visualization of the structure within **sim_blocks**, highlighting background *noise* from *signal*, can be obtained with package **ComplexHeatmap** ().

```
#Visualizing the simulated omic blocks with a heatmap.
require("ComplexHeatmap")
#> Loading required package: ComplexHeatmap
#> Loading required package: grid
#> ========================================
#> ComplexHeatmap version 2.2.0
#> Bioconductor page: http://bioconductor.org/packages/ComplexHeatmap/
#> Github page: https://github.com/jokergoo/ComplexHeatmap
#> Documentation: http://jokergoo.github.io/ComplexHeatmap-reference
#>
#> If you use it in published research, please cite:
#> Gu, Z. Complex heatmaps reveal patterns and correlations in multidimensional
#>   genomic data. Bioinformatics 2016.
#> ========================================
```

```r
#Highlighting groups of subjects and features.
Sub.ann <- rowAnnotation(
  `Sub labels`=lab.sub,
  col=list(`Sub labels`=c("Group 1" = "#44015480" ,"Group 2"="#F1605D80" ,"Background"="#21908C80")),
  show_annotation_name=F)

Feat.ann <- HeatmapAnnotation(
  `Feat labels`=lab.feat,
  col=list(`Feat labels`=c("Background features" = "#47039FCC" ,"Signal features"="#FA9E3BCC")),
  show_annotation_name = F)

#Creating heatmap.
draw(Heatmap(do.call("cbind", sim_blocks[-4]), #Excluding the categorical response.
        left_annotation = Sub.ann,
        top_annotation = Feat.ann,
        row_title = "Subjects",
        column_title = "Features",
        show_heatmap_legend = F,
        cluster_columns = T,
        cluster_rows = T),
     annotation_legend_side = "bottom")
```

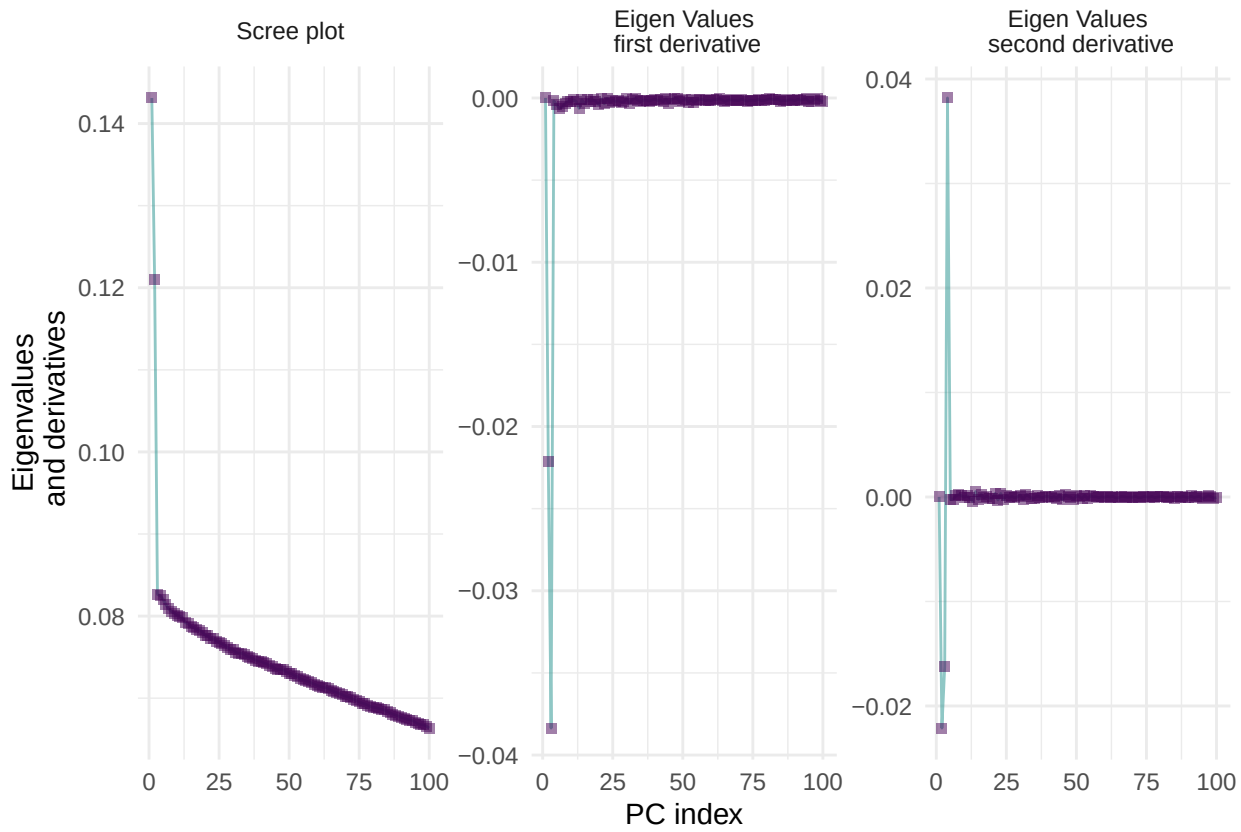


Now, suppose you want a fast PCA to have an idea of your data structure. You can do that with **MOSS** by giving a list of objects of class "matrix" or "FBM".

```
require("ggplot2")
#> Loading required package: ggplot2
require("ggthemes")
#> Loading required package: ggthemes
require("viridis")
#> Loading required package: viridis
#> Loading required package: viridisLite
set.seed(43)
out <-  moss(
  data.blocks = sim_blocks[-4], #a list with omic blocks (we exclude the response vector)
        method="pca", #This is the default method.
        K.X = 100, #Number of PCs require. Default is K.X = 5.
        plot=TRUE) #Allows graphical display.
#> Checking for missing values.
#> ------------------------------------------------------------------------------
#> | Principal Components Analysis (PCA) for 3 data blocks and 100 latent factors |
#> ------------------------------------------------------------------------------
#> Standardizing/Normalizing data blocks.
#> Getting SVD of predictors matrix (dimension 500 x 3000).
```

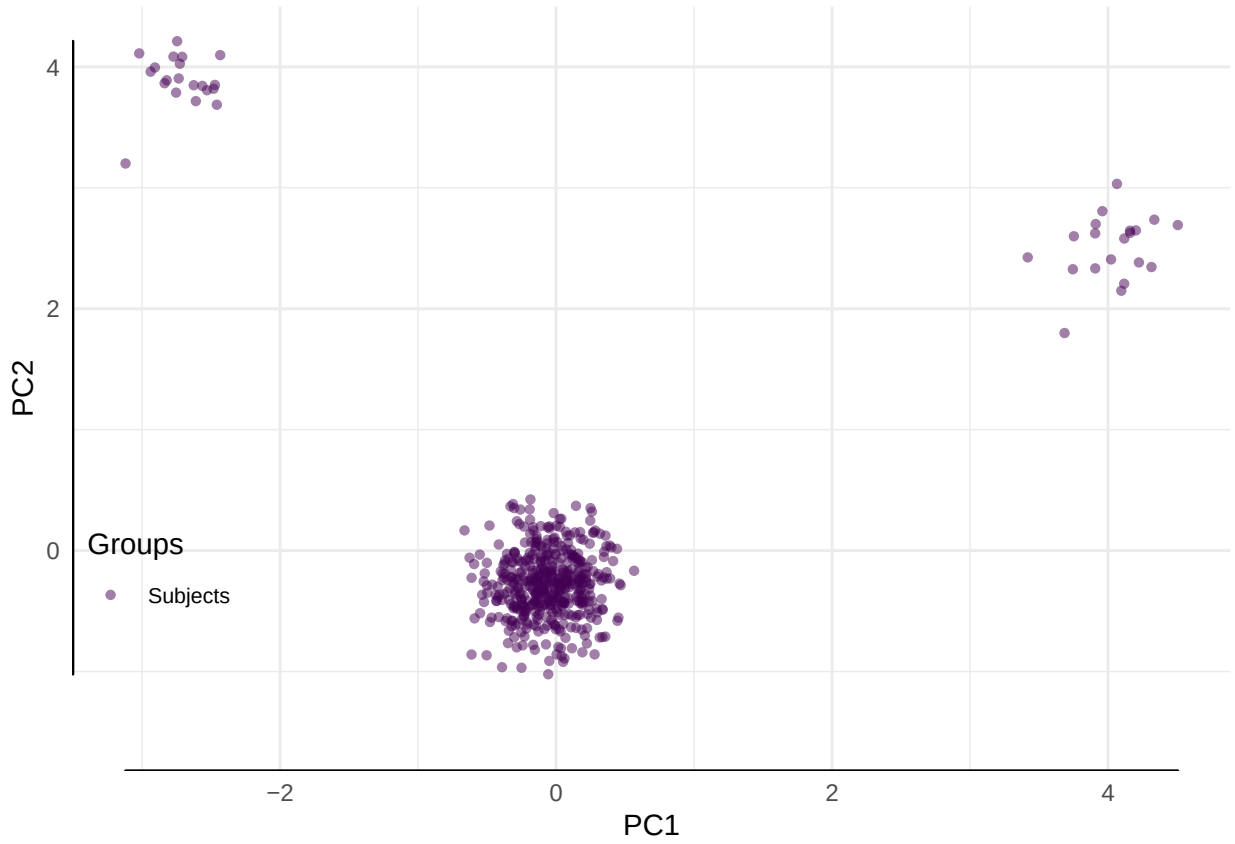
The function **moss** returns a plot of eigenvalues, together with their first and second derivative across PC index.

```
#Showing scree plot.
out$scree.plot
```



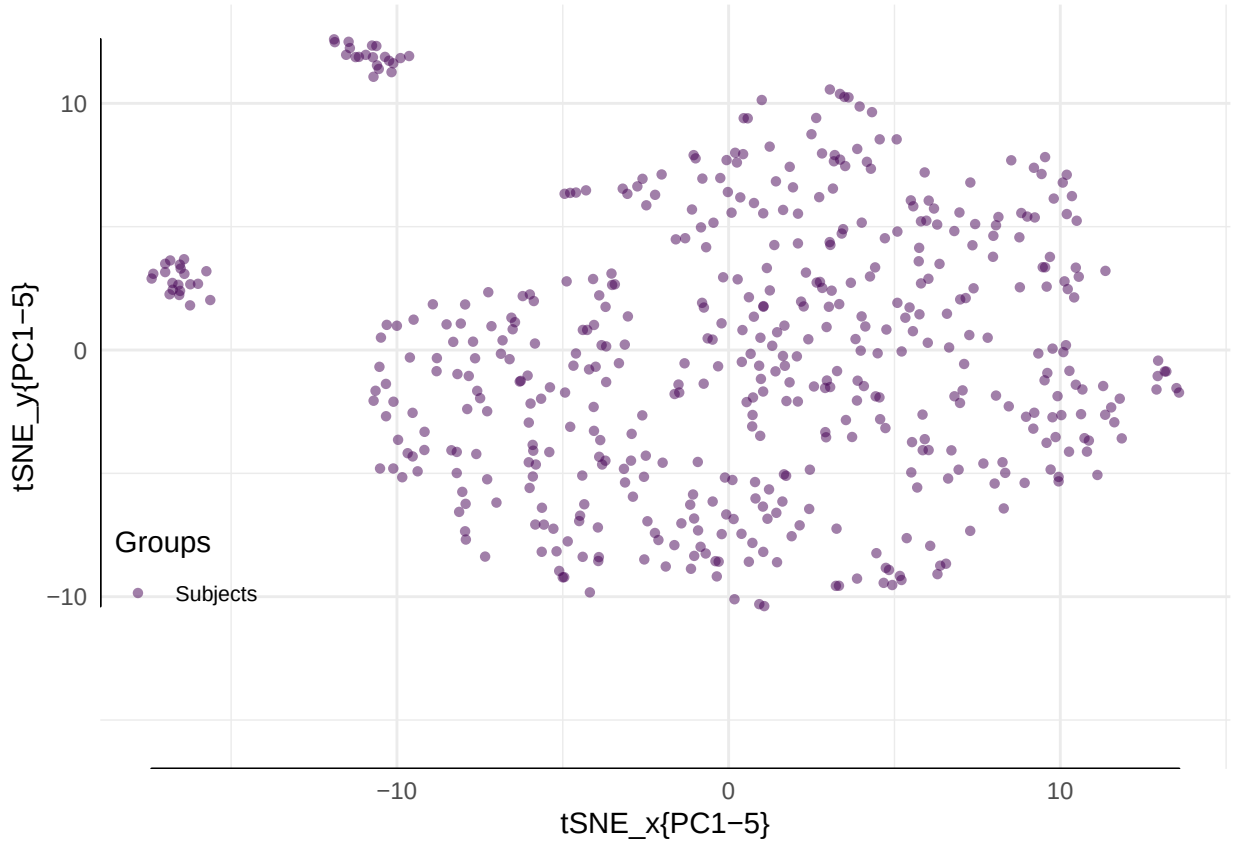
As well as a plot of the first two PCs.

```
#Showing scatterplot of the first two PCs.
out$PC1_2.plot
```



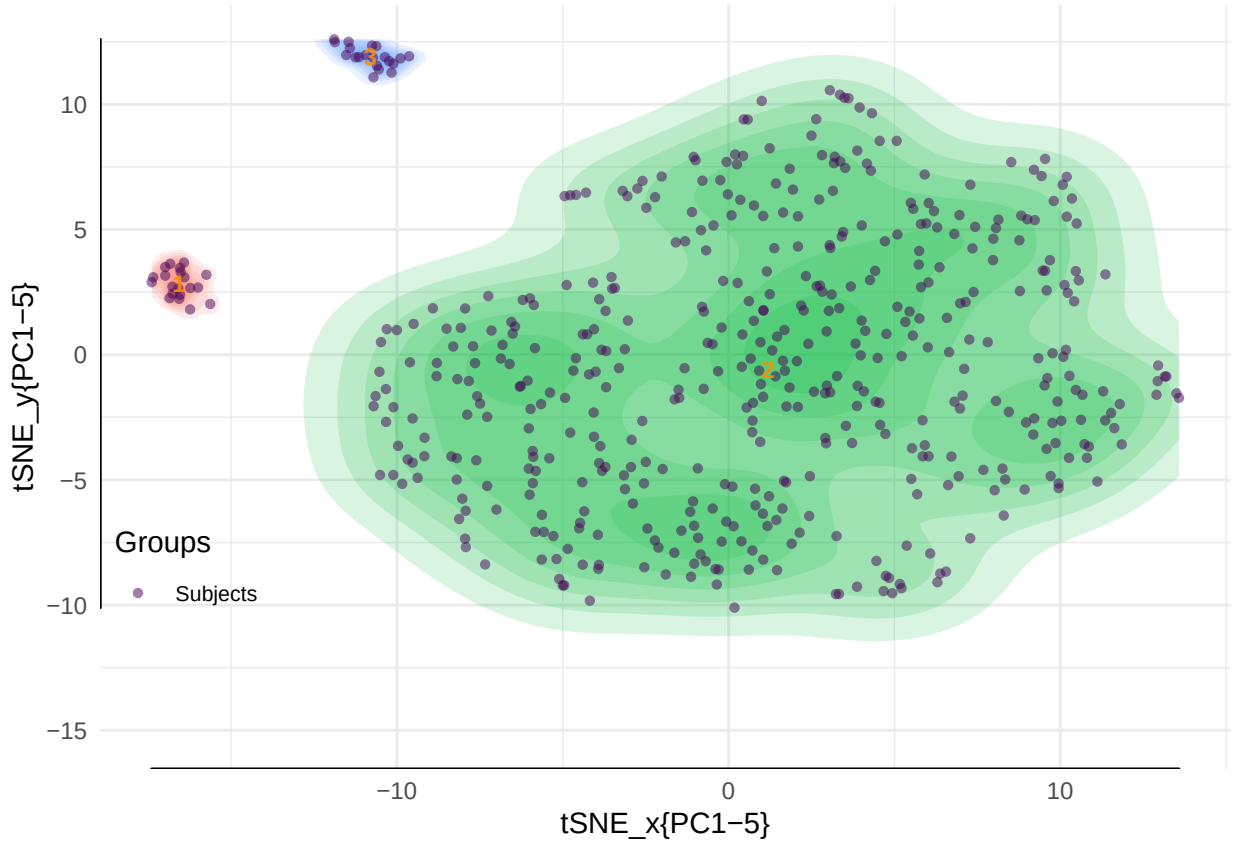
In this simple example, the data structure is clear. However, if the user wishes to map a set of more than two PC, the option *tSNE* can be used. This tell **MOSS** to call **Rtsne** to map more than two PCs onto a two dimensional display. By using *tSNE* = TRUE, one random realization of t-stochastic neighbor embedding is done, with perplexity equal 50, in one thousand iterations (e.g. *tSNE* = list("perp"=50, "n.samples"=1, "n.iter"=1e3). The user can vary this by passing arguments to *tSNE*, as in the following code.

```
require("Rtsne")
#> Loading required package: Rtsne
set.seed(43)
out <-  moss(
  data.blocks = sim_blocks[-4], #a list with omic blocks (we exclude the response vector)
        method="pca", #This is the default method.
        tSNE=list("perp"=50, "n.samples"=1, "n.iter"=1e3), #Mapping first five PC onto 2_D.
        plot=TRUE)
#> Checking for missing values.
#> ------------------------------------------------------------------------------
#> | Principal Components Analysis (PCA) for 3 data blocks and 5 latent factors |
#> ------------------------------------------------------------------------------
#> Standardizing/Normalizing data blocks.
#> Getting SVD of predictors matrix (dimension 500 x 3000).
#> Calculating a tSNE map
#> Embedding 5 axes onto two dimensions.
out$tSNE.plot
```



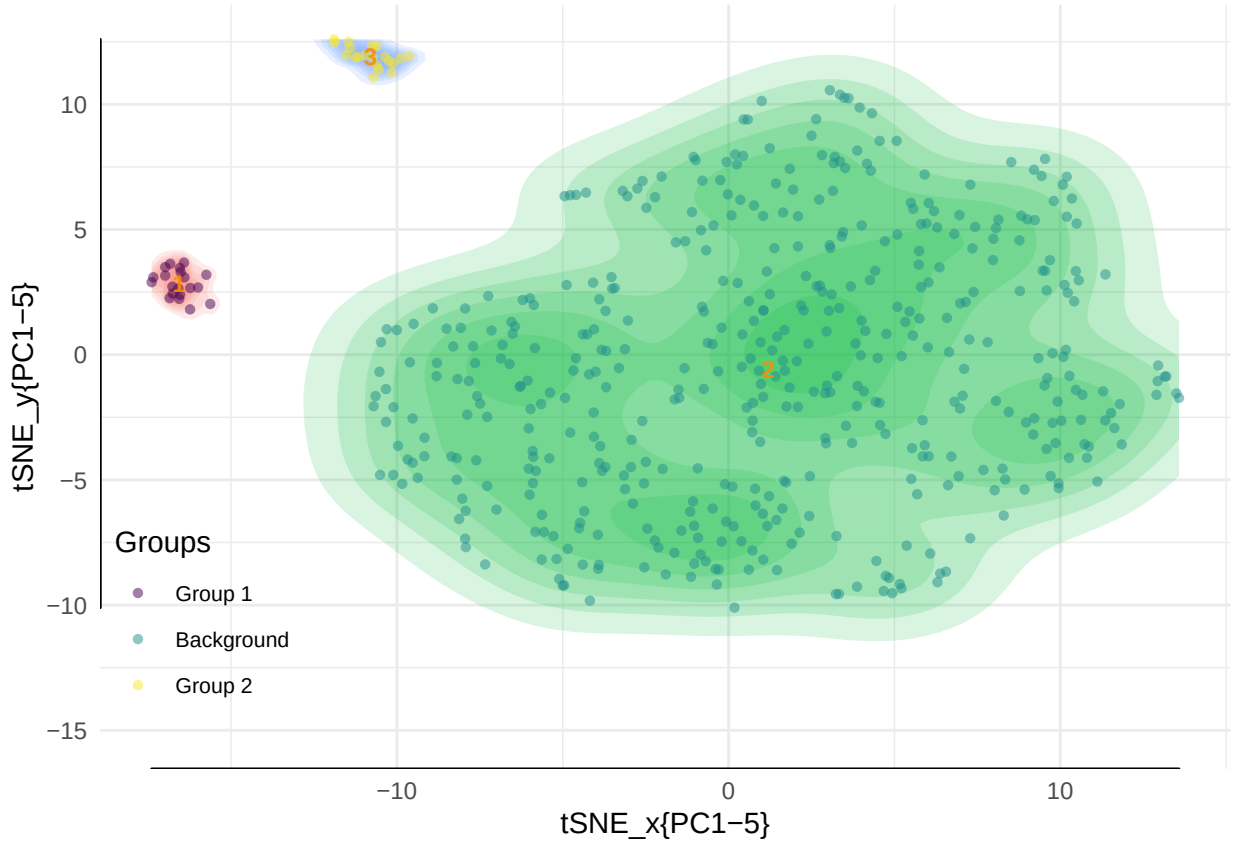
The user can also use the option *clus*=TRUE within *moss*. This option allows **MOSS** to perform a cluster delimitation using DBSCAN (), as inspired in the **meredith** package. For more personalized aesthetic and labeling, check **help(tsne2clus)** and examples in the extended vignette.

```
set.seed(43)
require("dbscan")
#> Loading required package: dbscan
out <-  moss(
  data.blocks = sim_blocks[-4], #a list with omic blocks (we exclude the response vector)
        method="pca", #This is the default method.
        tSNE=list("perp"=50, "n.samples"=1, "n.iter"=1e3), #Mapping first five PC onto 2_D.
        clus=T, #Delimiting cluster via DBSCAN.
        plot=TRUE)
#> Checking for missing values.
#> ------------------------------------------------------------------------------
#> | Principal Components Analysis (PCA) for 3 data blocks and 5 latent factors |
#> ------------------------------------------------------------------------------
#> Standardizing/Normalizing data blocks.
#> Getting SVD of predictors matrix (dimension 500 x 3000).
#> Calculating a tSNE map
#> Embedding 5 axes onto two dimensions.
#> Getting clusters via DBSCAN.
out$clus.obj
```



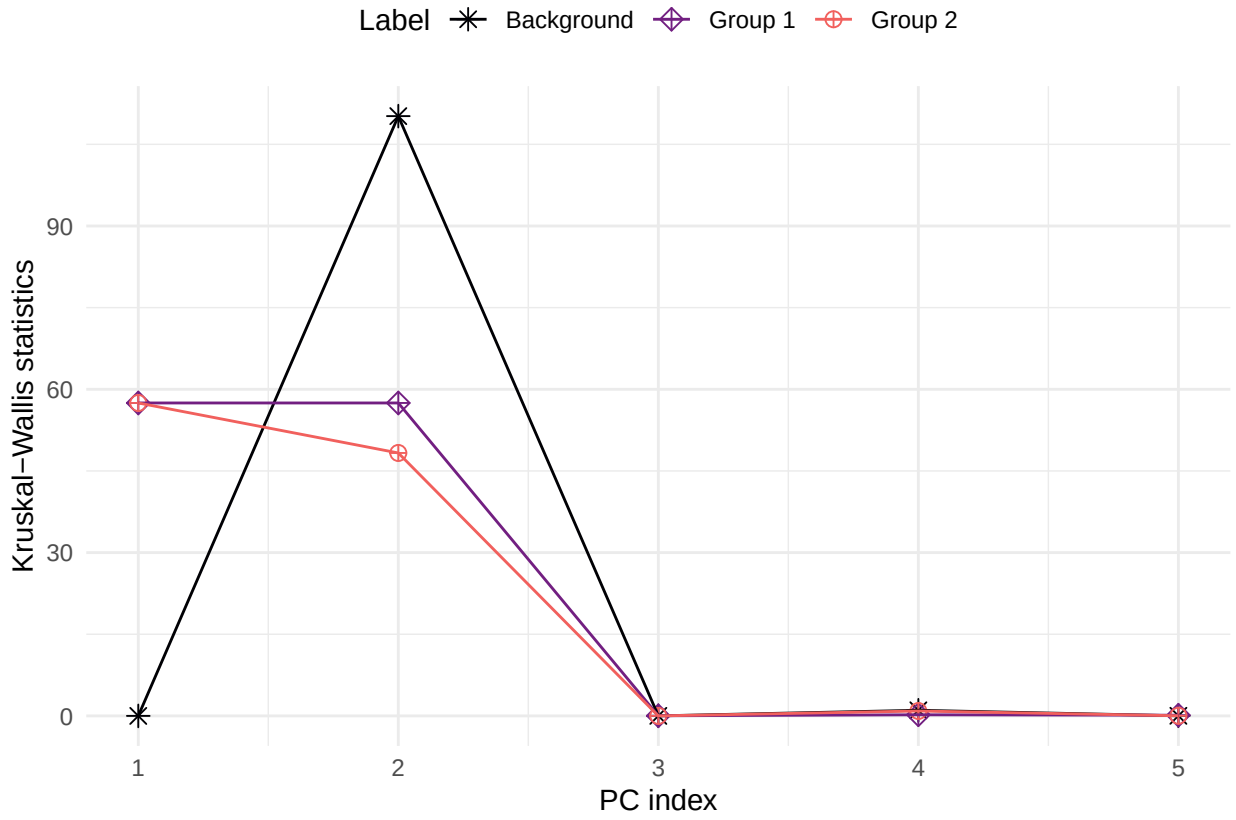
If information of pre-defined biologically relevant groups is available (e.g. histological subtypes of cancer), the user can *1)* use it to label points and visually assess the overlap between data-driven clusters and previous information and *2)* use to determine which PCs are significantly associated with each group.

```
set.seed(43)
out <-  moss(
  data.blocks = sim_blocks[-4], #a list with omic blocks (we exclude the response vector)
        method="pca", #This is the default method.
        tSNE=list("perp"=50, "n.samples"=1, "n.iter"=1e3), #Mapping first five PC onto 2_D.
        clus=T, #Delimiting cluster via DBSCAN.
        clus.lab = lab.sub, #We'll use the info of 'signal' subjects as an example.
        plot=TRUE)
#> Checking for missing values.
#> ---------------------------------------------------------------------------
#> | Principal Components Analysis (PCA) for 3 data blocks and 5 latent factors |
#> ---------------------------------------------------------------------------
#> Standardizing/Normalizing data blocks.
#> Getting SVD of predictors matrix (dimension 500 x 3000).
#> Calculating a tSNE map
#> Embedding 5 axes onto two dimensions.
#> Getting clusters via DBSCAN.
#> Evaluating overlap between subjects selected and pre-established labels.
out$clus.obj
```



The Kruskal-Wallis statistics (KW) is used to determine how much each PC is associated with each pre-defined group (the input for the *clus.lab* argument). Higher values of the KW statistics will representing stronger associations.

```
out$subLabels_vs_cluster
```



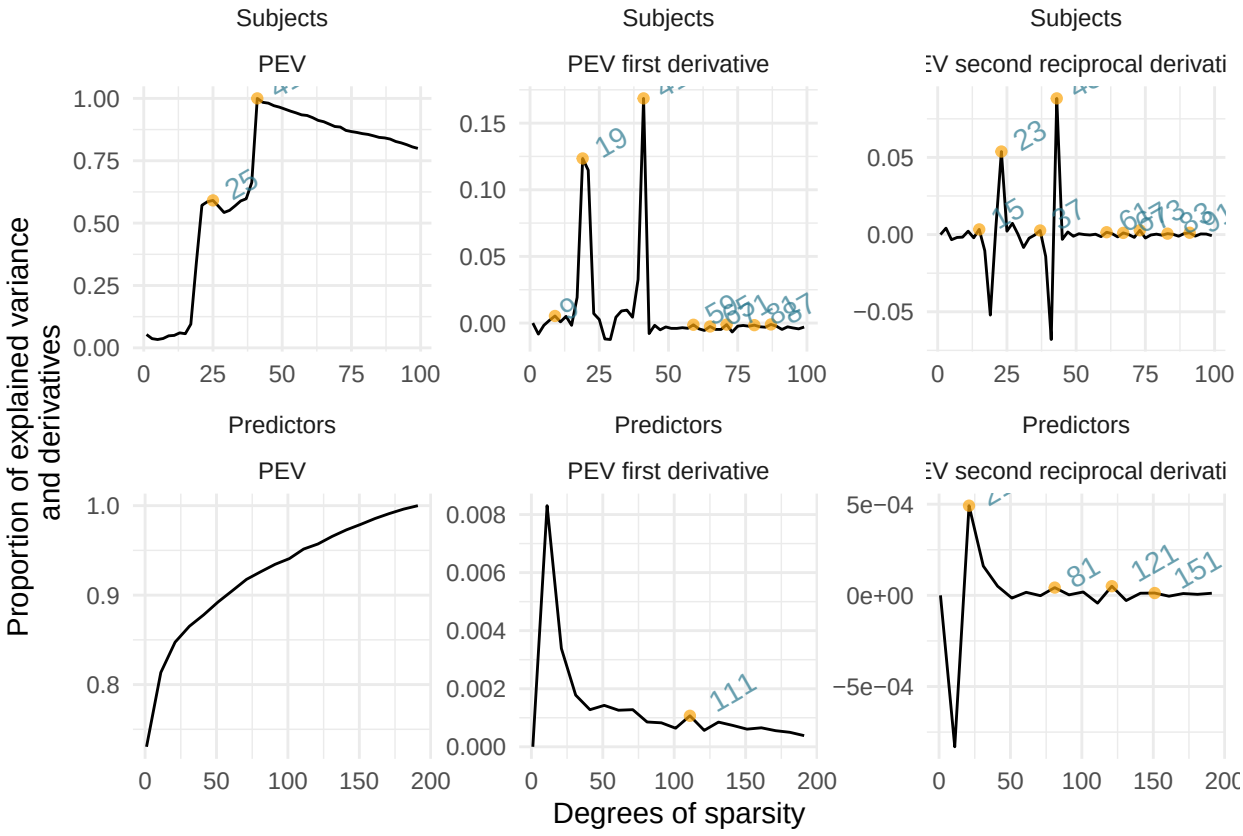
Now, suppose that the user is interested in detecting what groups of subjects and features form regulatory modules. The following code shows an application of MOSS using sparse principal components analysis. In this example, we tune the degree of sparsity for both subjects and features, using an elastic net parameter *alpha.* equal to one, for subjects (i.e. a LASSO penalty), and 0.5 for features.

```
set.seed(43)
out <-  moss(data.blocks = sim_blocks[-4], #Feed moss a list with omic blocks (we exclude the response
        method="pca", #This is the default method.
        K.X = 5, #Number of PC (Defaults to 5).
        dg.grid.right = seq(1,200,by=10), #This is a grid of increasing sparsity values for features.
        dg.grid.left = seq(1,100,by=2), #Same, but for subjects.
        alpha.right = 0.5, #This is the EN parameter for features.
        alpha.left = 1, #This is the EN parameter for subjects. Here we're doing LASSO.
        tSNE=T, #This tells moss to project the 5 PC onto 2-D via tSNE.
        clus=T, #This tells moss to use DBSCAN to delimt clusters on the tSNE map
        clus.lab=lab.sub, #This colors subjects according to the labels within lab.sub
        plot=TRUE)
#> Checking for missing values.
#> ------------------------------------------------------------------------------
#> | Principal Components Analysis (PCA) for 3 data blocks and 5 latent factors |
#> ------------------------------------------------------------------------------
#> Standardizing/Normalizing data blocks.
#> Elastic net smoothing & selection of right eigenvectors. Number of features selected increases with
#> Getting SVD of predictors matrix (dimension 500 x 3000).
#> Imposing sparsity constraints.
#> Calculating a tSNE map
#> Embedding 5 axes onto two dimensions.
```

```
#> Getting clusters via DBSCAN.
#> Evaluating overlap between subjects selected and pre-established labels.
```

The following shows the plot with the marginal variation in *PEV* across degrees of sparsity at the subjects and features levels, respectively. The numbers in the peaks of first and second-reciprocal derivatives show the point of rapid and/or accelerated changes. The minimum between the global maximum in the trajectory of first and second derivative, respectively, is automatically chosen as 'optimal'. Although it was tempting to include an automatic selection of both PCs and degrees of sparsity using the heuristic discussed above, we leave to the user the decision of how many PC to include in the model. We discuss benefits, limitation, and alternatives of this approach at the end of this document.

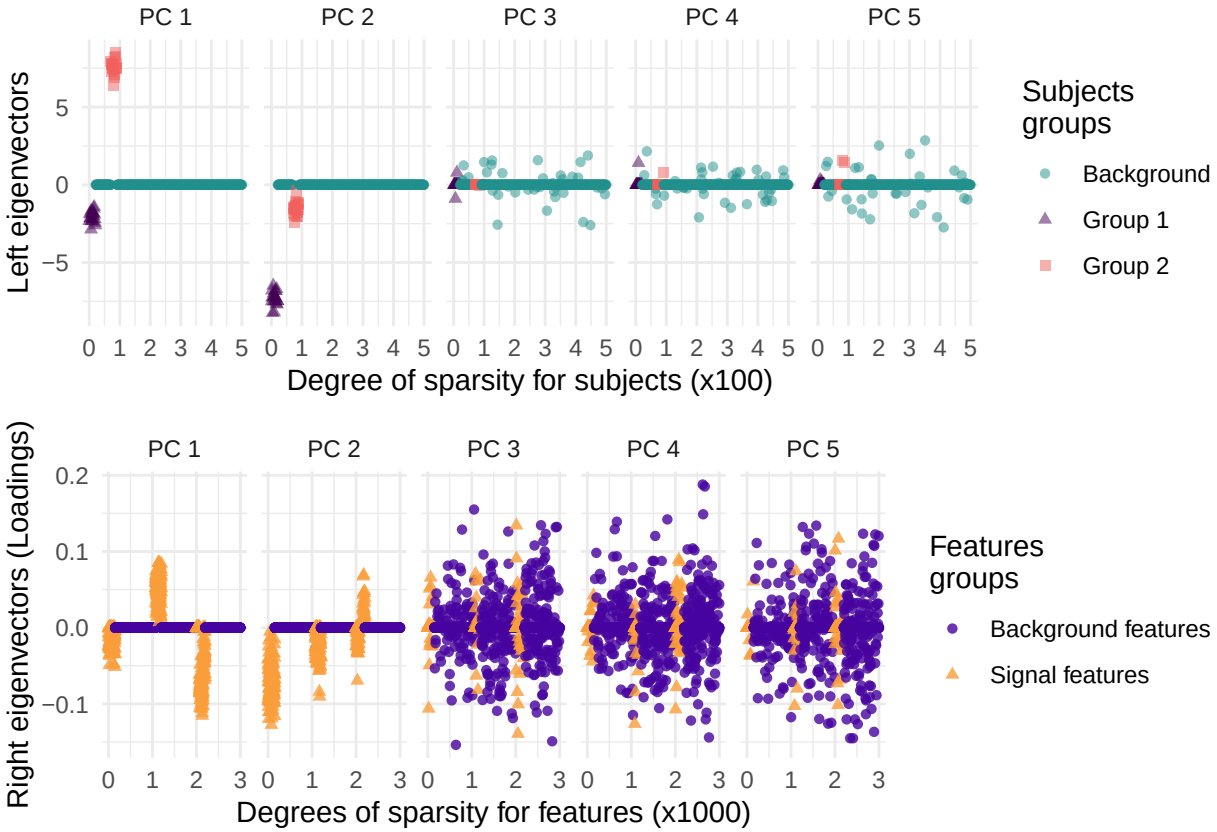
```
out$tun_dgSpar.plot
```



We now can use the solutions **MOSS** for the selected *dg* for subjects and/or features and and plot them against elements of *lab.sub* and *lab.feat* to evaluate how good **MOSS** does to identify relevant biological signals (see **help(moss)**).

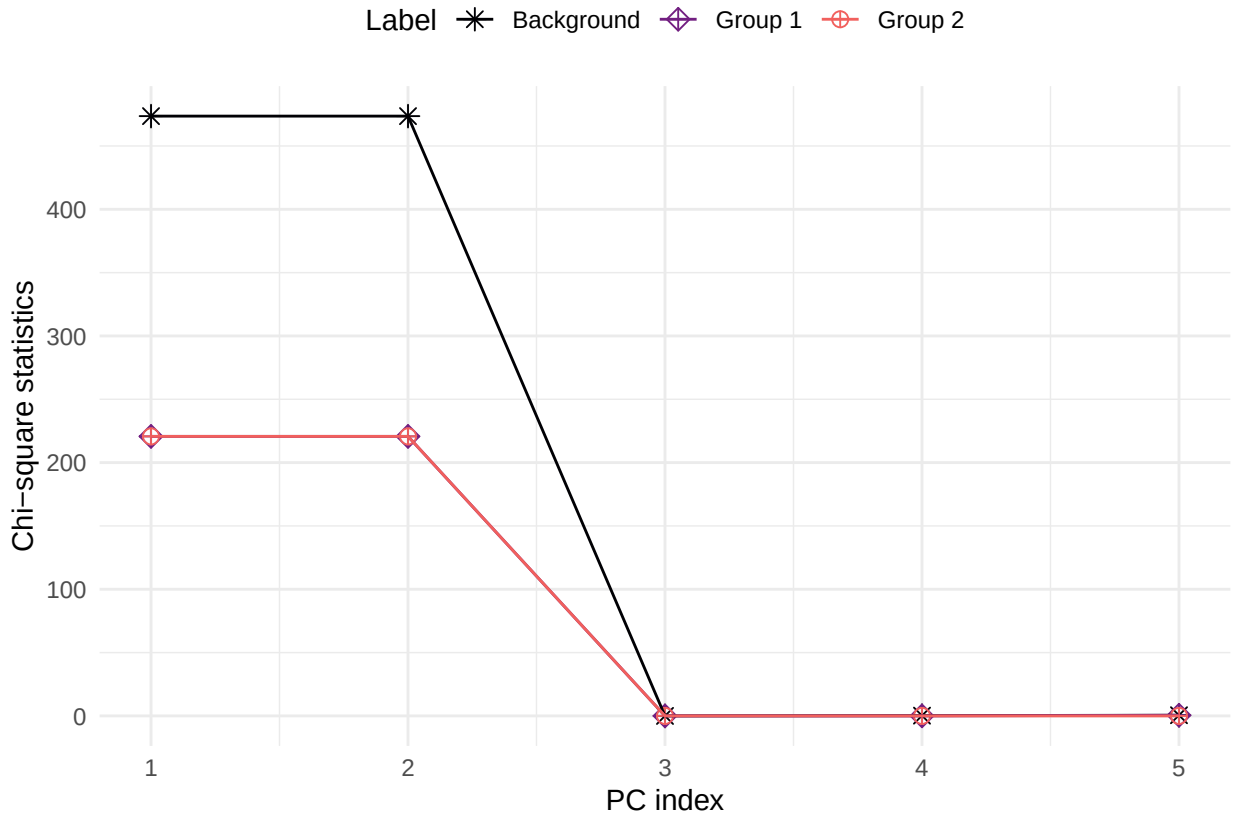
```
require("gridExtra")
#> Loading required package: gridExtra

#Did we select the correct 'signal subjects'?
d <- data.frame(Features=rep(1:500,5),
                Loadings=as.vector(out$sparse$u),
                Features_groups=rep(lab.sub,5),
                PC=paste0("PC ",rep(1:5,each=500)))

#Storing subjects' results.
```

```r
q.s <- ggplot(d,aes(x=Features,y=Loadings,col=Features_groups,pch=Features_groups))+
  facet_grid(~PC,scales = "free_y")+
  scale_color_manual(values=c("#21908C80","#44015480" ,"#F1605D80"))+
  scale_x_continuous("Degree of sparsity for subjects (x100)",breaks=seq(0,500,100),labels = seq(0,5,1))
  scale_y_continuous("Left eigenvectors")+
  geom_point()+
  theme_minimal()+
  labs(col='Subjects\ngroups',
       pch='Subjects\ngroups')

#Did we select the correct 'signal features'?
d <- data.frame(Features=rep(1:3e3,5),
                Loadings=as.vector(out$sparse$v),
                Features_groups=rep(lab.feat,5),
                PC=paste0("PC ",rep(1:5,each=3e3)))

#Storing features' results.
q.f <- ggplot(d,aes(x=Features,y=Loadings,col=Features_groups,pch=Features_groups))+
  facet_grid(~PC,scales = "free_y")+
  scale_color_manual(values=c("#47039FCC","#FA9E3BCC"))+
  scale_x_continuous("Degrees of sparsity for features (x1000)",breaks=seq(0,3e3,1e3),labels = seq(0,3,
  scale_y_continuous("Right eigenvectors (Loadings)")+
  geom_point()+
  theme_minimal()+
  labs(col='Features\ngroups',
       pch='Features\ngroups')

#Putting plots together.
grid.arrange(q.s,q.f)
```



The above results offer a good example of how the performance of the selection task can be closed tied to what PCs end being included in the model. In this simple example, the scree plot suggested the two first PCs as the ones explaining most of the data variability. If we are interested in finding the regulatory modules that define clusters of subjects, we would like first to identify what left-singular vectors are associated with these clusters. Then, determine what features have a relevant impact on their variation, by taking those with non-zero values along the matching right-singular vectors. Whenever sparsity is imposed at the subjects level and *clus.lab* is different than NULL, a Chi square statistic of the overlap between selected subjects and pre-defined labels is used instead of KW.

```
#What axes of variation are associated with clusters of subjects?
out$subLabels_vs_cluster
```



Finally, to have an idea of how we did in terms of specificity and sensitivity, we can use the following code, where the rows represent 'positive' versus 'negative' results, respectively, and columns represent 'true' versus 'false' results. From the tables below, it can be seen that in this simple example we had 100% specificity and 99% sensitivity for subjects, and 92% specificity and 99% sensitivity. Nevertheless, we expect lower performances as a result of more convoluted structure and combination of sparsity parameters.

```
#Evaluating performance of subjects selection.
table(rowMeans(out$sparse$u[,1:2]) != 0,lab.sub)

#Evaluating performance of features selection.
table(rowMeans(out$sparse$v[,1:2]) != 0,lab.feat)
```

Additional capabilities

By imposing the additional assumption of omic blocks being conditionally independent, each multivariate technique can be extended using a 'multi-block' approach, where the contribution of each omic block to the total (co)variance is addressed. When response $\mathbf{Y}$ is a character column matrix, with classes or categories by subject, each multivariate technique can be extended to perform linear discriminant analysis.

Limitations and future work.

1) The function does not return PEV for elastic net parameters, the user needs to provide a single value for each.
2) When number of PC index > 1, columns right singularvectors might not be orthogonal.
3) Although the user is encouraged to perform data projection and cluster separately, MOSS allows to do this automatically. However, both tasks might require finer tuning than the provided by default, and computations could become cumbersome. In those cases, the user can call functions **pca2tsne** and

tsne2clus, separately.

4) Tuning of degrees of sparsity is done heuristically on training set. In our experience, this results in high specificity, but rather low sensitivity (i.e. too liberal cutoffs, as compared with extensive cross-validation on testing set).
5) When *method* is an unsupervised technique, $K.X$ is the number of latent factors returned and used in further analysis. When *method* is a supervised technique, $K.X$ is used to perform a SVD that facilitates the product of large matrices and inverses. Use $K.Y$ intead, to define number of latent factors.
6) If $K.X$ (or $K.Y$) equal 1, no plots are returned.