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

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**, an 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 **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 **X**. To summarize the main sources of variability across subjects and features, package MOSS performs a sparse singular value decomposition (sSVD).

The following codes illustrate the main capabilities of package 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 three simulated data blocks (see **help(simulate_data)** for a description of the simulation procedure). The following code shows how to load these data and check its classes and dimensions.

```
#Load package MOSS.
library("MOSS")
#>
#>  ---------------------------------------------------------------------
#> |MOSS: Multi-omic integration via sparse singular value decomposition.|
#>  ---------------------------------------------------------------------
#>
#> Agustin Gonzalez-Reymundez, Alexander Grueneberg, and Ana I. Vazquez.
#>
#> Maintainer:
#>   Agustin Gonzalez-Reymundez <gonza650@msu.edu>,
#>
#> Documentation:
#>   Type 'help(moss)' for package overview and examples.
#>
#> Type 'sim.data <- simulate_data()',
#> to generate a small simulated data set with three omics and one categorical variable.

#Simulating omic blocks.
sim_data <- simulate_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 define the 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 code below shows how the data is simulated inside **simulated_data**.

```
#Check dimensions and objects class.
lapply(sim_blocks,dim)
lapply(sim_blocks,function(x) class(x[,1]))

#Showing how the data was generated.
set.seed(42)
O1 <- matrix(data=0, nrow=5e2,ncol=1e3)
O2 <- O1
O1[1:20, 1:150] <- 1
O1 <- O1 + rnorm(5e5, mean=0,sd=0.5)
O2[71:90, 71:200] <- 1
O2 <- O2 + rnorm(5e5, mean=0,sd=0.5)

#Simulating a continous response blocks.
O3 <- 3*O1 - 5*O2 + rnorm(5e5, mean=0,sd=0.5)

#Creating a vector labeling clusters of subjects.
aux <- rep("Background",500)
aux[1:20] <- "Group 1"
aux[71:90] <- "Group 2"
all.equal(aux,lab.sub)

#Generating a classification response.
O4 <- as.matrix(aux)

#Storing all blocks within a list.
all.equal(sim_blocks,list("Block 1" = O1,
   "Block 2" = O2,
   "Block 3" = O3,
   "Block 4" = O4))
```

```
#Creating a vector labeling signal and background features.
aux <- rep("Background features",3000)
aux[c(1:150,1072:1200,2001:2200)] <- "Signal features"
all.equal(aux,lab.feat)
```

The following code allows a nice visualization of the structure within **sim_blocks**, highlighting background *noise* from *signal*, can be obtained with package **ComplexHeatmap**.

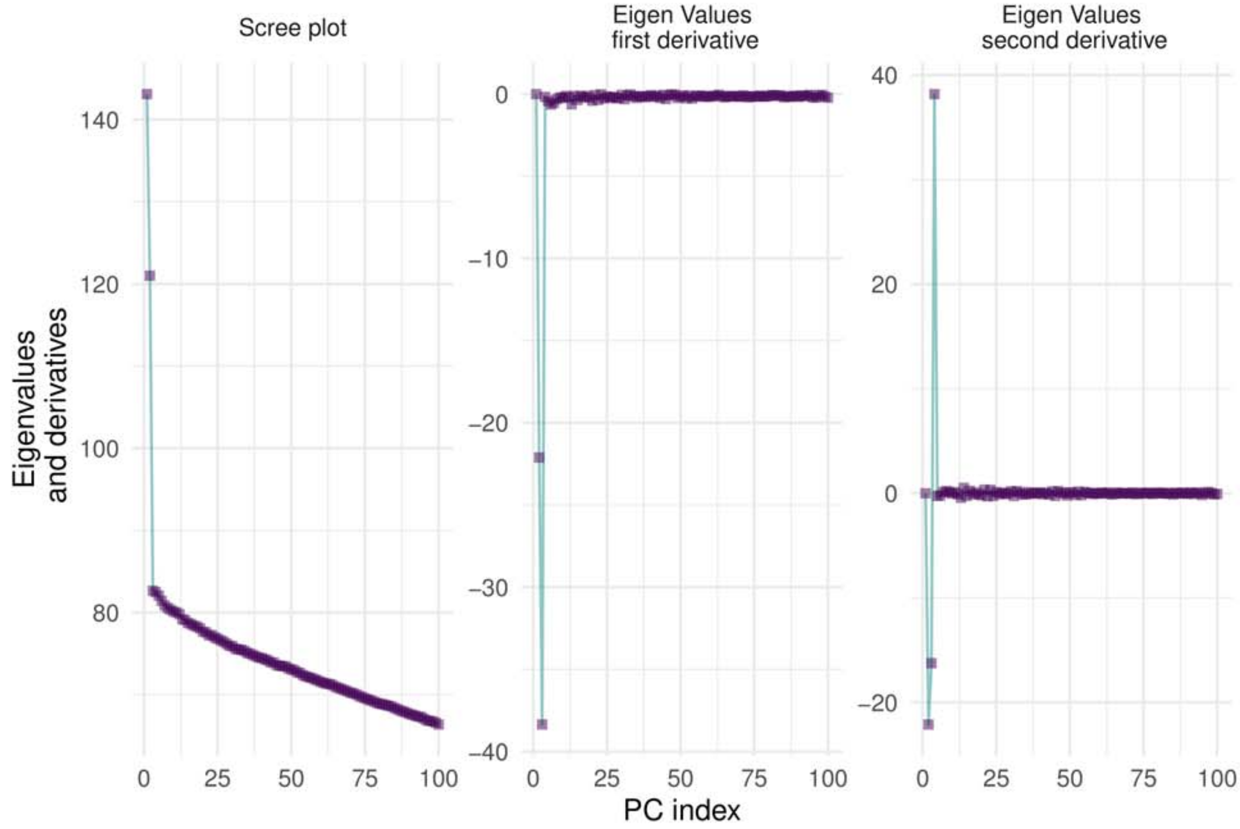
```
#Visualizing the simulated omic blocks with a heatmap.
library("ComplexHeatmap")

#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".

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

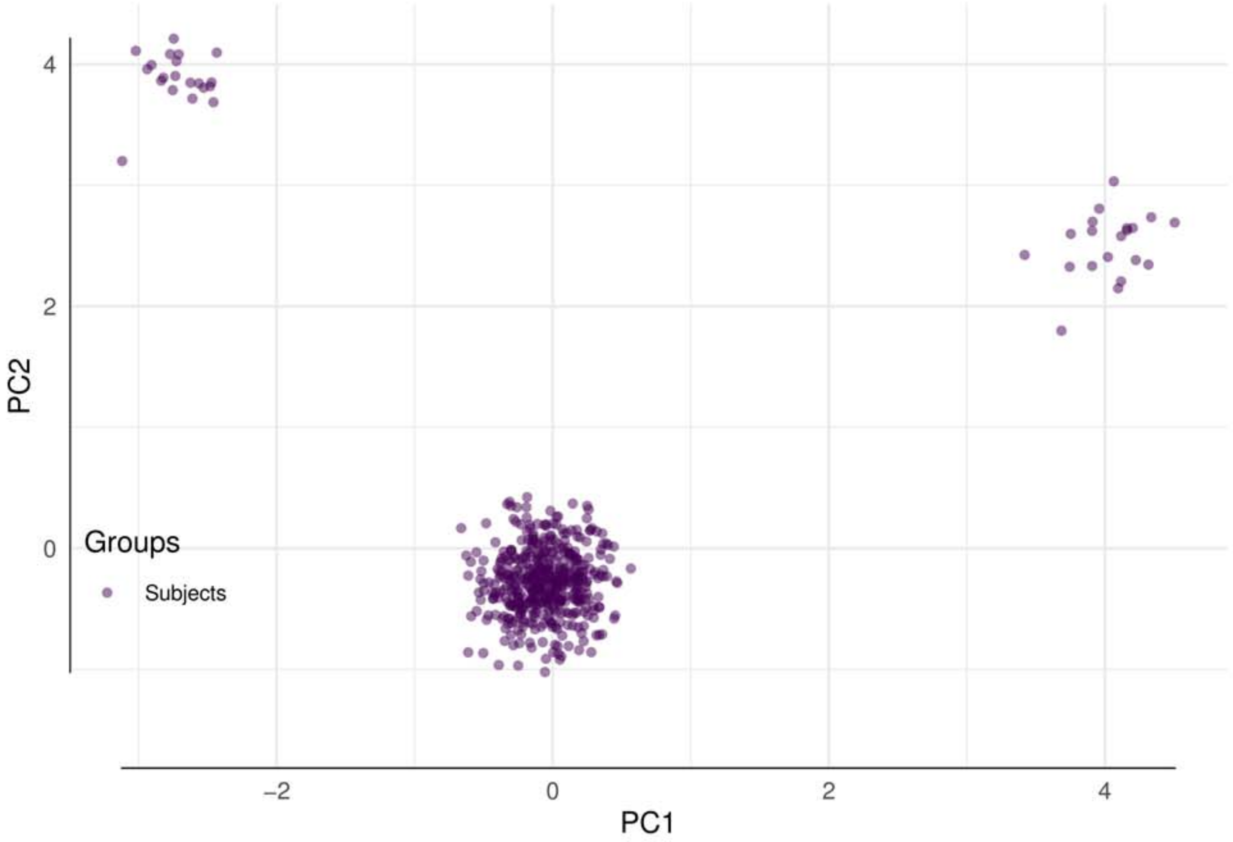
The function **moss** returns a plot of eigenvalues, together with their first and second derivative across PC index.

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



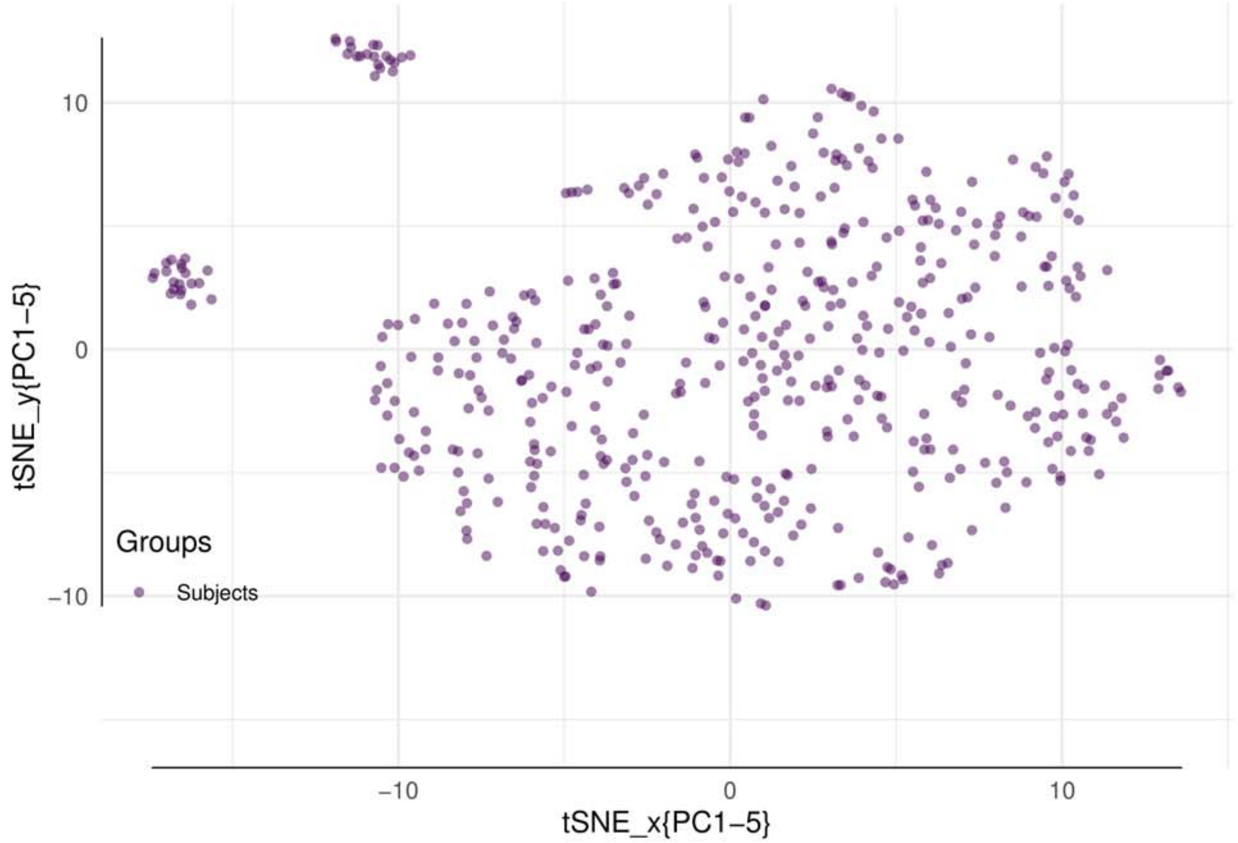
As well as a plot of the first two PCs.

```
#Showing scatterplot of the first two PCs.
out$PC1_2_plot
```



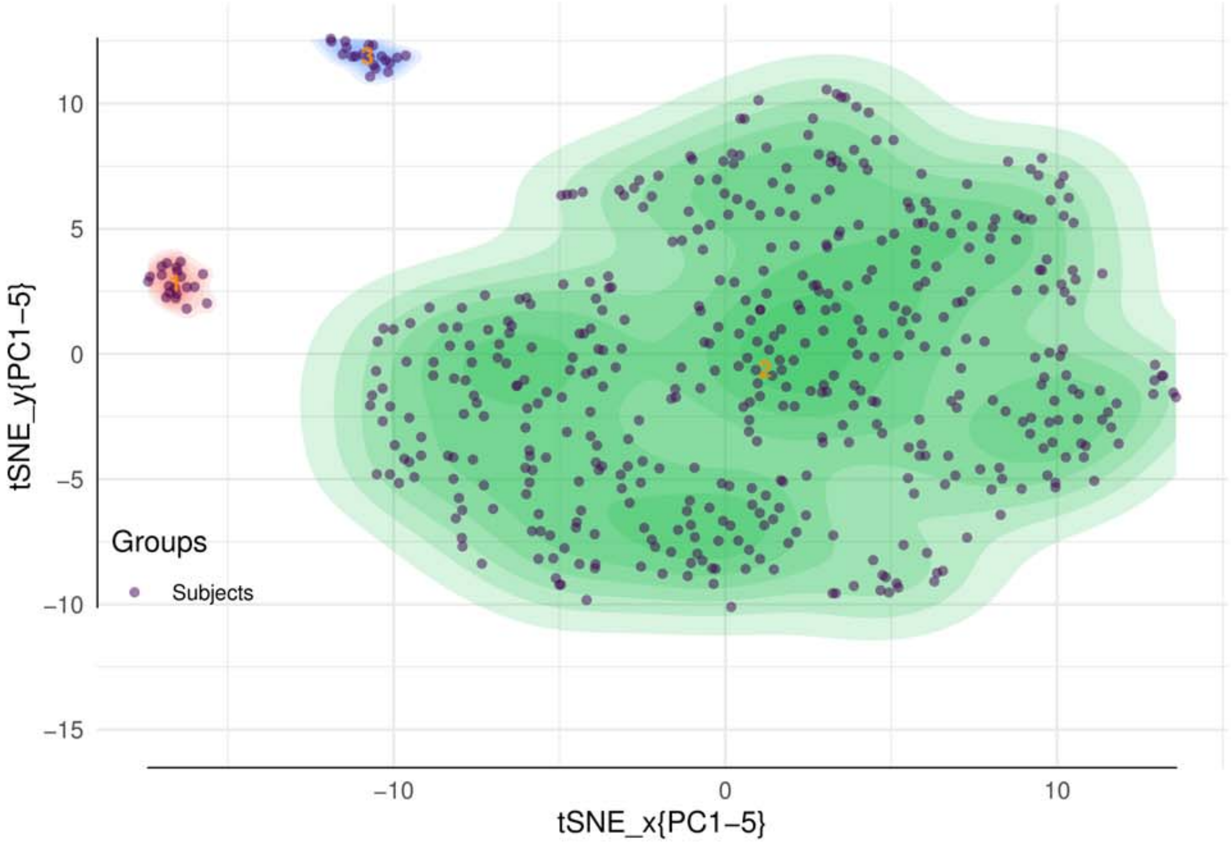
In this simple example, the data structure is clear. However, if the user wishes to map a set of more than two PC, the option *tSNE* can be used. This tell **MOSS** to call **Rtsne** to map more than two PCs onto a two dimensional display. By using *tSNE* = TRUE, one random realization of t-stochastic 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.

```
set.seed(43)
out <-  moss(
  data.blocks = sim_blocks[-4], #a list with omic blocks
        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)
#> ------------------------------------------------------------------------------
#> | Principal Components Analysis (PCA) for 3 data blocks and 5 latent factors |
#> ------------------------------------------------------------------------------
#> Checking for missing values.
#> 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
```



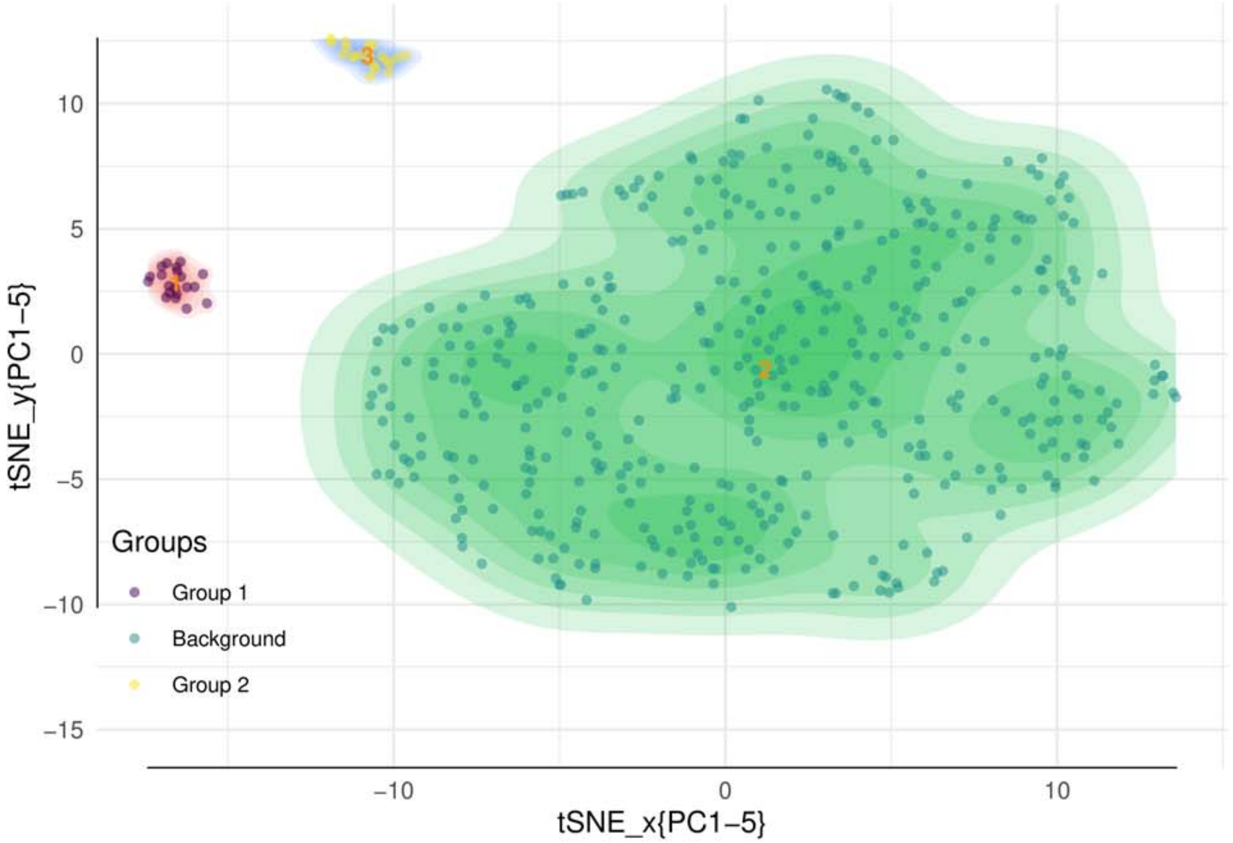
The user can also use the option *cluster*=TRUE within *moss*. This option allows **MOSS** to perform a cluster delimitation using DBSCAN, inspired by the **MEREDITH** procedure. For more personalized aesthetic and labeling, check **help(tsne2clus)**.

```
set.seed(43)
out <-  moss(
  data.blocks = sim_blocks[-4], #a list with omic blocks
        method="pca", #This is the default method.
        tSNE=list("perp"=50, "n.samples"=1, "n.iter"=1e3), #Mapping first five PC onto 2_D.
        cluster=T, #Delimiting cluster via DBSCAN.
        plot=TRUE)
#> ------------------------------------------------------------------------------
#> | Principal Components Analysis (PCA) for 3 data blocks and 5 latent factors |
#> ------------------------------------------------------------------------------
#> Checking for missing values.
#> 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 association between PCs and detected clusters.
out$clus_plot
```



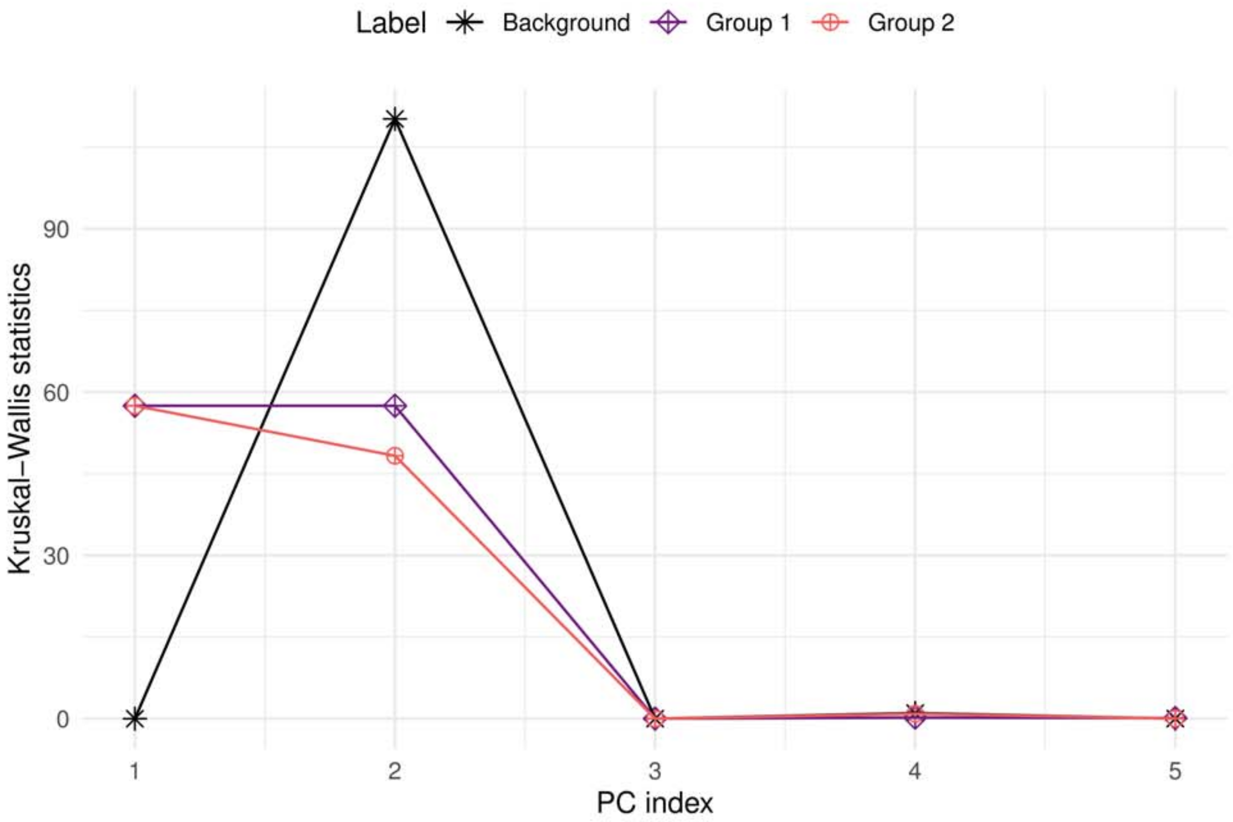
If information of pre-defined and biologically relevant groups is available (e.g. histological sub-types of cancer), the user can *1)* use it to label points and visually evaluate the overlap between data-driven clusters and previous information and *2)* use it to determine which PCs are significantly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

```
set.seed(43)
out <-  moss(
  data.blocks = sim_blocks[-4], #a list with omic blocks
        method="pca", #This is the default method.
        tSNE=list("perp"=50, "n.samples"=1, "n.iter"=1e3), #Mapping first five PC onto 2_D.
        cluster=T, #Delimiting cluster via DBSCAN.
        clus.lab = lab.sub, #We'll use the info of 'signal' subjects as an example.
        plot=TRUE)
#> ---------------------------------------------------------------------------
#> | Principal Components Analysis (PCA) for 3 data blocks and 5 latent factors |
#> ---------------------------------------------------------------------------
#> Checking for missing values.
#> 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 association between PCs selected and pre-established labels.
#> Evaluating association between PCs and detected clusters.
out$clus_plot
```



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_PCs
```

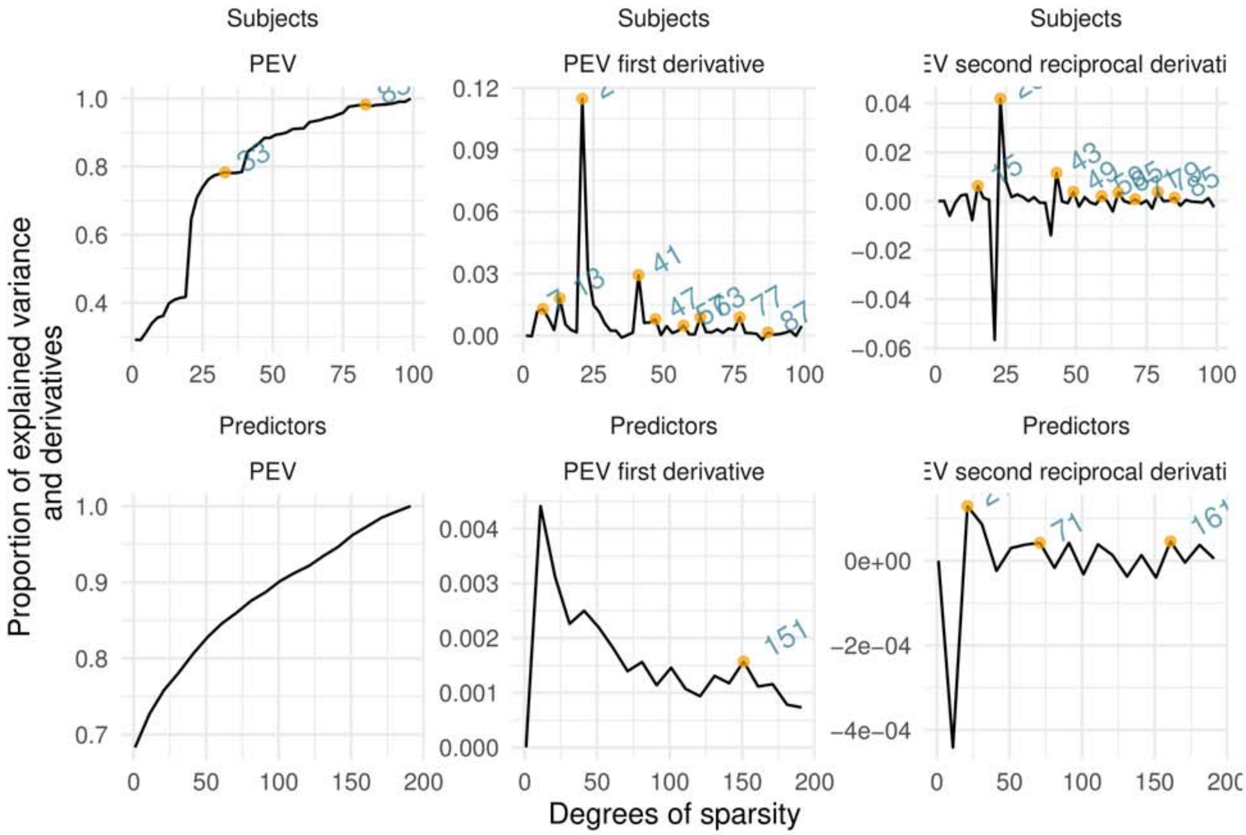


Now, suppose that the user is interested in detecting what groups of subjects and features form regulatory modules. In this example, we tune the degree of sparsity for both subjects and features, using an elastic net parameter *alpha.* equal to one, for subjects (i.e. a LASSO penalty), and 0.5 for features.

```
set.seed(43)
out <-  moss(data.blocks = sim_blocks[-4], #Feed moss a list with omic blocks
        method="pca", #This is the default method.
        K.X = 5, #Number of PC (Defaults to 5).
        nu.v = seq(1,200,by=10), #This is a grid of increasing sparsity values for features.
        nu.u = seq(1,100,by=2), #Same, but for subjects.
        alpha.v = 0.5, #This is the EN parameter for features.
        alpha.u = 1, #This is the EN parameter for subjects. Here we're doing LASSO.
        tSNE=TRUE, #This tells moss to project the 5 PC onto 2-D via tSNE.
        cluster=TRUE, #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)
#> ------------------------------------------------------------------------------
#> | Principal Components Analysis (PCA) for 3 data blocks and 5 latent factors |
#> ------------------------------------------------------------------------------
#> Checking for missing values.
#> Standardizing/Normalizing data blocks.
#> Elastic net shrinking & selection in RIGHT eigenvectors.
#> LASSO in LEFT eigenvectors.
#> 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 groups of selected subjects and pre-established labels.
#> Evaluating overlap between groups of selected subjects and detected clusters.
```

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
```



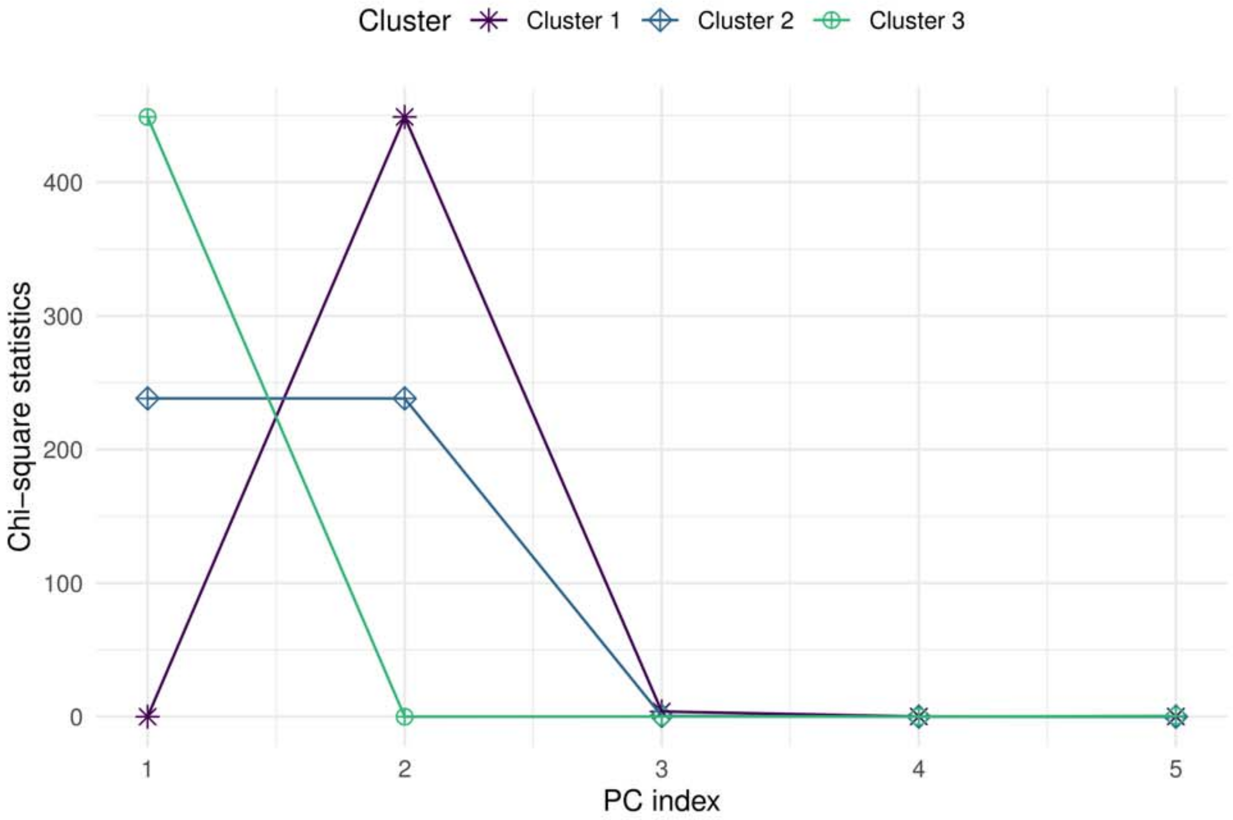
We now can use the solutions **MOSS** for the selected *dg* for subjects and/or features and and plot them against elements of *lab.sub* and *lab.feat* to evaluate how good **MOSS** does to identify relevant biological signals (see **help(moss)**).

```
library("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.
```

```
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)
```

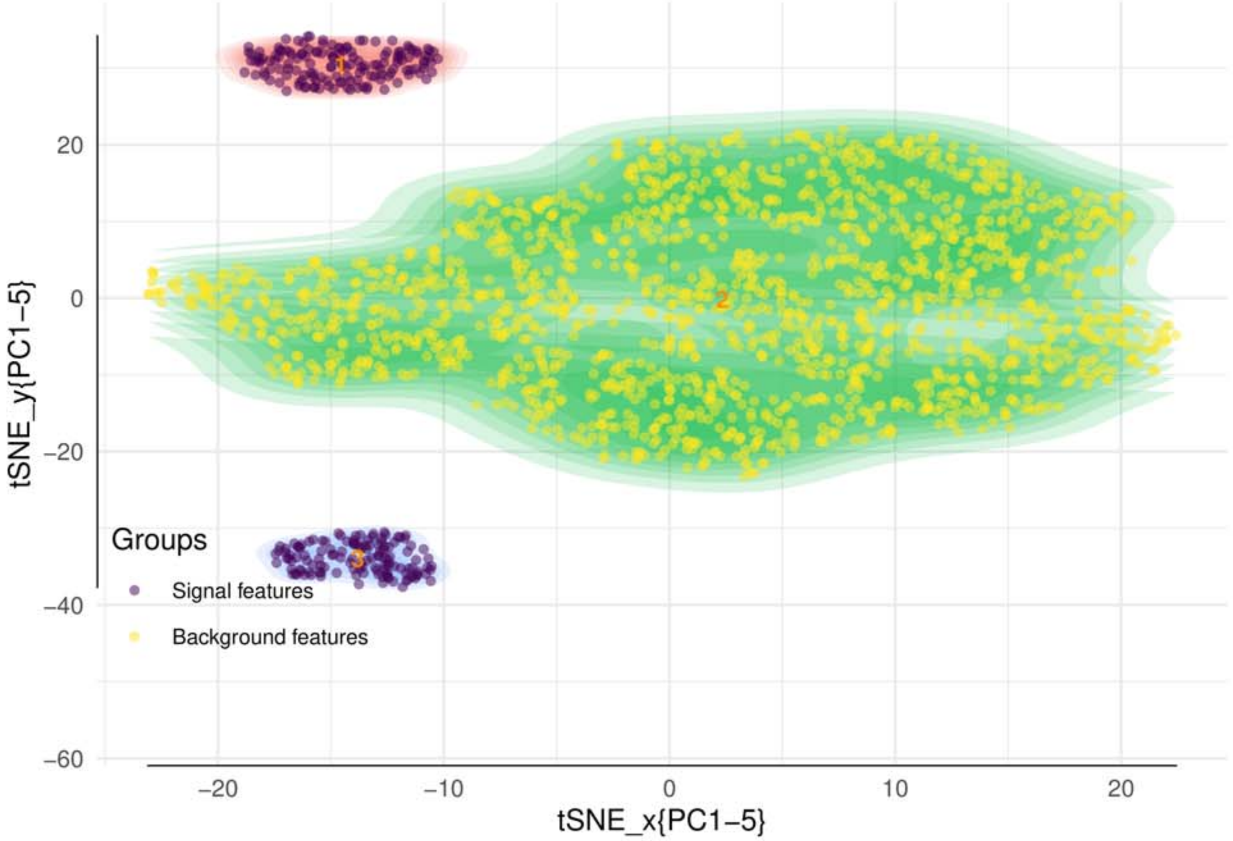
The above results offer a good example of how the performance of the selection task can be closely tied to how many and what PCs are used to identify regulatory modules. In this simple example, the scree plot suggests that the two first PCs are ones explaining most of the data variability. If we are 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*. Independently of having specified labels of subjects, whenever *cluster != FALSE*, the associations between clusters and PCs are stored in:

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



When one of the omic blocks is a multivariate response, MOSS can perform partial least squares. This specified by *method = pls*. The next codes shows how to run a PLS taking the third omic blocs as response.

```
set.seed(43)
out <-  moss(data.blocks = sim_blocks[-4], #Feed moss a list with omic blocks
        method="pls",
        K.X = 5, #Number of PC (Defaults to 5).
        nu.v = seq(1,200,by=10), #This is a grid of increasing sparsity values for features.
        nu.u = seq(1,100,by=2), #Same, but for subjects.
        alpha.v = 0.5, #This is the EN parameter for features.
        alpha.u = 1, #This is the EN parameter for subjects. Here we're doing LASSO.
        tSNE=TRUE, #This tells moss to project the 5 PC onto 2-D via tSNE.
        cluster=TRUE, #This tells moss to use DBSCAN to delimt clusters on the tSNE map
        clus.lab=lab.feat[1:2000], #This colors features in the predictors according to the labels with
        plot=TRUE,
        resp.block=3)# Using the third block as a multivariate response.
#> -----------------------------------------------------------------------
#> | Partial Least Squares (PLS) for 3 data blocks and 5 latent factors |
#> -----------------------------------------------------------------------
#> Checking for missing values.
#> Block 3 used as response.
#> Standardizing/Normalizing data blocks.
#> Elastic net shrinking & selection in RIGHT eigenvectors.
#> LASSO in LEFT eigenvectors.
#> Getting SVD of predictors matrix (dimension 500 x 2000).
#> Creating matrix 'R' (dimension 5 x 1000).
#> Creating matrix 'L' (dimension 2000 x 5).
#> Getting 'L*R' (dimensions 2000 x 1000).
#> Getting SVD of LR.
#> Imposing sparsity constraints.
#> Calculating a tSNE map
#> Embedding 5 axes onto two dimensions.
#> Getting clusters via DBSCAN.
#> Evaluating overlap between groups of selected subjects and pre-established labels.
#> Evaluating overlap between groups of selected subjects and detected clusters.
out$clus_plot
```



In all the above examples we used conventional R matrices to define omic blocks. MOSS allows the user to handle larger data sets by means of File-backed Big Matrices (FBM). If for example, omic blocks are stored in FBM, or if argument *use.fbm=TRUE*, MOSS uses package bigstats to calculate the first SVD and perform the matrix multiplications needed to impose sparsity. The following code allows MOSS to turn matrices into FBMs internally.

```
set.seed(43)
out <-  moss(data.blocks = sim_blocks[-4], #Feed moss a list with omic blocks
        method="pls",
        K.X = 5, #Number of PC (Defaults to 5).
        nu.v = seq(1,200,by=10), #This is a grid of increasing sparsity values for features.
        nu.u = seq(1,100,by=2), #Same, but for subjects.
        alpha.v = 0.5, #This is the EN parameter for features.
        alpha.u = 1, #This is the EN parameter for subjects. Here we're doing LASSO.
        tSNE=TRUE, #This tells moss to project the 5 PC onto 2-D via tSNE.
        clus=TRUE, #This tells moss to use DBSCAN to delimt clusters on the tSNE map
        clus.lab=lab.feat[1:2000], #This colors features in the predictors according to the labels with
        plot=TRUE,
        resp.block=3, # Using the third block as a multivariate response.
        use.fbm=TRUE)
out$clus_plot
```