sPLS model axes interpretation for 10x Single-cell RNA-Seq

Hadrien LORENZO and Robin GENUER

29/06/2017

Contents

Import datasets

PLS, sPLS and sPLS-DA

(Lê Cao, Boitard, and Besse 2011)

As a reminder, each axis of a **PLS** problem solves the following optimization problem :

$$\max_{\substack{\boldsymbol{u}^T\boldsymbol{u}=1\\ \boldsymbol{v}^T\boldsymbol{v}=1}} cov(\boldsymbol{X}\boldsymbol{u}, \boldsymbol{Y}\boldsymbol{v}) = \max_{\substack{\boldsymbol{u}^T\boldsymbol{u}=1\\ \boldsymbol{v}^T\boldsymbol{v}=1}} \boldsymbol{u}^T\boldsymbol{X}^T\boldsymbol{Y}\boldsymbol{v}.$$

The deflation permits to repeat that problem over successive axes in orthonormal ways.

The **sPLS** is the L_1 modified optimization problem, like for the LASSO, which constraints the weights to be smaller than the unconstrained problem (**PLS**).

sPLS-DA is a **PLS** method which account to answer to **2** questions in plus than the classical **PLS** model :

- Deal with classes instead of quantitative variables,
- Select variables for each axis.

So, the **sPLS-DA** permits to select variables on different axes which will discriminate the different classes. For this we will have to tune two parameters :

- $keep_X$ per component : the number of genes selected per component,
- n_{comp} the total number of components.

Remark : In the **sPLS** problem we do not have to tune $keep_X$ because we fix it to the number of variables.

Back to the 10X dataset!

We have decided to deal with the **10X** dataset and to treat firstly the case of 4 populations which are easily separable, which are :

- **b-cells**
- **cd14_monocytes**
- **cd34**
- **cd56_nk**

So, first of all we create new datasets with only the considered cells. We have to standardize the **X** dataset :

```
indices_4_pop <- which(y %in%  c("b_cells",
                                 "cd14_monocytes",
                                 "cd34",
                                 "cd56_nk"))
X_4_pop <- scale(X[indices_4_pop,])
y_4_pop <- droplevels(y[indices_4_pop])
```

Tune the parameters

The tricky part of the algorithm is to tune the different parameters n_{comp} and $keep_X$.

The common way is to tune $keep_X$ on the 1^{st} component, then do the deflation, then tune $keep_X$ on the 2^{nd} component and re-apply up to a *coherent* value for n_{comp}.

To fix n_{comp} we will most of times use the rule of thumb : n_{comp}**=K-1**, where **K** is the number of classes in the dataset.

Here **K=4** so we will build

$$n_{comp} = 3,$$

different components.

Now we have to find a way of selecting the number of variables per axis using a validation criterion. We wil use here the quality of classification in a cross-validation based method.

We will use

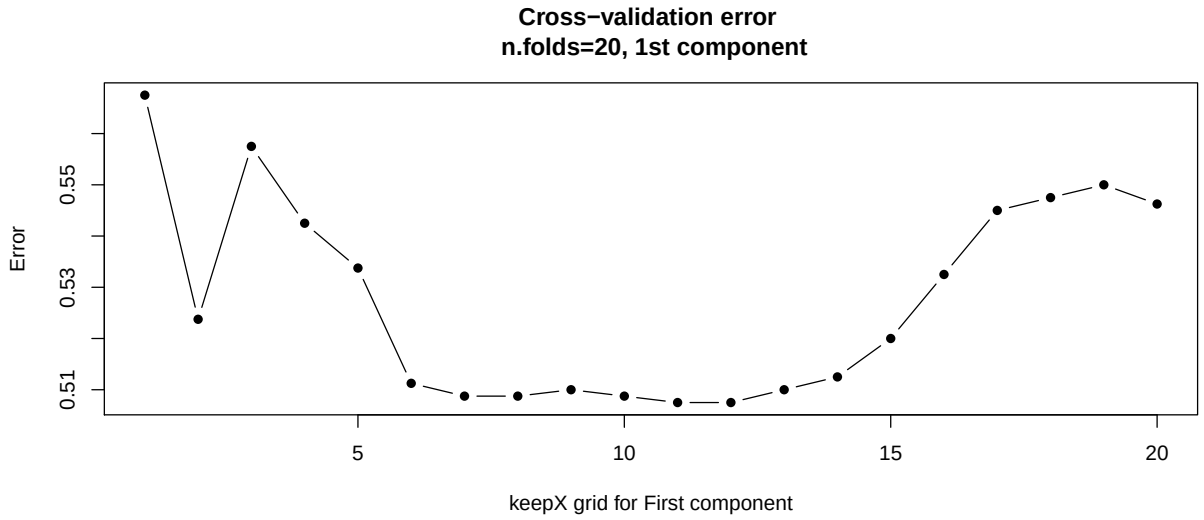
- $n.folds = 20$ cross-validation,
- $keep_{Xs} = 1 : 20$ the grid over which we test the model.

Tune $keep_{X_1}$: the number of variables selected along the first axis

Please use the function *crossValidate_splsDA* to perform cross-validation.

```
n.folds <- 20
keepXs <- 1:20

ncomp <- 1
previsous.keepX <- NULL
errors <- crossValidate_splsDA(X_4_pop,y_4_pop,
                               ncomp=ncomp,
                               keepXs=keepXs,
                               previsous.keepX=previsous.keepX,
                               n.folds=n.folds)
plot(errors,pch=16,type="b",
     xlab="keepX grid for First component",ylab="Error", main="Cross-validation error \n n.folds=20, 1s
```



. . . it took a few times but you found certainly that $keep_X \in (2, 7)$ are quite good possibilities. Tell us if you found anything different. . .

Whatever we will select the following parameter :

$$keep_{X_1} = 7.$$

Get $keep_{X_2}$ and $keep_{X_3}$

As we do not want to waste to much time, we have performed the other cross-validation procedures. The code is here :

```
# ncomp <- 2
# previsous.keepX <- 7
# errors <- crossValidate_splsDA(X_4_pop,y_4_pop,
#                                ncomp=ncomp,
#                                keepXs=keepXs,
#                                previsous.keepX=previsous.keepX,
#                                n.folds=20)
```

```
# plot(errors,pch=16,type="b")
#
# ncomp <- 3
# previsous.keepX <- c(7,4)
# errors <- crossValidate_splsDA(X_4_pop,y_4_pop,
#                                ncomp=ncomp,
#                                keepXs=keepXs,
#                                previsous.keepX=previsous.keepX,
#                                n.folds=20)
# plot(errors,pch=16,type="b")
```

Which permits to select the following values :

$$(keep_{X_2}, keep_{X_3}) = (5, 6)$$

Check the final model

As we have selected the parameters which gives a good level of satisfaction in terms of prediction, we want to check out what we selected.

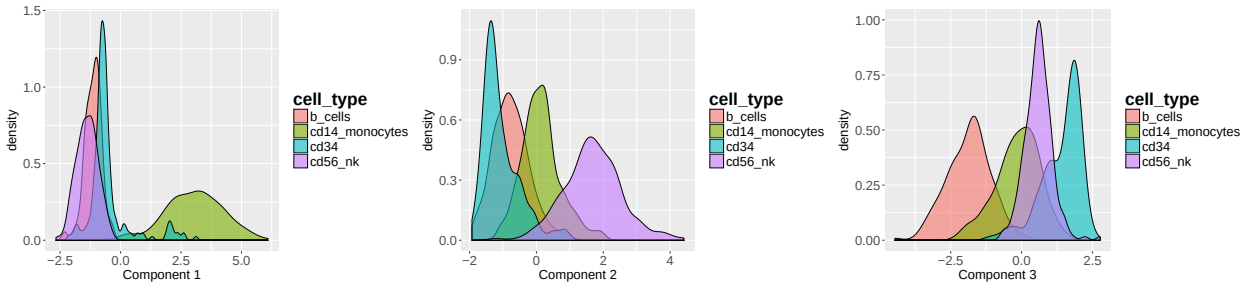
Model builing

We build the model as follows :

```
ncomp <- 3
modele <- splsda(X_4_pop,y_4_pop,ncomp = ncomp,keepX = c(7,5,6))
```

We get the following per axis representation :

```
plots <- list()
for(i in 1:ncomp){
  dat <- data.frame(VariateX = modele$variates$X[,i], cell_type = y_4_pop)
  a <- ggplot(dat, aes(x = VariateX, fill = cell_type)) +
    geom_density(alpha = 0.6)+theme(
      legend.title = element_text(size = 20, face = "bold"),
      legend.text = element_text(size = 15),
        axis.title = element_text(size=15),
        axis.text = element_text(size=15))+
    xlab(paste("Component",i))
  plots[[i]] <- a
}
do.call(gridExtra::grid.arrange,  c(grobs=plots, ncol=3))
```



A few things seem clear over the discrimination of the axes against the cell types :

- **Component 1** : Discriminates **CD 14** versus the others
- **Component 2** : Discriminates **CD 56** versus the others
- **Component 3** : Discriminates **CD 34 & B-cells** versus the others

Selected variables

For example we might want to get the gene names selected. For this we can do so :

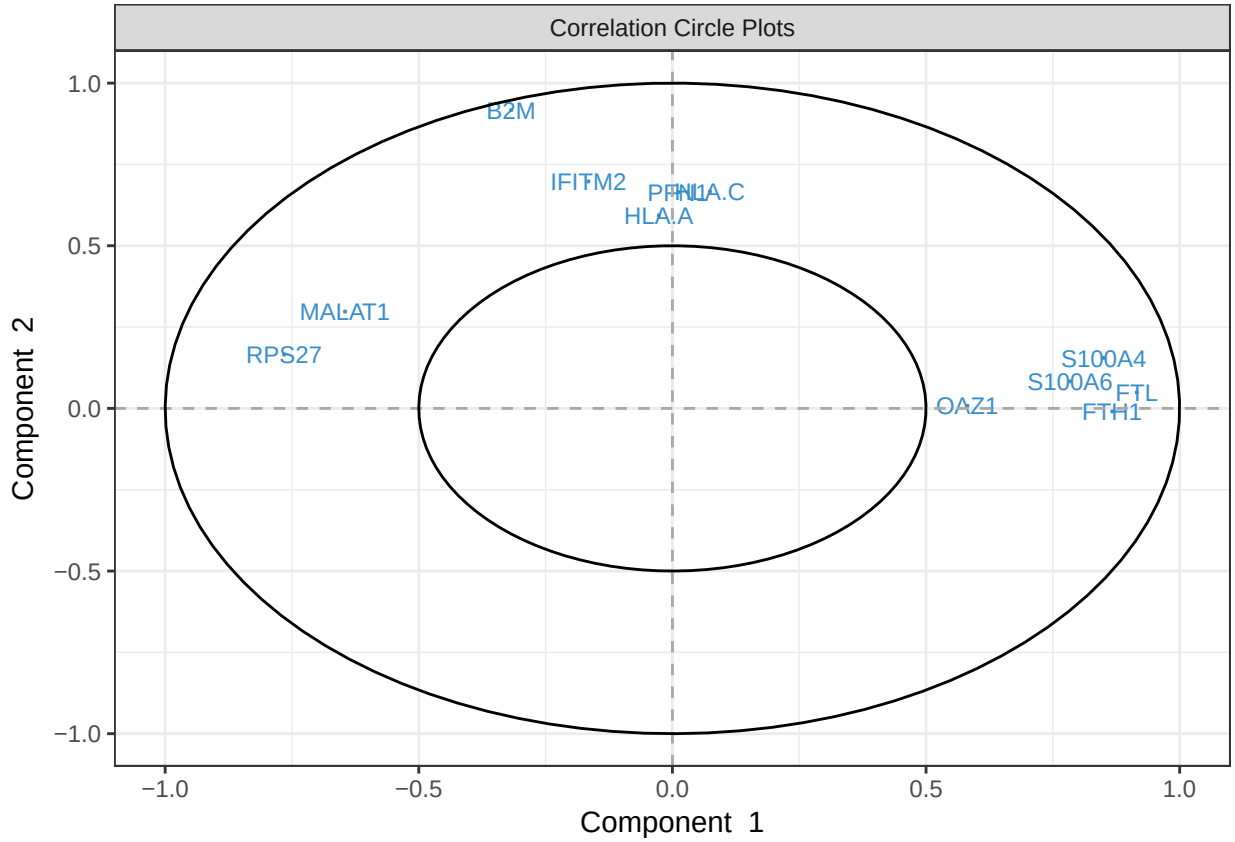
```
matGenes <- round(modele$loadings$X[-which(rowSums(modele$loadings$X)==0),],3)
matGenes[which(matGenes==0)] <- ''
kable(matGenes)
```

	comp 1	comp 2	comp 3
CD52			-0.084
S100A6	0.251		
S100A4	0.447		
RPS27	-0.349		
CD74			-0.846
HLA.A		0.006	
HLA.C		0.13	
LTB			-0.346
EEF1A1			-0.311
IFITM2		0.254	
FTH1	0.47		
MALAT1	-0.194		
B2M		0.943	
RPL13			-0.131
PFN1		0.172	
OAZ1	0.05		
FTL	0.595		
CD37			-0.212

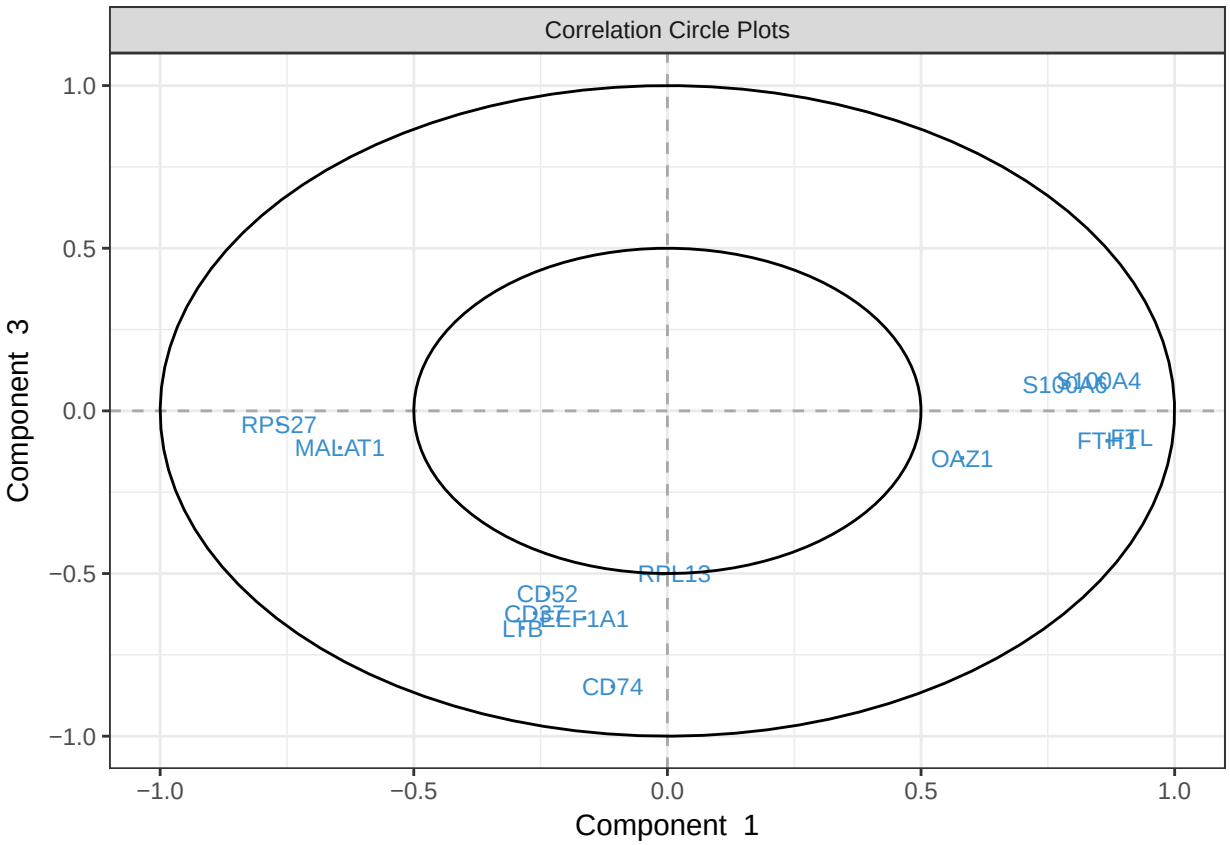
PlotVar representation

It permits to represent the weights in two-dimensionnal figures.

```
plotVar(modele,comp = 1:2,cex=3)
```




```
plotVar(modele,comp = c(1,3),cex=3)
```

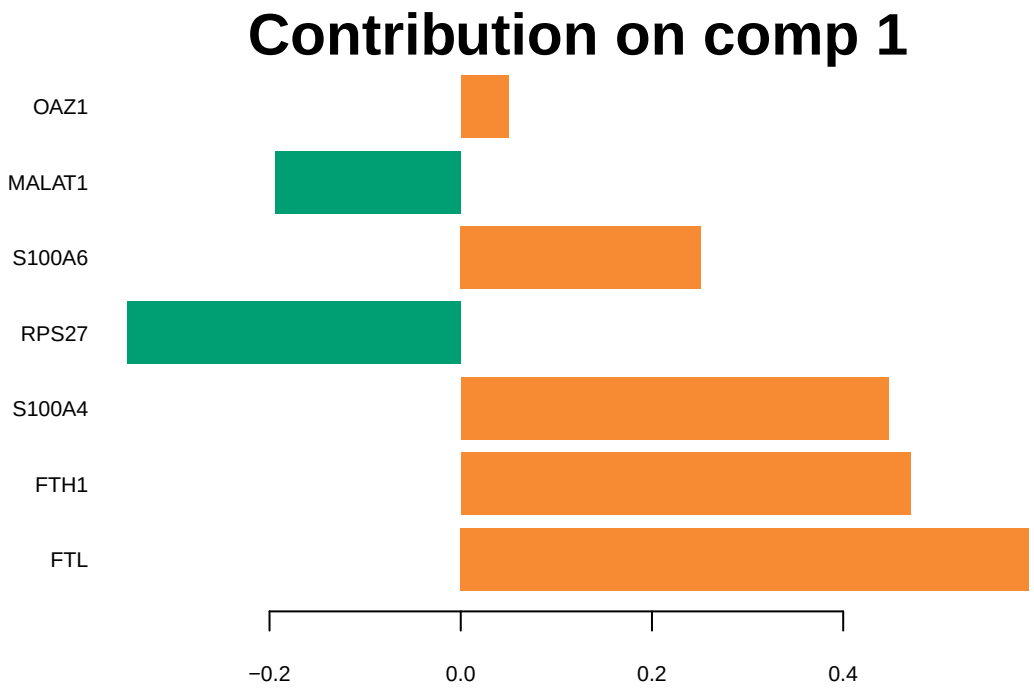


What are the problems of that representation ?

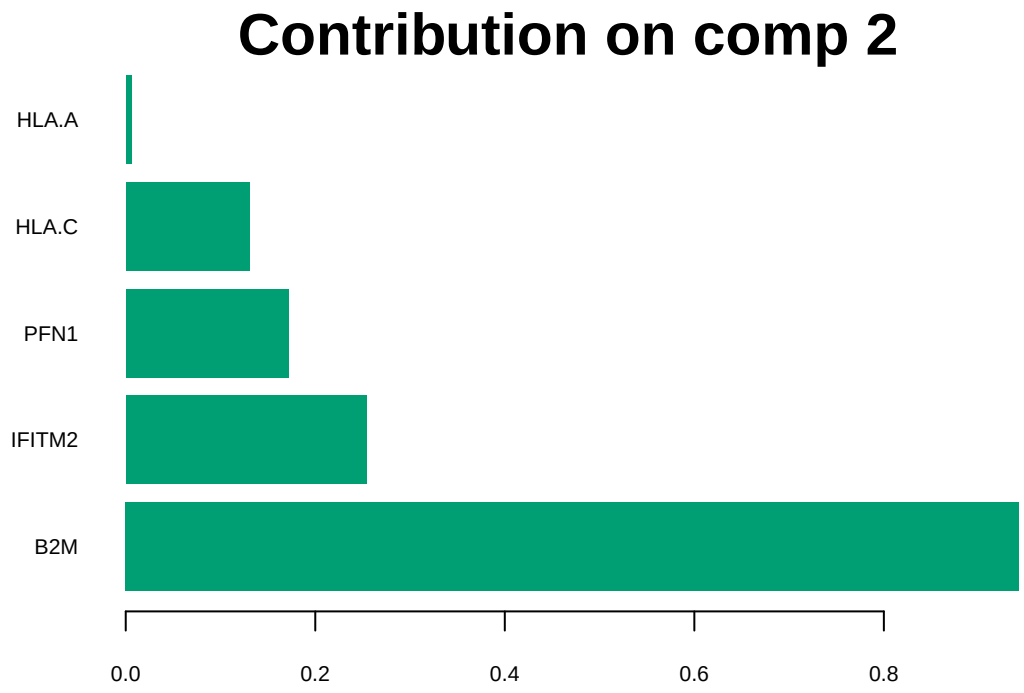
Contribution plots

Those plots are quite fancy to check the weights of each variable selected along its component

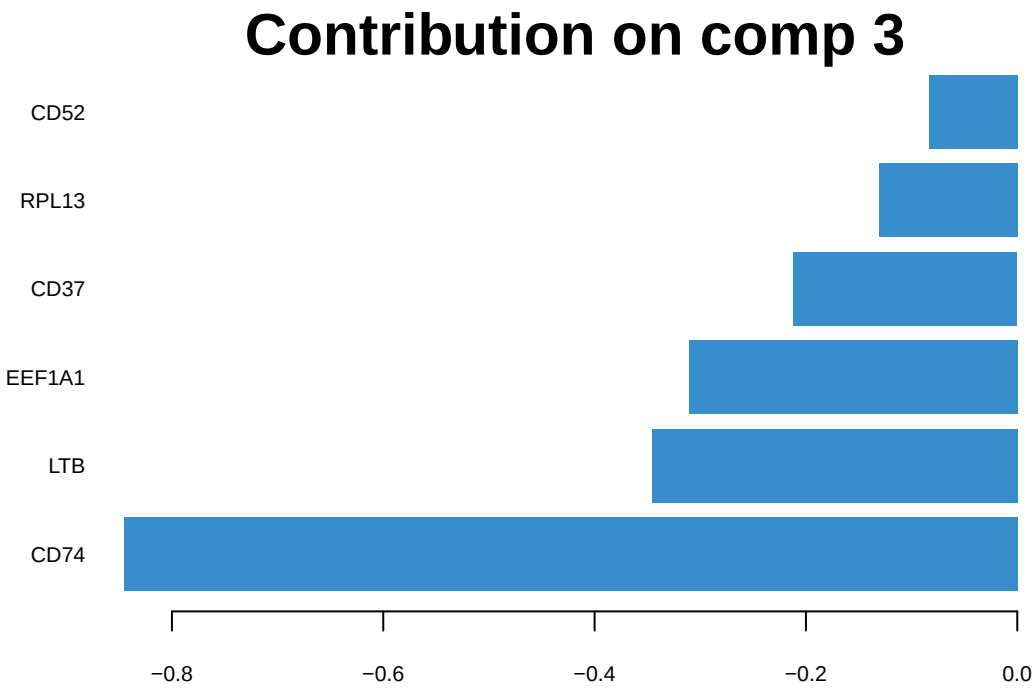
```
plotLoadings(modele, comp = 1, method = 'mean', contrib = 'max',legend=F)
```




```
plotLoadings(modele, comp = 2, method = 'mean', contrib = 'max',legend=F)
```



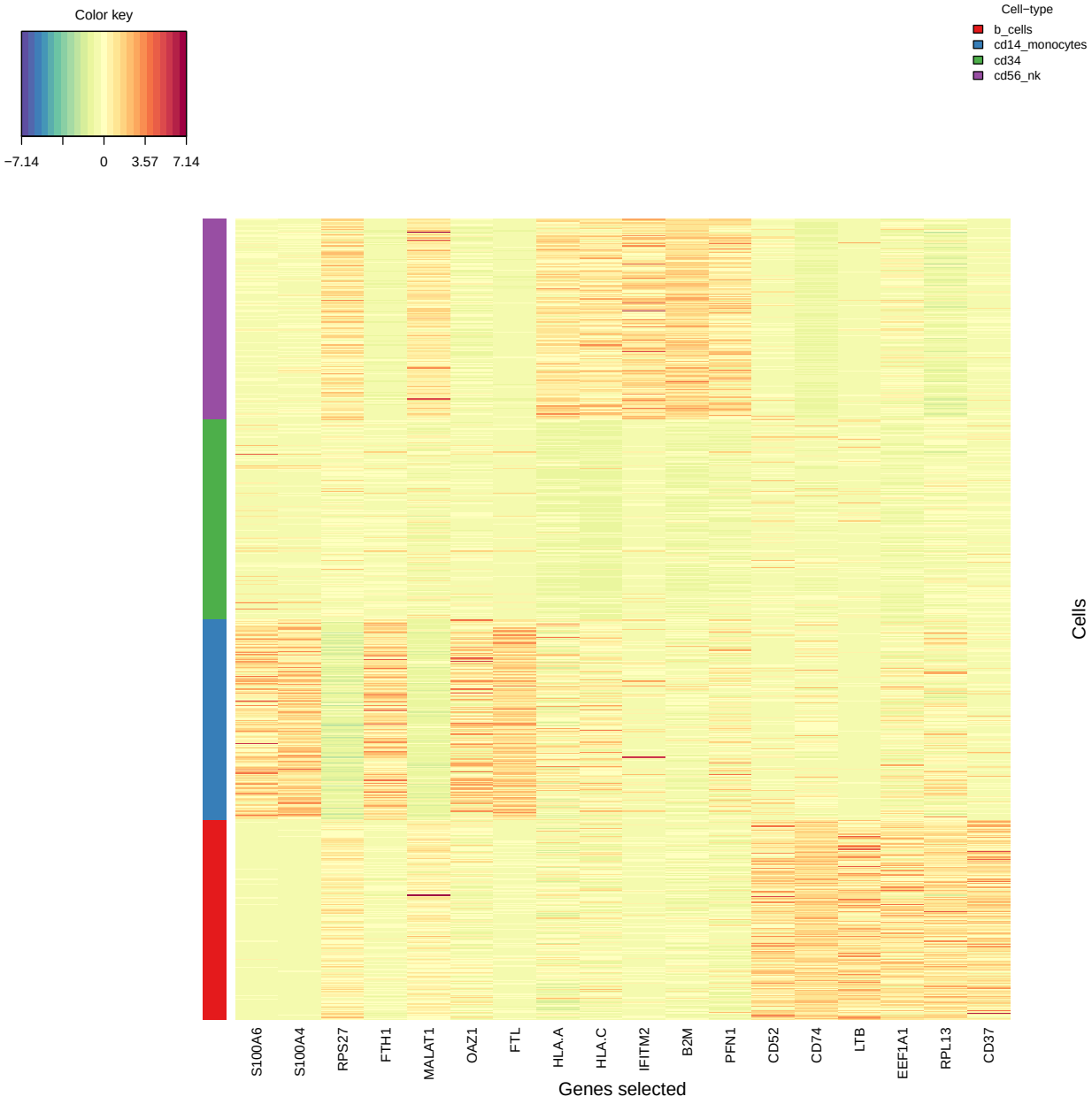

```
plotLoadings(modele, comp = 3, method = 'mean', contrib = 'max',legend=F)
```



cim representation

An interpretable way of representing the selected genes is to use **cim** representation from mixOmics. Actually it represents the values

```
matGenes <- modele$loadings$X[-which(rowSums(modele$loadings$X)==0),]
modele_cim <- plsda(X_4_pop[,
                            which(rowSums(abs(modele$loadings$X))!=0)[
                              order(apply(matGenes,1,function(a){which(a!=0)}))]],
                    y_4_pop,ncomp = 3)
cim(modele_cim,row.names = NA,
    row.sideColors = brewer.pal(4,name = "Set1")[y_4_pop],
    cluster = "none",
    xlab = "Genes selected",ylab = "Cells",
    legend=list( legend = levels(y_4_pop),
    col = brewer.pal(4,name = "Set1"),title = "Cell-type", cex = 0.7))
```



How can you comment that figure ?

Can you find information redundant with previous figures ?

Variances explained

In mixOmics we calculate the variance explain by each component related to each data (X or Y).

```
modele$explained_variance
```

```
## $X
##     comp 1     comp 2     comp 3
## 0.09226848 0.08613825 0.04301901
##
```

```
## $Y
##    comp 1    comp 2    comp 3
## 0.3333333 0.3331567 0.3312492
```

Are the variances increasing or decreasing ?

Is that necessarly the case

Predict functions

PLS-based methods permit to construct a regression model.

For example, if we want to test a few predictions :

```
id_test <- c(1,2,5,6,5,22,20,12,55,256,758,726,540,265,799)
predicted_classes <- predict(object = modele,newdata = X_4_pop[id_test,] )
# compare prediction to reality
kable(table(predicted_classes$class$max.dist[,ncomp], y_4_pop[id_test]))
```

	b_cells	cd14_monocytes	cd34	cd56_nk
b_cells	9	0	0	0
cd14_monocytes	0	2	0	0
cd34	0	0	1	0
cd56_nk	0	0	0	3

Do the same with the all dataset

No cheat here but ask for help if needed!

References

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). BioMed Central: 253.