The OmicsPLS R Package

Said el Bouhaddani

2016-11-08

The OmicsPLS R package

Welcome to the vignette of the O2PLS package for analyzing two Omics datasets!

Here you can find examples and explanation of the input options and output objects. As always: help is always found by using the `?` operator. Try to type `?OmicsPLS` for an overview of the package and `?o2m` for description of the main fitting function.

Background

The O2PLS method

The O2PLS method is proposed in (Trygg & Wold, 2003). It decomposes the variation of two datasets in three parts:

- A Joint part for X and Y: $TW^\top$ and $UC^\top$,
- A Systematic/Specific/Orthogonal part for X and Y: $T_\perp W_\perp^\top$ and $U_\perp C_\perp^\top$,
- A noise part for X and Y: E and F.

The number of columns in T, U, W and C are denoted by as n and are referred to as the number of joint components. The number of columns in $T_\perp$ and $W_\perp$ are denoted by as n_X and are referred to as the number of X-specific components. Analoguous for Y, where we use n_Y to denote the number of Y-specific components. The relation between T and U makes the joint part the joint part: $U = TB + H$ or $U = TB' + H'$. The number of components (n, n_X, n_Y) are chosen beforehand (e.g. with Cross-Validation).

Cross-Validation

In cross-validation (CV) one minimizes a certain measure of error over some parameters that should be determined a priori. In our case we have three parameters: (n, n_X, n_Y). A popular measure is the prediction error $||\hat{Y} - Y||$, where $\hat{Y}$ is a prediction of Y. In our case the O2PLS method is symmetric in X and Y, so we minimize the sum of the prediction errors: $||\hat{X} - X|| + ||\hat{Y} - Y||$. The idea is to fit O2PLS to our data X and Y and compute the prediction errors for a grid of values for n, n_X and n_Y. Here n should be a positive integer, and n_X and n_Y should be non-negative. The 'best' integers are then the minimizers of the prediction error.

Proposed cross-validation approach

We proposed an alternative way for choosing the number of components (el Bouhaddani, 2016). Here we construct a grid of values for n. For each n we consider then the R^2 between T and U for different n_X and n_Y. If T and U are contaminated with data-specific variation the R^2 will be lower. If too many specific components are removed the R^2 will again be lower. Somewhere in between is the maximum, with its maximizers n_X and n_Y. With these two integers we now compute the prediction error for our n that we have kept fixed. This process we repeat for each n on the one-dimensional grid and get our maximizers. This can provide a (big) speed-up and often yields similar values for (n, n_X, n_Y).

Installing and loading

The easiest way is to run `devtools::install_github("selbouhaddani/OmicsPLS")`. If this doesn't work, check if there is a package missing. It imports the **ggplot2** and **parallel** package, so these should be installed first. If there still is an error, try to download the .tar or .zip (for Windows) and install offline. These two files can be found also in the *selbouhaddani/ZippedPackage* repository. Also feel free to send an email with the error message you are receiving.

The OmicsPLS package is loaded by running `library(OmicsPLS)`. Maybe you get a message saying that the `loadings` object is masked from `package::stats`. This basically means that whenever you type `loadings` (which is generic), you'll get the `loadings.o2m` variant.

A first test case

First we generate some data

```
set.seed(564785412L)
X = rnorm(100) %*% t(c(rep(1,5), rep(0,45))/sqrt(5)) + # Component 1 = joint
  rnorm(100) %*% t(c(rep(0,45), rep(1,5))/sqrt(5)) # Component 2 = specific
Y = X[,c(6:25, 1:5, 26:45)] # Reorder columns of X and leave out last 5
X = X + matrix(rnorm(100*50), nrow=100) # add noise
Y = Y + matrix(rnorm(100*45), nrow=100) # add noise

X = scale(X, scale=F)
Y = scale(Y, scale=F)
```

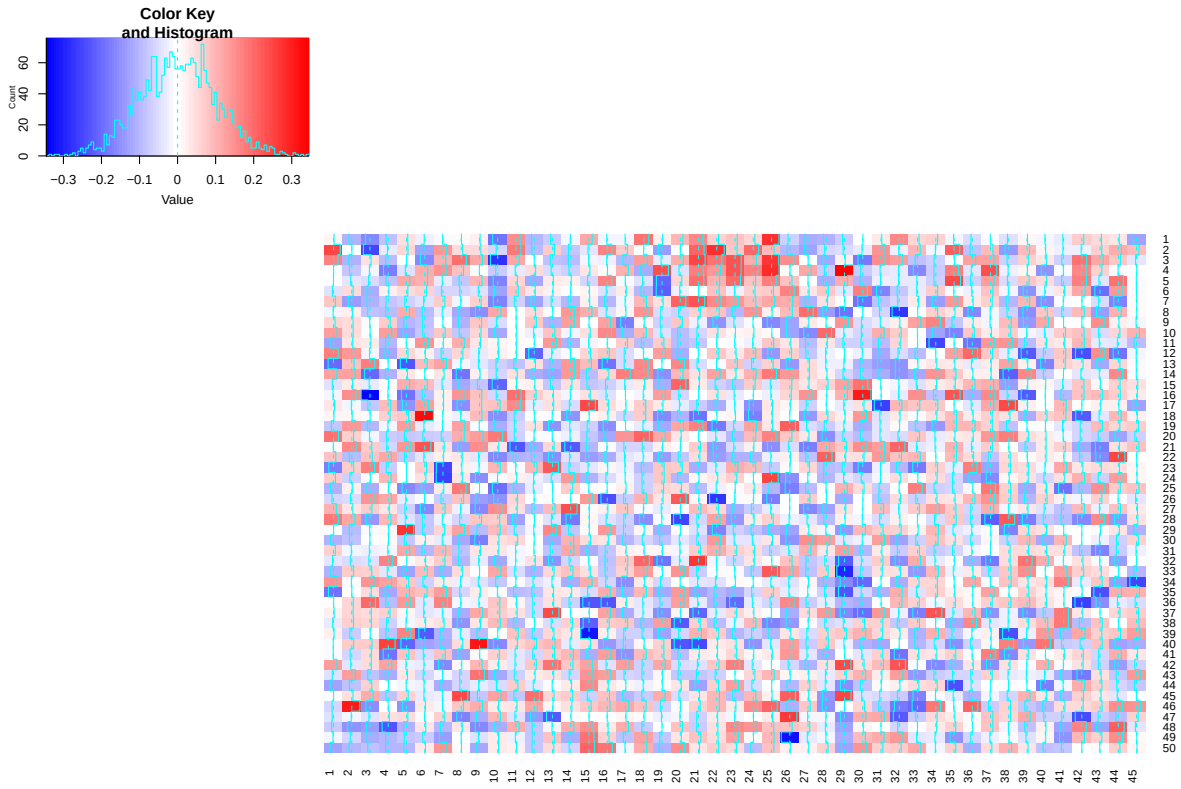
Now `X` has 100 rows and 50 columns while `Y` has 100 rows and 45 columns. We used two latent components in X, which are hidden in the first five and last five variables. The first five variables are also present in Y_{20} to Y_{25}. We add noise so we do not exactly observe the latent structures.

We will use the `gplots` package to create heatmaps of correlations.

```
try(
  gplots::heatmap.2(cor(X,Y), Rowv=F,Colv=F, col=gplots::bluered(100)),
  silent = TRUE)
```

```
## Warning in gplots::heatmap.2(cor(X, Y), Rowv = F, Colv = F, col =
## gplots::bluered(100)): Discrepancy: Rowv is FALSE, while dendrogram is
## `both'. Omitting row dendogram.
```

```
## Warning in gplots::heatmap.2(cor(X, Y), Rowv = F, Colv = F, col =
## gplots::bluered(100)): Discrepancy: Colv is FALSE, while dendrogram is
## `column'. Omitting column dendogram.
```



It is difficult to see where the correlated part lies. We will try to find out with O2PLS. First we need to determine the number of components.

```
library(OmicsPLS)
set.seed(1221L)
crossval_o2m_adjR2(X, Y, 1:3, 0:3, 0:3, nr_folds = 2)
```

```
## minimum is at n = 1
```

```
## Elapsed time: 0.55 sec
```

```
##        MSE n nx ny
## 1 2.030027 1  1  1
## 2 2.053605 2  3  3
## 3 2.083403 3  3  3
```

```
crossval_o2m(X, Y, 1:3, 0:3, 0:3, nr_folds = 10)
```

```
## ********************
## Elapsed time: 3.76 sec
## *******
## Minimal 10-CV error is at ax=0 ay=1 a=1
## *******
## Minimum is 2.022266
## ********************
```

The alternative cross-validation suggests one component in all parts. The full cross-validation suggests one joint and one X-specific component. Although the full CV got it right, the alternative yielded similar answers in much less CPU time (a factor of ten faster). This is partly because we use more folds, but decreasing the number of folds to two yielded unreliable results for the full CV.

We now fit the O2PLS model.

```
fit0 = o2m(X, Y, 1, 1, 0)
fit0
```

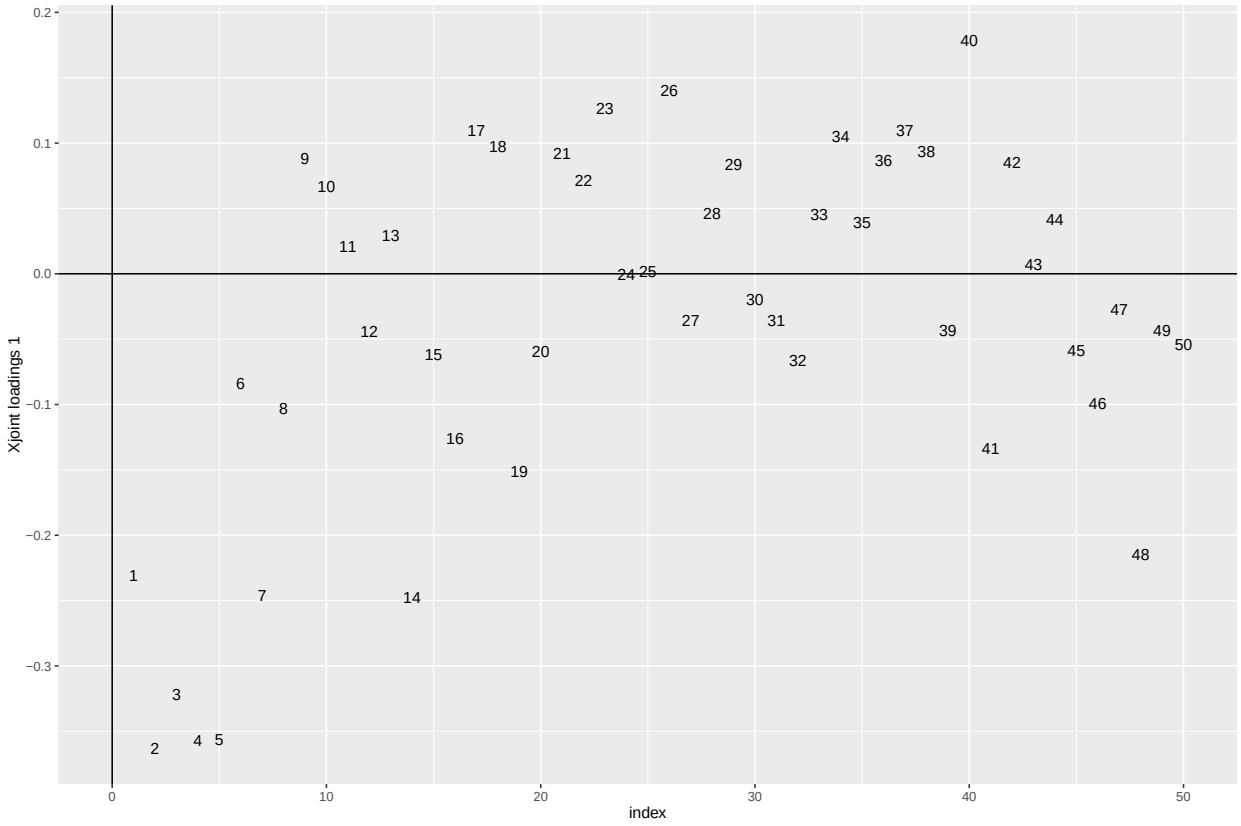
```
## O2PLS fit
## with 1 joint components
## and  1 orthogonal components in X
## and  0 orthogonal components in Y
## Elapsed time: 0 sec
```

```
summary(fit0)
```

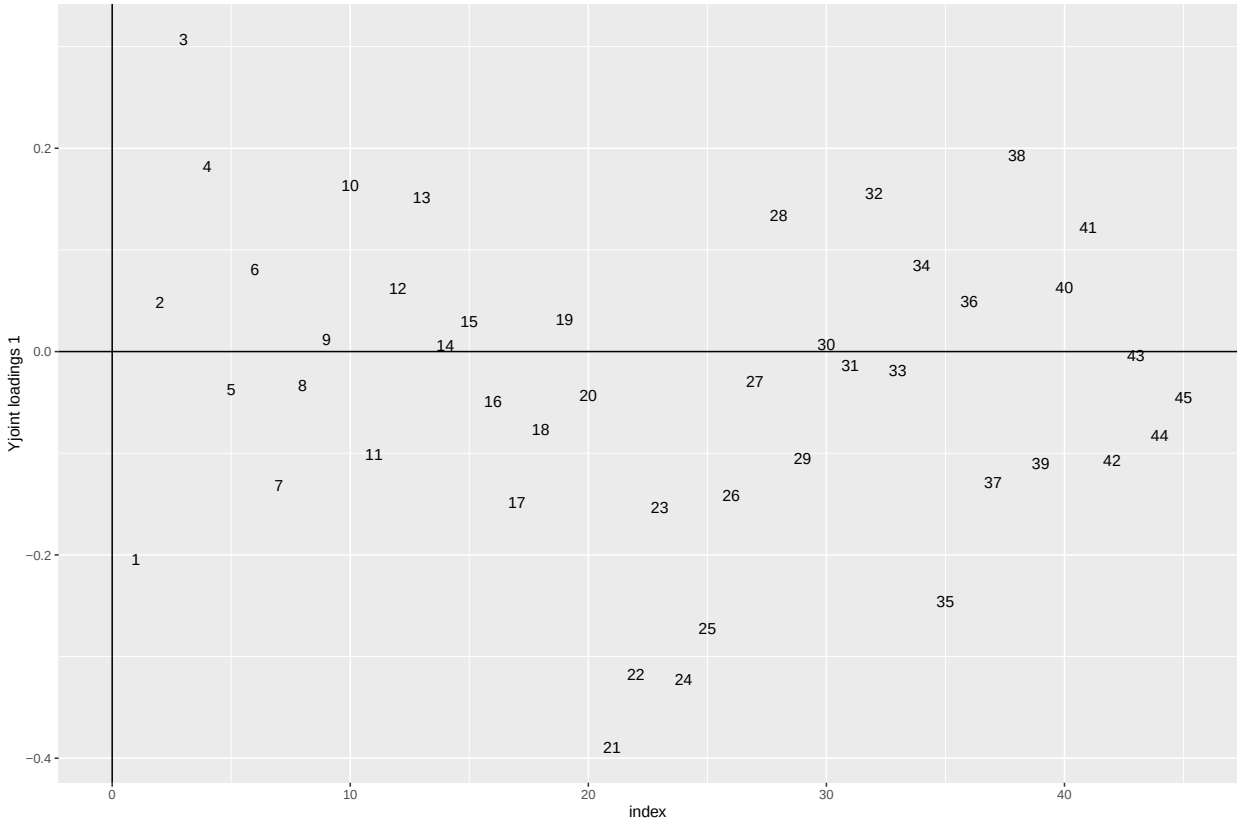
```
##
## *** Summary of the O2PLS fit ***
##
## -  Call: o2m(X = X, Y = Y, n = 1, nx = 1, ny = 0)
##
## -  Modeled variation
## -- Total variation:
## in X: 5100.552
## in Y: 4576.141
##
## -- Joint, Orthogonal and Noise as proportions:
##
##            data X data Y
## Joint       0.040  0.054
## Orthogonal  0.042  0.000
## Noise       0.918  0.946
##
## -- Predictable variation in Y-joint part by X-joint part:
## Variation in Yhat relative to U: 0.693
## -- Predictable variation in X-joint part by Y-joint part:
## Variation in Xhat relative to T: 0.693
##
## -- Variances per component:
##
##          Comp 1
## X joint 206.273
## Y joint 247.776
##
##         Comp 1
## X Orth 212.371
##
##
## -  Coefficient in 'U = T B_T + H_U' model:
## -- Diagonal elements of B_T =
##  0.912
```

We can see that there is a lot of noise (92% and 95%), and only about 5% joint variation. However relative to this variation, 69% is predictable. To see which variables induce the joint variation, we plot the joint loadings of X and Y.

```
plot(fit0)
```



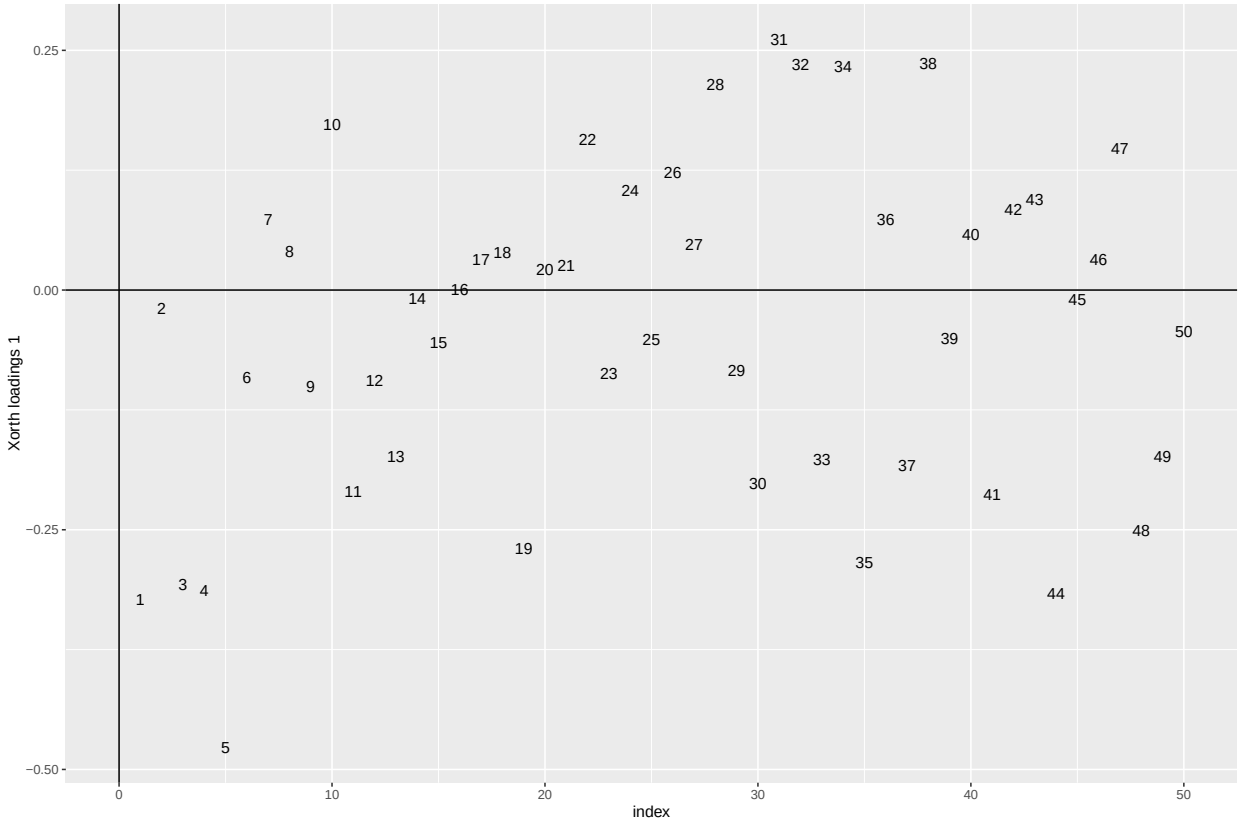

```
plot(fit0, "Yj")
```



We see that more or less the first five X variables and columns 21 to 25 of Y have high absolute loading values.

The X-specific loadings are not recovered unfortunately, probably due to the high noise level.

```
plot(fit0, "Xo")
```



Real data example

First we define a function to pick only the top `100*prop` percent of the genes that have highest expression level, intersected with the top `100*prop` percent with highest Inter Quantile Range.

```
filter_rna <- function(rna=rna, prop = 0.75){
  #first, calculate the maximum of gene expression per each gene and take the quantiles, we are interes
  maxGE <- apply(rna, 2, max)
  propGEmax <- quantile(maxGE, prop)
  #next, take the IQR of each gene, to capture the variability and check the top 25% genes according to
  IQRGE <- apply(rna, 2, IQR, na.rm=TRUE)
  propGEIQR <- quantile(IQRGE, prop)
  #selected genes/probes is the intersection of the two previous sets
  filter2 <- (intersect(which(maxGE> propGEmax), which(IQRGE> propGEIQR)))
  return(filter2)
}
```

Load in the data

Packages needed

- `install.packages("data.table")`

Now we load in the data, assumed it is downloaded and stored on disk, and prepare it in the right format (samples as rows and genes as columns) and give the rows and columns the right names. We use the package `data.table` as this reads in large data sets much faster.

```
set.seed(31*12*2016)
rna0 <- data.table::fread("C:/Users/selbouhaddani/MEGANZ/Downloads/O2PLS_BiB/test.tab",header=T, sep='\t
```

```
##
Read 28.2% of 35420 rows
Read 56.5% of 35420 rows
Read 84.7% of 35420 rows
Read 35420 rows and 519 (of 519) columns from 0.289 GB file in 00:00:08
```

```
rna1 <- t(as.data.frame.matrix(rna0[-1,-1,with=F]))
rna2 <- matrix(as.numeric(rna1), nrow = nrow(rna1))
dimnames(rna2) <- list(colnames(rna0)[-1],unlist(rna0[-1,1,with=F]))
rna2 <- rna2[order(row.names(rna2)), ] # Order rows according to the participant ID
rna3 <- rna2[,filter_rna(rna2)]
rm(rna0)
rm(rna1)
```

We removed unneeded copies of the expression data set.

We also load in the metabolite data and prepare it to have samples as rows and set the columns names.

```
metab0 <- read.table("C:/Users/selbouhaddani/MEGANZ/Downloads/O2PLS_BiB/metabonomic_data.txt",header=T,
metab1 <- t(metab0[,-1])
colnames(metab1) <- metab0$Metabolite
```

Missing data imputation

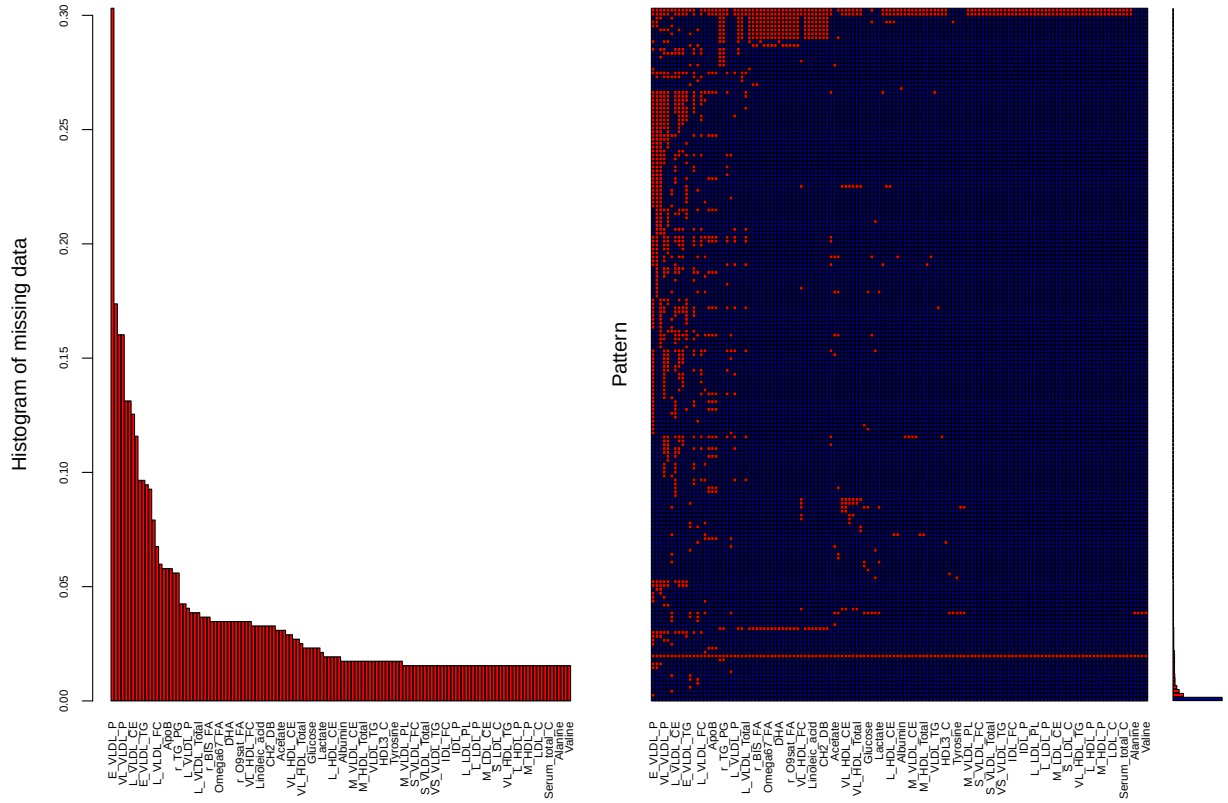
Packages needed

- `install.packages("VIM")`
- `install.packages("missForest")`

Note that we have missingness in the metabolite data. The functions in OmicsPLS currently cannot impute this, so we need to do some imputation ourselves. Some diagostics on the missingness in the metabolite data can be obtained. Firstly we plot a histogram of the missing data. We need the `VIM` package for this.

```
VIM::aggr(metab1, col=c('navyblue','red'), numbers=TRUE, sortVars=TRUE,
     labels=names(data), cex.axis=.7, gap=3, ylab=c("Histogram of missing data","Pattern"))
```

```
## Warning in plot.aggr(res, ...): not enough vertical space to display
## frequencies (too many combinations)
```




```
##
##  Variables sorted by number of missings:
##            Variable      Count
##           E_VLDL_P 0.30308880
##          E_VLDL_PL 0.17374517
##       E_VLDL_Total 0.16023166
##          VL_VLDL_P 0.16023166
##         VL_VLDL_PL 0.13127413
##           Creatine 0.13127413
##          L_VLDL_CE 0.12548263
##       VL_VLDL_Total 0.11583012
##          VL_VLDL_TG 0.09652510
##          E_VLDL_TG 0.09652510
##       Acetoacetate 0.09459459
##          L_VLDL_C 0.09266409
##  a3hydroxybutyrate 0.07915058
##          L_VLDL_FC 0.06756757
##         Isoleucine 0.05984556
##               ApoAI 0.05791506
##                ApoB 0.05791506
##         ApoB_ApoAI 0.05791506
##           Total_PG 0.05598456
##            r_TG_PG 0.05598456
##          L_VLDL_PL 0.04247104
##                 GP 0.04247104
##           L_VLDL_P 0.04054054
##           Total_TG 0.03861004
```

```
##                SM 0.03861004
##       L_VLDL_Total 0.03861004
##                PC 0.03667954
##             DB_FA 0.03667954
##           r_BIS_FA 0.03667954
##             Free_C 0.03474903
##          Omega3_FA 0.03474903
##         Omega67_FA 0.03474903
##      Omega9_sat_FA 0.03474903
##           Total_FA 0.03474903
##                DHA 0.03474903
##            r_O3_FA 0.03474903
##           r_O67_FA 0.03474903
##         r_O9sat_FA 0.03474903
##             CH2_FA 0.03474903
##            Avg_FAL 0.03474903
##          VL_HDL_FC 0.03474903
##            Total_C 0.03281853
##       Esterified_C 0.03281853
##      Linoleic_acid 0.03281853
##         Other_PUFA 0.03281853
##     Total_cholines 0.03281853
##             CH2_DB 0.03281853
##           r_BIS_DB 0.03281853
##          L_VLDL_TG 0.03088803
##            Acetate 0.03088803
##            Citrate 0.03088803
##          VL_HDL_C 0.02895753
##          VL_HDL_CE 0.02895753
##          VL_HDL_PL 0.02702703
##           VL_HDL_P 0.02702703
##       VL_HDL_Total 0.02509653
##           L_HDL_FC 0.02316602
##         Creatinine 0.02316602
##            Glucose 0.02316602
##          Glutamine 0.02316602
##      Phenylalanine 0.02316602
##            Lactate 0.02123552
##          M_VLDL_TG 0.01930502
##            L_HDL_C 0.01930502
##           L_HDL_CE 0.01930502
##           M_HDL_CE 0.01930502
##            S_HDL_P 0.01930502
##            Albumin 0.01737452
##           M_VLDL_C 0.01737452
##          M_VLDL_FC 0.01737452
##          M_VLDL_CE 0.01737452
##           S_VLDL_C 0.01737452
##            M_HDL_C 0.01737452
##        M_HDL_Total 0.01737452
##           S_HDL_TG 0.01737452
##        S_HDL_Total 0.01737452
##            VLDL_TG 0.01737452
##     VLDL_TG_lipido 0.01737452
```

```
##       IDL_C_lipido 0.01737452
##             HDL3_C 0.01737452
##          Histidine 0.01737452
##            Leucine 0.01737452
##           Tyrosine 0.01737452
##  Unsaturatedlipids 0.01737452
##               Urea 0.01737452
##          M_VLDL_PL 0.01544402
##       M_VLDL_Total 0.01544402
##           M_VLDL_P 0.01544402
##          S_VLDL_FC 0.01544402
##          S_VLDL_PL 0.01544402
##          S_VLDL_TG 0.01544402
##       S_VLDL_Total 0.01544402
##           S_VLDL_P 0.01544402
##         VS_VLDL_PL 0.01544402
##         VS_VLDL_TG 0.01544402
##      VS_VLDL_Total 0.01544402
##           VS_VLDL_P 0.01544402
##              IDL_FC 0.01544402
##              IDL_PL 0.01544402
##          IDL_Total 0.01544402
##              IDL_P 0.01544402
##            L_LDL_C 0.01544402
##           L_LDL_FC 0.01544402
##           L_LDL_PL 0.01544402
##           L_LDL_CE 0.01544402
##        L_LDL_Total 0.01544402
##            L_LDL_P 0.01544402
##            M_LDL_C 0.01544402
##           M_LDL_PL 0.01544402
##           M_LDL_CE 0.01544402
##        M_LDL_Total 0.01544402
##            M_LDL_P 0.01544402
##            S_LDL_C 0.01544402
##        S_LDL_Total 0.01544402
##            S_LDL_P 0.01544402
##          VL_HDL_TG 0.01544402
##           L_HDL_PL 0.01544402
##        L_HDL_Total 0.01544402
##            L_HDL_P 0.01544402
##           M_HDL_FC 0.01544402
##           M_HDL_PL 0.01544402
##            M_HDL_P 0.01544402
##             IDL_TG 0.01544402
##              IDL_C 0.01544402
##              LDL_C 0.01544402
##              HDL_C 0.01544402
##     Serum_total_TG 0.01544402
##      Serum_total_C 0.01544402
##       LDL_C_lipido 0.01544402
##             HDL2_C 0.01544402
##            Alanine 0.01544402
##                CH2 0.01544402
```

```
##                CH3 0.01544402
##             Valine 0.01544402
```

We remove participants with 100% missing metabolite measurements, i.e. missing rows.

```
NAs_in_metab1 <- which(apply(metab1, 1, function(e) sum(is.na(e))/length(e))==1)
metab2 <- metab1[-NAs_in_metab1,]
rna4 <- rna3[-NAs_in_metab1,]
```

Random Forests can be used to impute missing metabolites. We use the `missForest` package to do this. It takes some time, a couple of minutes on a modest second generation i5 laptop.

```
metab2.imp <- missForest::missForest(metab2, verbose = T)
```

```
##   missForest iteration 1 in progress...done!
##     estimated error(s): 0.4234906
##     difference(s): 0.02714137
##     time: 76.54 seconds
##
##   missForest iteration 2 in progress...done!
##     estimated error(s): 0.4189017
##     difference(s): 0.0005727861
##     time: 75.98 seconds
##
##   missForest iteration 3 in progress...done!
##     estimated error(s): 0.4185268
##     difference(s): 0.0002720186
##     time: 75.55 seconds
##
##   missForest iteration 4 in progress...done!
##     estimated error(s): 0.4203561
##     difference(s): 0.0002325965
##     time: 76.02 seconds
##
##   missForest iteration 5 in progress...done!
##     estimated error(s): 0.4195952
##     difference(s): 0.0002338311
##     time: 75.78 seconds
```

```
X2 <- scale(metab2.imp$ximp, scale=F)
X1 <- scale(rna4, scale = F)
```

In the last two lines, we took one imputed instance of the metabolite data and centered the columns of the RNA and metabolite data to have zero mean. We denote them by `X1` (transcripts) and `X2` (metabolites).

Inspect the data: descriptives

Packages needed

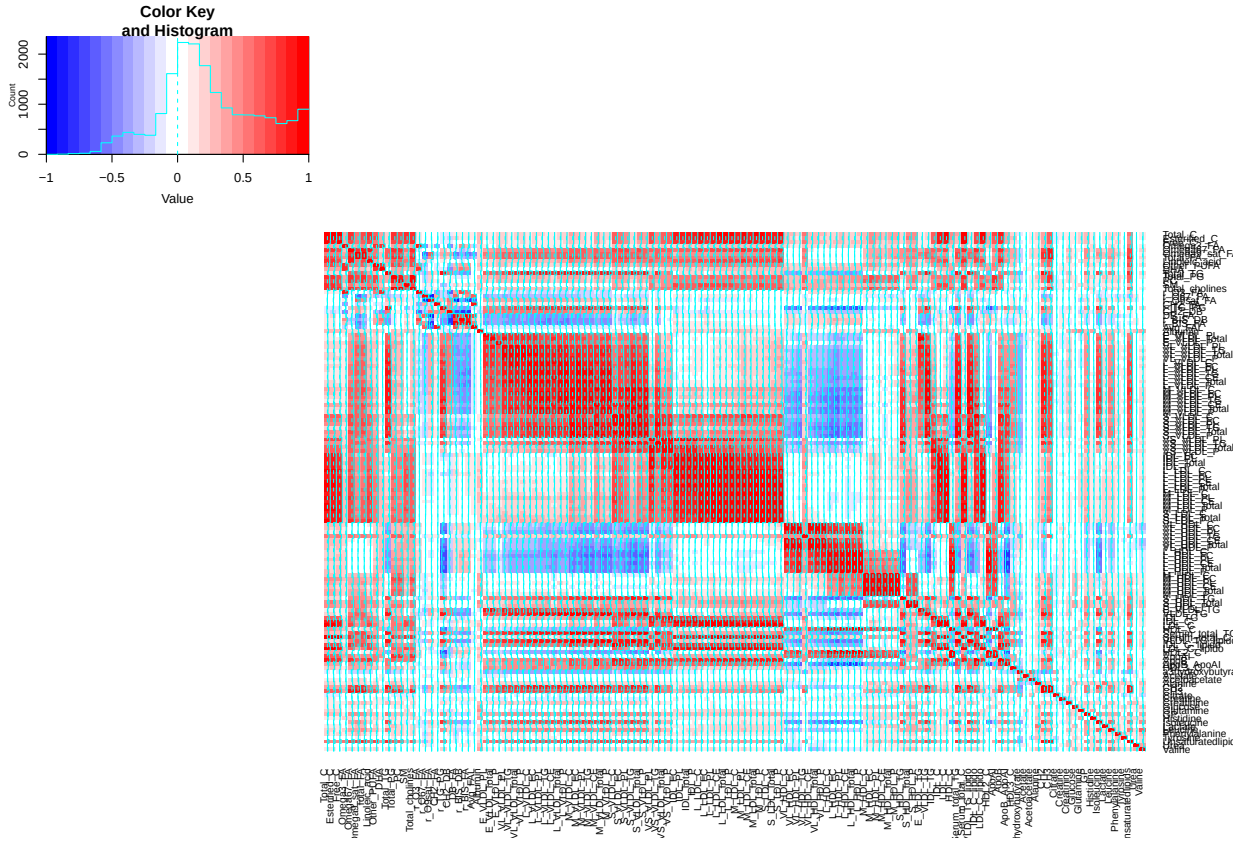
- `install.packages("gplots")`

A heatmap of metabolites, before and after imputation

```
try(
  gplots::heatmap.2(cor(metab1,use = 'pair'), Rowv=F, Colv=F,
                    breaks=seq(-1,1,length.out = 25), col=gplots::bluered),
  silent = TRUE)
```

```
## Warning in gplots::heatmap.2(cor(metab1, use = "pair"), Rowv = F, Colv =
## F, : Discrepancy: Rowv is FALSE, while dendrogram is `both'. Omitting row
## dendogram.
```

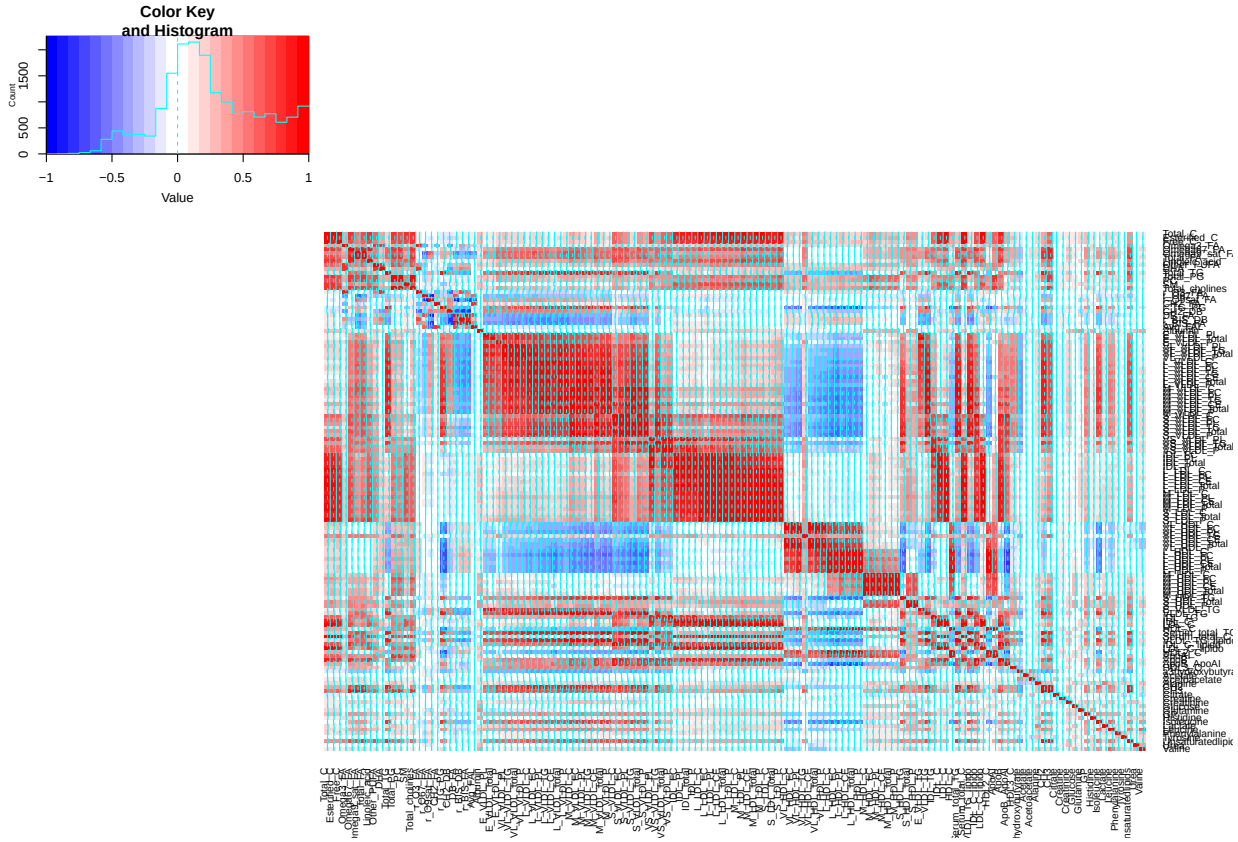
```
## Warning in gplots::heatmap.2(cor(metab1, use = "pair"), Rowv = F, Colv =
## F, : Discrepancy: Colv is FALSE, while dendrogram is `column'. Omitting
## column dendogram.
```




```
try(
  gplots::heatmap.2(cor(X2,use = 'pair'), Rowv=F, Colv=F,
                    breaks=seq(-1,1,length.out = 25), col=gplots::bluered),
  silent = TRUE)
```

```
## Warning in gplots::heatmap.2(cor(X2, use = "pair"), Rowv = F, Colv =
## F, : Discrepancy: Rowv is FALSE, while dendrogram is `both'. Omitting row
## dendogram.
```

```
## Warning in gplots::heatmap.2(cor(X2, use = "pair"), Rowv = F, Colv = F, :
## Discrepancy: Colv is FALSE, while dendrogram is `column'. Omitting column
## dendogram.
```



They are almost the same, indicating that the correlation structure within metabolites hasn't changed much.

To get an idea on the latent structure of the data one may look at eigenvalues of covariance matrix of `X1` and `X2`.

```
svd(X1, 0, 0)$d[1:10]^2 / sum(X1^2)
```

```
##  [1] 0.19568455 0.12670559 0.09534211 0.05334638 0.03931151 0.03294359
##  [7] 0.01963035 0.01663232 0.01575400 0.01499442
```

```
svd(X2, 0, 0)$d[1:10]^2 / sum(X2^2)
```

```
##  [1] 0.37544553 0.21076846 0.10669151 0.04796332 0.03171004 0.02697083
##  [7] 0.02085717 0.01758810 0.01569650 0.01329923
```

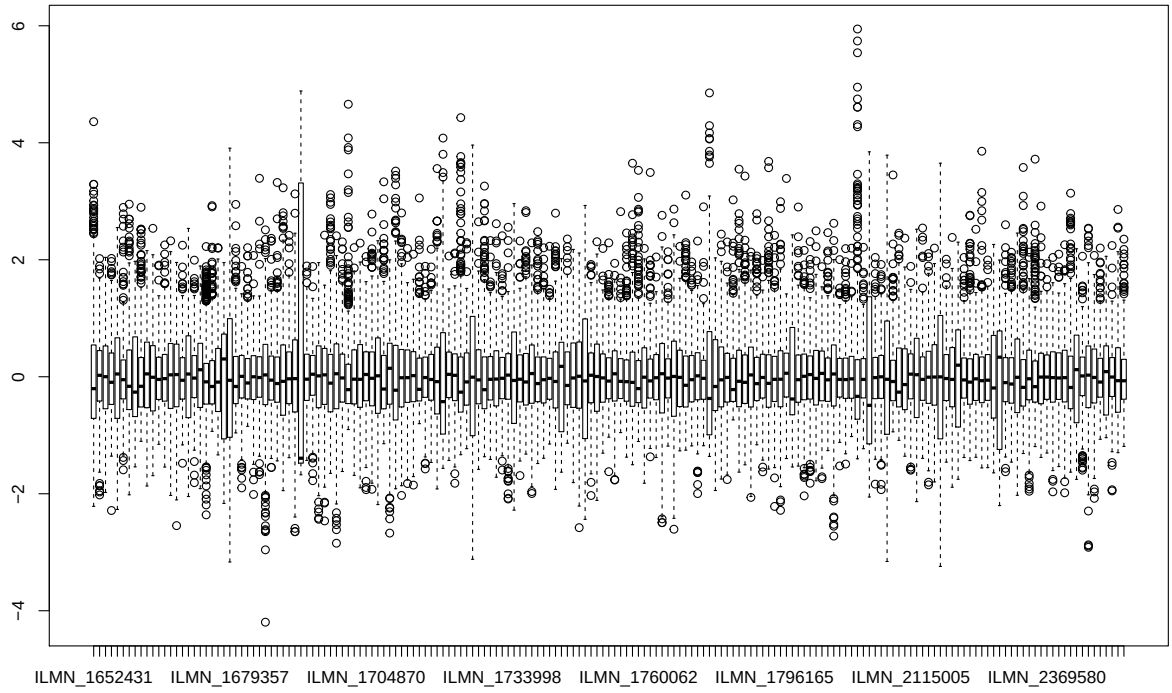
```
svd(crossprod(X1,X2),0,0)$d[1:20]
```

```
##  [1] 7109.2911 3885.6227 2453.7749 2364.2937 1990.4501 1226.8889 1156.0470
##  [8]  986.7474  780.6249  747.3281  691.4328  616.8352  579.9895  514.3137
## [15]  490.8930  429.8551  410.8972  368.9870  356.8286  346.3882
```

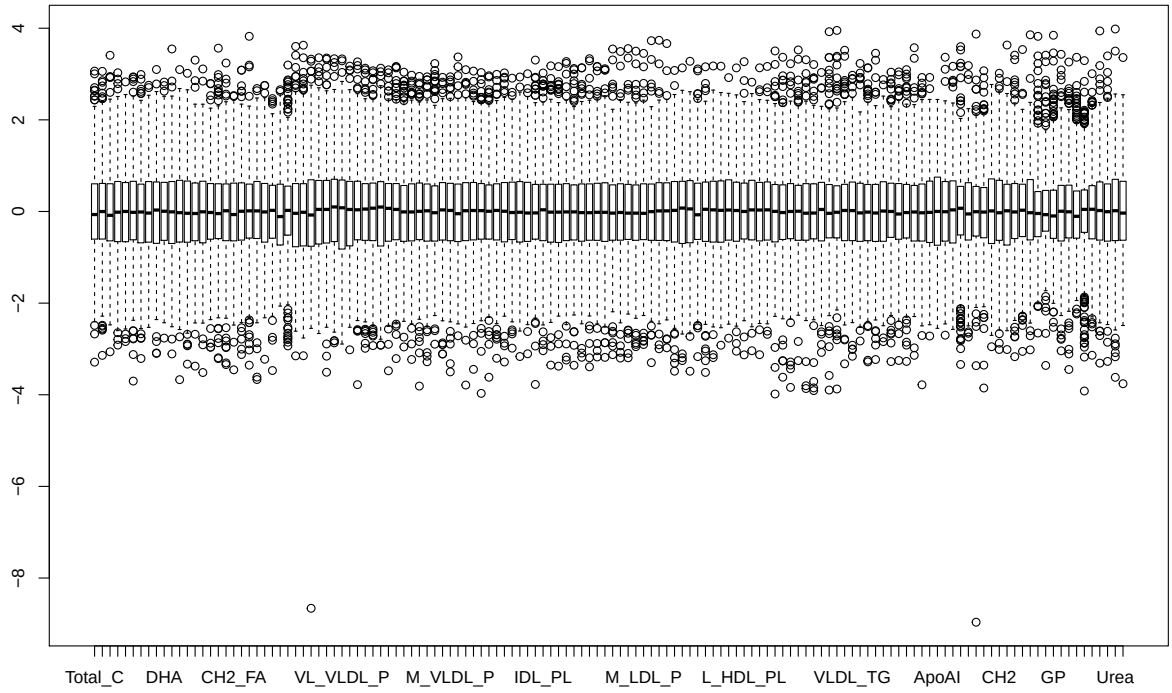
The first two lines contain relative variances explained by each principal component. Strong latent structure is indicated by a sharp decline of the relative variances at the first few components.

Boxplots are a good summary to check the distribution of the variables relative to each other. Properties such as comparable means, variances and symmetry are often good to have. To reduce the number of boxplots we filter the transcriptomic data to include genes with 95% highest expression and IQR.

```
boxplot(X1[,filter_rna(X1, .95)])
```


6
4
2
0
−2
−4
ILMN_1652431
ILMN_1679357
ILMN_1704870
ILMN_1733998
ILMN_1760062
ILMN_1796165
ILMN_2115005
ILMN_2369580

```
boxplot(X2)
```



The OmicsPLS package

Cross-validation

We load the OmicsPLS package and set a seed for the cross-validation. The strategy is to have a relatively large grid to search on and apply the faster alternative Cross-Validation (Cv) approach to find a solution. Then on a smaller grid containing these integers we do a full CV to determine the best choice for the number of components. The objective function to minimize is the sum of the two prediction errors $||X - \hat{X}||$ and $||Y - \hat{Y}||$.

```
library(OmicsPLS)
set.seed(1+1+2016)
#CV1 <- crossval_o2m_adjR2(X1, X2, 1:3, c(0,1,5,10), c(0,1,5,10), nr_folds = 2, nr_cores = 4)
#CV2 <- crossval_o2m(X1, X2, 1:2, 0:2, 9:11, nr_folds = 10, nr_cores = 4)
#CV1
#CV2
```

Following the advice of the last CV output, we select two joint, one transcript-specific and ten metabolite-specific components. We fit the O2PLS model with default values as follows.

```
fit = o2m(X1, X2, 2, 1, 10)
fit
```

```
## O2PLS fit
## with 2 joint components
## and  1 orthogonal components in X
## and  10 orthogonal components in Y
## Elapsed time: 3.12 sec
```

A summary of the variation modeled is obtained via

```
summary(fit)
```

```
##
## *** Summary of the O2PLS fit ***
##
## -  Call: o2m(X = X1, Y = X2, n = 2, nx = 1, ny = 10)
##
## -  Modeled variation
## -- Total variation:
## in X: 332035.7
## in Y: 68381.71
##
## -- Joint, Orthogonal and Noise as proportions:
##
##            data X data Y
## Joint       0.124  0.410
## Orthogonal  0.171  0.243
## Noise       0.705  0.348
##
## -- Predictable variation in Y-joint part by X-joint part:
## Variation in Yhat relative to U: 0.116
## -- Predictable variation in X-joint part by Y-joint part:
## Variation in Xhat relative to T: 0.151
##
## -- Variances per component:
##
##           Comp 1    Comp 2
## X joint 25039.63 16203.728
## Y joint 18456.89  9555.336
##
##          Comp 1
## X Orth 56637.85
##
##          Comp 1   Comp 2   Comp 3   Comp 4   Comp 5   Comp 6  Comp 7
## Y Orth 6715.068 3315.059 2055.242 1199.923 1091.131 1103.341 838.014
##         Comp 8  Comp 9 Comp 10
## Y Orth 950.716 570.621 850.223
##
##
## -  Coefficient in 'U = T B_T + H_U' model:
## -- Diagonal elements of B_T =
##  0.401 0.339
```

Plotting

Packages needed

- install.packages("magrittr")
- install.packages("ggplot2")
- install.packages("gridExtra")
- install.packages("stringr")
- install.packages("gplots")

We want to see which (groups of) metabolites and transcripts tend to correlate with each other. To do this we plot the loadings. The individual loading values per component indicate the relative importance of each variable to the corresponding component. We plot the two joint loadings against each other to see which metabolites are more important for each component. To do this we need three packages for convenience: `magrittr` for the piping operator, `ggplot2` for plotting and `gridExtra` to put multiple ggplots in one figure. Also `stringr` will be needed to extract substrings of column names.

```
require(magrittr)
```

```
## Loading required package: magrittr
```

```
require(ggplot2)
```

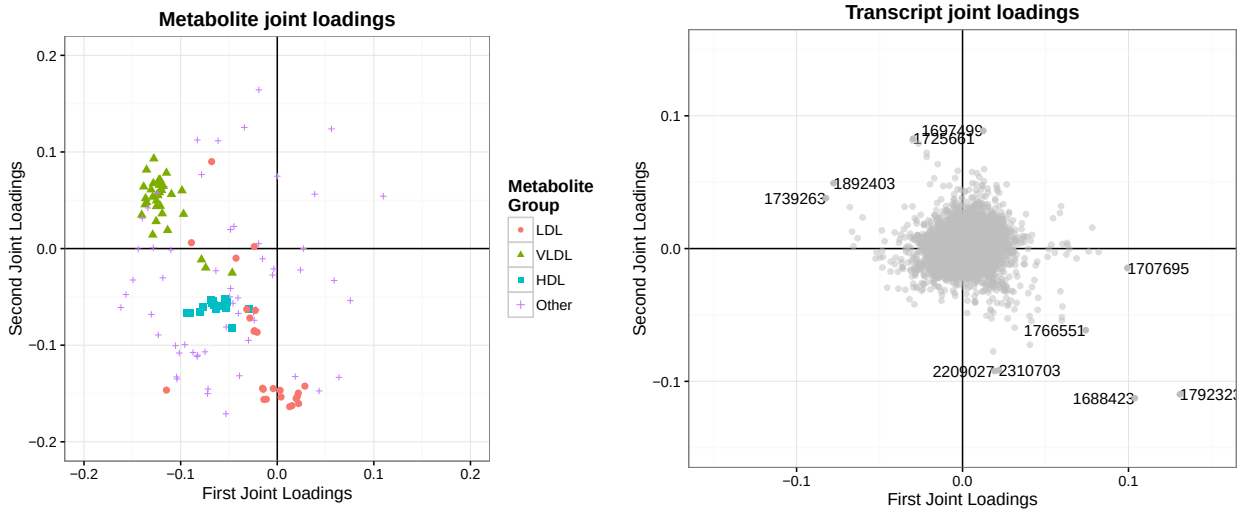
```
## Loading required package: ggplot2
```

```
require(gridExtra)
```

```
## Loading required package: gridExtra
```

```
# Color names
name_col = 1 + sapply( #First sapply loops over column names
  X = colnames(X2),
  FUN = function(arg){
    crossprod(
      c(1, 1, 3), # Weights to be used as categories
      sapply(c("VLDL", "LDL", "HDL"), # metabolite classes
             function(arg2){grepl(arg2, arg)} # compare class of metabolites
      )
    )
    }
  )

alpX2 <- loadings(fit, "Yjoint", 1:2) %>%  # Retreive loadings
  abs %>% # Absolute loading values for positive weights
  rowSums %>% # Sum over the components
  sqrt + (name_col>1) # Take square root

######### Plot loadings with OmicsPLS plot method ###
p_X2 <- plot(fit, loading_name="Yj", i=1, j=2, label="c", # Plot the loadings
             alpha=0) + # set points to be 100% transparant
###################### Add all layers ###
  theme_bw() +
  coord_fixed(ratio = 1, xlim=c(-.2,.2),ylim=c(-.2,.2)) +
  geom_point( # Set color and size
    aes(col=factor(name_col, levels = 4:1), size = I(1+(name_col>1)), shape =
          factor(name_col, levels = 4:1)),show.legend = T) +
  theme(legend.position="right") +
  scale_color_discrete(name="Metabolite\nGroup",
                       labels=c("LDL","VLDL","HDL","Other")) +
  guides(size=F) + scale_shape_discrete(name="Metabolite\nGroup",
                                        labels=c("LDL","VLDL","HDL","Other")) +
labs(title = "Metabolite joint loadings",
     x = "First Joint Loadings", y = "Second Joint Loadings") +
  theme(plot.title = element_text(face='bold'),
        legend.title=element_text(face='bold'))

alpX1 <- loadings(fit, "Xjoint", 1:2) %>% raise_to_power(2) %>% rowSums
```

```
alpX1[-(order(alpX1,decreasing=T)[1:10])] = 0
alpX1 <- sign(alpX1)
######### Plot loadings with OmicsPLS plot method ###
p_X1 <- plot(fit, loading_name="Xj", i=1, j=2, label = "c", use_ggplot2 = TRUE,
             alpha = alpX1,
             aes(label = stringr::str_sub(colnames(X1), start = 6)),
             hjust = rep(c(0, 1), length.out = ncol(X1))) +
###################### Add all layers ###
  theme_bw() +
  coord_fixed(.8, c(-.15,.15),c(-.15,.15)) +
  geom_point(alpha = 0.5+0.5*alpX1, col = 'grey') +
  labs(title = "Transcript joint loadings",
       x = "First Joint Loadings", y = "Second Joint Loadings") +
  theme(plot.title = element_text(face='bold'))

## Finally plot both plots in one figure.
grid.arrange(p_X2, p_X1, ncol=2)
```



Tweaking and adjusting the plots for publication can take some time, which also was here the case.

To visualize the decomposition, we will plot heatmaps of the correlation induced by the different parts. To do this we define a shortcut of the `gplots::heatmap.2` function. Here we need the `gplots` package.

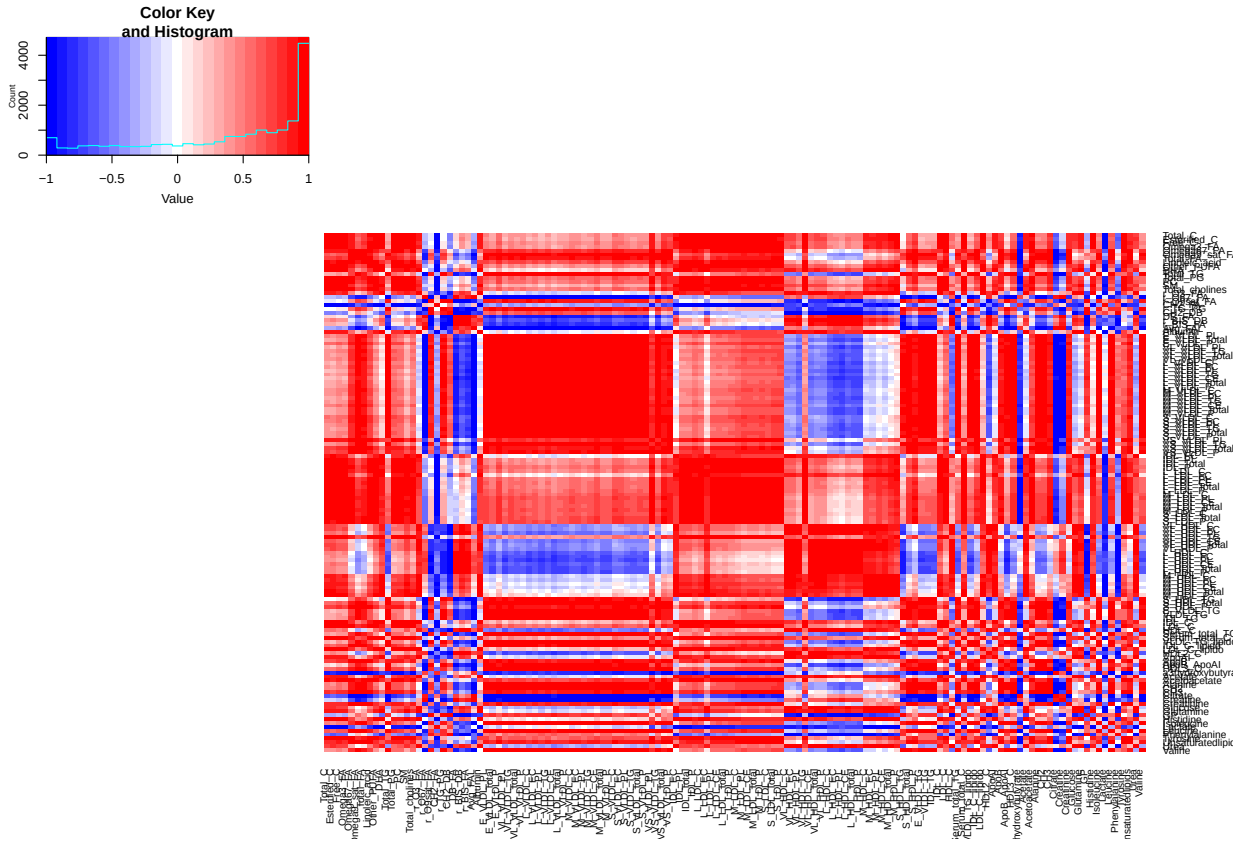
```
library(gplots)
```

```
##
## Attaching package: 'gplots'
```

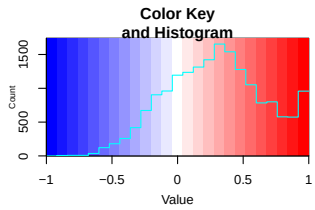
```
## The following object is masked from 'package:stats':
```

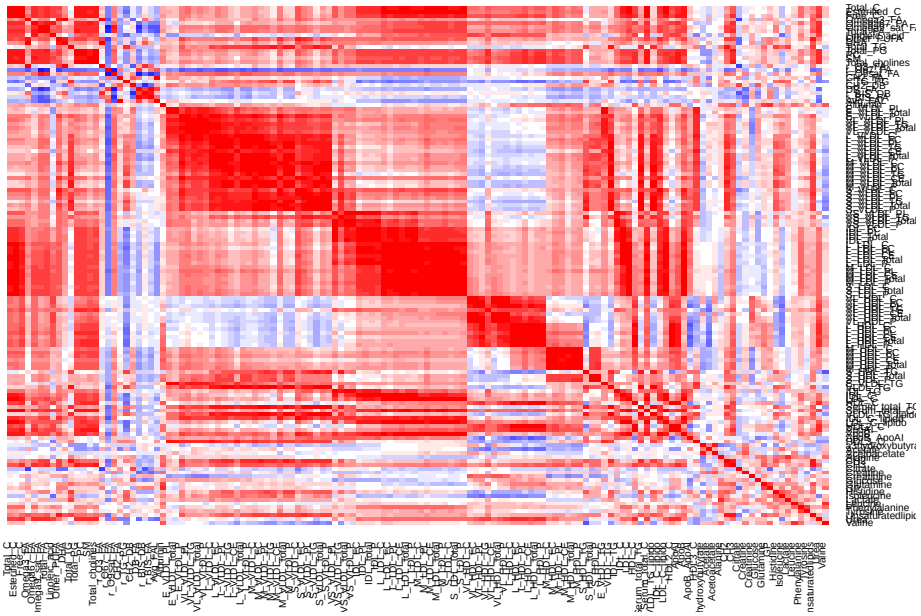
```
##
##     lowess
```

```
hm.2 <- function(obj){
  try(
    heatmap.2(obj,Rowv=F,symm=T,col=colorRampPalette(c("blue","white","red"))(25),
              dendrogram='none',trace='none',symkey=TRUE,breaks=seq(-1,1,length=26),
              key = T),
    silent = TRUE)
}
hm.2(cor(with(fit,U%*%t(C.))))
```

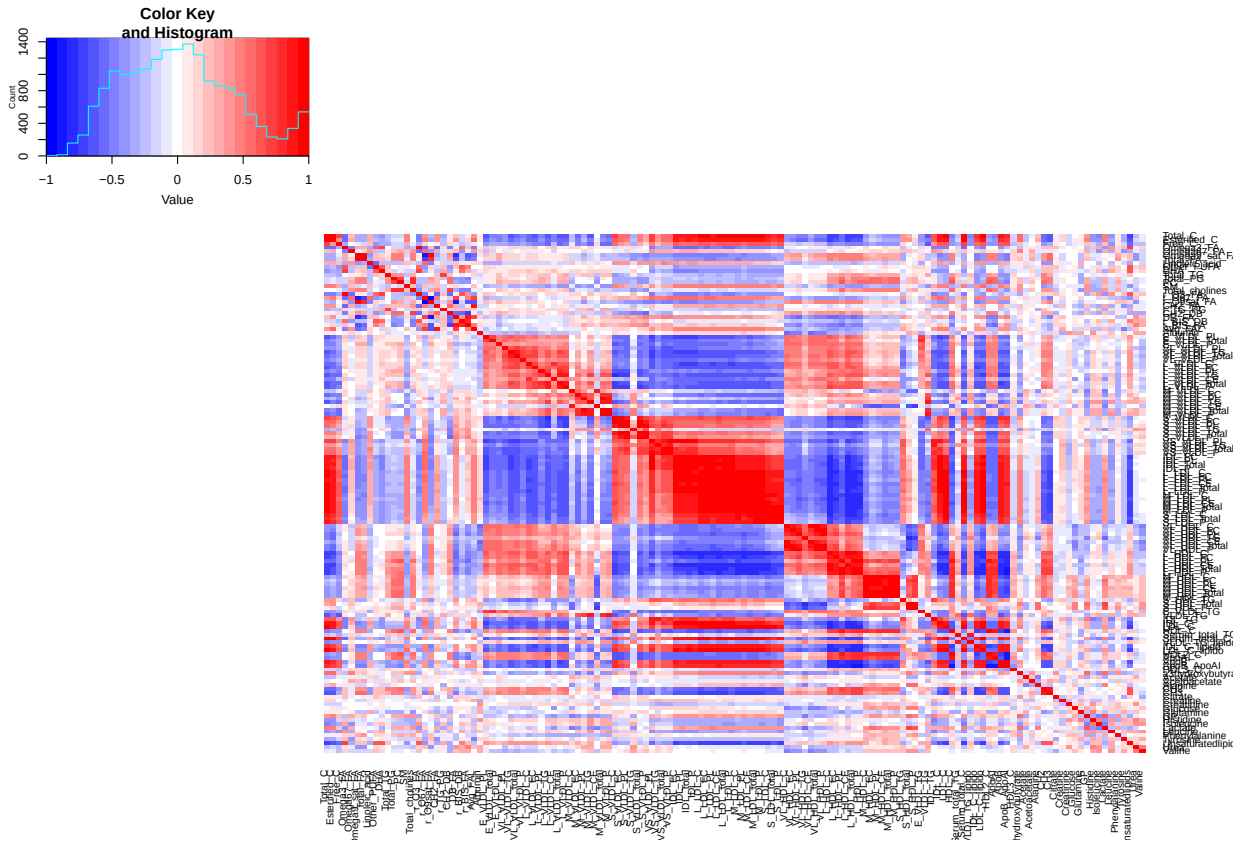



```
hm.2(cor(with(fit,U_Xosc%*%t(P_Xosc.))))
```





```
hm.2(cor(with(fit,Ff)))
```



CPU times

Packages needed

- `install.packages("microbenchmark")`

In OmicsPLS we added an alternative, memory-efficient, fitting algorithm (NIPALS) for high-dimensional data. This omits storing the whole covariance matrix of size p times q. In case p and q are large, say larger than 3000 both, storing this becomes a memory intensive operation. To see how long `o2m` takes to fit, we consider three scenarios. They are timed with the `microbenchmark` function.

```
set.seed(2016^2)
fake_X <- scale(matrix(rnorm(1e2*1e4),1e2))
fake_Y <- scale(matrix(rnorm(1e2*1e2),1e2))
suppressMessages(
  scenario1 <- microbenchmark::microbenchmark(
    default=o2m(fake_X, fake_Y, 1, 1, 1),
    stripped=o2m(fake_X, fake_Y, 1, 1, 1, stripped=T),
    highD = o2m(fake_X, fake_Y, 1, 1, 1, stripped=T, p_thresh=1),
    times = 5, unit = 's',control=list(warmup=1))
)

fake_X <- scale(matrix(rnorm(1e2*2e3),1e2))
fake_Y <- scale(matrix(rnorm(1e2*2e3),1e2))
suppressMessages(
  scenario2 <- microbenchmark::microbenchmark(
    default=o2m(fake_X, fake_Y, 1, 1, 1),
```

```
    stripped=o2m(fake_X, fake_Y, 1, 1, 1, stripped=T),
    highD = o2m(fake_X, fake_Y, 1, 1, 1, stripped=T, p_thresh=1),
    times = 5, unit = 's',control=list(warmup=1))
)

fake_X <- scale(matrix(rnorm(1e2*5e4),1e2))
fake_Y <- scale(matrix(rnorm(1e2*5e4),1e2))
try(o2m(fake_X, fake_Y, 1, 1, 1, p_thresh=1e6))
```

```
## Warning in as.matrix(x): Reached total allocation of 8096Mb: see
## help(memory.size)

## Warning in as.matrix(x): Reached total allocation of 8096Mb: see
## help(memory.size)

## Warning in as.matrix(x): Reached total allocation of 8096Mb: see
## help(memory.size)

## Warning in as.matrix(x): Reached total allocation of 8096Mb: see
## help(memory.size)
```

```
try(o2m(fake_X, fake_Y, 1, 1, 1, stripped=T, p_thresh=1e6))
```

```
## Warning in as.matrix(x): Reached total allocation of 8096Mb: see
## help(memory.size)

## Warning in as.matrix(x): Reached total allocation of 8096Mb: see
## help(memory.size)

## Warning in as.matrix(x): Reached total allocation of 8096Mb: see
## help(memory.size)

## Warning in as.matrix(x): Reached total allocation of 8096Mb: see
## help(memory.size)
```

```
o2m(fake_X, fake_Y, 1, 1, 1, stripped=T)
```

```
## Using Power Method with tolerance 1e-10 and max iterations 100

## Power Method (comp 1) stopped after 100 iterations.

## Power Method (comp 2) stopped after 100 iterations.

## Power Method (comp 1) stopped after 100 iterations.

## O2PLS fit: Stripped
## with 1 joint components
## and  1 orthogonal components in X
## and  1 orthogonal components in Y
## Elapsed time: 14.35 sec
```

```
rm(fake_X)
rm(fake_Y)
```