OMICs analyses: Dimensionality reduction
Document with solutions

Computational Epidemiology

MSc Health Data Analytics and Machine Learning

11^{th} of February 2021

Contents

Overview of the session

In this session you are going to reproduce the results from the paper **R Vermeulen, F Saberi Hosnijeh et al. Pre-diagnostic blood immune markers, incidence and progression of B-cell lymphoma and multiple myeloma: Univariate and functionally informed multivariate analyses, Int J Cancer (2018)**. In this paper the authors used linear mixed models and the sparse and sparse-group pls to identify sets of inflammatory proteins associated with prospective lymphoma status. The practical has two parts: a first dedicated to the linear mixed model analysis and the second dedicated to the application of the PLS models to the data.

Take a few minutes to read the sections “Introduction” and “Study subjects” in “Materials and Methods” in the paper to get background information on the disease and the study design.

Data provided

In the folder “BCL_analyses”, you will find the folders “Paper”, “Data” and “Scripts”.

The scientific publication along with supplementary materials is in “Paper”. Throughout the session, you will be able to check your results by comparing them to those presented in the paper.

In “Data”, you will find three files:

- "Proteins.rds": protein levels for the 536 participants
- "Proteins_denoised.rds": residuals from a linear mixed model adjusting on technical covariates and potential confounders
- "Covariates.rds": individual characteristics and technical covariate
- "wbc.rds": white blood cell counts for 448 individuals (224 case/control pairs)

Finally, the folder “Scripts” contains two R scripts (“pls_functions.R” and “pls_analyses.R”). You are going to run and complete the script “pls_analyses.R”, which calls functions from “pls_functions.R”.

Descriptive analyses and tables

Set your working directory in R to be in “BCL_analyses” using the command `setwd()`.

You can load the proteins and covariates data using the code below:

```r
covars = readRDS("Data/Covariates.rds")
proteins = readRDS("Data/Proteins.rds")
dim(proteins)
```

```
## [1] 536  28
```

```
dim(covars)
```

```
## [1] 536  18
```

```
print(all(rownames(covars) == rownames(proteins)))
```

```
## [1] TRUE
```

The `proteins` object contains log-transformed levels of 28 proteins in 536 participants. The `covars` object contains individual characteristics:

- `type`: the disease status (0 if healthy control and 1 if case)
- `age`: the individual age in years
- `sex`: binary variable
- `cohort`: E if EPIC-Italy and N if NSHDS
- `phase`: the phase in which samples were analysed (1 or 2)
- `Imputed_bmi`: numerical variable, imputed if missing from questionnaire data
- `Imputed_activity`: physical activity (4 categories)
- `Imputed_smoking_status`: smoking status (never, former or current smokers)
- `Imputed_alcohol`: alcohol intake (in g/day)
- `plate`: technical variable
- `Status`: same as type
- `LY_subtype`: BCL subtypes (10 categories)
- `egm_set`: ID of the case/control pair
- `egm_id`: sample ID
- `Sample`: date of sample collection
- `Diag`: date of diagnosis
- `Time_to_diag`: time to diagnosis in days

```
str(covars)
```

```
## 'data.frame':    536 obs. of  18 variables:
##  $ type                  : Factor w/ 2 levels "0","1": 1 2 2 1 2 1 1 2 1 2 ...
##  $ age                   : num  57 57.1 50.3 50.9 46.6 ...
##  $ sex                   : Factor w/ 2 levels "F  ","M  ": 1 1 1 1 2 2 2 2 2 2 ...
##  $ cohort                : Factor w/ 2 levels "E  ","N  ": 1 1 1 1 1 1 1 1 1 1 ...
##  $ phase                 : Factor w/ 2 levels "1","2": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Imputed_bmi           : num  23 25.9 24.1 26.2 23.8 ...
##  $ Imputed_education     : Factor w/ 5 levels "       0                ",..: 5 4 2 2 3 5 2 4 4 2 ...
##  $ Imputed_activity      : Factor w/ 4 levels "       1                ",..: 2 1 2 1 4 3 2 2 1 2 ...
##  $ Imputed_smoking_status: Factor w/ 3 levels "C                       ",..: 2 3 3 3 1 3 2 2 2 2 ...
##  $ Imputed_alcohol       : num  26.21 36.33 5.71 12.49 4.9 ...
##  $ plate                 : Factor w/ 11 levels "1","2","3","4",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ Status                : chr  "CO" "CA" "CA" "CO" ...
##  $ LY_subtype            : chr  "" "FL" "FL" "" ...
##  $ egm_set               : chr  "ELY006" "ELY006" "ELY017" "ELY017" ...
##  $ egm_id                : chr  "E001LY" "E002LY" "E003LY" "E004LY" ...
##  $ Sample                : Date, format: "1994-09-22" "1994-10-25" ...
##  $ Diag                  : Date, format: "9980-01-01" "1999-07-05" ...
##  $ Time_to_diag          : num  NA 1714 3391 NA 3201 ...
```

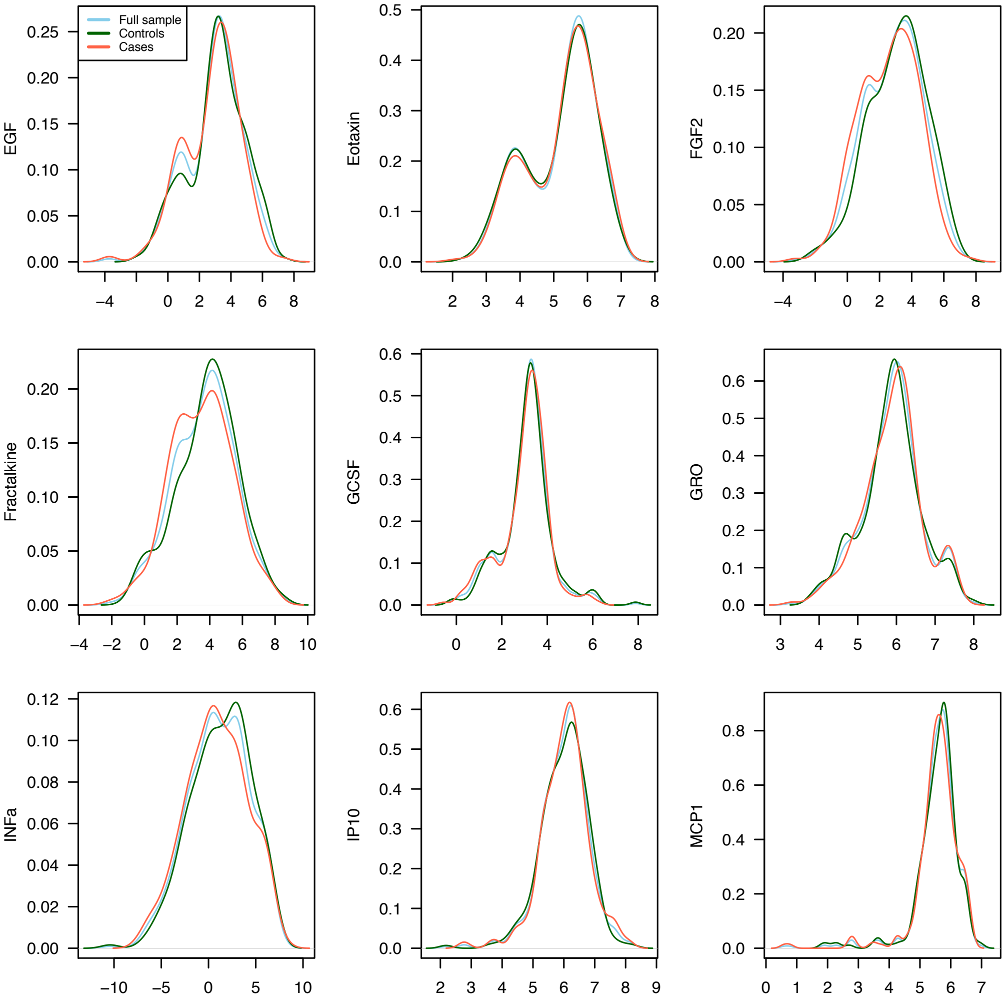
Challenge 1: Reproduce the summary table of general characteristics of BCL cases and matched controls (Table 1 in the paper). Hint: You might want to look at the content of previous practicals or use the package `tableone`.

See previous practicals for code and Table 1 in the paper for the exact values.

Challenge 2: Plot the distribution of individual proteins in cases, controls and in the full sample. What can you conclude?

```
dir.create("Figures", showWarnings = FALSE)

pdf("Figures/Distributions.pdf")
par(mfrow = c(3, 3), mar = c(3, 4.5, 1, 1))
for (k in 1:ncol(proteins)) {
    xfull = density(proteins[, k])
    x0 = density(proteins[as.character(covars$type) ==
        "0", k])
    x1 = density(proteins[as.character(covars$type) ==
        "1", k])
    plot(density(proteins[, k]), col = "skyblue", xlab = "",
        main = "", ylab = colnames(proteins)[k], las = 1,
        ylim = range(c(xfull$y, x0$y, x1$y)))
    lines(x0, col = "darkgreen")
    lines(x1, col = "tomato")
    if (k == 1) {
        legend("topleft", lwd = 2, col = c("skyblue",
            "darkgreen", "tomato"), legend = c("Full sample",
            "Controls", "Cases"), cex = 0.7)
    }
}
dev.off()
```

The figure below shows the distributions of the first 9 proteins. For some proteins, there is very little difference in the distributions between cases and controls (e.g. Eotaxin). The difference is bigger for Fractalkine.



Univariate analyses

Base model on the full sample

Linear mixed models will be used to investigate the relationship between each of the immune marker levels separately and the disease outcome. The matching criteria (age, gender, cohort), the experimental phase and other potential confounders (BMI, education, physical activity, smoking status and alcohol intake) are set as fixed effects in the model. Due to the hierarchical structure arising from the technical covariate (plate), it is set as a random intercept. The strength of the association between each protein level and BCL case/control status is inferred using a likelihood ratio test comparing the model described above (m_0) and a model additionally including BCL status (m_1).

(m_0) protein ~ age + sex + cohort + phase + Imputed_bmi + Imputed_education + Imputed_activity + Imputed_smoking_status + Imputed_alcohol + (1 | plate)

(m_1) protein ~ age + sex + cohort + phase + Imputed_bmi + Imputed_education + Imputed_activity + Imputed_smoking_status + Imputed_alcohol + (1 | plate) + BCL status

Challenge 3: Apply the linear mixed models described above on the full sample. Compare the results with Supplementary table 5.

The code below compares both linear mixed models and denoises the data. In this case, the denoised data is obtained from the sum of the intercept, the product of the β coefficient corresponding to the disease status by the binary vector of disease status, and the residuals. This approach reduces potential confounding between the disease status and confounders compared to the one extracting only residuals. However, it strongly relies on the assumption that the relationship with the outcome is linear.

```
suppressPackageStartupMessages(library(lme4))
suppressPackageStartupMessages(library(stringr))
suppressPackageStartupMessages(library(RColorBrewer))

denoised = NULL
Beta_pooled = NULL
pvalue_pooled = NULL

f0 = "proteins[,k] ~ age + sex + cohort + phase + Imputed_bmi
+ Imputed_education + Imputed_activity + Imputed_smoking_status
+ Imputed_alcohol + (1 | plate)"
f1 = paste(f0, "+ type")

for (k in seq(1:ncol(proteins))) {
    print(k)
    model = lmer(as.formula(f1), data = covars, REML = FALSE,
        control = lmerControl(check.conv.singular = .makeCC(action = "ignore",
            tol = 1e-04)))
    model0 = lmer(as.formula(f0), data = covars, REML = FALSE,
        control = lmerControl(check.conv.singular = .makeCC(action = "ignore",
            tol = 1e-04)))
    pvalue_pooled = c(pvalue_pooled, anova(model, model0)$"Pr(>Chisq)"[2])

    beta = fixef(model)[c("(Intercept)", "type1")]
    Beta_pooled = c(Beta_pooled, fixef(model)["type1"])

    X = cbind(rep(1, length(covars$type)), as.numeric(covars$type))
    denoised = cbind(denoised, (X %*% beta + resid(model)))
}

colnames(denoised) = colnames(proteins)
rownames(denoised) = rownames(proteins)
saveRDS(denoised, "Data/Proteins_denoised.rds")

Table_pooled = cbind(Beta_pooled, pvalue_pooled)
rownames(Table_pooled) = colnames(proteins)
```

Model further adjusted on white blood cell counts

To investigate potential confounding by white blood cell counts, the models were further adjusted on the proportions of 5 of the 6 cell types (namely CD8, CD4, NK, B cells and monocytes).

White blood cell proportions are stored in the file “wbc.rds” in the folder “Data”.

```
wbc = readRDS("Data/wbc.rds")
```

Challenge 4: Apply the linear mixed models further adjusted on white blood cell (WBC) proportions on the sample for which this measure was available (224 case/control pairs):

1. Create the object `covars_wbc` that will contain all variables in `covars` as well as the WBC variables for all 224 pairs.

2. Create the object `proteins_wbc` as the subset of `proteins` with the 224 pairs.

3. Apply linear mixed models further adjusted on 5 of the 6 WBC proportions on these 224 pairs.

4. Compare the results with Supplementary table 6.

5. Reproduce Figure 1-a from the paper

The code below can be used to produce the objects `covars_wbc` and `proteins_wbc` and perform linear mixed models on these datasets.

```r
wbc = readRDS("Data/wbc.rds")
covars_wbc = merge(covars, wbc, by = "row.names", all = FALSE)
rownames(covars_wbc) = covars_wbc$Row.names
proteins_wbc = proteins[rownames(covars_wbc), ]

Beta_pooled_wbc = NULL
pvalue_pooled_wbc = NULL

f0 = "proteins_wbc[,k] ~ age + sex + cohort + phase + Imputed_bmi
+ Imputed_education + Imputed_activity + Imputed_smoking_status
+ Imputed_alcohol
+ CD4 + NK + Bcells + Mono + CD8 + (1 | plate)"
f1 = paste(f0, "+ type")

for (k in seq(1:ncol(proteins_wbc))) {
    print(k)
    model = lmer(as.formula(f1), data = covars_wbc,
        REML = FALSE, control = lmerControl(check.conv.singular = .makeCC(action = "ignore",
            tol = 1e-04)))
    model0 = lmer(as.formula(f0), data = covars_wbc,
        REML = FALSE, control = lmerControl(check.conv.singular = .makeCC(action = "ignore",
            tol = 1e-04)))
    pvalue_pooled_wbc = c(pvalue_pooled_wbc, anova(model,
        model0)$"Pr(>Chisq)"[2])

    Beta_pooled_wbc = c(Beta_pooled_wbc, fixef(model)["type1"])
}

Table_pooled_wbc = cbind(Beta_pooled_wbc, pvalue_pooled_wbc)
rownames(Table_pooled_wbc) = colnames(proteins_wbc)
```

STRATIFICATION ANALYSES

Challenge 5: Apply both the base models (LRT between m_0 and m_1) and models further adjusted on WBC on the following histological subtypes:

- chronic lymphocytic leukemia (CLL)
- diffuse large BCL (DLBCL)
- follicular lymphoma (FL)
- multiple myeloma (MM)

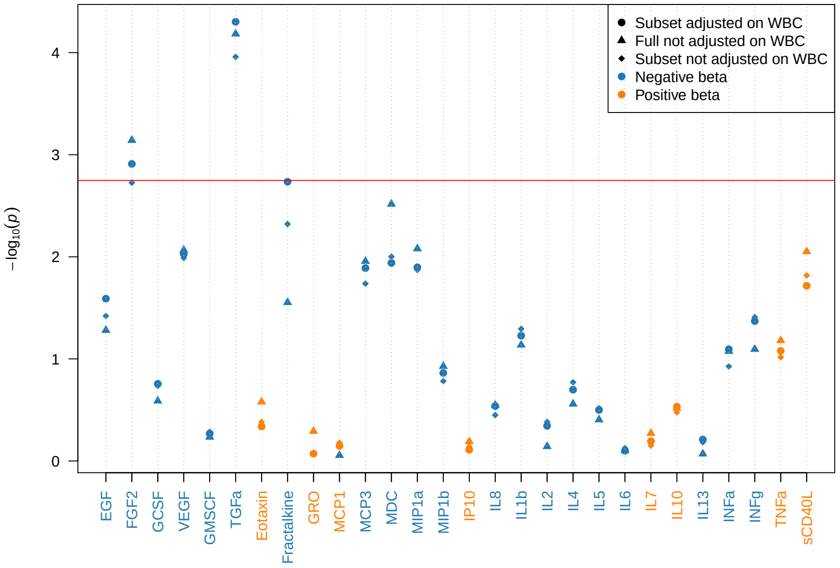
Check the consistency of your results with Supplementary Table 5 from the paper. You can now reproduce Figure 1-b, c, d and e from the paper.

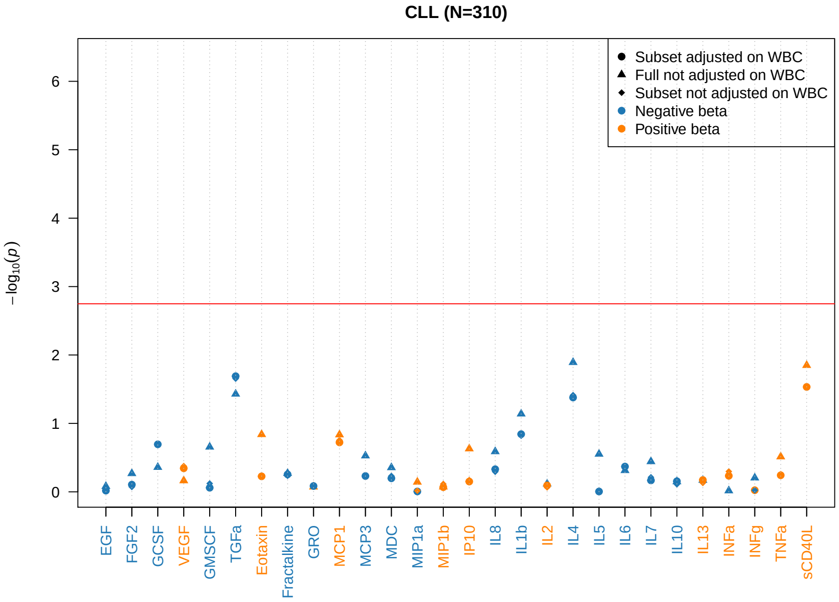
You can use the script “LMM_analyses.R” to perform these analyses. The portion of code copy-pasted below can be used to perform stratified analyses by BCL subtype on the full sample, without adjusting on WBC. Supplementary Table 5 is reproduced and saved as “Univariate_full_non_adjusted_wbc.txt”

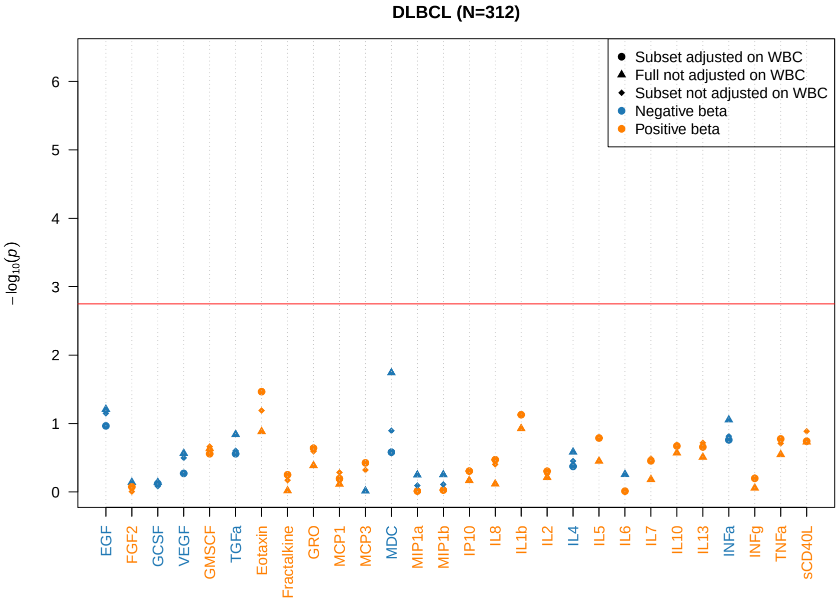
```
f0 = "proteins_subtype[,k] ~ age + sex + cohort + phase + Imputed_bmi
+ Imputed_education + Imputed_activity + Imputed_smoking_status
+ Imputed_alcohol + (1 | plate)"
f1 = paste(f0, "+ type")

for (subtype in c("BCLL", "DLBL", "FL", "MM")) {
    print(subtype)

    ids = c(covars$egm_id[covars$LY_subtype == ""],
        covars$egm_id[covars$LY_subtype == subtype])
    covars_subtype = covars[covars$egm_id %in% ids,
        ]
    proteins_subtype = proteins[covars_subtype$egm_id,
        ]
    Beta_subtype = NULL
    pvalue_subtype = NULL

    for (k in seq(1:ncol(proteins_subtype))) {
        model = lmer(as.formula(f1), data = covars_subtype,
            REML = FALSE, control = lmerControl(check.conv.singular = .makeCC(action = "ignore",
                tol = 1e-04)))
        model0 = lmer(as.formula(f0), data = covars_subtype,
            REML = FALSE, control = lmerControl(check.conv.singular = .makeCC(action = "ignore",
                tol = 1e-04)))
        pvalue_subtype = c(pvalue_subtype, anova(model,
            model0)$"Pr(>Chisq)"[2])

        Beta_subtype = c(Beta_subtype, fixef(model)["type1"])
    }

    Table_subtype = cbind(Beta_subtype, pvalue_subtype)
    rownames(Table_subtype) = colnames(proteins_subtype)

    assign(paste0("Table_", subtype), Table_subtype)

    Table_subtype = as.data.frame(Table_subtype)
    Table_subtype = cbind(round(Table_subtype$Beta_subtype,
        digits = 2), sprintf("%.2e", Table_subtype$pvalue_subtype))
    rownames(Table_subtype) = colnames(proteins)
    colnames(Table_subtype) = c("Beta", "pvalue")

    assign(paste0("Table_ready_", subtype), Table_subtype)
}

Table = cbind(Table_pooled, Table_BCLL, Table_DLBL,
    Table_FL, Table_MM)
colnames(Table) = paste(rep(c("Pooled", "CLL", "DLBCL",
    "FL", "MM"), each = 2), rep(c("Beta", "Pvalue"),
    5))
dir.create("Tables", showWarnings = FALSE)
write.table(Table, "Tables/Univariate_full_non_adjusted_wbc.txt",
    sep = "\t")
```

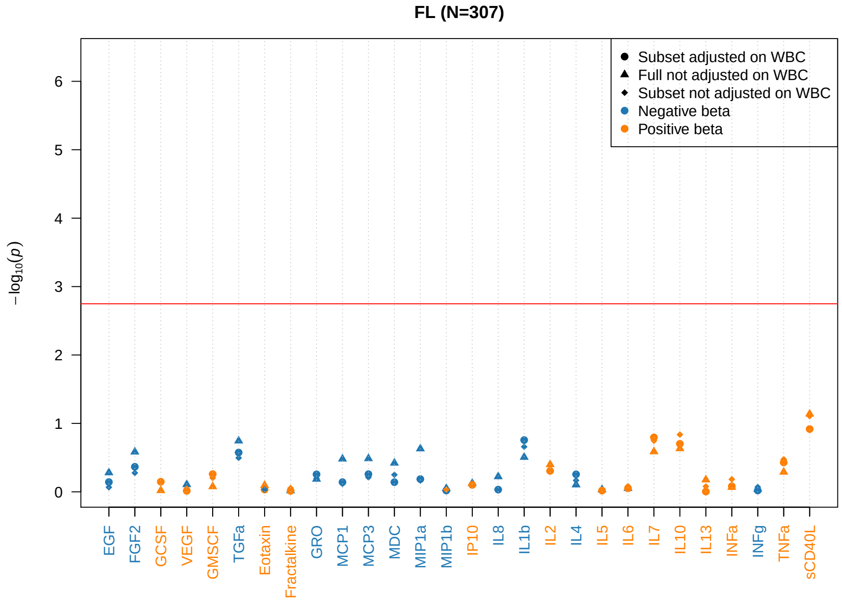
The same approach is used for the models adjusting on WBC (see script “LMM_analyses.R”), results are stored in “Univariate_subset_adjusted_wbc.txt”. Additionally, the base model was applied on the 224 case/control pairs, with results stored in “Univariate_subset_non_adjusted_wbc.txt”. Finally, the figures can be reproduced using the chunk of code below:

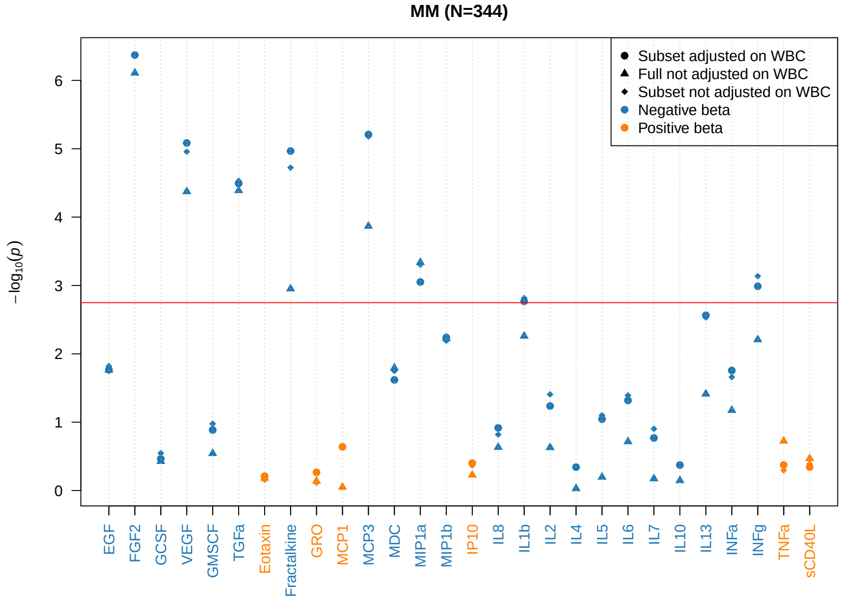
```
Table = read.table("Tables/Univariate_full_non_adjusted_wbc.txt")
Table_wbc = read.table("Tables/Univariate_subset_adjusted_wbc.txt")
Table_no_wbc = read.table("Tables/Univariate_subset_non_adjusted_wbc.txt")

MyPal = brewer.pal("Paired", n = 12)
MyPalPairsExt <- colorRampPalette(MyPal)(28)

Groups = c("EGF", "FGF2", "GCSF", "VEGF", "GMSCF",
    "TGFa", "Eotaxin", "Fractalkine", "GRO", "MCP1",
    "MCP3", "MDC", "MIP1a", "MIP1b", "IP10", "IL8",
    "IL1b", "IL2", "IL4", "IL5", "IL6", "IL7", "IL10",
    "IL13", "INFa", "INFg", "TNFa", "sCD40L")

Table = Table[Groups, ]
Table_wbc = Table_wbc[Groups, ]
Table_no_wbc = Table_no_wbc[Groups, ]

dir.create("Figures", showWarnings = FALSE)
pdf("Figures/Fig1_A.pdf", height = 7, width = 10)
par(mar = c(6, 5, 3, 3))
plot(-log10(Table$Pooled.Pvalue), pch = 17, col = ifelse(Table$Pooled.Beta <
    0, yes = MyPal[2], no = MyPal[8]), xaxt = "n",
    ylab = expression(-log[10](italic(p))), xlab = "",
    las = 1, ylim = c(min(c(-log10(Table$Pooled.Pvalue),
        -log10(Table_wbc$Pooled.Pvalue))), max(c(-log10(Table$Pooled.Pvalue),
        -log10(Table_wbc$Pooled.Pvalue)))))
points(-log10(Table_wbc$Pooled.Pvalue), pch = 19, col = ifelse(Table_wbc$Pooled.Beta <
    0, yes = MyPal[2], no = MyPal[8]))
points(-log10(Table_no_wbc$Pooled.Pvalue), pch = 18,
    col = ifelse(Table_no_wbc$Pooled.Beta < 0, yes = MyPal[2],
        no = MyPal[8]))
abline(h = -log10(0.05/28), col = "red")
abline(v = seq(1, 28), lty = 3, col = "grey")
axis(1, at = 1:28, ifelse(Table_wbc$Pooled.Beta < 0,
    yes = rownames(Table), no = ""), las = 2, col.axis = MyPal[2])
axis(1, at = 1:28, ifelse(Table_wbc$Pooled.Beta < 0,
    yes = "", no = rownames(Table)), las = 2, col.axis = MyPal[8])
legend("topright", pch = c(19, 17, 18, 19, 19), col = c("black",
    "black", "black", MyPal[2], MyPal[8]), legend = c("Subset adjusted on WBC",
    "Full not adjusted on WBC", "Subset not adjusted on WBC",
    paste("Negative", expression(beta)), paste("Positive",
        expression(beta))))
dev.off()


## By subtype

for (k in 1:4) {
    subtype = c("BCLL", "DLBL", "FL", "MM")[k]
    subtype_name = c("CLL", "DLBCL", "FL", "MM")[k]

    Table1 = eval(parse(text = paste0("Table_", subtype)))
    Table2 = eval(parse(text = paste0("Table_", subtype,
        "_wbc")))
    Table3 = eval(parse(text = paste0("Table_", subtype,
        "_no_wbc")))

    Table1 = Table1[Groups, ]
    Table2 = Table2[Groups, ]
    Table3 = Table3[Groups, ]

    pdf(paste0("Figures/Fig1_", subtype_name, ".pdf"),
        height = 7, width = 10)
    par(mar = c(6, 5, 3, 3))
    plot(-log10(Table1[, 2]), pch = 17, col = ifelse(Table1[,
        1] < 0, yes = MyPal[2], no = MyPal[8]), xaxt = "n",
        ylab = expression(-log[10](italic(p))), xlab = "",
        las = 1, ylim = c(min(c(-log10(Table$MM.Pvalue),
            -log10(Table_wbc$MM.Pvalue))), max(c(-log10(Table$MM.Pvalue),
            -log10(Table_wbc$MM.Pvalue)))), main = paste0(subtype_name,
            " (N= ", sum(covars$LY_subtype == subtype),
            ")"))
    points(-log10(Table2[, 2]), pch = 19, col = ifelse(Table2[,
        1] < 0, yes = MyPal[2], no = MyPal[8]))
    points(-log10(Table3[, 2]), pch = 18, col = ifelse(Table3[,
        1] < 0, yes = MyPal[2], no = MyPal[8]))
    abline(h = -log10(0.05/28), col = "red")
    abline(v = seq(1, 28), lty = 3, col = "grey")
    axis(1, at = 1:28, ifelse(Table2[, 1] < 0, yes = rownames(Table),
        no = ""), las = 2, col.axis = MyPal[2])
    axis(1, at = 1:28, ifelse(Table2[, 1] < 0, yes = "",
        no = rownames(Table)), las = 2, col.axis = MyPal[8])
    legend("topright", pch = c(19, 17, 18, 19, 19),
        col = c("black", "black", "black", MyPal[2],
            MyPal[8]), legend = c("Subset adjusted on WBC",
            "Full not adjusted on WBC", "Subset not adjusted on WBC",
            paste("Negative", expression(beta)), paste("Positive",
                expression(beta))))
    dev.off()
}
```




CLL (N=310)
−log10(p)
0
1
2
3
4
5
6
● Subset adjusted on WBC
▲ Full not adjusted on WBC
◆ Subset not adjusted on WBC
● Negative beta
● Positive beta
EGF
FGF2
GCSF
VEGF
GMSCF
TGFa
Eotaxin
Fractalkine
GRO
MCP1
MCP3
MDC
MIP1a
MIP1b
IP10
IL8
IL1b
IL2
IL4
IL5
IL6
IL7
IL10
IL13
INFa
INFg
TNFa
sCD40L


DLBCL (N=312)
−log10(p)
0
1
2
3
4
5
6
● Subset adjusted on WBC
▲ Full not adjusted on WBC
◆ Subset not adjusted on WBC
● Negative beta
● Positive beta
EGF
FGF2
GCSF
VEGF
GMSCF
TGFa
Eotaxin
Fractalkine
GRO
MCP1
MCP3
MDC
MIP1a
MIP1b
IP10
IL8
IL1b
IL2
IL4
IL5
IL6
IL7
IL10
IL13
INFa
INFg
TNFa
sCD40L


FL (N=307)
Subset adjusted on WBC
Full not adjusted on WBC
Subset not adjusted on WBC
Negative beta
Positive beta
$-\log_{10}(p)$
0
1
2
3
4
5
6
EGF
FGF2
GCSF
VEGF
GMSCF
TGFa
Eotaxin
Fractalkine
GRO
MCP1
MCP3
MDC
MIP1a
MIP1b
IP10
IL8
IL1b
IL2
IL4
IL5
IL6
IL7
IL10
IL13
INFa
INFg
TNFa
sCD40L


MM (N=344)
Subset adjusted on WBC
Full not adjusted on WBC
Subset not adjusted on WBC
Negative beta
Positive beta
$-\log_{10}(p)$
0
1
2
3
4
5
6
EGF
FGF2
GCSF
VEGF
GMSCF
TGFa
Eotaxin
Fractalkine
GRO
MCP1
MCP3
MDC
MIP1a
MIP1b
IP10
IL8
IL1b
IL2
IL4
IL5
IL6
IL7
IL10
IL13
INFa
INFg
TNFa
sCD40L

PLS analyses

A series of PLS-DA analyses will be performed to assess the joint inflammatory signal related to BCL or any of its histological subtypes. The PLS-DA (where DA stands for Discriminant Analysis) is an extension of PLS models to deal with categorical outcomes. The sparse and sparse-group extensions allow us to perform variable selection.

Both models have been implemented in R, in the package `sgPLS`. We are going to use functions from this package in what follows below. These functions depend on functions from another package: `mixOmics`.

Analyses on all BCL cases

We are going to perform sparse and sparse-group PLS-DA on the 28 proteins in association with all BCL status. That is, we want to identify the proteins that best relate to differences between B-cell lymphoma cases and healthy controls.

Running your first (penalised) PLS models

Loading packages and functions

First of all, set the working directory in R to be in "BCL_analyses".

Load all packages and functions required using the code below:

```r
suppressPackageStartupMessages(library(sgPLS))
suppressPackageStartupMessages(library(pheatmap))
suppressPackageStartupMessages(library(RColorBrewer))
suppressPackageStartupMessages(library(utils))
suppressPackageStartupMessages(library(pheatmap))

source("Scripts/pls_functions.R")
```

This script loads functions from a previous version of mixOmics. In general, we do not recommend to do this, however, in this case we know that the functions work in the R version 3.5.1.

Loading the data

As in the tutorial on penalised regression, we are going to use a two-step strategy to account for technical confounders and potential covariates. In the first step, a series of linear mixed models adjusted on these variables are run. Then, the residuals from these models are extracted and PLS models are applied to the residuals. We have denoised the data, using the process described and you can load it using the code below.

```r
X_pooled = readRDS("Data/Proteins_denoised.rds")
covars = readRDS("Data/Covariates.rds")
Y_pooled = covars$type
```

Running a PLS-DA model

Non-penalised PLS-DA models can be run using the `plsda()` function as follows:

```r
MyPLSDA_pooled <- plsda(X_pooled, Y_pooled, ncomp = 1)
```

The loadings coefficients on X and Y can be extracted as illustrated below:

```r
MyPLSDA_pooled$loadings$X
```

```
##                   comp1
## EGF         -0.19882817
## Eotaxin      0.11476941
## FGF2        -0.34542652
## Fractalkine -0.22514502
## GCSF        -0.11645709
## GRO          0.06716995
## INFa        -0.17806199
## IP10         0.04748458
## MCP1        -0.01541968
## MCP3        -0.26075388
## MDC         -0.30392463
## MIP1a       -0.27042592
## MIP1b       -0.16005826
## sCD40L       0.26715264
## TGFa        -0.40834743
## VEGF        -0.26778251
## IL1b        -0.18493104
## IL2         -0.03694567
## IL4         -0.11239825
## IL5         -0.08792728
## IL6         -0.02768646
## IL7          0.06391408
## IL8         -0.11069130
## IL10         0.10604398
## IL13        -0.01936663
## INFg        -0.18007002
## GMSCF       -0.05661752
## TNFa         0.18918607
```

```
MyPLSDA_pooled$loadings$Y
```

```
##        comp1
## 0 -0.7071068
## 1  0.7071068
```

The proportion of variance explained by X on X and by Y on Y are also stored in the output of `plsda()`:

```
MyPLSDA_pooled$explained_variance
```

```
## $X
##   comp 1
## 0.259475
##
## $Y
## comp 1
##      1
```

As the outcome is binary, the 1st component on Y explains 100% of the variance in Y. For this reason, the number of components to use when running (penalised) PLS-DA models with a binary outcome should not exceed 1.

Running an uncalibrated sparse PLS-DA model

In the sgPLS package, the sparse PLS-DA has been implemented so that the user has to provide a number of variables to be selected by the model rather than a penalty parameter to ease interpretability. The sPLS-DA parameter has to be specified in the keepX argument of the `splsda()` function.

For example, the command below will fit a sparse PLS-DA model including 5 variables.

```
MysPLSDA_pooled <- splsda(X_pooled, Y_pooled, ncomp = 1,
    keepX = 5)
```

We can see which variables were selected by looking at the loadings coefficients in X:

```
MysPLSDA_pooled$loadings$X
```

```
##                   comp1
## EGF          0.000000000
## Eotaxin      0.000000000
## FGF2        -0.472645096
## Fractalkine  0.000000000
## GCSF         0.000000000
## GRO          0.000000000
## INFa          0.000000000
## IP10          0.000000000
## MCP1          0.000000000
## MCP3          0.000000000
## MDC          -0.222042192
## MIP1a        -0.019765251
## MIP1b         0.000000000
## sCD40L        0.000000000
## TGFa         -0.852583572
## VEGF         -0.003803395
## IL1b          0.000000000
## IL2           0.000000000
## IL4           0.000000000
## IL5           0.000000000
## IL6           0.000000000
## IL7           0.000000000
## IL8           0.000000000
## IL10          0.000000000
## IL13          0.000000000
## INFg          0.000000000
## GMSCF         0.000000000
## TNFa          0.000000000
```

The selected variables are the ones with non-zero loadings coefficients:

```
MysPLSDA_pooled$loadings$X[MysPLSDA_pooled$loadings$X !=
    0, ]
```

```
##         FGF2          MDC        MIP1a         TGFa         VEGF
## -0.472645096 -0.222042192 -0.019765251 -0.852583572 -0.003803395
```

Incorporating a priori knowledge: setting group membership

The group and sparse group PLS functions in the sgPLS package require the variables to be ordered by group. The parameter defining group membership, Xgroups, is a vector of length $G-1$, where G is the number of groups and it contains the position of the last element in the respective group. See example below for further clarification.

In our protein data, the proteins have been grouped based on their protein functions to generate 3 groups:

- Growth Factors (N=6 proteins)
- Chemokines (N=10)

- Cytokines (N=12)

In the code below, the proteins dataset is rearranged such that the 6 Growth Factors come first, followed by the 10 Chemokines and lastly the 12 Cytokines.

```
Groups = c("EGF", "FGF2", "GCSF", "VEGF", "GMSCF",
    "TGFa", "Eotaxin", "Fractalkine", "GRO", "MCP1",
    "MCP3", "MDC", "MIP1a", "MIP1b", "IP10", "IL8",
    "IL1b", "IL2", "IL4", "IL5", "IL6", "IL7", "IL10",
    "IL13", "INFa", "INFg", "TNFa", "sCD40L")
X_pooled = X_pooled[, Groups]
```

The parameter given to the sgPLSDA function is written as follows in this case:

```
Xgroups = c(6, 16)
```

By specifying that `Xgroups=c(6,16)` we are saying that the first 6 variables form a group, the variables 7 to 16 form another group and variables 17 to 28 form the last group.

Group membership is set with the `ind.block.x` argument of the `gPLSda` or `sgPLSda` functions. The number of groups to be selected by the model is then defined by the `keepX` argument. For example, the model below will include only one of the three groups of proteins:

```
MygPLSDA_pooled <- gPLSda(X_pooled, Y_pooled, ncomp = 1,
    ind.block.x = Xgroups, keepX = 1)
```

By looking at the loadings coefficients, we can see which group was selected:

```
MygPLSDA_pooled$loadings$X
```

```
##                  comp 1
## EGF         -0.30898572
## FGF2        -0.53680452
## GCSF        -0.18097826
## VEGF        -0.41614310
## GMSCF       -0.08798554
## TGFa        -0.63458574
## Eotaxin      0.00000000
## Fractalkine  0.00000000
## GRO          0.00000000
## MCP1         0.00000000
## MCP3         0.00000000
## MDC          0.00000000
## MIP1a        0.00000000
## MIP1b        0.00000000
## IP10         0.00000000
## IL8          0.00000000
## IL1b         0.00000000
## IL2          0.00000000
## IL4          0.00000000
## IL5          0.00000000
## IL6          0.00000000
## IL7          0.00000000
## IL10         0.00000000
## IL13         0.00000000
## INFa         0.00000000
## INFg         0.00000000
## TNFa         0.00000000
## sCD40L       0.00000000
```

In this case, only the Growth Factors were selected by the model, and they all have non-zero coefficients. Sparse-group PLS-DA can be used to add a second layer of sparsity by selecting the variables within the groups. In the `sgPLSda()` function, this is done via a second sparsity parameter `alpha.x` that is between 0 and 1. As illustrated below, the closer to 1 alpha is, the more the coefficients are shrunk.

```
MysgPLSDA_pooled <- sgPLSda(X_pooled, Y_pooled, ncomp = 1,
    ind.block.x = Xgroups, keepX = 1, alpha.x = 0.1)
MysgPLSDA_pooled$loadings$X
```

```
##                    [,1]
## EGF         -0.30371552
## FGF2        -0.53927943
## GCSF        -0.17135623
## VEGF        -0.41451591
## GMSCF       -0.07520206
## TGFa        -0.64038490
## Eotaxin      0.00000000
## Fractalkine  0.00000000
## GRO          0.00000000
## MCP1         0.00000000
## MCP3         0.00000000
## MDC          0.00000000
## MIP1a        0.00000000
## MIP1b        0.00000000
## IP10         0.00000000
## IL8          0.00000000
## IL1b         0.00000000
## IL2          0.00000000
## IL4          0.00000000
## IL5          0.00000000
## IL6          0.00000000
## IL7          0.00000000
## IL10         0.00000000
## IL13         0.00000000
## INFa         0.00000000
## INFg         0.00000000
## TNFa         0.00000000
## sCD40L       0.00000000
```

```
MysgPLSDA_pooled <- sgPLSda(X_pooled, Y_pooled, ncomp = 1,
    ind.block.x = Xgroups, keepX = 1, alpha.x = 0.9)
MysgPLSDA_pooled$loadings$X
```

```
##                   [,1]
## EGF          0.0000000
## FGF2        -0.5528206
## GCSF         0.0000000
## VEGF        -0.2567278
## GMSCF        0.0000000
## TGFa        -0.7927674
## Eotaxin      0.0000000
## Fractalkine  0.0000000
## GRO          0.0000000
## MCP1         0.0000000
## MCP3         0.0000000
## MDC          0.0000000
## MIP1a        0.0000000
## MIP1b        0.0000000
## IP10         0.0000000
## IL8          0.0000000
## IL1b         0.0000000
## IL2          0.0000000
## IL4          0.0000000
## IL5          0.0000000
## IL6          0.0000000
## IL7          0.0000000
## IL10         0.0000000
## IL13         0.0000000
## INFa         0.0000000
## INFg         0.0000000
## TNFa         0.0000000
## sCD40L       0.0000000
```

Model calibration

Both sparse and sparse-group extensions depend on parameters that have to be calibrated. In the case of the sPLS, the parameter to be calibrated is the number of variables to be selected (i.e. with non zero loadings coefficients). In the case of the sgPLS, there are two parameters:

- the number of groups to be selected,
- a sparsity parameter.

You have been provided with the script "pls_functions.R". We are going to go through how to use some of the functions contained there within to understand how the calibration of sparse and sparse group PLS-DA models works.

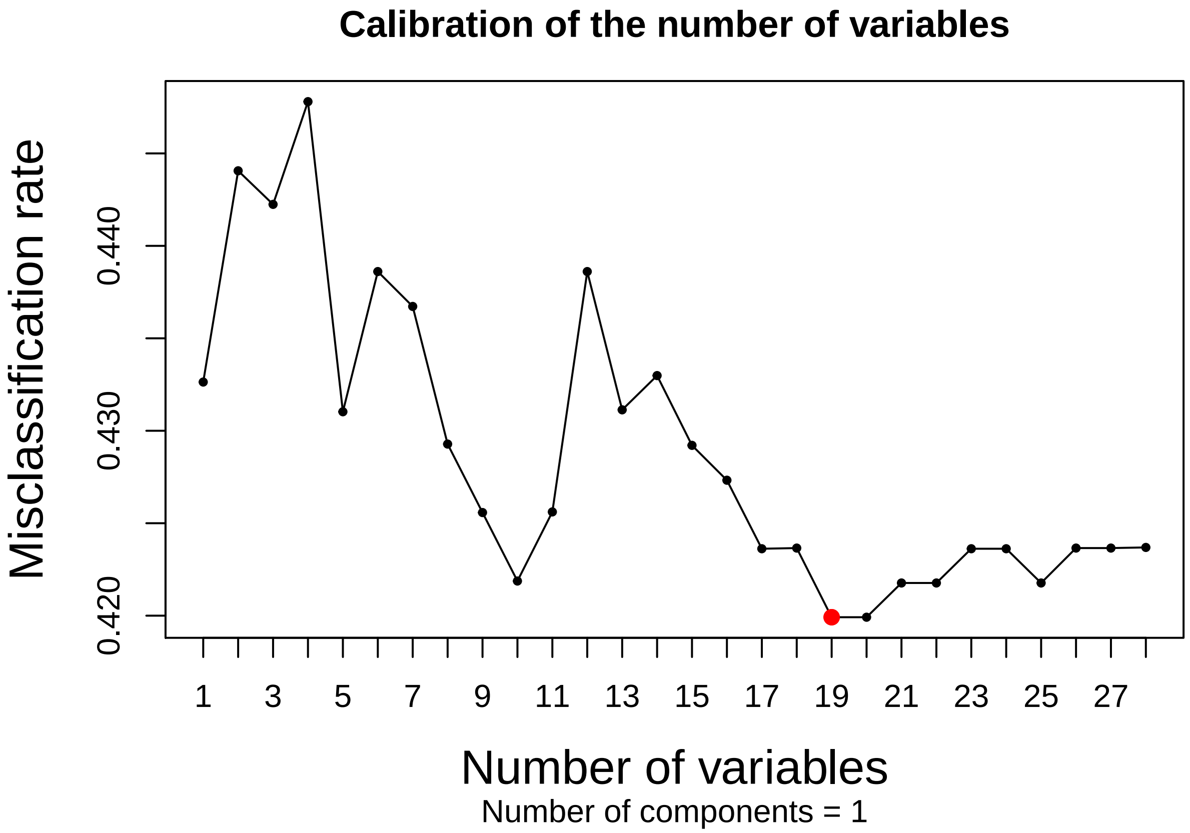
Running the function CalibratesPLSDA()

This function performs 5-fold cross-validation to calibrate the number of variables selected by sPLS-DA. The optimisation criterion is the misclassification rate (i.e. proportion of samples that were predicted to belong to the wrong category). The parameter `ncomp` is the number of components for which we want to calibrate the number of variables to be selected. In the example below, we only calibrate the first component. The parameter `Nrepeat` is the number of CV iterations to perform, for each of the possible numbers of variables (i.e. for numbers between 1 and 28 in this case).

Try to run the CV for 5 iterations using the code below. The command `PlotCalib` allows you to visualise the results from the CV iterations.

```r
set.seed(1)
res_splsda = CalibratesPLSDA(dataX = X_pooled, dataY = Y_pooled,
    ncomp = 1, Nrepeat = 5)
```

```r
PlotCalib(res = res_splsda)
```



In this instance, the optimal number of variables is 19 (leading to the smallest misclassification rate). However, the results that you obtain depend on the number of iterations of the CV procedure. In the paper, the calibration was run with 100 iterations. The optimal number of variables was found to be 14.

Running the function `CalibratesgPLSDA()`

The two parameters of the sgPLS-DA model can be calibrated using the function `CalibratesgPLSDA()` as illustrated below:

```
Groups = c("EGF", "FGF2", "GCSF", "VEGF", "GMSCF",
    "TGFa", "Eotaxin", "Fractalkine", "GRO", "MCP1",
    "MCP3", "MDC", "MIP1a", "MIP1b", "IP10", "IL8",
    "IL1b", "IL2", "IL4", "IL5", "IL6", "IL7", "IL10",
    "IL13", "INFa", "INFg", "TNFa", "sCD40L")
X_pooled = X_pooled[, Groups]
```

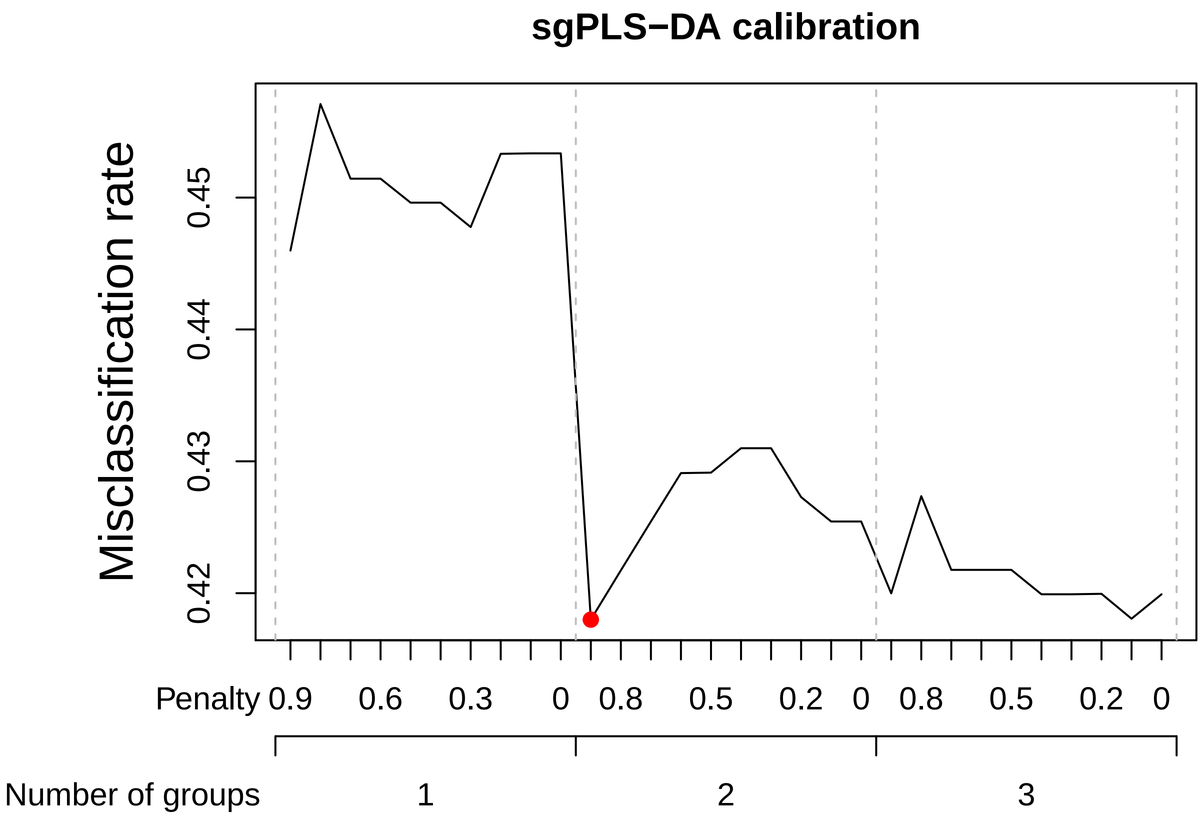
The parameter given to the sgPLSDA function is written as follows in this case:

```
Xgroups = c(6, 16)
```

By specifying that `Xgroups=c(6,16)` we are saying that the first 6 variables form a group, the variables 7 to 16 form another group and variables 17 to 28 form the last group.

```r
set.seed(1)
res_sgplsda = CalibratesgPLSDA(dataX = X_pooled, dataY = Y_pooled,
    ncomp = 1, Nrepeat = 5, Xgroups = Xgroups)
```

```r
PlotCalib(res = res_sgplsda, type = "sgPLSDA")
```



The parameters are displayed on the X-axis as “number of groups” - “sparsity parameter”. As for sPLSDA, the optimal pair of parameters is the one leading to smallest MSEP. In this case, it would be: 2 groups with a sparsity parameter of 0.9. But again, the number of iterations was very small in this case (to avoid long computation times). The optimal set of parameters in the paper was obtained with 100 iterations: 2 groups with a sparsity coefficient of 0.6.

Fitting PLS models

In this section, we are going to run the sPLS-DA and sgPLS-DA models with the parameters that were calibrated as above.

First, we will prepare the stratified datasets in preparation for the subtype analyses and the BCL case dataset for the time-to-diagnosis analyses:

```r
for (subtype in c("BCLL", "DLBL", "FL", "MM")) {
    sets = covars$egm_set[covars$LY_subtype == subtype]
    X = X_pooled[covars$egm_set %in% sets, ]
    Y = Y_pooled[covars$egm_set %in% sets]
    assign(paste0("X_", subtype), X)
    assign(paste0("Y_", subtype), Y)
}

Y_diagnostic = covars$Time_to_diag[as.character(covars$type) ==
    "1"]  ## time to diagnostic in days
X_diagnostic = X_pooled[as.character(covars$type) ==
    "1", ]
names(Y_diagnostic) = rownames(X_diagnostic)

for (subtype in c("BCLL", "DLBL", "FL", "MM")) {
    ids = covars$egm_id[covars$LY_subtype == subtype]
    X = X_diagnostic[ids, ]
    Y = Y_diagnostic[ids]
    assign(paste0("X_diag_", subtype), X)
    assign(paste0("Y_diag_", subtype), Y)
}

MyPal = brewer.pal("Paired", n = 12)
```

The calibrated parameters are read from the file "MyParameters.rds" in the folder "Data":

```
MyParameters = readRDS("Data/MyParameters.rds")
```

The object `MyParameters` contains parameters of the s, g and sg PLS-DA models arranged by type of analysis:

```
str(MyParameters)
```

```
## List of 6
##  $ Pooled:List of 3
##   ..$ sPLSDA :List of 1
##   .. ..$ NVar: num 14
##   ..$ gPLSDA :List of 1
##   .. ..$ NGroups: num 1
##   ..$ sgPLSDA:List of 2
##   .. ..$ NGroups: num 2
##   .. ..$ alpha  : num 0.6
##  $ DLBL  :List of 3
##   ..$ sPLSDA :List of 1
##   .. ..$ NVar: num 2
##   ..$ gPLSDA :List of 1
##   .. ..$ NGroups: num 1
##   ..$ sgPLSDA:List of 2
##   .. ..$ NGroups: num 1
##   .. ..$ alpha  : num 0.8
##  $ FL    :List of 3
##   ..$ sPLSDA :List of 1
##   .. ..$ NVar: num 1
##   ..$ gPLSDA :List of 1
##   .. ..$ NGroups: num 1
##   ..$ sgPLSDA:List of 2
##   .. ..$ NGroups: num 1
##   .. ..$ alpha  : num 0.9
##  $ BCLL  :List of 3
##   ..$ sPLSDA :List of 1
##   .. ..$ NVar: num 6
##   ..$ gPLSDA :List of 1
##   .. ..$ NGroups: num 1
##   ..$ sgPLSDA:List of 2
##   .. ..$ NGroups: num 1
##   .. ..$ alpha  : num 0.8
##  $ MM    :List of 3
##   ..$ sPLSDA :List of 1
##   .. ..$ NVar: num 9
##   ..$ gPLSDA :List of 1
##   .. ..$ NGroups: num 1
##   ..$ sgPLSDA:List of 2
##   .. ..$ NGroups: num 1
##   .. ..$ alpha  : num 0.6
##  $ Time  :List of 3
##   ..$ sPLS :List of 1
##   .. ..$ NVar: int 1
##   ..$ gPLS :List of 1
##   .. ..$ NGroups: int 1
##   ..$ sgPLS:List of 2
##   .. ..$ NGroups: num 1
##   .. ..$ alpha  : num 0.9
```

The code below can be used to run PLS-DA, sPLS-DA, gPLS-DA and sgPLS-DA models with the calibrated parameters:

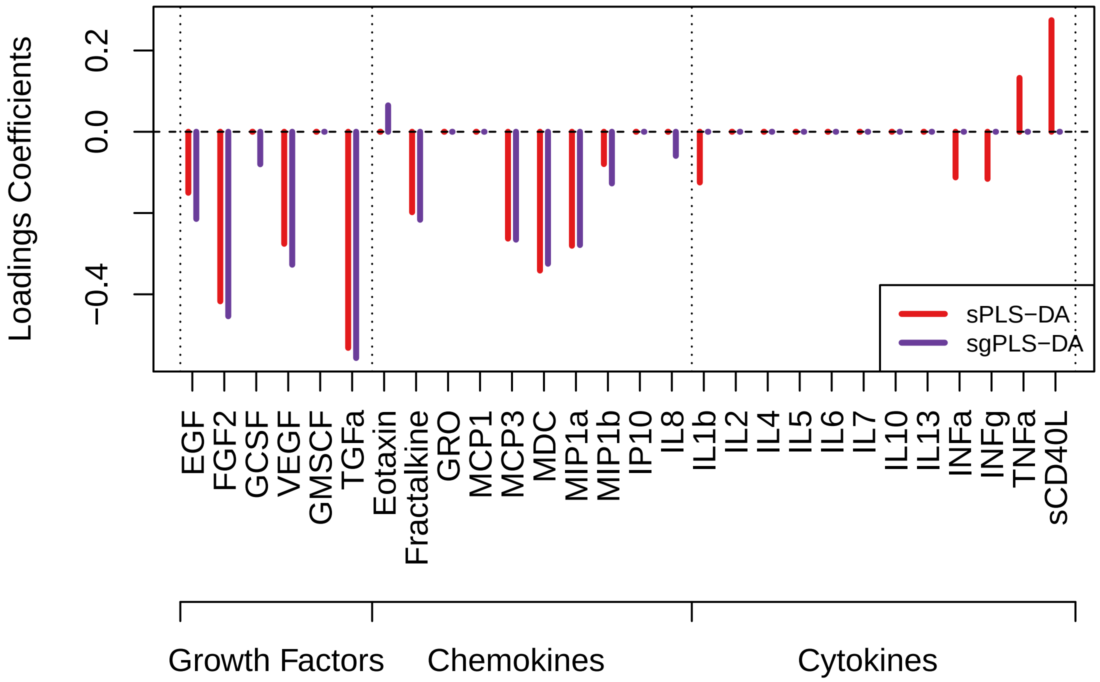
```
MyPLSDA_pooled <- plsda(X_pooled, Y_pooled, ncomp = 1)
MysPLSDA_pooled <- splsda(X_pooled, Y_pooled, keepX = MyParameters$Pooled$sPLSDA$NVar,
    ncomp = 1)
MygPLSDA_pooled <- gPLSda(X_pooled, Y_pooled, ncomp = 1,
    ind.block.x = Xgroups, keepX = MyParameters$Pooled$gPLSDA$NGroups)
```

```
MysgPLSDA_pooled <- sgPLSda(X_pooled, Y_pooled, ncomp = 1,
    ind.block.x = Xgroups, keepX = MyParameters$Pooled$sgPLSDA$NGroups,
    alpha.x = MyParameters$Pooled$sgPLSDA$alpha)
```

Visualising the loadings coefficients

We are only going to look at the sPLS-DA and sgPLS-DA models. You can now reproduce Fig2-a from the paper:

```
Loadings = cbind(MysPLSDA_pooled$loadings$X, MysgPLSDA_pooled$loadings$X,
    rep(NA, 28), rep(NA, 28))
Loadings = as.vector(t(Loadings))
Loadings = Loadings[-c(length(Loadings) - 1, length(Loadings))]

par(mar = c(10, 5, 3, 3))
plot(Loadings, col = c(MyPal[6], MyPal[10], NA, NA),
    xaxt = "n", ylab = "Loadings Coefficients", type = "h",
    lwd = 3, xlab = "")
axis(1, at = seq(1.5, 28 * 4, by = 4), labels = colnames(MyPLSDA_pooled$X),
    las = 2)
axis(1, at = c(0, Xgroups, 28) * 4, line = 6, labels = NA)
axis(1, at = c(3, 10.5, 21.5) * 4, labels = c("Growth Factors",
    "Chemokines", "Cytokines"), line = 6, tick = FALSE)
abline(v = c(0, Xgroups, 28) * 4, lty = 3, col = "black")
abline(h = 0, lty = 2)
legend("bottomright", legend = c("sPLS-DA", "sgPLS-DA"),
    lty = 1, lwd = 3, col = c(MyPal[6], MyPal[10]),
    cex = 0.75)
```



There is a good consistency between the sPLS-DA and sgPLS-DA models. See paper for further interpretation of the results.

Visualising the misclassification rate

The fitted values of the sPLS-DA model can be obtained using the function `predict()`:

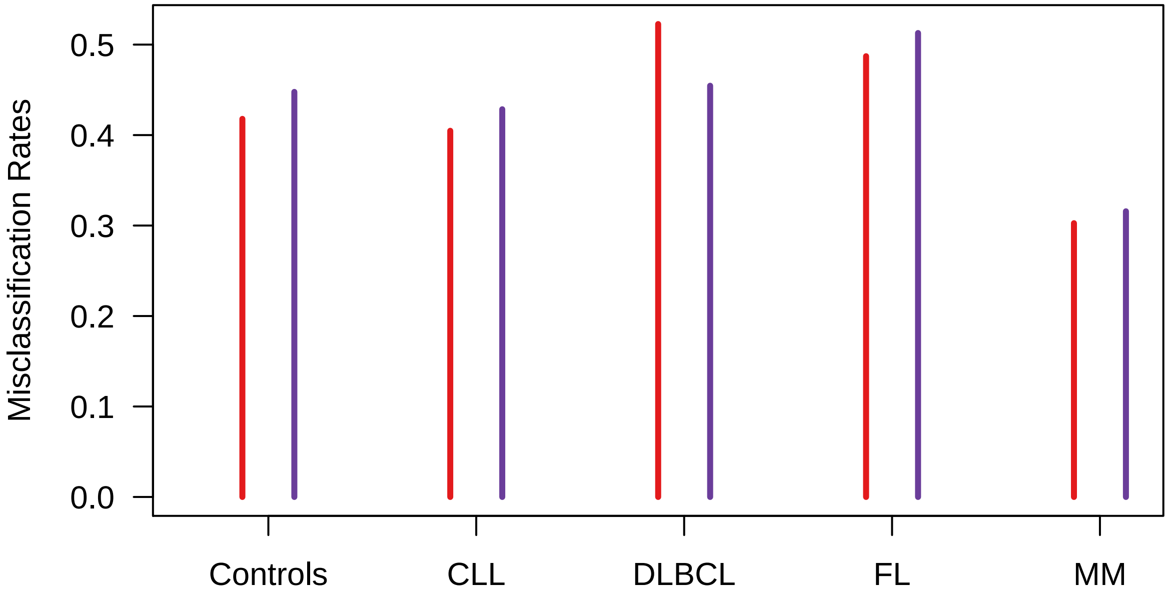
```
MyPredict = predict(MysPLSDA_pooled, newdata = X_pooled)
fitted = MyPredict$class$max.dist
table(fitted)
```

```
## fitted
##   0   1
## 264 272
```

Challenge 6: Compute the misclassification rate of the sPLS-DA model on the dataset used for inference, i.e. the proportion of fitted values that differ from the true values. Compute the misclassification rate by subtype for the sPLS-DA and sgPLS-DA models to reproduce Figure 2-c from the paper.

You have been provided with the script “PLS_analyses_solutions.R” that contains code that can be used for the whole set of PLS analyses. In particular, the chunk of code below can be used for this challenge:

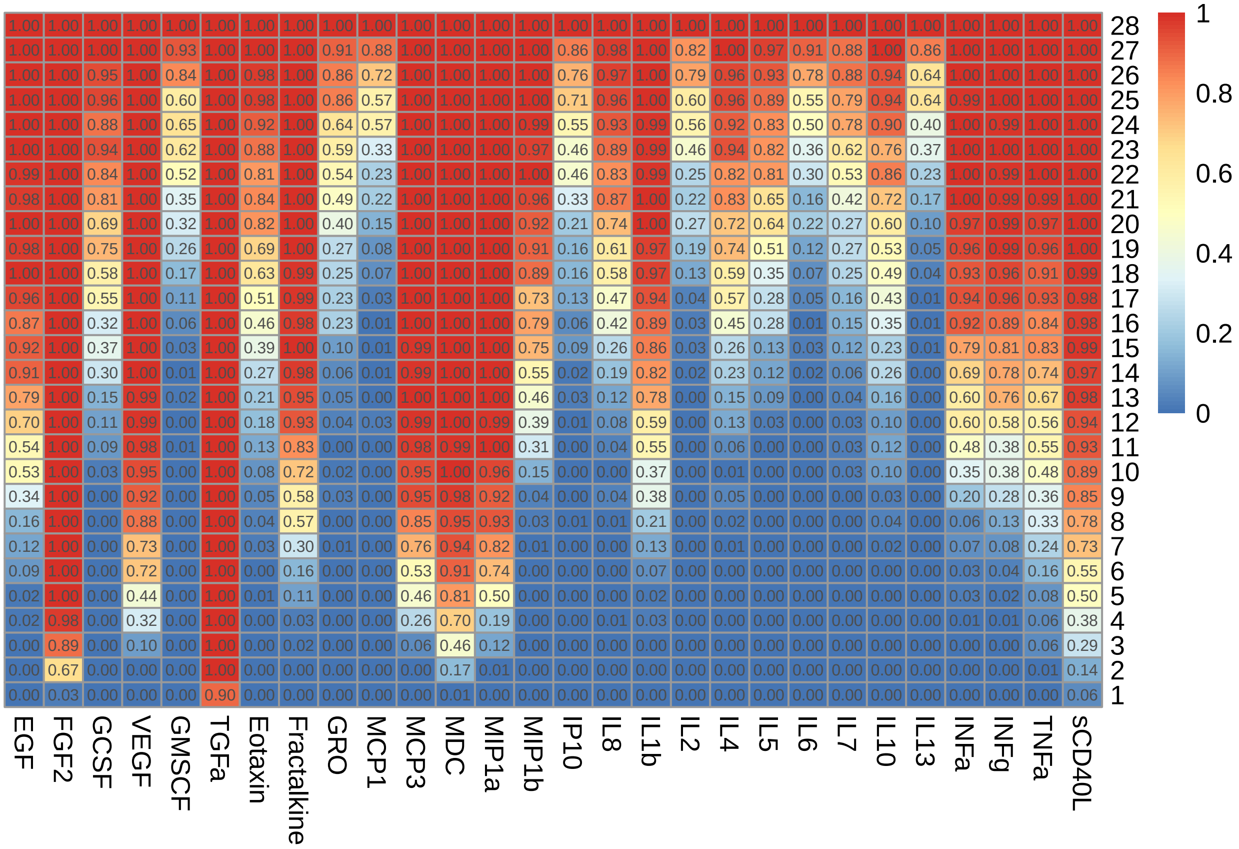
```
MSEP_sPLSDA = NULL
for (subtype in c("", "BCLL", "DLBL", "FL", "MM")) {
    idx = which(covars$LY_subtype == subtype)
    MSEP_sPLSDA[[subtype]] = 1 - sum(diag(table(Y_pooled[idx],
        MyPredict$class$max.dist[idx])))/length(idx)
}

MyPredict = predict(MysgPLSDA_pooled, newdata = X_pooled)

MSEP_sgPLSDA = NULL
for (subtype in c("", "BCLL", "DLBL", "FL", "MM")) {
    idx = which(covars$LY_subtype == subtype)
    MSEP_sgPLSDA[[subtype]] = 1 - sum(diag(table(Y_pooled[idx],
        MyPredict$class$max.dist[idx])))/length(idx)
}

MSEP = cbind(MSEP_sPLSDA, MSEP_sgPLSDA)
rownames(MSEP) = c("Controls", "CLL", "DLBCL", "FL",
    "MM")

MSEP = cbind(MSEP, rep(NA, 5), rep(NA, 5))
MSEP = unlist(t(MSEP)[1:(nrow(t(MSEP)) * ncol(t(MSEP)))])
# MSEP=as.vector(t(MSEP))
MSEP = MSEP[-c(length(MSEP) - 1, length(MSEP))]

plot(MSEP, type = "h", lwd = 3, xaxt = "n", xlab = "",
    ylab = "Misclassification Rates", col = c(MyPal[6],
        MyPal[10]), xlim = c(0, length(MSEP)), ylim = c(0,
        max(MSEP[!is.na(MSEP)])), las = 1)
axis(1, at = seq(1.5, 5 * 4, by = 4), labels = c("Controls",
    "CLL", "DLBCL", "FL", "MM"))
```



Stability analyses

In order to explore the consistency in variable selection in sPLS-DA over different parameters, we performed stability analyses. These are based on a subsampling procedure. For each subsampling iteration, and each parameter, the sPLS-DA model is fitted and the selected variables are stored. The proportions of selection of the variables can be visualised using a heatmap:

```r
set.seed(1)
Stability_results = StabilityPlot(X = X_pooled, Y = Y_pooled,
    NIter = 100)

pheatmap(Stability_results, cluster_rows = FALSE, cluster_cols = FALSE,
    display_numbers = TRUE, filename = "Figures/Fig2_B.pdf",
    height = 5, width = 10)
```



EGF	FGF2	GCSF	VEGF	GMSCF	TGFa	Eotaxin	Fractalkine	GRO	MCP1	MCP3	MDC	MIP1a	MIP1b	IP10	IL8	IL1b	IL2	IL4	IL5	IL6	IL7	IL10	IL13	INFa	INFg	TNFa	sCD40L	
1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	28
1.00	1.00	1.00	1.00	0.93	1.00	1.00	1.00	0.91	0.88	1.00	1.00	1.00	1.00	0.86	0.98	1.00	0.82	1.00	0.97	0.91	0.88	1.00	0.86	1.00	1.00	1.00	1.00	27
1.00	1.00	0.95	1.00	0.84	1.00	0.98	1.00	0.86	0.72	1.00	1.00	1.00	1.00	0.76	0.97	1.00	0.79	0.96	0.93	0.78	0.88	0.94	0.64	1.00	1.00	1.00	1.00	26
1.00	1.00	0.96	1.00	0.60	1.00	0.98	1.00	0.86	0.57	1.00	1.00	1.00	1.00	0.71	0.96	1.00	0.60	0.96	0.89	0.55	0.79	0.94	0.64	0.99	1.00	1.00	1.00	25
1.00	1.00	0.88	1.00	0.65	1.00	0.92	1.00	0.64	0.57	1.00	1.00	1.00	0.99	0.55	0.93	0.99	0.56	0.92	0.83	0.50	0.78	0.90	0.40	1.00	0.99	1.00	1.00	24
1.00	1.00	0.94	1.00	0.62	1.00	0.88	1.00	0.59	0.33	1.00	1.00	1.00	0.97	0.46	0.89	0.99	0.46	0.94	0.82	0.36	0.62	0.76	0.37	1.00	1.00	1.00	1.00	23
0.99	1.00	0.84	1.00	0.52	1.00	0.81	1.00	0.54	0.23	1.00	1.00	1.00	1.00	0.46	0.83	0.99	0.25	0.82	0.81	0.30	0.53	0.86	0.23	1.00	0.99	1.00	1.00	22
0.98	1.00	0.81	1.00	0.35	1.00	0.84	1.00	0.49	0.22	1.00	1.00	1.00	0.96	0.33	0.87	1.00	0.22	0.83	0.65	0.16	0.42	0.72	0.17	1.00	0.99	0.99	1.00	21
1.00	1.00	0.69	1.00	0.32	1.00	0.82	1.00	0.40	0.15	1.00	1.00	1.00	0.92	0.21	0.74	1.00	0.27	0.72	0.64	0.22	0.27	0.60	0.10	0.97	0.99	0.97	1.00	20
0.98	1.00	0.75	1.00	0.26	1.00	0.69	1.00	0.27	0.08	1.00	1.00	1.00	0.91	0.16	0.61	0.97	0.19	0.74	0.51	0.12	0.27	0.53	0.05	0.96	0.99	0.96	1.00	19
1.00	1.00	0.58	1.00	0.17	1.00	0.63	0.99	0.25	0.07	1.00	1.00	1.00	0.89	0.16	0.58	0.97	0.13	0.59	0.35	0.07	0.25	0.49	0.04	0.93	0.96	0.91	0.99	18
0.96	1.00	0.55	1.00	0.11	1.00	0.51	0.99	0.23	0.03	1.00	1.00	1.00	0.73	0.13	0.47	0.94	0.04	0.57	0.28	0.05	0.16	0.43	0.01	0.94	0.96	0.93	0.98	17
0.87	1.00	0.32	1.00	0.06	1.00	0.46	0.98	0.23	0.01	1.00	1.00	1.00	0.79	0.06	0.42	0.89	0.03	0.45	0.28	0.01	0.15	0.35	0.01	0.92	0.89	0.84	0.98	16
0.92	1.00	0.37	1.00	0.03	1.00	0.39	1.00	0.10	0.01	0.99	1.00	1.00	0.75	0.09	0.26	0.86	0.03	0.26	0.13	0.03	0.12	0.23	0.01	0.79	0.81	0.83	0.99	15
0.91	1.00	0.30	1.00	0.01	1.00	0.27	0.98	0.06	0.01	0.99	1.00	1.00	0.55	0.02	0.19	0.82	0.02	0.23	0.12	0.02	0.06	0.26	0.00	0.69	0.78	0.74	0.97	14
0.79	1.00	0.15	0.99	0.02	1.00	0.21	0.95	0.05	0.00	1.00	1.00	1.00	0.46	0.03	0.12	0.78	0.00	0.15	0.09	0.00	0.04	0.16	0.00	0.60	0.76	0.67	0.98	13
0.70	1.00	0.11	0.99	0.00	1.00	0.18	0.93	0.04	0.03	0.99	1.00	0.99	0.39	0.01	0.08	0.59	0.00	0.13	0.03	0.00	0.03	0.10	0.00	0.60	0.58	0.56	0.94	12
0.54	1.00	0.09	0.98	0.01	1.00	0.13	0.83	0.00	0.00	0.98	0.99	1.00	0.31	0.00	0.04	0.55	0.00	0.06	0.00	0.00	0.03	0.12	0.00	0.48	0.38	0.55	0.93	11
0.53	1.00	0.03	0.95	0.00	1.00	0.08	0.72	0.02	0.00	0.95	1.00	0.96	0.15	0.00	0.00	0.37	0.00	0.01	0.00	0.00	0.03	0.10	0.00	0.35	0.38	0.48	0.89	10
0.34	1.00	0.00	0.92	0.00	1.00	0.05	0.58	0.03	0.00	0.95	0.98	0.92	0.04	0.00	0.04	0.38	0.00	0.05	0.00	0.00	0.00	0.03	0.00	0.20	0.28	0.36	0.85	9
0.16	1.00	0.00	0.88	0.00	1.00	0.04	0.57	0.00	0.00	0.85	0.95	0.93	0.03	0.01	0.01	0.21	0.00	0.02	0.00	0.00	0.00	0.04	0.00	0.06	0.13	0.33	0.78	8
0.12	1.00	0.00	0.73	0.00	1.00	0.03	0.30	0.01	0.00	0.76	0.94	0.82	0.01	0.00	0.00	0.13	0.00	0.01	0.00	0.00	0.00	0.02	0.00	0.07	0.08	0.24	0.73	7
0.09	1.00	0.00	0.72	0.00	1.00	0.00	0.16	0.00	0.00	0.53	0.91	0.74	0.00	0.00	0.00	0.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.04	0.16	0.55	6
0.02	1.00	0.00	0.44	0.00	1.00	0.01	0.11	0.00	0.00	0.46	0.81	0.50	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	0.02	0.08	0.50	5
0.02	0.98	0.00	0.32	0.00	1.00	0.00	0.03	0.00	0.00	0.26	0.70	0.19	0.00	0.00	0.01	0.03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.06	0.38	4
0.00	0.89	0.00	0.10	0.00	1.00	0.00	0.02	0.00	0.00	0.06	0.46	0.12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	0.29	3
0.00	0.67	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.17	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.14	2
0.00	0.03	0.00	0.00	0.00	0.90	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.06	1

Stratified analyses

As for univariate analyses, we are going to apply PLS models on subsets of the data with all controls and cases of one particular subtype only. We will focus on CLL, DLBCL and MM as the other subtypes have very low sample sizes.

Sparse PLS-DA models

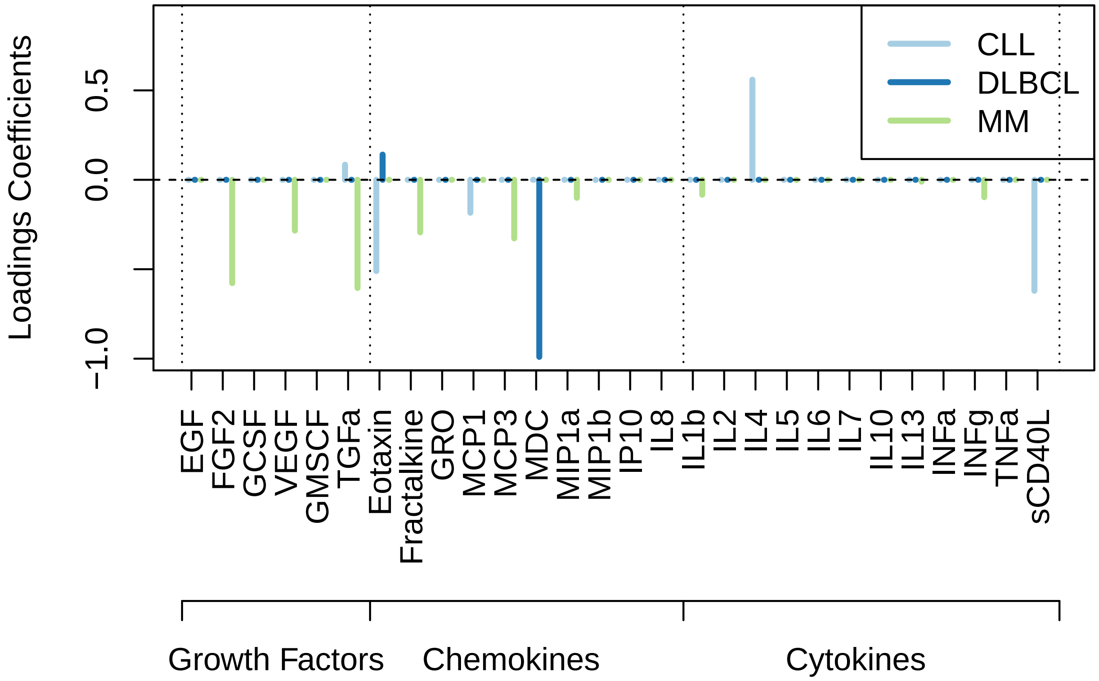
Challenge 7: Perform sPLS-DA on the data stratified by CLL, DLBCL and MM with theparameters provided in MyParameters.rds and reproduce Figure 3-a.

Note that, for example, the parameters of the sPLS-DA models for subtype MM can be accessed by the following code, from which we can see that the number of variables selected is 9.

```
MyParameters$MM$sPLSDA$NVar
```

```
## [1] 9
```

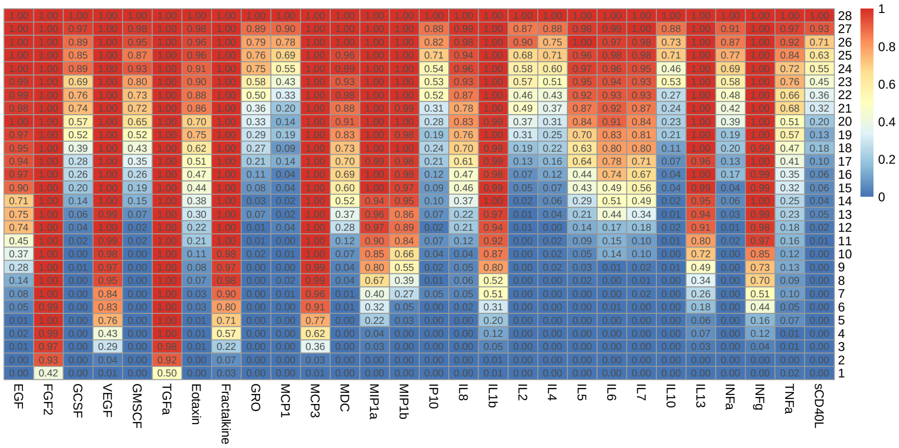
```
for (subtype in c("BCLL", "DLBL", "FL", "MM")) {
    X = eval(parse(text = paste0("X_", subtype)))
    Y = eval(parse(text = paste0("Y_", subtype)))

    MysPLSDA <- splsda(X, Y, keepX = eval(parse(text = paste0("MyParameters$",
        subtype, "$sPLSDA$NVar"))), ncomp = 1)
    assign(paste0("MysPLSDA_", subtype), MysPLSDA)

    MysgPLSDA <- sgPLSda(X, Y, ncomp = 1, ind.block.x = Xgroups,
        keepX = eval(parse(text = paste0("MyParameters$",
            subtype, "$sgPLSDA$NGroups"))), alpha.x = eval(parse(text = paste0("MyParameters$",
            subtype, "$sgPLSDA$alpha"))))
    assign(paste0("MysgPLSDA_", subtype), MysgPLSDA)
}

Loadings = cbind(MysPLSDA_BCLL$loadings$X, MysPLSDA_DLBL$loadings$X,
    MysPLSDA_MM$loadings$X, rep(NA, 28), rep(NA, 28))
Loadings = as.vector(t(Loadings))

par(mar = c(10, 5, 3, 3))
plot(Loadings, col = c(MyPal[1], MyPal[2], MyPal[3],
    NA, NA), xaxt = "n", ylab = "Loadings Coefficients",
    type = "h", lwd = 3, xlab = "", ylim = c(min(Loadings[!is.na(Loadings)]),
        0.9))
axis(1, at = seq(1.5, 28 * 5, by = 5), labels = colnames(X_pooled),
    las = 2)
axis(1, at = c(0, Xgroups, 28) * 5, line = 6, labels = NA)
axis(1, at = c(3, 10.5, 21.5) * 5, labels = c("Growth Factors",
    "Chemokines", "Cytokines"), line = 6, tick = FALSE)
abline(v = c(0, Xgroups, 28) * 5, lty = 3, col = "black")
abline(h = 0, lty = 2)
legend("topright", lty = 1, lwd = 3, col = c(MyPal[1],
    MyPal[2], MyPal[3]), legend = c("CLL", "DLBCL",
    "MM"))
```



Challenge 8: Perform stability analyses on the sPLS-DA models by subtype. Compare your results to Supplementary Figure 2.

```
for (subtype in c("BCLL", "DLBL", "FL", "MM")) {
    print(subtype)

    MyStab = NULL

    X = eval(parse(text = paste0("X_", subtype)))
    Y = eval(parse(text = paste0("Y_", subtype)))

    Stability_results = Stability_plot(X = X, Y = Y,
        NIter = 100)

    pheatmap(Stability_results, cluster_rows = FALSE,
        cluster_cols = FALSE, display_numbers = TRUE,
        filename = paste0("Figures/Supp_fig2_", subtype,
            ".pdf"), height = 5, width = 10)
}
```

For example, the stability plot of MM is shown below:



Sparse-group PLS-DA models

Challenge 9: Perform sgPLS-DA on the data stratified by CLL, DLBCL and MM with the with the parameters provided in "MyParameters.rds" and reproduce Figure 3-b.

As for sPLS-DA models, the parameters are stored in `MyParameters`. For example for MM the two parameters can be accessed using the following commands:

```
MyParameters$MM$sgPLSDA$NGroups
```

```
## [1] 1
```

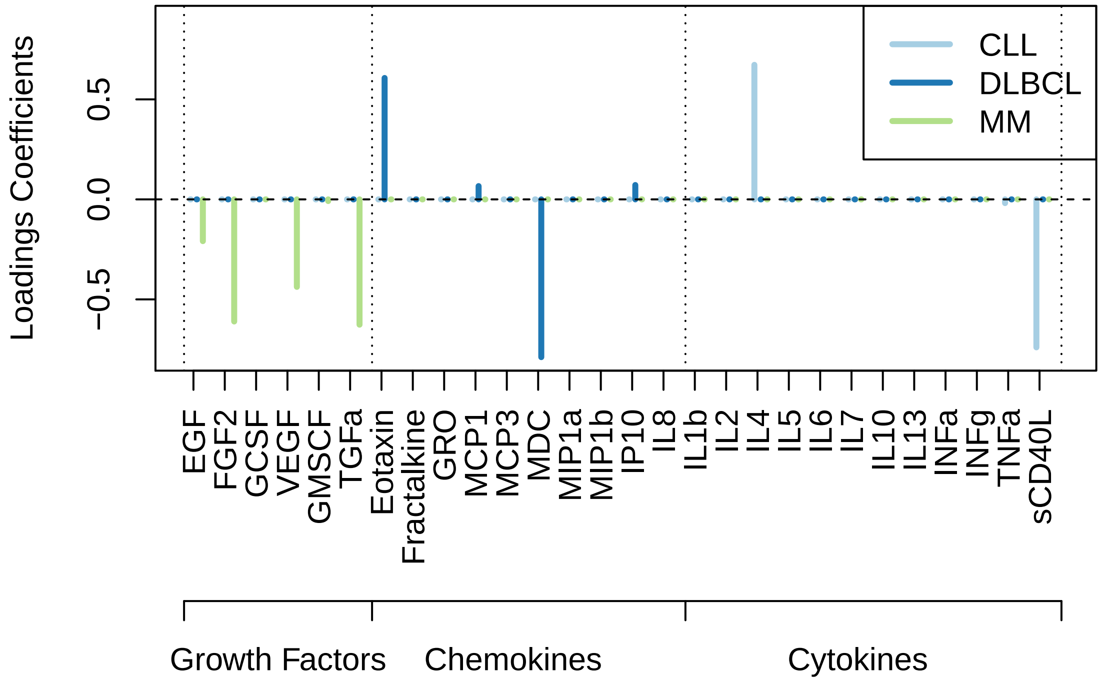
```
MyParameters$MM$sgPLSDA$alpha
```

```
## [1] 0.6
```

The sgPLS-DA models were run in the code above (see answer to Challenge 7 in section "Sparse PLS-DA models"). The code below can be used to reproduce the figure:

```
Loadings = cbind(MysgPLSDA_BCLL$loadings$X, MysgPLSDA_DLBL$loadings$X,
    MysgPLSDA_MM$loadings$X, rep(NA, 28), rep(NA, 28))
Loadings = as.vector(t(Loadings))

par(mar = c(10, 5, 3, 3))
plot(Loadings, col = c(MyPal[1], MyPal[2], MyPal[3],
    NA, NA), xaxt = "n", ylab = "Loadings Coefficients",
    type = "h", lwd = 3, xlab = "", ylim = c(min(Loadings[!is.na(Loadings)]),
        0.9))
axis(1, at = seq(1.5, 28 * 5, by = 5), labels = colnames(X_pooled),
    las = 2)
```

```r
axis(1, at = c(0, Xgroups, 28) * 5, line = 6, labels = NA)
axis(1, at = c(3, 10.5, 21.5) * 5, labels = c("Growth Factors",
    "Chemokines", "Cytokines"), line = 6, tick = FALSE)
abline(v = c(0, Xgroups, 28) * 5, lty = 3, col = "black")
abline(h = 0, lty = 2)
legend("topright", lty = 1, lwd = 3, col = c(MyPal[1],
    MyPal[2], MyPal[3]), legend = c("CLL", "DLBCL",
    "MM"))
```



Analyses of time to diagnosis

In this section, we are going to restrict ourselves to the sample of cases, to analyse the time to diagnosis of disease. The protein data for cases (`X_diagnostic`) and the vector of times to diagnosis in days (`Y_diagnostic`) have been prepared above:

```r
dim(X_diagnostic)
```

```
## [1] 268  28
```

```r
length(Y_diagnostic)
```

```
## [1] 268
```

```r
summary(Y_diagnostic)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     733    1378    2192    2267    2964    5828
```

The restricted subsets of CLL, DLBCL and MM containing cases are contained in:

- X_diag_BCLL and Y_diag_BCLL
- X_diag_DLBL and Y_diag_DLBL
- X_diag_MM and Y_diag_MM

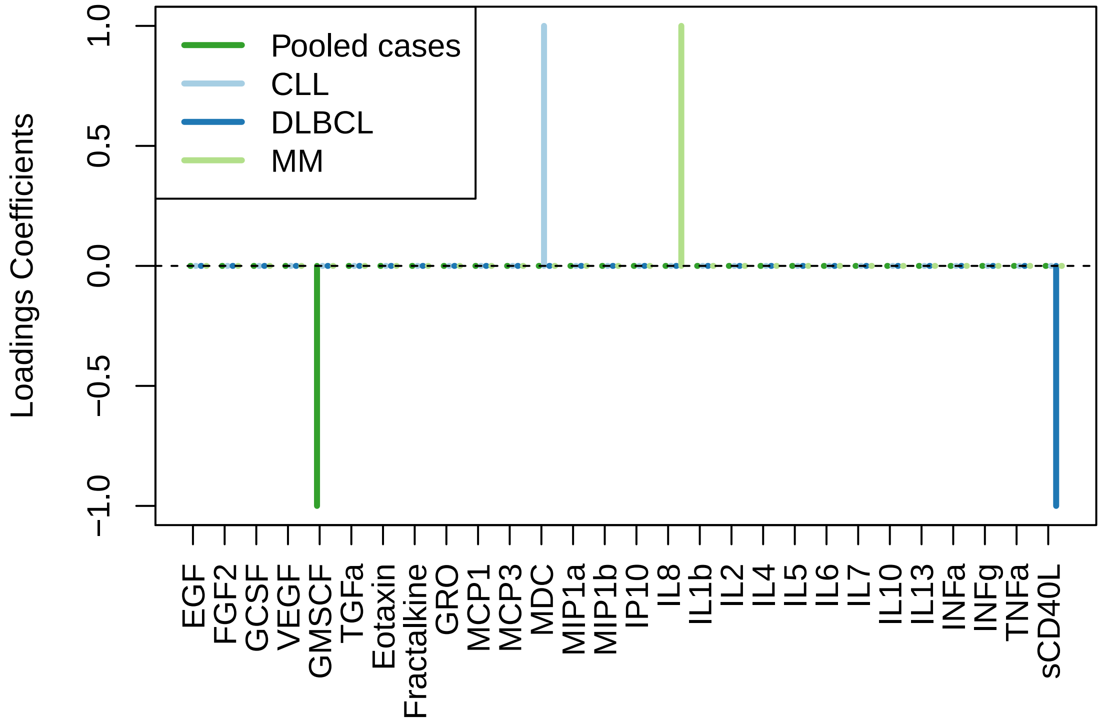
Challenge 10: Run sPLS analyses on the full sample and on the three subsets using the parameters provided.

```
MyParameters$Time$sPLS$NVar
```

```
## [1] 1
```

```
MysPLS_diag = spls(X = X_diagnostic, Y = Y_diagnostic,
    ncomp = 1, keepX = MyParameters$Time$sPLS$NVar)
MysPLS_diag_BCLL = spls(X = X_diag_BCLL, Y = Y_diag_BCLL,
    ncomp = 1, keepX = 1)
MysPLS_diag_DLBL = spls(X = X_diag_DLBL, Y = Y_diag_DLBL,
    ncomp = 1, keepX = 1)
MysPLS_diag_MM = spls(X = X_diag_MM, Y = Y_diag_MM,
    ncomp = 1, keepX = 1)
```

```
Loadings = cbind(MysPLS_diag$loadings$X, MysPLS_diag_BCLL$loadings$X,
    MysPLS_diag_DLBL$loadings$X, MysPLS_diag_MM$loadings$X,
    rep(NA, 28), rep(NA, 28))
Loadings = as.vector(t(Loadings))
Loadings = Loadings[-c(length(Loadings) - 1, length(Loadings))]

par(mar = c(6, 5, 3, 3))
plot(Loadings, col = c(MyPal[4], MyPal[1], MyPal[2],
    MyPal[3], NA, NA), xaxt = "n", ylab = "Loadings Coefficients",
    type = "h", lwd = 3, xlab = "")
axis(1, at = seq(1.5, 28 * 6, by = 6), labels = rownames(MysPLS_diag$loadings$X),
    las = 2)
legend("topleft", lty = 1, lwd = 3, col = c(MyPal[4],
    MyPal[1], MyPal[2], MyPal[3]), legend = c("Pooled cases",
    "CLL", "DLBCL", "MM"))
abline(h = 0, lty = 2)
```



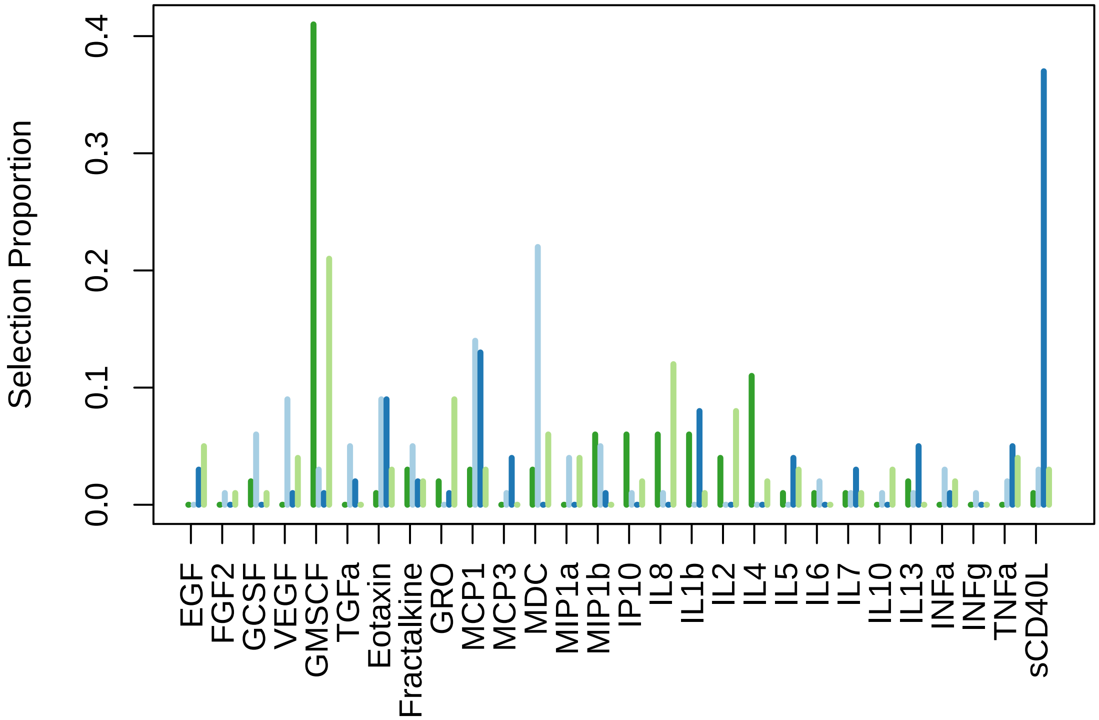
Challenge 11: Compute the selection proportion of each variable for each subtype by setting the number of selected variables to 1.

```
NVar = c(1, 1, 1, 1)
X_diag_pooled = X_diagnostic
Y_diag_pooled = Y_diagnostic

NIter = 100

for (i in 1:4) {
    print(i)
    subtype = c("pooled", "DLBL", "BCLL", "MM")[i]
    X = eval(parse(text = paste0("X_diag_", subtype)))
    Y = eval(parse(text = paste0("Y_diag_", subtype)))

    TmpStab = NULL

    for (k in 1:NIter) {
        s = sample(seq(1, nrow(X)), size = nrow(X),
            replace = TRUE)   # BOOTSTRAP
        X_boot = X[s, ]
        rownames(X_boot) = seq(1:nrow(X))
        Y_boot = Y[s]

        TmpsPLS = spls(X_boot, Y_boot, keepX = NVar[i],
            ncomp = 1)
        TmpStab = rbind(TmpStab, TmpsPLS$loadings$X[,
            1])
    }

    Stab = apply(TmpStab, 2, FUN = function(x) {
        sum(x != 0)
    })/NIter
    assign(paste0("Stab_", subtype), Stab)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
```

```
Stab = cbind(Stab_pooled, Stab_BCLL, Stab_DLBL, Stab_MM,
    rep(NA, 28), rep(NA, 28))
Stab = as.vector(t(Stab))
```

```
par(mar = c(6, 5, 3, 3))
plot(Stab, col = c(MyPal[4], MyPal[1], MyPal[2], MyPal[3],
    NA, NA), xaxt = "n", ylab = "Selection Proportion",
    type = "h", lwd = 3, xlab = "")
axis(1, at = seq(1.5, 28 * 6, by = 6), labels = rownames(MysPLS_diag$loadings$X),
    las = 2)
```



Please refer to the paper for the biological interpretation of the results.