# NMR-based metabolomic analysis of the dataset MTBLS242: serum samples

Institute for Bioengineering of Catalonia

May 25, 2020

This is an example of using AlpsNMR package on the MTBLS242 dataset structured as a pipeline so that inputs are needed and outputs are obtained in the selected folders. Edit “inputs” to match your parameters and just run the “code to run” for the pipeline execution. However, you can see the vignettes and use the functions as you wish. You can download the MTBLS242 dataset from MetaboLights database: https://www.ebi.ac.uk/metabolights/MTBLS242

```r
library(AlpsNMR)
```

```
##
## Attaching package: 'AlpsNMR'

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

## Pipeline preparation

To work as in a pipeline manner we need to set an output directory. We can set the number of cores of your computer for parallelization.

```r
# Set a folder to keep results
output_dir <- "C:/Users/hgracia/Desktop/IBEC/results"

# How many cores to use for parallelization
num_workers <- 12
```

## Node 1: Load samples

Loads samples from a specified directory into a `nmr_dataset` object. Then we can save the loaded data into the output directory.

### Input parameters

```r
# Path of NMR samples downloaded from https://www.ebi.ac.uk/metabolights/MTBLS242:
dataset_path_nmr <- "C:/Users/hgracia/Desktop/IBEC/data/MTBLS242"

# Files/directories ending in "s" corresponding to the spectra in the dataset:
filename_glob <- "*s"
```

### Code to run

```r
NMRExperiments <- as.character(fs::dir_ls(dataset_path_nmr, glob = filename_glob))
plan(multiprocess, workers = num_workers)
nmr_dataset <- nmr_read_samples(NMRExperiments)
plan(sequential)
```

### Save results to disk

```r
output_dir_node1 <- file.path(output_dir, "01-load-samples")
fs::dir_create(output_dir_node1)
nmr_dataset_rds <- fs::path(output_dir_node1, "nmr_dataset.rds")
nmr_dataset_save(nmr_dataset, nmr_dataset_rds)
```

```
## An nmr_dataset (391 samples)
```

```r
nmr_meta_export(nmr_dataset, fs::path(output_dir_node1, "nmr_dataset_metadata.xlsx"))
message(nmr_dataset$num_samples, " samples loaded.")
```

```
## 391 samples loaded.
```

## Node 2: Append metadata

We now merge the metadata. To do that, you need an Excel file containing a first column called "NMRExperiments" with the name of the imported spectra (it does not have to be the name of the individuals).

### Input parameters

```r
# Path where metada is contained
excel_file <- "C:/Users/hgracia/Desktop/IBEC/data/MTBLS242/nmr_dataset_metadata_tidy.xlsx"
```

### Code to run

```r
nmr_dataset <- nmr_meta_add_tidy_excel(nmr_dataset, excel_file)
```

### Save results

```r
output_dir_node2 <- file.path(output_dir, "02-add-metadata")
fs::dir_create(output_dir_node2)

metadata_added_xlsx <- file.path(output_dir_node2, "nmr_metadata_added.xlsx")
nmr_dataset_rds <- fs::path(output_dir_node2, "nmr_dataset.rds")

nmr_meta_export(nmr_dataset, xlsx_file = metadata_added_xlsx, groups = "external")
nmr_dataset_save(nmr_dataset, nmr_dataset_rds)
```

```
## An nmr_dataset (391 samples)
```

## Node 3: Interpolation

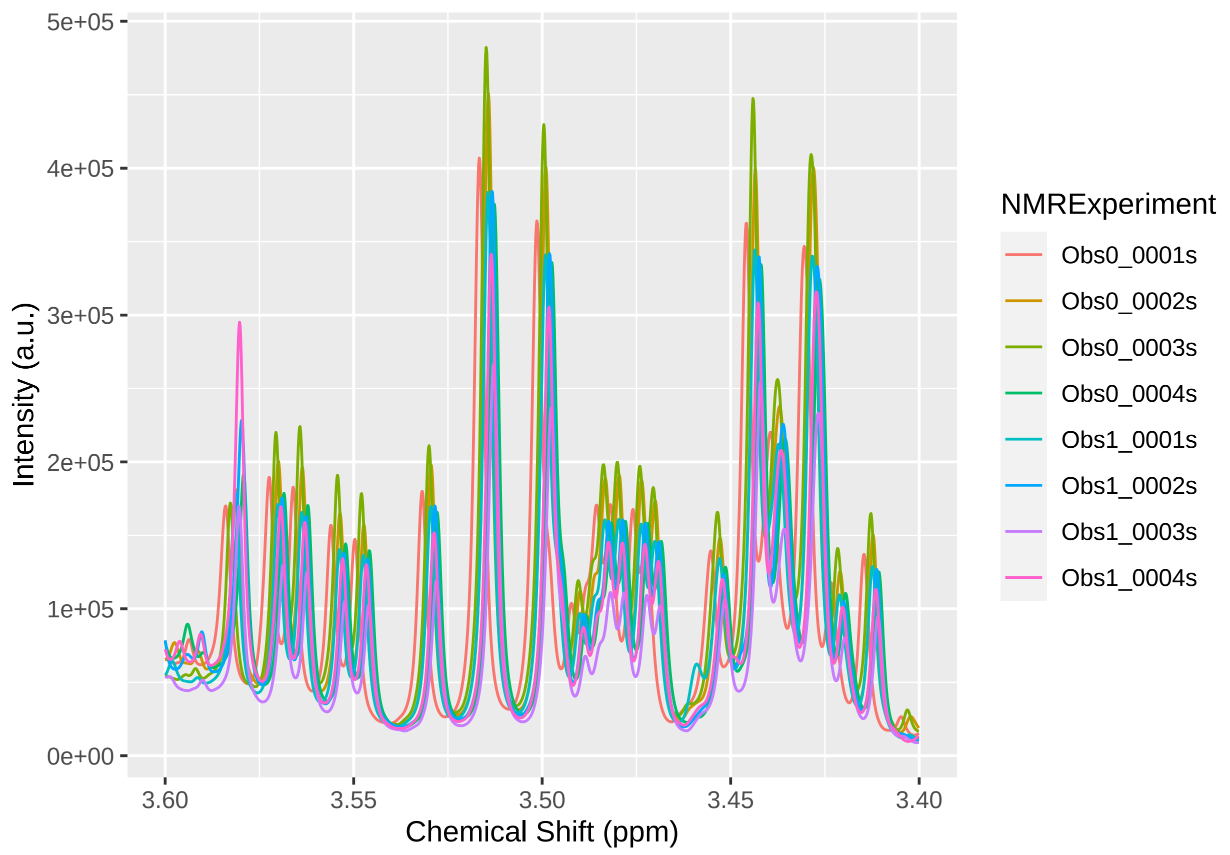
Interpolation is used to unify the ppm axis from all spectra. However, you also can set a range for next steps avoiding noise regions from here. Note that ppm resolution is automatically calculated with the function `nmr_ppm_resolution` in "Code to run".

### Input parameters

```r
ppm_range_start <- 0.7
ppm_range_end <- 9.5
```

### Code to run

```r
ppm_resolution <- unlist(nmr_ppm_resolution(nmr_dataset[1]))
axis <- c(min = ppm_range_start, max = ppm_range_end, by = ppm_resolution)
nmr_dataset <- nmr_interpolate_1D(nmr_dataset, axis = axis)

plot(nmr_dataset,
  NMRExperiment = c(
  "Obs0_0001s",
  "Obs0_0002s",
  "Obs0_0003s",
  "Obs0_0004s",
  "Obs1_0001s",
  "Obs1_0002s",
  "Obs1_0003s",
  "Obs1_0004s"),
  chemshift_range = c(3.40, 3.60))
```



### Save results

```
output_dir_node3 <- file.path(output_dir, "03-interpolate-1D")
fs::dir_create(output_dir_node3)

raw_data_matrix_fn <- file.path(output_dir_node3, "raw_data.csv")
metadata_fn <- file.path(output_dir_node3, "metadata.xlsx")
nmr_dataset_outfile <- file.path(output_dir_node3, "nmr_dataset.rds")
plot_html <- file.path(output_dir_node3, "plot-samples.html")

nmr_export_data_1r(nmr_dataset, raw_data_matrix_fn)
```

```
## An nmr_dataset_1D (391 samples)
```

```
nmr_meta_export(nmr_dataset, metadata_fn, groups = "external")
nmr_dataset_save(nmr_dataset, nmr_dataset_outfile)
```

```
## An nmr_dataset_1D (391 samples)
```

## Node 4: Region Exclusion

Here it is important to know what type of signals can mask the results due to their intensity or what type of solvent has been used in sample processing since this can create artifacts in the spectra and should be removed. In this case, the biological samples correspond to serum, which contains a lot of water and its signal should be removed from further steps. To do this, we define a vector containing the range (min ppm value, max ppm value) of the water signal, but other signals can be eliminated, for example: exclude_regions <- list(water = c(4.5, 5.1), methanol = c(3.33, 3.34))

### Input parameters

```r
exclude_regions <-  list(water = c(4.5, 5.1))
```

### Code to run

```r
nmr_dataset <- nmr_exclude_region(nmr_dataset, exclude = exclude_regions)
```

### Save Results

```r
output_dir_node4 <- file.path(output_dir, "04-exclude-regions")

fs::dir_create(output_dir_node4)

raw_data_matrix_fn <- file.path(output_dir_node4, "raw_data.csv")
metadata_fn <- file.path(output_dir_node4, "metadata.xlsx")
nmr_dataset_outfile <- file.path(output_dir_node4, "nmr_dataset.rds")
plot_html <- file.path(output_dir_node4, "plot-samples.html")

nmr_export_data_1r(nmr_dataset, raw_data_matrix_fn)
```

```
## An nmr_dataset_1D (391 samples)
```

```r
nmr_meta_export(nmr_dataset, metadata_fn, groups = "external")
nmr_dataset_save(nmr_dataset, nmr_dataset_outfile)
```

```
## An nmr_dataset_1D (391 samples)
```

## Node 5: Initial Outlier Rejection

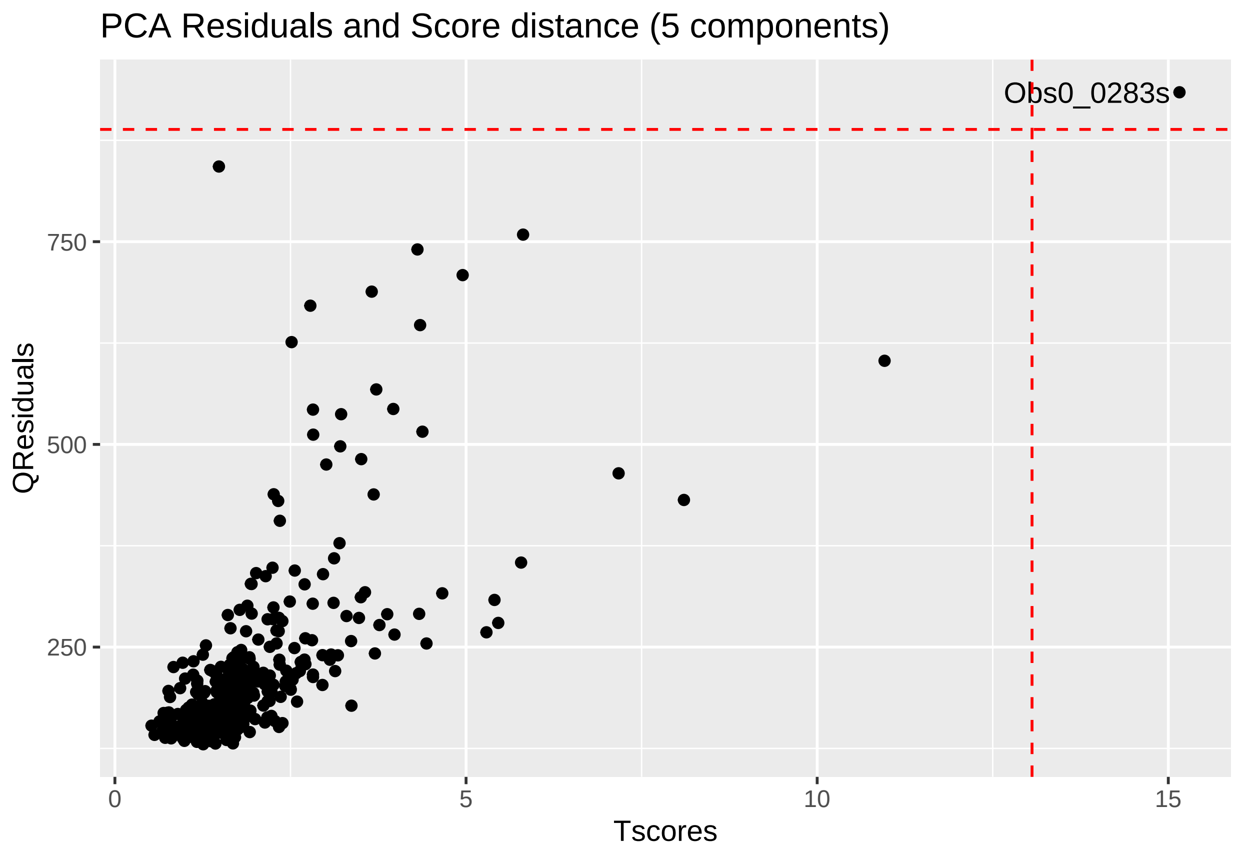
The robust principal component analysis (rPCA) for outlier detection gives an idea of potential outliers. A proposed threshold, based on quantiles, for Q residual and T2 score values, results less sensitive to extreme intensities. Then you choose if any sample should be excluded. The plot below indicated that a sample "Obs0_0283s" is extremely different than the other samples. The function is prepared to annotated samples that are in the top-right corner, exhibiting high differences.

### Input parameters

```r
# Nothing
```

### Code to run

```r
pca_outliers <- nmr_pca_outliers_robust(nmr_dataset)
nmr_pca_outliers_plot(nmr_dataset, pca_outliers)
```



Then, if we decide to discard this sample, we just run the function below. Otherwise, just ignore this:

```r
nmr_dataset_with_outliers <- nmr_dataset
nmr_dataset <- nmr_pca_outliers_filter(nmr_dataset, pca_outliers)
```

### Save results

```r
output_dir_node5 <- file.path(output_dir, "05-outlier-detection")
fs::dir_create(output_dir_node5)

full_spectra_matrix_fn <- file.path(output_dir_node5, "full_spectra_matrix.csv")
metadata_fn <- file.path(output_dir_node5, "metadata.xlsx")
nmr_dataset_outfile <- file.path(output_dir_node5, "nmr_dataset.rds")
```

```
plot_outlier_QT2 <- file.path(output_dir_node5, "plot-Qresiduals_vs_Tscores.png")
plot_outlier_html <- file.path(output_dir_node5, "plot-outlier-samples.html")

nmr_export_data_1r(nmr_dataset, full_spectra_matrix_fn)
```

```
## An nmr_dataset_1D (390 samples)
```

## Node 6: Filter samples

### Input parameters

The filter node takes care of keeping only some samples. In this case, we want to compare two time points of the MTBLS242 dataset to compare them: "preop" and "3 months after surgery". However, you can filter to keep other conditions kept in the metadata. Some examples: - `Cohort == "A"`: Keeps the A cohort - `TimePoint %in% c("preop", "3 months after surgery")`: Keeps timepoints "preop" and "3 months after surgery" - `Gender == "Female"`: Keeps Female samples - others

```
samples_to_keep_conditions <- 'Timepoint %in% c("preop", "3 months after surgery")'
```

### Code to run

```
conditions_expr <- rlang::parse_exprs(samples_to_keep_conditions)
nmr_dataset <- AlpsNMR::filter(nmr_dataset, !!!conditions_expr)
```

### Save results

```
output_dir_node6 <- file.path(output_dir, "06-filter-samples")
fs::dir_create(output_dir_node6)

raw_data_matrix_fn <- file.path(output_dir_node6, "raw_data.csv")
metadata_fn <- file.path(output_dir_node6, "metadata.xlsx")
nmr_dataset_outfile <- file.path(output_dir_node6, "nmr_dataset.rds")


nmr_export_data_1r(nmr_dataset, raw_data_matrix_fn)
```

```
## An nmr_dataset_1D (201 samples)
```

```
nmr_meta_export(nmr_dataset, metadata_fn, groups = "external")
nmr_dataset_save(nmr_dataset, nmr_dataset_outfile)
```

```
## An nmr_dataset_1D (201 samples)
```

```
pasted_conditions <- glue::glue_collapse(samples_to_keep_conditions, sep = ", ",
                                         width = 80, last = ", ")
message("Dataset filtered by: ", pasted_conditions)
```

```
## Dataset filtered by: Timepoint %in% c("preop", "3 months after surgery")
```

## Node 7: Peak detection and Alignment

Peak detection is based on a combination of an automated baseline threshold, signal to noise ratio and maximum tolerance. Alignment is based on hierarchical cluster-based peak alignment (CluPA) (Vu et al., 2011).

### Input parameters

```
# Leave those as default/recommended for serum.
# Size of peak detection segments
nDivRange_ppm <- 0.1

# Baseline threshold
baselineThresh <- NULL

# Signal to noise ratio
SNR.Th <- 3

# Maximum alignment shift
maxShift_ppm <- 0.0015
```

### Code to run

```
scales <- seq(1, 16, 2)
acceptLostPeak <- FALSE

plan(multiprocess, workers = num_workers)
message("Detecting peaks...")
```

```
## Detecting peaks...
```

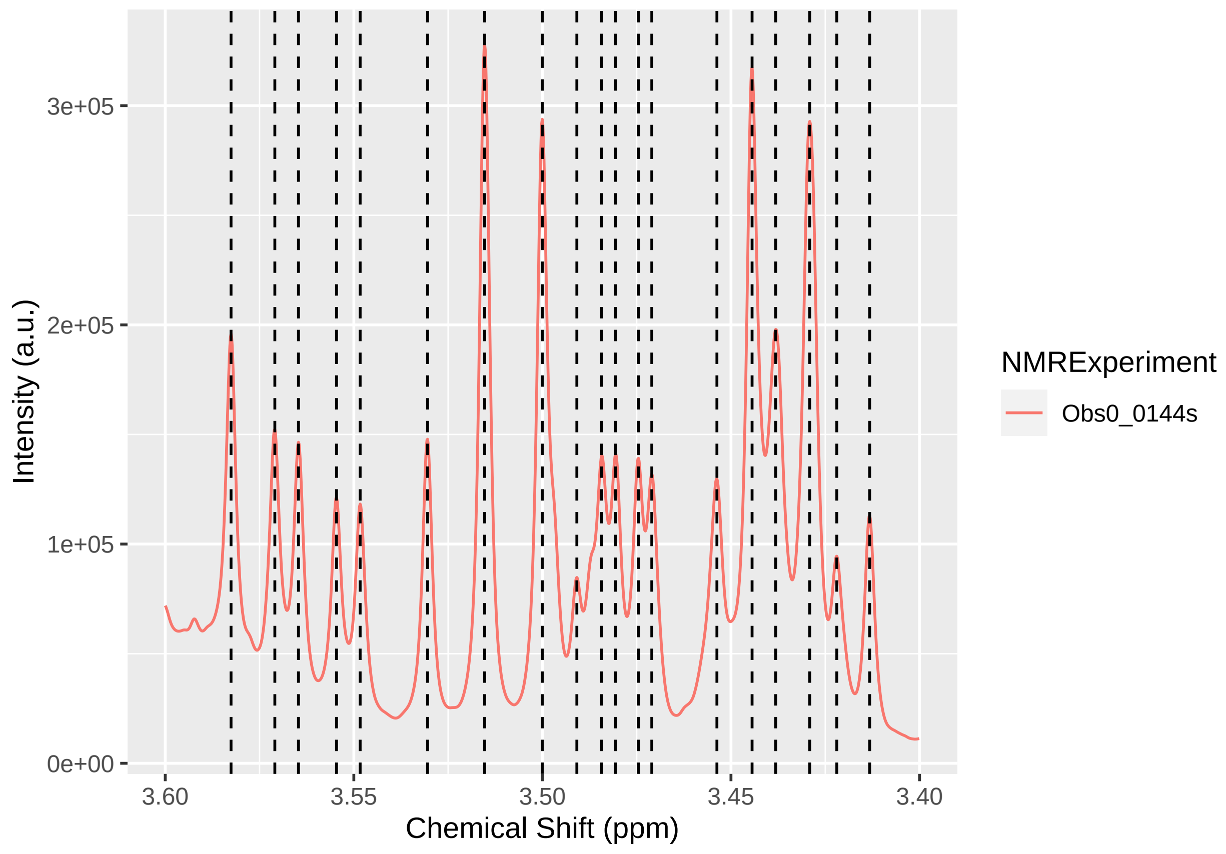
```
peak_data <- nmr_detect_peaks(nmr_dataset,
                              nDivRange_ppm = nDivRange_ppm,
                              scales = scales,
                              baselineThresh = baselineThresh,
                              SNR.Th = SNR.Th)

message("Choosing alignment reference...")
```

```
## Choosing alignment reference...
```

```r
NMRExp_ref <- nmr_align_find_ref(nmr_dataset, peak_data)

message("Starting alignment...")
```

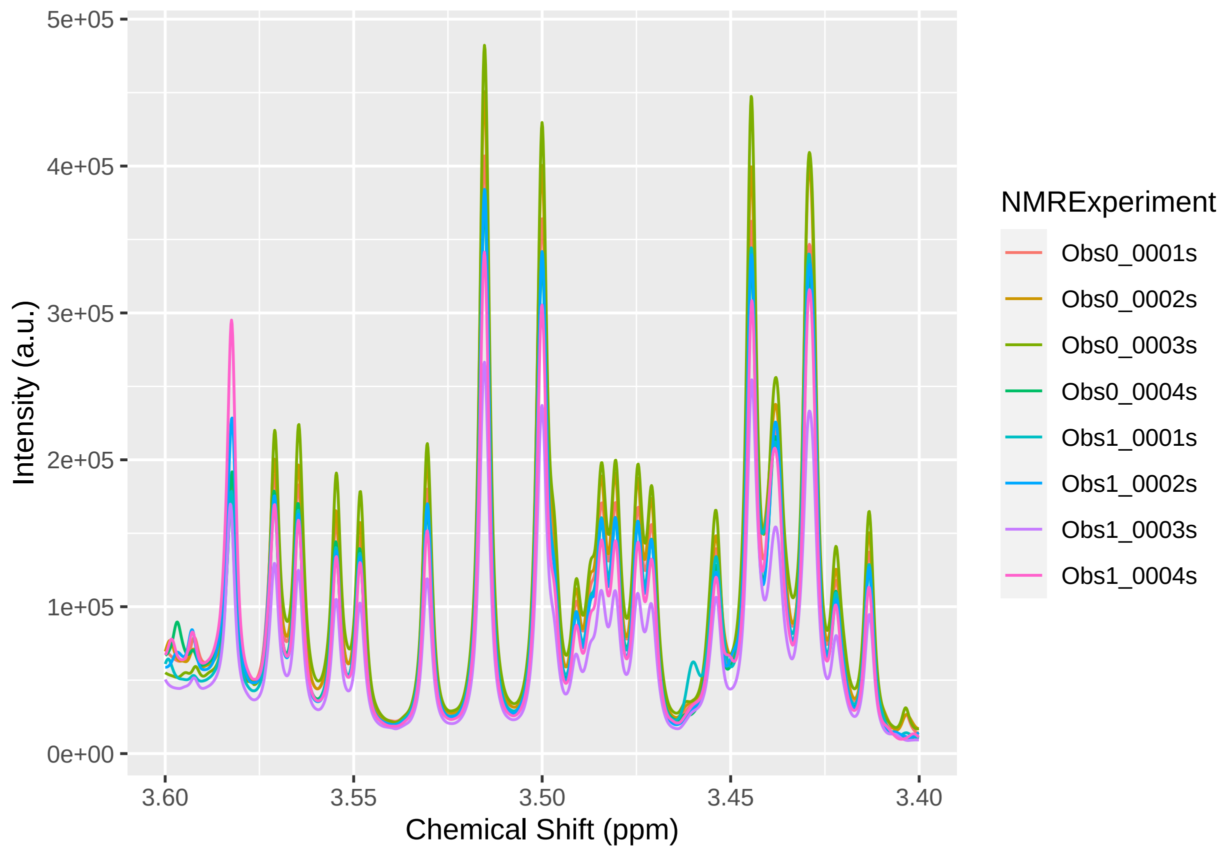
```
## Starting alignment...
```

```r
nmr_dataset <- nmr_align(nmr_dataset, peak_data,
                         NMRExp_ref = NMRExp_ref,
                         maxShift_ppm = maxShift_ppm,
                         acceptLostPeak = acceptLostPeak)

gplt <- nmr_detect_peaks_plot(nmr_dataset, peak_data, NMRExperiment = NMRExp_ref)
plan(sequential)

nmr_detect_peaks_plot(
  nmr_dataset,
  peak_data,
  NMRExperiment = NMRExp_ref,
  chemshift_range = c(3.40, 3.60)
  )
```



we can take a look into the detected peaks. The interactive plot allows you to zoom in in HTML files.

```
plot(nmr_dataset,
  NMRExperiment = c(
  "Obs0_0001s",
  "Obs0_0002s",
  "Obs0_0003s",
  "Obs0_0004s",
  "Obs1_0001s",
  "Obs1_0002s",
  "Obs1_0003s",
  "Obs1_0004s"),
  chemshift_range = c(3.40, 3.60))
```



### Save results

```
output_dir_node7 <- file.path(output_dir, "07-alignment")
fs::dir_create(output_dir_node7)

raw_data_matrix_fn <- file.path(output_dir_node7, "raw_data.csv")
metadata_fn <- file.path(output_dir_node7, "metadata.xlsx")
nmr_dataset_outfile <- file.path(output_dir_node7, "nmr_dataset.rds")

plot_peak_detection_html <- file.path(output_dir_node7, "peak-detection-diagnostic.html")
plot_html <- file.path(output_dir_node7, "plot-samples.html")
```

```r
peak_data_fn <- file.path(output_dir_node7, "peak_data.csv")
NMRExp_ref_fn <- file.path(output_dir_node7, "NMRExperiment_align_ref.txt")

nmr_export_data_1r(nmr_dataset, raw_data_matrix_fn)
nmr_meta_export(nmr_dataset, metadata_fn, groups = "external")
nmr_dataset_save(nmr_dataset, nmr_dataset_outfile)

plot_interactive(gplt, plot_peak_detection_html)
plot_webgl(nmr_dataset, html_filename = plot_html)
utils::write.csv(peak_data, peak_data_fn, row.names = FALSE)
write(NMRExp_ref, NMRExp_ref_fn)
```

## Node 8: Normalization

We can normalize the dataset. This is recommended for biosamples, controlling for dilution factors, irregular pipetting, etc. Probabilistic quotient normalization is one of the most used model-based techniques NMR-based metabolomics.

### Input parameters

```r
# nothing
```

### Code to run

```r
nmr_dataset <- nmr_normalize(nmr_dataset, method = "pqn")
norm_pqn_diagnostic <- nmr_normalize_extra_info(nmr_dataset)
gplt_norm_factor_pqn <- norm_pqn_diagnostic$plot
```

### Save results

```r
output_dir_node8 <- file.path(output_dir, "08-normalization")

fs::dir_create(output_dir_node8)

plot_norm_factor_ic <- file.path(output_dir_node8, "normalization_factor_ic.png")
plot_norm_factor_pqn <- file.path(output_dir_node8, "normalization_factor_pqn.png")

full_spectra_matrix_fn <- file.path(output_dir_node8, "full_spectra_matrix.csv")
metadata_fn <- file.path(output_dir_node8, "metadata.xlsx")
nmr_dataset_outfile <- file.path(output_dir_node8, "nmr_dataset.rds")
plot_html <- file.path(output_dir_node8, "plot-samples.html")

ggplot2::ggsave(filename = plot_norm_factor_pqn, plot = gplt_norm_factor_pqn,
                width = 14, height = 8, unit = "cm", dpi = 300)

nmr_export_data_1r(nmr_dataset, full_spectra_matrix_fn)
```

```
## An nmr_dataset_1D (201 samples)
```

```r
nmr_meta_export(nmr_dataset, metadata_fn, groups = "external")
nmr_dataset_save(nmr_dataset, nmr_dataset_outfile)
```

```
## An nmr_dataset_1D (201 samples)
```

## Node 9: Integration

For peak integration, calculation of peak width may be performed automatically (set peak_width_ppm = NULL), from the detected peaks in the reference spectrum (if you wish, you can combine detected peaks other than the reference spectrum, see help), or manually, in which users can select a specific peak width for integrating the detected peaks. This differs than the bucketing approach in which spectra are equally divided into buckets (for example of 0.01 ppm) and this normally leads to a higher number of total variables. this has the inconvenient that several peaks might be split into several parts, lowering the statistical power, and vice-versa, certain overlapping tails might result in false positives because of this noisy parts. However, a good match between them is expected.

### Input parameters

```r
peak_width_ppm <- NULL
```

### Code to run

```r
# be carefull, you integrate based on peaks from a unique ref sample
peak_data_integ <- dplyr::filter(peak_data, NMRExperiment == !!NMRExp_ref)

nmr_peak_table <- nmr_integrate_peak_positions(
  samples = nmr_dataset,
  peak_pos_ppm = peak_data_integ$ppm,
  peak_width_ppm = peak_width_ppm)
```

```
## calculated width for integration is 0.00427905515639537 ppm
```

```r
nmr_peak_table_completed <- get_integration_with_metadata(nmr_peak_table)
```

### Save Results

```r
output_dir_node9 <- file.path(output_dir, "09-peak-integration")
fs::dir_create(output_dir_node9)

peak_table_fn <- file.path(output_dir_node9, "peak_table.csv")
metadata_fn <- file.path(output_dir_node9, "metadata.xlsx")
nmr_peak_table_rds <- file.path(output_dir_node9, "nmr_peak_table.rds")
```

```
utils::write.csv(nmr_peak_table_completed, peak_table_fn, row.names = FALSE)
nmr_meta_export(nmr_peak_table, metadata_fn, groups = "external")
nmr_dataset_save(nmr_peak_table, nmr_peak_table_rds)
```

```
## An nmr_dataset_peak_table (201 samples, and 122 peaks)
```

## Node 10: Machine learning

We will use a bootstrap and permutation method in a k-fold cross validation for VIPs selection. More information about this methodology can be found here, Ref: http://dx.doi.org/10.1016/j.aca.2013.01.004

Double cross validation based in random subsampling is used to select the number of components for plsda models in the bootstrap step.

The classification models in all steps will be PLSDA models, exploring at maximum 5 latent variables. Different plots will be displayed to visually check the process.

### Input parameters

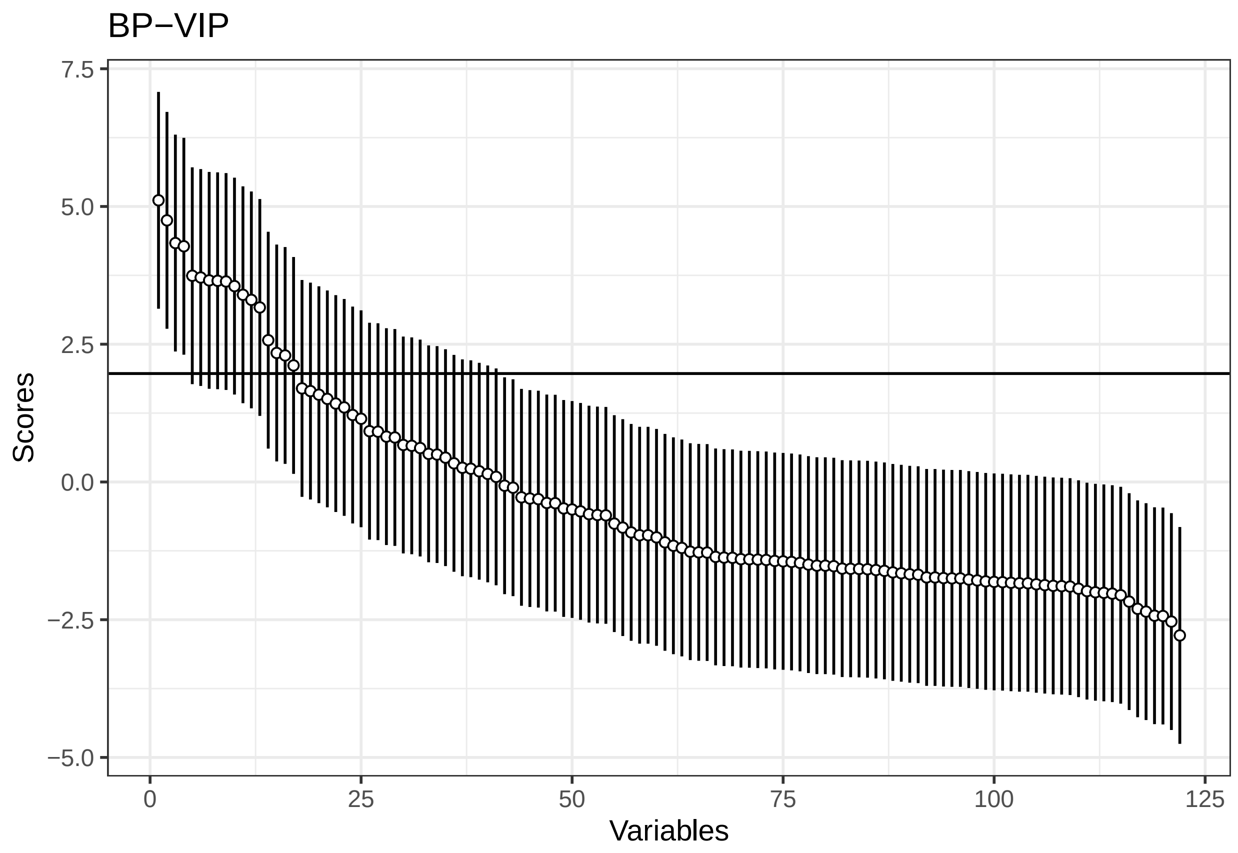
```
# Plsda parameters
max_ncomp = 5
auc_threshold = 0.01
label = "Timepoint"
external_iterations = 2
internal_iterations = 3
test_size = 0.25

# Set the number of bootstraps and k-folds
nbootstraps <- 300
k <- 4

# For permutation test
number_of_permutations = 100
```

### Code to run

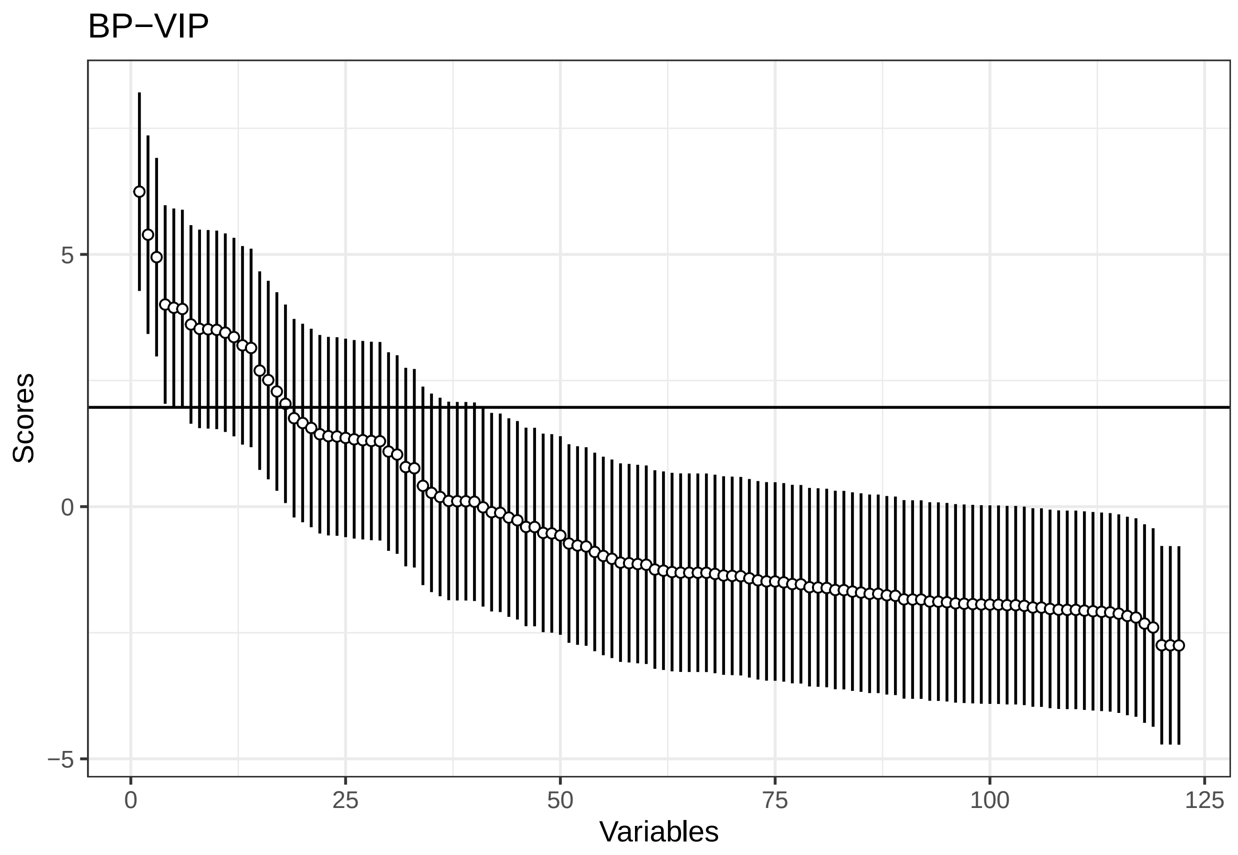
```
# We select the maximun number of components to use and the auc
# threshold that a component must increase to be selected
methodology <- plsda_auroc_vip_method(ncomp = max_ncomp, auc_increment_threshold = auc_threshold)
model <- nmr_data_analysis(
    nmr_peak_table, # Data to be analized
    y_column = label, # Label inside data that indicates the class
    identity_column = NULL,
    external_val = list(iterations = external_iterations,
                        test_size = test_size),
    internal_val = list(iterations = internal_iterations,
                        test_size = test_size),
    data_analysis_method = methodology
```

```
)
# The number of components for the bootstrap models is selected
ncomps <- max(unlist(sapply(sapply(model$outer_cv_results, "[", "model"), "[", "ncomp")))
```

```
# Bootstrap and permutation method in k-fold cross validation
bp_results <-
    bp_kfold_VIP_analysis(nmr_peak_table, # Data to be analized
                          y_column = label,
                          k = k,
                          ncomp = ncomps,
                          nbootstrap = nbootstraps)

# We can plot the mean of the vips of all kfolds and its standard
# desviation. The ones that have the lower bound over the
# threshold are considered important vips. If the lower bound is over 0,
# are considered relevant vips and the rest irrelevant.
plot(bp_results$vip_score_plot)
```



```
# We can also plot the scores of a particular fold
plot_vip_scores(
    bp_results$kfold_results[[1]]$pls_vip_means,
    bp_results$kfold_results[[1]]$error[1],
    nbootstraps
)
```



### Autoselected features

Then, we can extract VIP values from the autoselected features.

```
message("Selected VIPs are: ")
```

```
## Selected VIPs are:
```

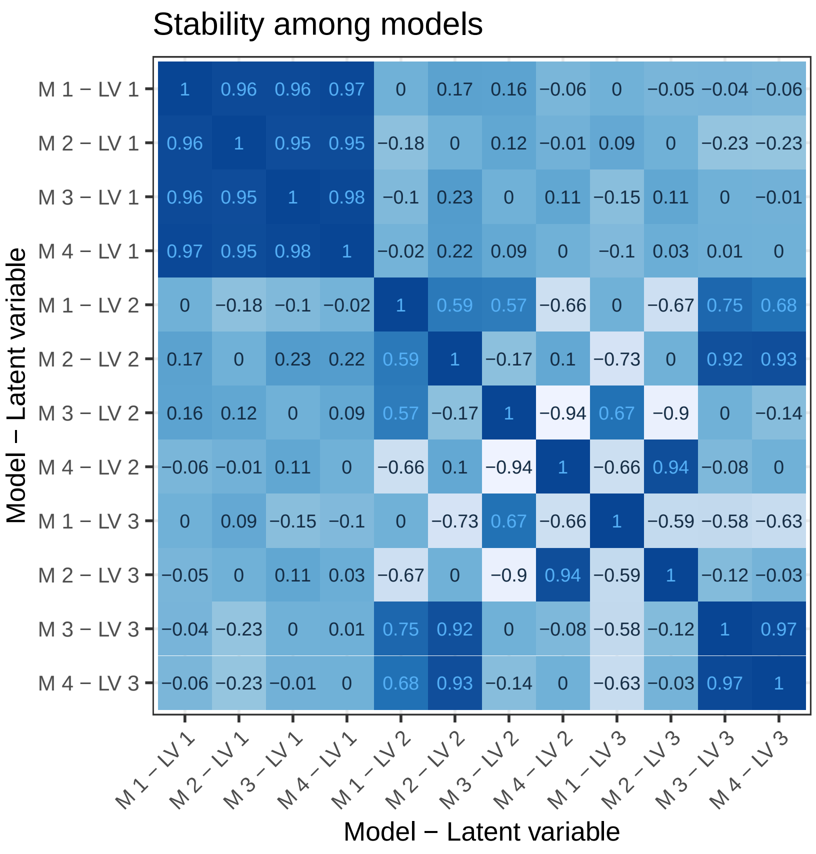
```
bp_results$important_vips
```

```
## [1] "ppm_1.2141" "ppm_3.5826"
```

```
VIPs <- sort(bp_results$important_vips)

# You can select the relevant_vips instead of importat ones if you want
# a less restrictive vip selection
#bp_results$relevant_vips
#VIPs <- sort(bp_results$relevant_vips)

# As a even less restrictive method, you can select the vips that pass a
# wilcoxon test
#bp_results$wilcoxon_vips
#VIPs <- sort(bp_results$wilcoxon_vips)
```

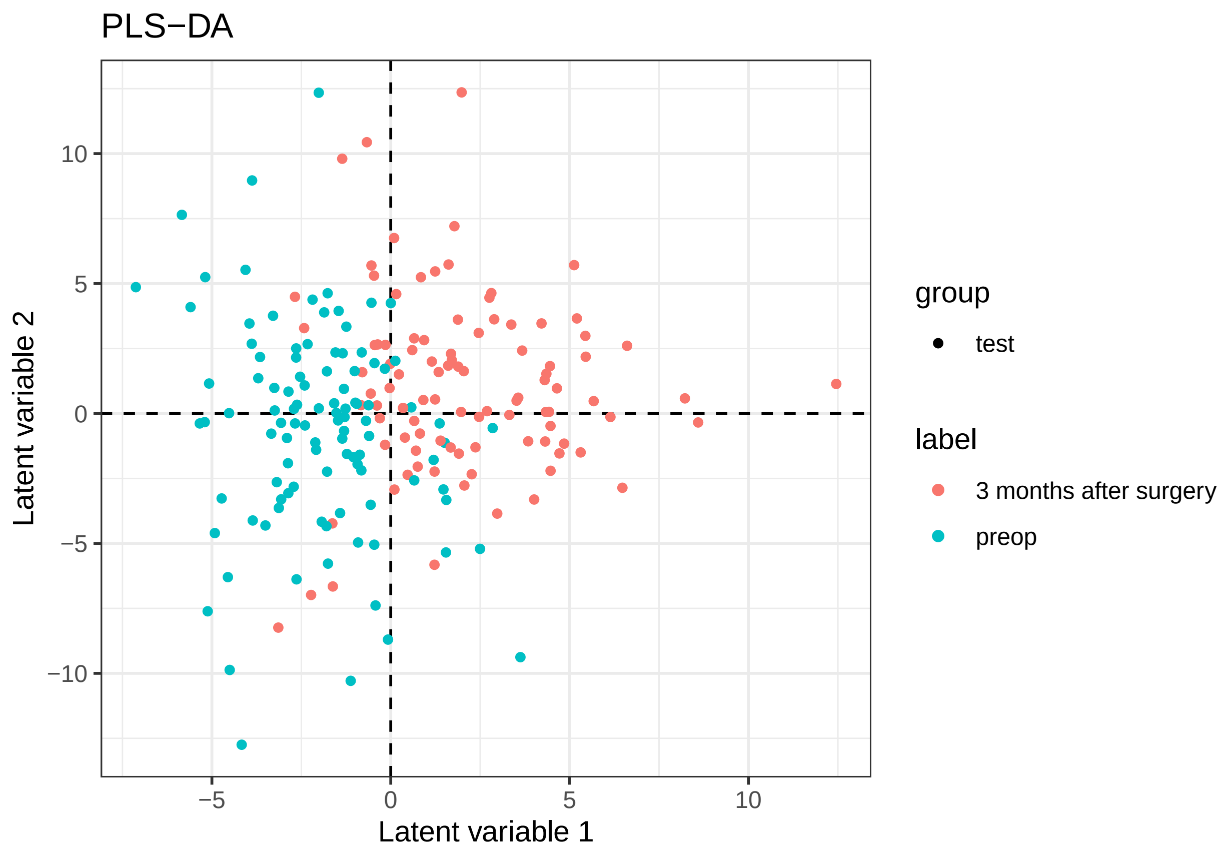
## Checking performance

```r
# We can check the stability of the models of the diferent kfolds to see
# if they are stable. If the values of the same latent variable among
# different models is close to 1, we can consider it similar models
models_stability_plot_bootstrap(bp_results)
```

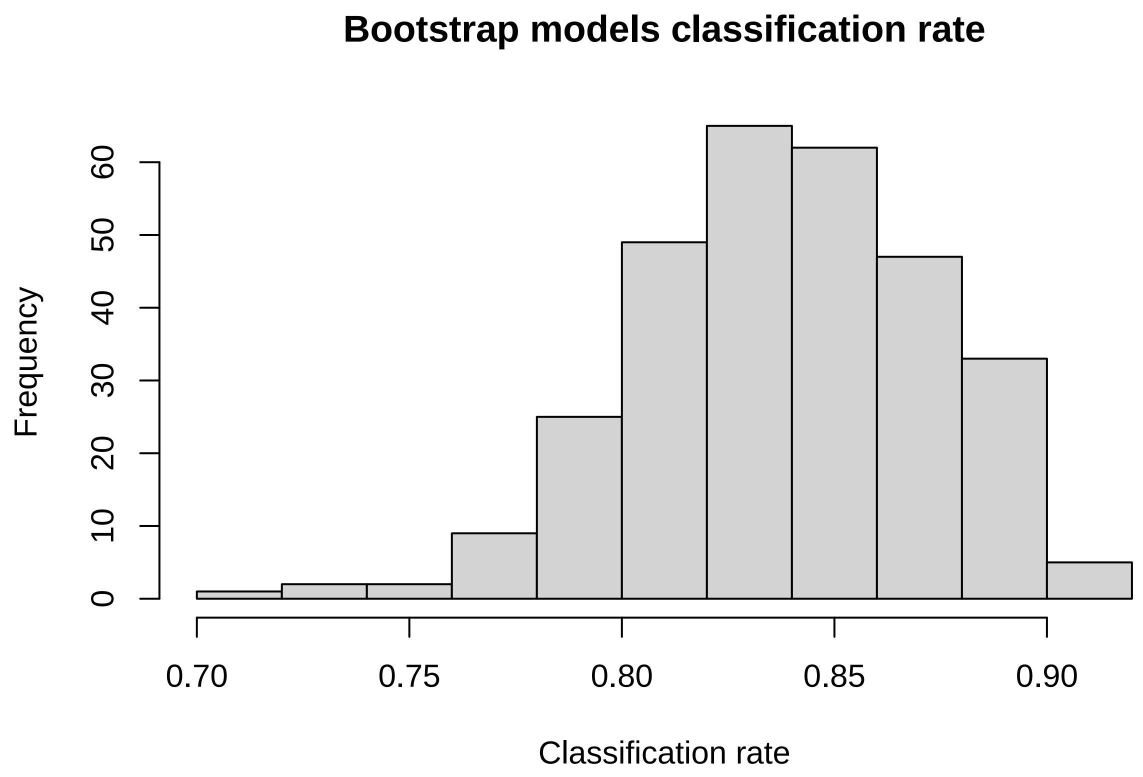


| | M 1 − LV 1 | M 2 − LV 1 | M 3 − LV 1 | M 4 − LV 1 | M 1 − LV 2 | M 2 − LV 2 | M 3 − LV 2 | M 4 − LV 2 | M 1 − LV 3 | M 2 − LV 3 | M 3 − LV 3 | M 4 − LV 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| M 1 − LV 1 | 1 | 0.96 | 0.96 | 0.97 | 0 | 0.17 | 0.16 | −0.06 | 0 | −0.05 | −0.04 | −0.06 |
| M 2 − LV 1 | 0.96 | 1 | 0.95 | 0.95 | −0.18 | 0 | 0.12 | −0.01 | 0.09 | 0 | −0.23 | −0.23 |
| M 3 − LV 1 | 0.96 | 0.95 | 1 | 0.98 | −0.1 | 0.23 | 0 | 0.11 | −0.15 | 0.11 | 0 | −0.01 |
| M 4 − LV 1 | 0.97 | 0.95 | 0.98 | 1 | −0.02 | 0.22 | 0.09 | 0 | −0.1 | 0.03 | 0.01 | 0 |
| M 1 − LV 2 | 0 | −0.18 | −0.1 | −0.02 | 1 | 0.59 | 0.57 | −0.66 | 0 | −0.67 | 0.75 | 0.68 |
| M 2 − LV 2 | 0.17 | 0 | 0.23 | 0.22 | 0.59 | 1 | −0.17 | 0.1 | −0.73 | 0 | 0.92 | 0.93 |
| M 3 − LV 2 | 0.16 | 0.12 | 0 | 0.09 | 0.57 | −0.17 | 1 | −0.94 | 0.67 | −0.9 | 0 | −0.14 |
| M 4 − LV 2 | −0.06 | −0.01 | 0.11 | 0 | −0.66 | 0.1 | −0.94 | 1 | −0.66 | 0.94 | −0.08 | 0 |
| M 1 − LV 3 | 0 | 0.09 | −0.15 | −0.1 | 0 | −0.73 | 0.67 | −0.66 | 1 | −0.59 | −0.58 | −0.63 |
| M 2 − LV 3 | −0.05 | 0 | 0.11 | 0.03 | −0.67 | 0 | −0.9 | 0.94 | −0.59 | 1 | −0.12 | −0.03 |
| M 3 − LV 3 | −0.04 | −0.23 | 0 | 0.01 | 0.75 | 0.92 | 0 | −0.08 | −0.58 | −0.12 | 1 | 0.97 |
| M 4 − LV 3 | −0.06 | −0.23 | −0.01 | 0 | 0.68 | 0.93 | −0.14 | 0 | −0.63 | −0.03 | 0.97 | 1 |

```r
# We can plot the prediction of all kfold test sets to visually inspect
# separation of classes
# Caution: next plot only is informative if the models stability is good,
# If the models only have 1 component, a histogram instead of a diagram
# will be ploted
plot_bootstrap_multimodel(bp_results, nmr_peak_table, label)
```



```r
# We can inspect the folds to see the classification rate (CR) of the
# bootstrap models
hist(unlist(bp_results$kfold_results[[1]]$classif_rate), main = "Bootstrap models classification rate",
```



```
# Also we can compare the general CR with the vips CR, to see how it
# change when we only use the important vips selected in that fold
print(paste("PLS CR of fold 1 is: ",
            bp_results$kfold_results[[1]]$general_CR))
```

```
## [1] "PLS CR of fold 1 is:  0.827814569536424"
```

```
print(paste("PLS CR of fold 1, using only important vips is: ",
            bp_results$kfold_results[[1]]$vips_CR))
```

```
## [1] "PLS CR of fold 1, using only important vips is:  0.887417218543046"
```

## Permutation test

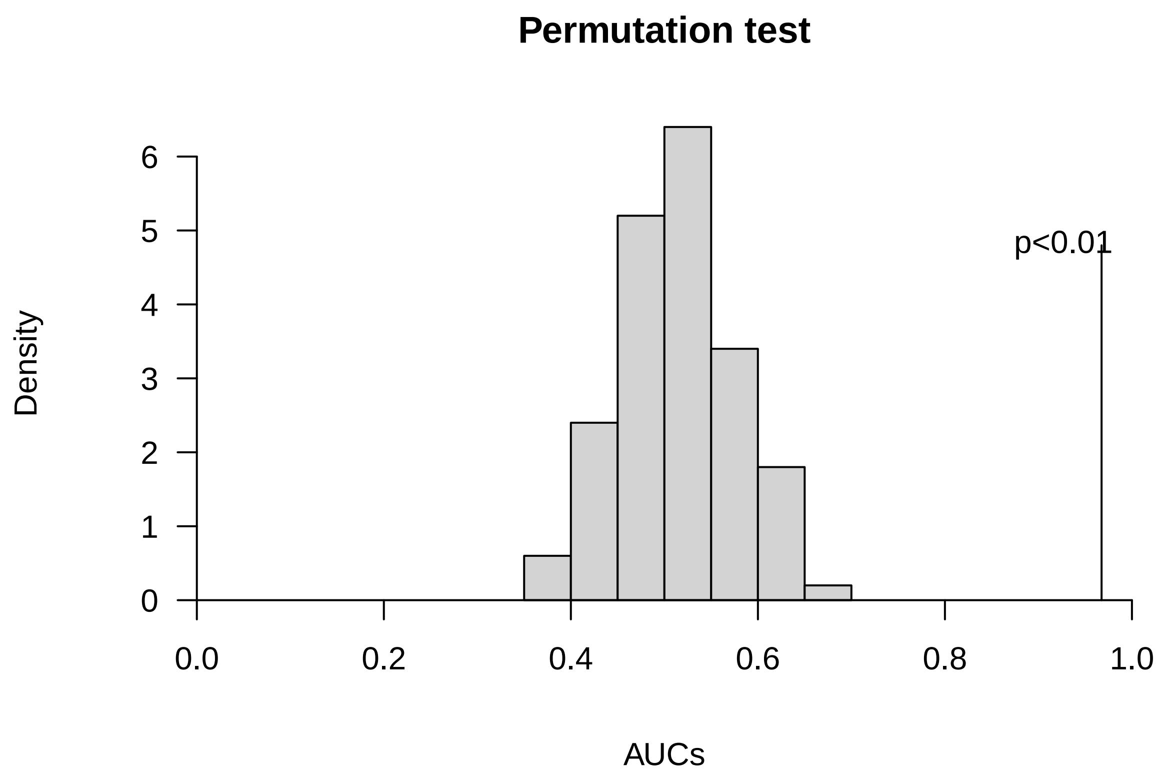
The function permutation_test_model performs the permutation test. Set the number of permutation at “nPerm”.

```
# We use the same parameters as the data analysis model
permutations = permutation_test_model(
    nmr_peak_table,
    y_column = label,
    # Label inside data that indicates the class
    identity_column = NULL,
    external_val = list(iterations = external_iterations,
                        test_size = test_size),
    internal_val = list(iterations = internal_iterations,
                        test_size = test_size),
    data_analysis_method = methodology,
    nPerm = number_of_permutations
)
```

### Permutation test plot

Permutation test plot for multilevel PLS-DA model gives a p-value for the model performance. The actual misclassification departs from the null hypothesis distribution.

```
permutation_test_plot(model, permutations)
```



### Save Results

```
output_dir_node10 <- file.path(output_dir, "10-machine_learning")
fs::dir_create(output_dir_node10)

model_rds <- file.path(output_dir_node10, "nmr_data_analysis_model.rds")
#confusion_matrix_fn <- file.path(output_dir_node10, "confusion_matrix.csv")
```

```r
VIPs_table_fn <- file.path(output_dir_node10, "VIPs.csv")
#permutations_fn <- file.path(output_dir_node10, "permutations.csv")

saveRDS(model, model_rds)
utils::write.csv(VIPs, VIPs_table_fn, row.names = FALSE)
#utils::write.csv(permutations, permutations_fn)
```

## Node 11: Identification

Finally, AlpsNMR allows an identification step, in which we can select between plasma/serum, urine and cell functions giving a ranked dataframe with ppm and proposed candidates by the Human Metabolome Database (http://www.hmdb.ca). However, this needs to be double-checked, as there is overlap between several potential compounds for a given ppm value and different specific metabolite-shifts, this should be carefully taken as a first step in the identification. First, we extract a vector with significant ppm values. Then, we run "nmr_identify_regions_blood". You can set the number of proposed candidates, but in this particular case, we set to 4. Even though, several NAs in the identification corresponded to Supplemental Table 1 in Supplemental Material.

### Autoselected features (blood samples)

```r
ppm_to_assign <- as.numeric(gsub("ppm_", "", VIPs))
assignation <-
  dplyr::select(nmr_identify_regions_blood(ppm_to_assign,
                                           num_proposed_compounds = 4),
                -Height)
assignation_NArm <- na.omit(assignation)
assignation
```

```
##                    Metabolite HMDB_code Shift_ppm Type  J_Hz Blood_concentration
## 12    3-Hydroxybutyric acid HMDB00357     1.219    d 6.262            147.7400
## 84          Isobutyric acid HMDB01873     1.225    d  7.02              2.3000
## NA                     <NA>      <NA>        NA <NA>  <NA>                  NA
## NA.1                   <NA>      <NA>        NA <NA>  <NA>                  NA
## 168             L-Threonine HMDB00167     3.590    d 4.867            124.2308
## 57                D-Mannose HMDB00169     3.579    t 9.705             51.5000
## NA1                    <NA>      <NA>        NA <NA>  <NA>                  NA
## NA.11                  <NA>      <NA>        NA <NA>  <NA>                  NA
##       n_reported_in_Blood ppm_to_assign
## 12                     13        1.2141
## 84                      1        1.2141
## NA                     NA        1.2141
## NA.1                   NA        1.2141
## 168                    13        3.5826
## 57                      2        3.5826
## NA1                    NA        3.5826
## NA.11                  NA        3.5826
```

```r
assignation_NArm
```

```
##                 Metabolite HMDB_code Shift_ppm Type  J_Hz Blood_concentration
## 12  3-Hydroxybutyric acid HMDB00357     1.219    d 6.262            147.7400
## 84        Isobutyric acid HMDB01873     1.225    d  7.02              2.3000
## 168           L-Threonine HMDB00167     3.590    d 4.867            124.2308
## 57              D-Mannose HMDB00169     3.579    t 9.705             51.5000
##     n_reported_in_Blood ppm_to_assign
## 12                   13        1.2141
## 84                    1        1.2141
## 168                  13        3.5826
## 57                    2        3.5826
```

### Save Results

```
output_dir_node11 <- file.path(output_dir, "11-identification")
fs::dir_create(output_dir_node11)

identification_fn <- file.path(output_dir_node11, "NMR_identification.csv")
identification_NArm_fn <- file.path(output_dir_node11, "NMR_identification_NArm.csv")

utils::write.csv(assignation, identification_fn, row.names = FALSE)
utils::write.csv(assignation_NArm, identification_NArm_fn, row.names = FALSE)
```