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

Institute for Bioengineering of Catalonia

November 13, 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

```
library(AlpsNMR)
```

```
## Loading required package: dplyr
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
## Loading required package: future
```

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

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.

```
# 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

```
# Execute next commented line to download demo data from
# https://www.ebi.ac.uk/metabolights/MTBLS242, it will take some time
# For download demo data:
#download_demo()

# Path of NMR samples
dataset_path_nmr <- "C:/Users/hgracia/Desktop/IBEC/MTBLS242"
# Files/directories ending in "s" corresponding to the spectra in the dataset:
filename_glob <- "*s"
```

Code to run

```
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

```
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)
```

```
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

```
# Path where metada is contained
metadata_file <- "C:/Users/hgracia/Desktop/IBEC/MTBLS242/s_mtbls242.txt"
```

Code to run

```
# We read metadata
metadata <- read.delim(metadata_file, header = TRUE, sep = "\t")
# we add needed columns NMRExperiment and Timepoint renaming existing
# columns. Sample.Name column needs the string "Obs" added  and
# change final "_S" to "s" to match existing NMRExperiment metadata
samplename <- substr(metadata[["Sample.Name"]], 1, nchar(metadata[["Sample.Name"]])-2)
meta <- data.frame("NMRExperiment" = paste0("Obs", samplename, "s"),
                   "Timepoint" = metadata[["Factor.Value.time.point."]])

nmr_dataset <- nmr_meta_add(nmr_dataset, meta)
```

Save results

```
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

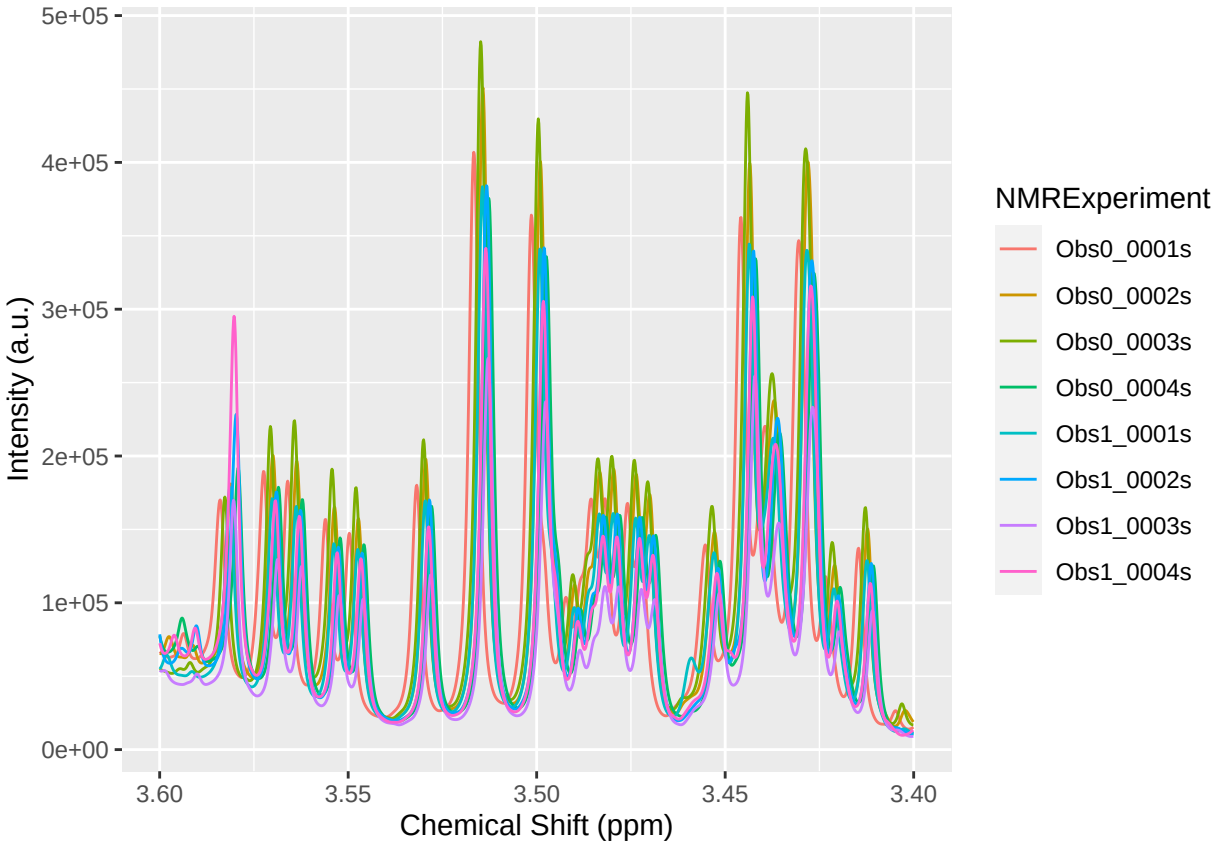
Interpolation is used to unify the ppm axis from all spectra. However, you also can set a range for next steps avoiding noise regions from here. Note that ppm resolution is automatically calculated with the function `nmr_ppm_resolution` in "Code to run".

Input parameters

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

Code to run

```
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

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

Code to run

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

Save Results

```
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)
```

```
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

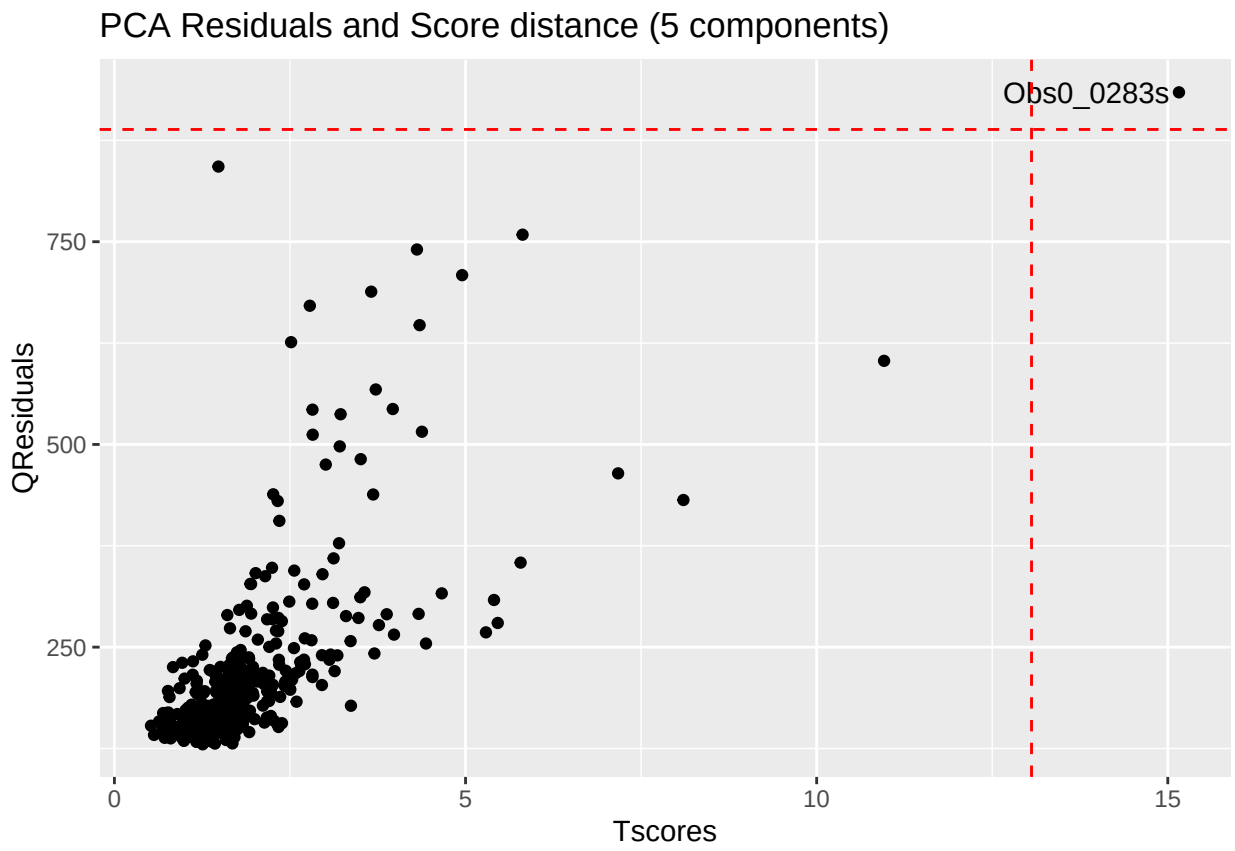
The robust principal component analysis (rPCA) for outlier detection gives an idea of potential outliers. A proposed threshold, based on quantiles, for Q residual and T2 score values, results less sensitive to extreme intensities. Then you choose if any sample should be excluded. The plot below indicated 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

```
# Nothing
```

Code to run

```
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:

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

Save results

```
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 <- 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 (195 samples)
```

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

```
## An nmr_dataset_1D (195 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). Ref: http://dx.doi.org/doi:10.1186/1471-2105-12-405

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...")
```

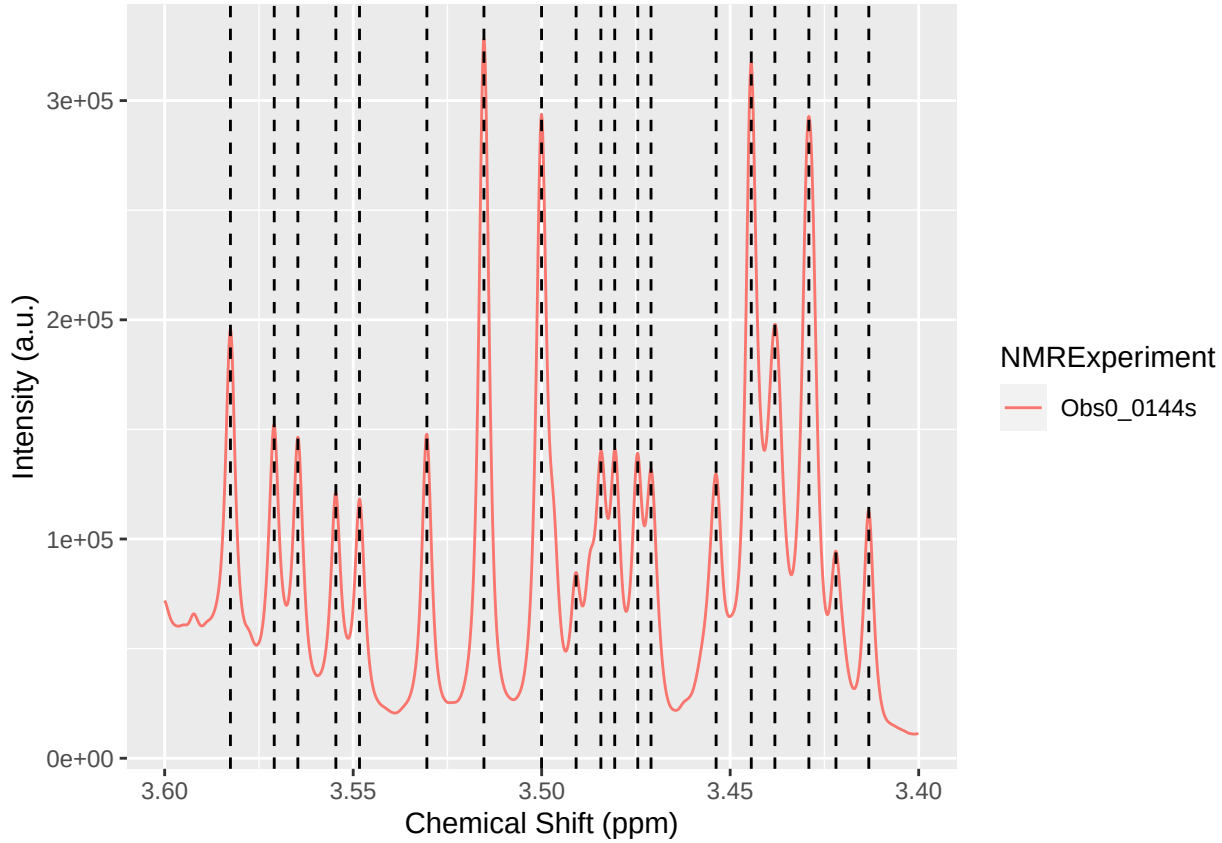
```
## Choosing alignment reference...
```

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

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

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

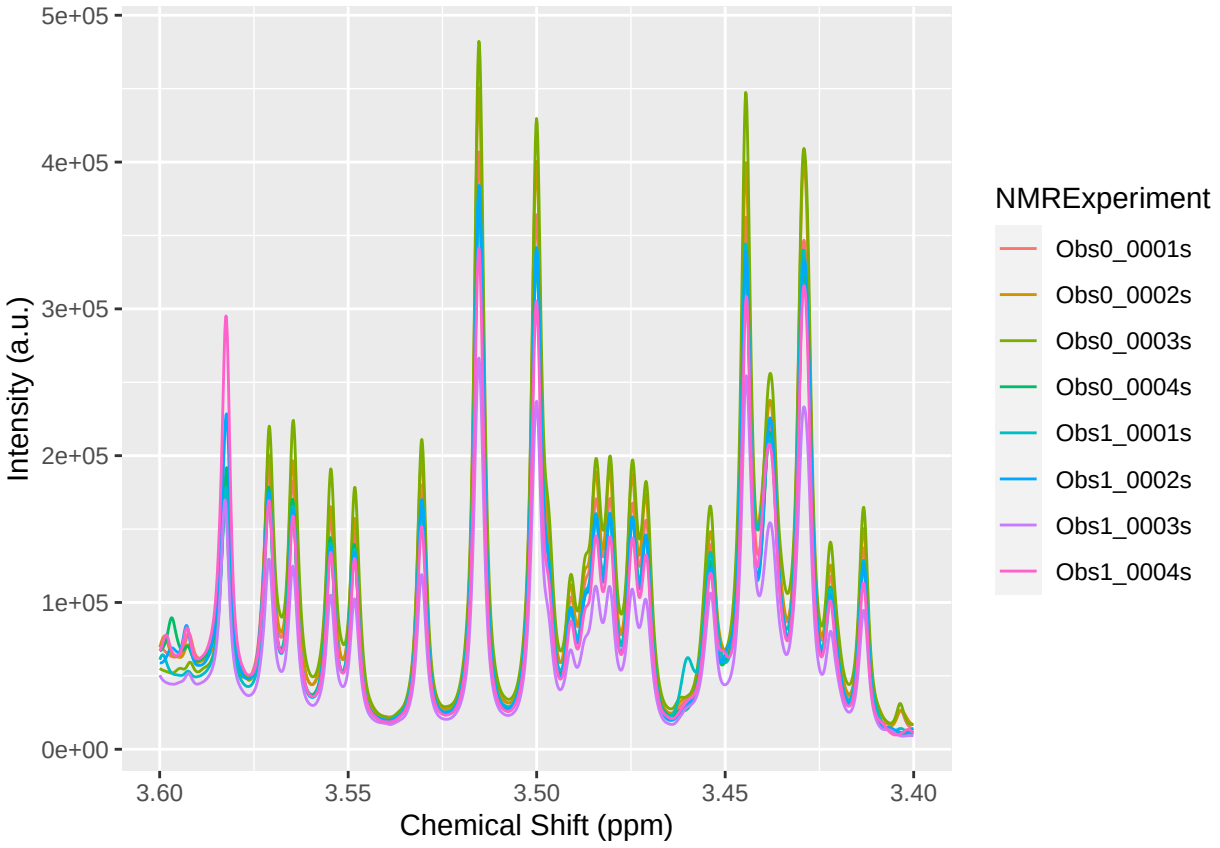
```
nmr_dataset <- nmr_align(nmr_dataset, peak_data,
                         NMRExp_ref = NMRExp_ref,
                         maxShift_ppm = maxShift_ppm,
                         acceptLostPeak = acceptLostPeak)

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")
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

```
# nothing
```

Code to run

```
nmr_dataset <- nmr_normalize(nmr_dataset, method = "pqn")
norm_pqn_diagnostic <- nmr_normalize_extra_info(nmr_dataset)
head(norm_pqn_diagnostic$norm_factor)
```

```
##   NMRExperiment norm_factor norm_factor_norm
## 1    Obs0_0001s  2388747542        1.3537324
## 2    Obs0_0002s  1731483641        0.9812529
## 3    Obs0_0003s  2171523386        1.2306289
## 4    Obs0_0004s  1675390586        0.9494643
## 5    Obs0_0008s  1752738628        0.9932984
## 6    Obs0_0009s  2122541997        1.2028705
```

```
# It is possible also plot the results as follows
#gplt_norm_factor_pqn <- norm_pqn_diagnostic$plot
```

Save results

```
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")

# To save the plot_norm_factor_pqn in case of need
# 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 (195 samples)
```

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

```
## An nmr_dataset_1D (195 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

```
peak_width_ppm <- NULL
```

Code to run

```
# 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
```

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

Save Results

```
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")
```

```r
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 (195 samples, and 122 peaks)
```

Node 10: Machine learning

Once the pre-processing stage is accomplished, relevant chemical information can be extracted from the data. We are interested in the automatic detection of features responsible for data separation by sample class. These autoselected features will be posteriorly associated to their corresponding metabolites in Node 11. Sample classification is performed using Partial Least Squares Discriminant Analysis (PLSDA). This allows us to use the Variable Important for Projection (VIP) score to rank data features according to their contribution to sample class separation. Feature selection process will be based on setting a threshold value for VIP score of each variable. Additionally, model stability for different train - test partitions of the dataset can be assessed using functions of this package. Different plots will be displayed to visually check the process.

Input parameters

```r
# Plsda parameters
max_ncomp = 5
auc_threshold = 0.05
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

The optimum number of latent variables for a PLDA model built from the data is obtained from a a double cross-validation based on random subsampling. All models used for selecting data features are created using this number of latent variables.

```r
# 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
)
```

```
##
## Attaching package: 'purrr'
```

```
## The following object is masked from 'package:magrittr':
##
##     set_names
```

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

Model Performance: Permutation test

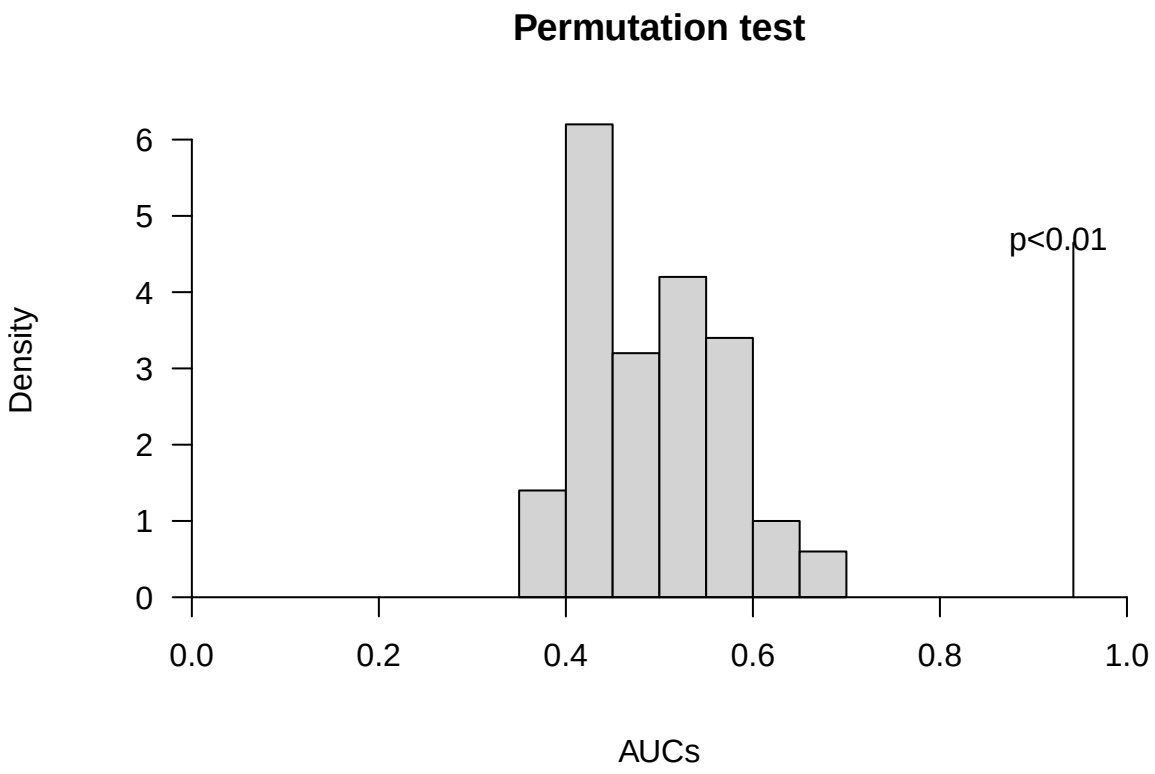
The function `permutation_test_model` performs the permutation test. It computes the AUC values of the external loop of the double cross-validation process for each permutation of the labels (Set the number of permutation at "nPerm").

```
# 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 the PLS-DA models gives a p-value for the model performance. The actual AUC departs from the null hypothesis distribution.

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



Feature selection

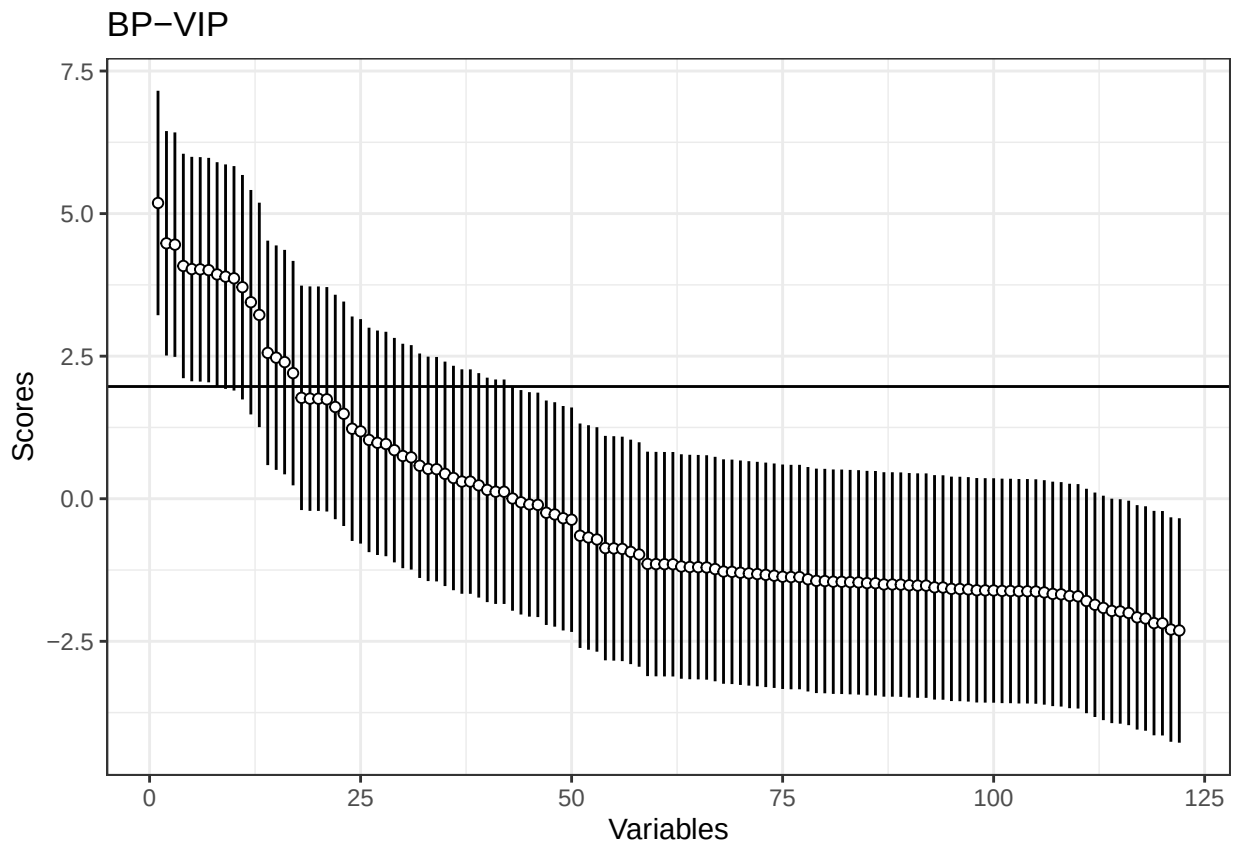
Feature selection process is carried out based on the bootstrap and permutation method. In the following paragraph this approach is briefly described. More information about this bootstrap and permutation method can be found here, Ref: http://dx.doi.org/10.1016/j.aca.2013.01.004.

Data is partitioned in k-folds. In each fold, m datasets are generated via bootstrapping. PLSDA models are created for the bootstrapped datasets. Each of these models will provide a VIP scores vector (that is a vector with the VIP score value for every feature). In the same fold, the previous process is repeated but this time each feature is randomly permuted within each bootstrap dataset. As a consequence, two distributions of VIP scores values are obtained for each feature: The first one is related to the true performance of the feature and the second one to the performance of a random selected feature. These two distributions are compared feature by feature. If they are similar, the feature is not relevant. Otherwise it is selected. So, for each fold, a number of features is selected. At the end of the process, only the features that were selected in each of the folds are considered relevant for data separation by sample class.

We can plot the mean of the vips of all k-folds and its standard deviation. The ones that have the lower bound over the threshold are considered important VIPs. There are different criteria for selecting will be discussed in the following section.

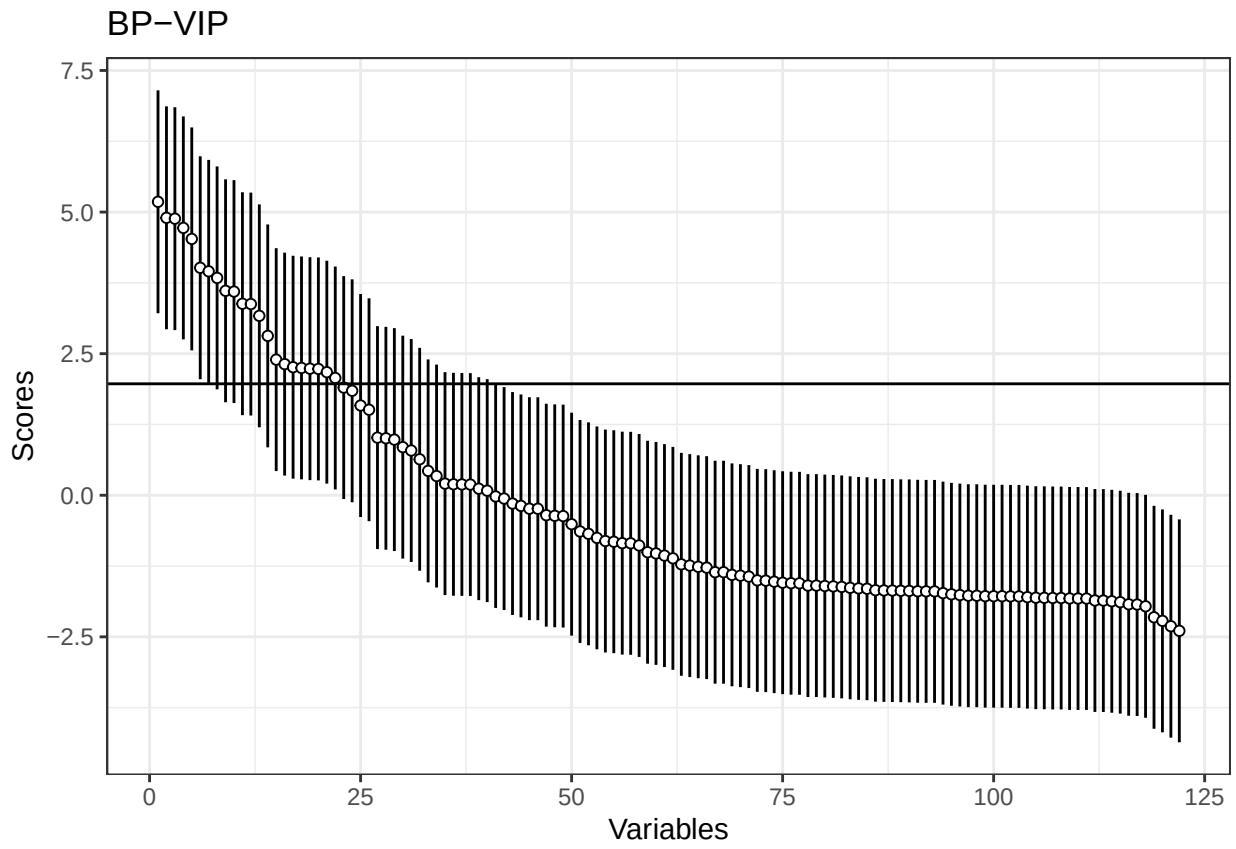
```
# Bootstrap and permutation method in k-fold cross validation
bp_results <-
    bp_kfold_VIP_analysis(nmr_peak_table, # Data to be analized
                          y_column = label,
                          k = k,
                          ncomp = ncomps,
```

```
                          nbootstrap = nbootstraps)


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
)
```



Different criteria for feature selection

There are three ways of selecting data feautes using the bootstrap and permutation method. The first way (1) is the most permissive strategy. It allows you selecting many features, although some of them may not be significant from the statistical point of view.

The threshold of selection is the mean value of the distribution corresponding the difference of the VIP scores distributions (that is 0). If the mean of the distribution is higher that the threshold, the variable is selected. Second (2) and (3) strategies are much more restrictive. In (2) the two distributions of VIP scores are compared using a Wilcoxon test at a significance level of 0.05. The null hypothesis is that the two distributions are the same, while the alternative hypothesis they don't. If the null hypothesis is rejected, the variable is selected. Finally, in strategy (3), the two distributions of VIP scores values are subtracted and the 95% confidence intervals around the differences are computed. Then, the t-statistic ($t_{1\text{-}\alpha/2,n\text{-}1}$ where α = 0.05 and n is equal to the number of bootstraps). If the 95% confidence interval lowe-bound is higher than $t_{1\text{-}\alpha/2,n\text{-}1}$ the variable is selected. Noteworthy, strategy (3) generally provides less number of autoselected features than strategies (1) and (2), giving raise to the most parsimonious final PSLDA models.

In this example, the most restrictive strategy (3) for feature selection is applied. You can find less restrictive choices commented in the chunk of code below:

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

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

```
bp_results$important_vips
```

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

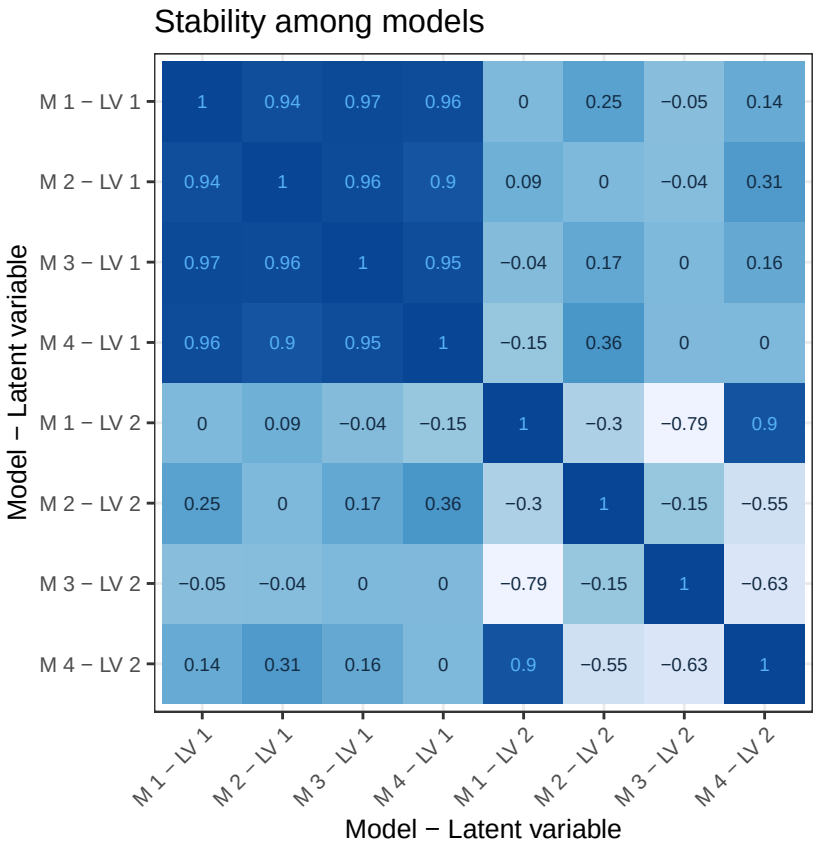
```
VIPs <- sort(bp_results$important_vips)

# You can select the relevant_vips instead of importat ones if you want
# a less restrictive vip selection (strategy (3) slightly modified)
#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  (strategy (2)
#bp_results$wilcoxon_vips
#VIPs <- sort(bp_results$wilcoxon_vips)
```

Model stability

Note that the previous feature selection process only makes sense if data behaves similar across the different k-fold partitions of the dataset. For stable data, the PLSDA models generated for different data partitions will be similar among them, and consequently, the feature selection process will be consistent. To check that, one thing we can do is to compare the loadings of the different models obtained from the training set in each k-fold by performing its dot product. Loadings from different models but the same latent variable are expected to get dot product values close to 1 (in absolute value) if the models are stable. Likewise, the dot product of loadings from different models and different latent variables is expected to be close to zero.

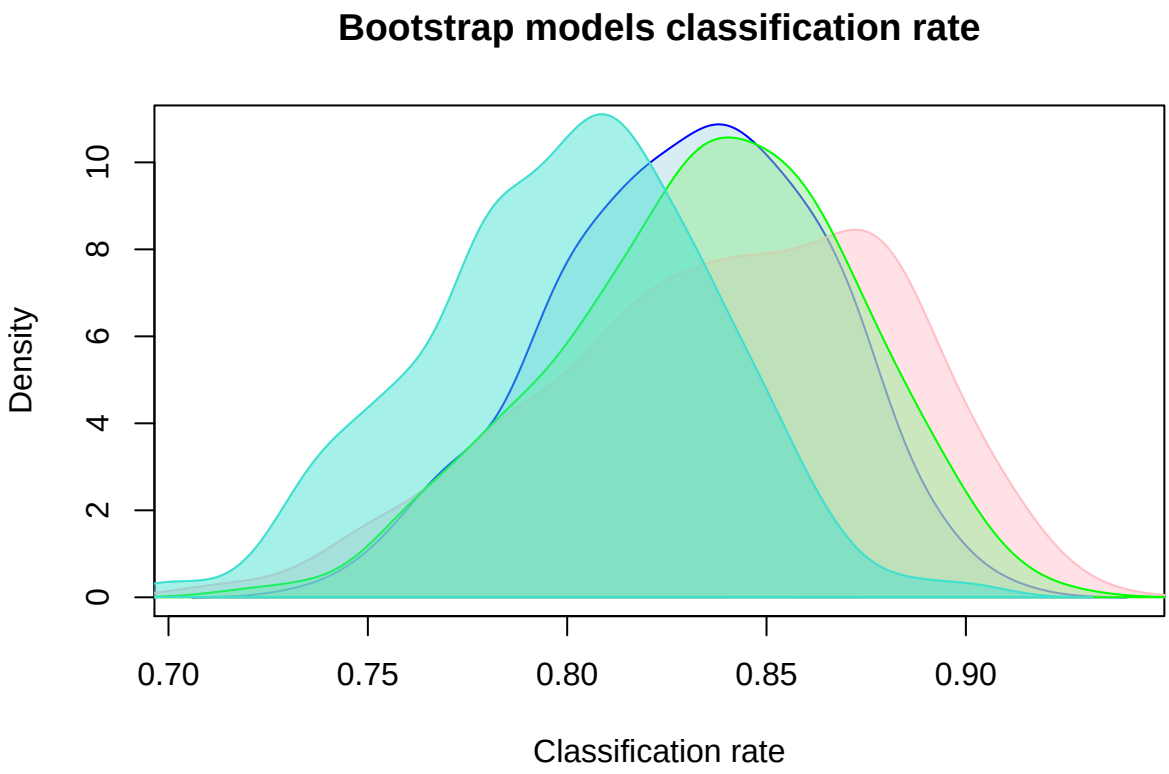
The following figure shows all combinations of dot products between loadings from different models (4-fold), and for the particular case of PLSDA models of 1 latent variable:

```
# Check the stability of the models of the diferent kfolds
models_stability_plot_bootstrap(bp_results)
```



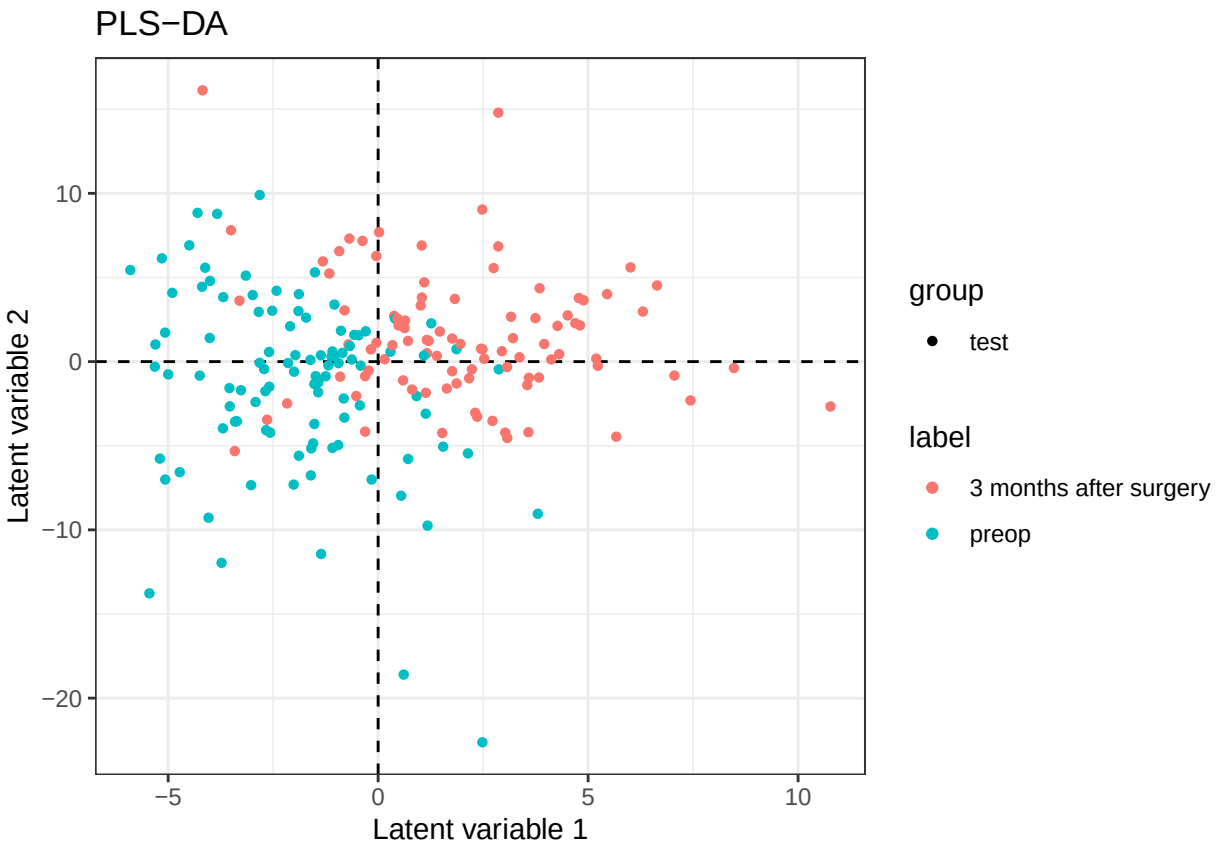
Another check of model stability can be the inspection of the classification rate (CR) distributions of the bootstrap models obtained for each of the k-folds. CR Distributions with similar mean and standard deviation are expected.

```
# Inspect the classification rate (CR) distribution of the
# bootstrap models, for all the folds
h1 <- unlist(bp_results$kfold_results[[1]]$classif_rate)
h2 <- unlist(bp_results$kfold_results[[2]]$classif_rate)
h3 <- unlist(bp_results$kfold_results[[3]]$classif_rate)
h4 <- unlist(bp_results$kfold_results[[4]]$classif_rate)
c1 <- rgb(173,216,230,max = 255, alpha = 120, names = "lt.blue")
c2 <- rgb(255,192,203, max = 255, alpha = 120, names = "lt.pink")
c3 <- rgb(144,238,144, max = 255, alpha = 120, names = "lt.green")
c4 <- rgb(64,224,208, max = 255, alpha = 120, names = "lt.yellow")

plot(density(h1), col = c1, main = "Bootstrap models classification rate",
     xlab = "Classification rate")
polygon(density(h1), col=c1, border="blue")
lines(density(h2), col = c2)
polygon(density(h2), col=c2, border="pink")
lines(density(h3), col = c3)
polygon(density(h3), col=c3, border="green")
lines(density(h4), col = c4)
polygon(density(h4), col=c4, border="turquoise")
```



Since model stability seems good enough, we can inspect the separability of the classes projecting on the same latent variable space the test samples for the different k partitions of the dataset. This way, all samples in the dataset are employed as test samples. Since each of the PLSDA models have just one latent variable this result is automatically represented with histograms. In case of having models with more of one latent variable, the first two latent variables are used to generate a scores plot.

```
# 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 scores plot
# will be plotted
plot_bootstrap_multimodel(bp_results, nmr_peak_table, label)
```



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.

```
paste("PLSDA CR of fold 1 is: ",
            bp_results$kfold_results[[1]]$general_CR)
```

```
## [1] "PLSDA CR of fold 1 is:  0.843537414965986"
```

```
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:  0.884353741496599"
```

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")
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)

```
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
```

```
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)

sessionInfo()
```

```
## R version 4.0.3 (2020-10-10)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 18363)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=Spanish_Spain.1252  LC_CTYPE=Spanish_Spain.1252
## [3] LC_MONETARY=Spanish_Spain.1252 LC_NUMERIC=C
## [5] LC_TIME=Spanish_Spain.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] purrr_0.3.4   AlpsNMR_3.1.3 magrittr_1.5  future_1.20.1 dplyr_1.0.2
##
## loaded via a namespace (and not attached):
##  [1] fs_1.5.0                    bitops_1.0-6
##  [3] matrixStats_0.57.0          RColorBrewer_1.1-2
##  [5] httr_1.4.2                  GenomeInfoDb_1.26.0
##  [7] tools_4.0.3                 R6_2.5.0
##  [9] BiocGenerics_0.36.0         colorspace_1.4-1
## [11] tidyselect_1.1.0            gridExtra_2.3
## [13] mQTL_1.0                    compiler_4.0.3
## [15] MassSpecWavelet_1.56.0      rvest_0.3.6
## [17] Biobase_2.50.0              xml2_1.3.2
## [19] DelayedArray_0.16.0         labeling_0.4.2
## [21] scales_1.1.1                randomForest_4.6-14
## [23] stringr_1.4.0               digest_0.6.27
## [25] rmarkdown_2.5               XVector_0.30.0
```

```
## [27] pkgconfig_2.0.3            htmltools_0.5.0
## [29] parallelly_1.21.0          MatrixGenerics_1.2.0
## [31] itertools_0.1-3            rlang_0.4.8
## [33] impute_1.64.0              generics_0.1.0
## [35] farver_2.0.3               BiocParallel_1.24.0
## [37] speaq_2.6.1                RCurl_1.98-1.2
## [39] qtl_1.46-2                 GenomeInfoDbData_1.2.4
## [41] Matrix_1.2-18              Rcpp_1.0.5
## [43] munsell_0.5.0              S4Vectors_0.28.0
## [45] lifecycle_0.2.0            RcppZiggurat_0.1.6
## [47] furrr_0.2.1                stringi_1.5.3
## [49] yaml_2.2.1                 MASS_7.3-53
## [51] SummarizedExperiment_1.20.0 zlibbioc_1.36.0
## [53] plyr_1.8.6                 grid_4.0.3
## [55] parallel_4.0.3             listenv_0.8.0
## [57] ggrepel_0.8.2              crayon_1.3.4
## [59] doSNOW_1.0.19              lattice_0.20-41
## [61] knitr_1.30                 pillar_1.4.6
## [63] igraph_1.2.6               GenomicRanges_1.42.0
## [65] corpcor_1.6.9              reshape2_1.4.4
## [67] codetools_0.2-18           mixOmics_6.14.0
## [69] stats4_4.0.3               glue_1.4.2
## [71] evaluate_0.14              outliers_0.14
## [73] data.table_1.13.2          vctrs_0.3.4
## [75] missForest_1.4             foreach_1.5.1
## [77] gtable_0.3.0               tidyr_1.1.2
## [79] assertthat_0.2.1           ggplot2_3.3.2
## [81] xfun_0.19                  Rfast_2.0.1
## [83] RSpectra_0.16-0            rARPACK_0.11-0
## [85] tibble_3.0.4               snow_0.4-3
## [87] iterators_1.0.13           IRanges_2.24.0
## [89] ellipse_0.4.2              cluster_2.1.0
## [91] globals_0.13.1             ellipsis_0.3.1
```