Package 'TIGERr'

January 20, 2025

Type Package

Title Technical Variation Elimination with Ensemble Learning Architecture

Version 1.0.0

Author Siyu Han [aut, cre], Jialing Huang [aut], Francesco Foppiano [aut], Cornelia Prehn [aut], Jerzy Adamski [aut], Karsten Suhre [aut], Ying Li [aut], Giuseppe Matullo [aut], Freimut Schliess [aut], Christian Gieger [aut], Annette Peters [aut], Rui Wang-Sattler [aut]

Maintainer Siyu Han <siyu.han@helmholtz-muenchen.de>

Acknowledgments TAI Yun-hsiu, WANG Ruoyu, CHENG Ming, GUO Yuan, LI Han, FAN Linrui

Description The R implementation of TIGER.
TIGER integrates random forest algorithm into an innovative ensemble learning architecture. Benefiting from this advanced architecture, TIGER is resilient to outliers, free from model tuning and less likely to be affected by specific hyperparameters.
TIGER supports targeted and untargeted metabolomics data and is competent to perform both intra- and inter-batch technical variation removal. TIGER can also be used for cross-kit adjustment to ensure data obtained from different analytical assays can be effectively combined and compared.
Reference: Han S. et al. (2022) <doi:10.1093/bib/bbab535>.

License GPL (>= 3)

Depends R (>= 3.5.0)

Imports parallel (>= 2.1.0), pbapply (>= 1.4-3), ppcor (>= 1.1), randomForest (>= 4.6-14), stats (>= 3.0.0)

BugReports https://github.com/HAN-Siyu/TIGER/issues

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2022-01-06 14:00:02 UTC

Contents

`compute_RSD` *Compute RSD (relative standard deviation)*

Description

This function computes the RSD (relative standard deviation) of the values in `input_data`. Missing values are removed before the computation automatically.

Usage

```
compute_RSD(input_data)
```

Arguments

`input_data`	a numeric vector

Details

The RSD in this function is computed by:

`sd(input_data, na.rm = TRUE) / mean(input_data, na.rm = TRUE)`.

Value

The RSD of the values in `input_data` is computed, as a numeric of length one.

Examples

```
RSD_1 <- compute_RSD(c(1:10))

data(FF4_qc) # load demo dataset

# RSD of QC:
RSD_2 <- sapply(FF4_qc[FF4_qc$sampleType == "QC", -c(1:5)], compute_RSD)
quantile(RSD_2)

# RSD of different types of QC samples:
# (each metabolote has its own RSD)
RSD_3 <- aggregate(FF4_qc[-c(1:5)], by = list(Type = FF4_qc$sampleType),
                   FUN = compute_RSD)
```

`compute_targetVal` *Compute target values for ensemble learning architecture*

Description

This function provides an advanced option to calculate the target values of one reference dataset (i.e. `QC_num`, numeric values of quality control samples). The generated target values (a list) can be further passed to argument `targetVal_external` in function `run_TIGER` such that TIGER can align the `test_samples` with the reference dataset. This is useful for longitudinal datasets correction and cross-kit adjustment. See case study section of our original paper for detailed explanation.

Usage

```
compute_targetVal(
  QC_num,
  sampleType,
  batchID = NULL,
  targetVal_method = c("mean", "median"),
  targetVal_batchWise = FALSE,
  targetVal_removeOutlier = !targetVal_batchWise,
  coerce_numeric = FALSE
)
```

Arguments

`QC_num`	a numeric data.frame including the metabolite values of quality control (QC) samples. Missing values and infinite values will not be taken into account. Row: sample. Column: metabolite variable. See Examples.
`sampleType`	a vector corresponding to `QC_num` to specify the type of each QC sample. QC samples of the **same type** should have the **same type name**. See Examples.
`batchID`	a vector corresponding to `QC_num` to specify the batch of each sample. Ignored if `targetVal_batchWise = FALSE`. See Examples.
`targetVal_method`	a character string specifying how the target values are computed. Can be `"mean"` (default) or `"median"`. See Details.
`targetVal_batchWise`	logical. If `TRUE`, the target values will be computed based on each batch, otherwise, based on the whole dataset. Setting `TRUE` might be useful if your dataset has very obvious batch effects, but this may also make the algorithm less robust. See Details. Default: `FALSE`.
`targetVal_removeOutlier`	logical. If `TRUE`, outliers will be removed before the computation. Outliers are determined with 1.5 * IQR (interquartile range) rule. We recommend turning this off when the target values are computed based on batches. See Details. Default: `!targetVal_batchWise`.

`coerce_numeric`	logical. If `TRUE`, values in `QC_num` will be coerced to numeric before the computation. The columns cannot be coerced will be removed (with warnings). See Examples. Default: `FALSE`.

Details

See `run_TIGER`.

Value

If `targetVal_batchWise = FALSE`, the function returns a list of length one containing the target values computed on the whole dataset.

If `targetVal_batchWise = TRUE`, a list containing the target values computed on different batches is returned. The length of the returned list equals the number of batch specified by `batchID`.

Examples

```
data(FF4_qc) # load demo dataset
QC_num <- FF4_qc[-c(1:5)] # only contain numeric metabolite values.

# target values computed on the whole dataset:
tarVal_1 <- compute_targetVal(QC_num = QC_num,
                              sampleType = FF4_qc$sampleType,
                              batchID = FF4_qc$plateID,
                              targetVal_method = "mean",
                              targetVal_batchWise = FALSE,
                              targetVal_removeOutlier = TRUE)

# target values computed on batches:
tarVal_2 <- compute_targetVal(QC_num = QC_num,
                              sampleType = FF4_qc$sampleType,
                              batchID = FF4_qc$plateID,
                              targetVal_method = "mean",
                              targetVal_batchWise = TRUE,
                              targetVal_removeOutlier = FALSE)

# If coerce_numeric = TRUE,
# columns cannot be coerced to numeric will be removed (with warnings):
tarVal_3 <- compute_targetVal(QC_num = FF4_qc[-c(4:5)],
                              sampleType = FF4_qc$sampleType,
                              batchID = FF4_qc$plateID,
                              targetVal_method = "mean",
                              targetVal_batchWise = TRUE,
                              targetVal_removeOutlier = FALSE,
                              coerce_numeric = TRUE)
identical(tarVal_2, tarVal_3)  # identical to tarVal_2

## Not run:

# will throw errors if input data have non-numeric columns
# and coerce_numeric = FALSE:
```

```
tarVal_4 <- compute_targetVal(QC_num = FF4_qc,
                              sampleType = FF4_qc$sampleType,
                              batchID = FF4_qc$plateID,
                              targetVal_method = "mean",
                              targetVal_batchWise = TRUE,
                              targetVal_removeOutlier = FALSE,
                              coerce_numeric = FALSE)

## End(Not run)
```

FF4_qc *Accompanying QC samples of KORA FF4 (demo data)*

Description

This demo dataset, a data.frame with 232 samples (rows) and 108 variables (columns). The dataset includes four types of quality control (QC) samples from 29 kit plates:

- QC1 ($N = 29$, one per plate),
- QC2 ($N = 29$, one per plate),
- QC3 ($N = 29$, one per plate),
- QC ($N = 145$, five per plate).

The columns include sample ID, sample type, plate ID, well position, injection order and the concentrations of 103 selected targeted metabolites. These QC samples are measured with the cohort samples of KORA FF4 (Cooperative Health Research in the Augsburg Region, the second follow-up study, 2013–2014) using the analytical assay Biocrates Absolute*IDQ*® p180 (BIOCRATES Life Sciences AG, Innsbruck, Austria).

In our paper, we used QC as training samples, while QC1, QC2, QC3 and cohort samples were used as test samples. The cohort data are operated by Helmholtz Zentrum München and available via KORA platform `https://www.helmholtz-munich.de/en/kora/` upon reasonable request. See Reference for detailed information.

Usage

```
data(FF4_qc)
```

Reference

Han S. *et al*. TIGER: technical variation elimination for metabolomics data using ensemble learning architecture. *Briefings in Bioinformatics* (2022) bbab535. doi: 10.1093/bib/bbab535.

`run_TIGER` *Run TIGER to eliminate technical variation*

Description

Use TIGER algorithm to eliminate the technical variation in metabolomics data. TIGER supports targeted and untargeted metabolomics data and is competent to perform both intra- and inter-batch technical variation removal.

Usage

```
run_TIGER(
  test_samples,
  train_samples,
  col_sampleID,
  col_sampleType,
  col_batchID,
  col_order = NULL,
  col_position = NULL,
  targetVal_external = NULL,
  targetVal_method = c("mean", "median"),
  targetVal_batchWise = FALSE,
  targetVal_removeOutlier = !targetVal_batchWise,
  selectVar_external = NULL,
  selectVar_corType = c("cor", "pcor"),
  selectVar_corMethod = c("pearson", "spearman"),
  selectVar_minNum = 5,
  selectVar_maxNum = 10,
  selectVar_batchWise = FALSE,
  mtry_percent = seq(0.2, 0.8, 0.2),
  nodesize_percent = seq(0.2, 0.8, 0.2),
  ...,
  parallel.cores = 2
)
```

Arguments

`test_samples` (required) a data.frame containing the samples to be corrected (for example, subject samples). This data.frame should contain columns of

- sample ID (required): name or label for each sample,
- sample type (required): indicating the type of each sample,
- batch ID (required): the batch of each sample,
- order information (optional): injection order or temporal information of each sample,
- position information (optional): well position of each sample,

	• metabolite values (required): values to be normalised. Infinite values are not allowed. Row: sample. Column: variable. See Examples.
`train_samples`	(required) a data.frame containing the quality control (QC) samples used for model training. The columns in this data.frame should correspond to the columns in `test_samples`. And `test_samples` and `train_samples` should have the identical column names.
`col_sampleID`	(required) a character string indicating the name of the column that specifies the sample ID of each sample. The values in this column will not affect the data correction process but can act as labels for different samples. See Examples.
`col_sampleType`	(required) a character string indicating the name of the column that specifies the type (such as QC1, QC2, subject) of each sample. This column can be used to indicate different kinds of QC samples in `train_samples`. QC samples of the **same type** should have the **same type name**. See Examples.
`col_batchID`	(required) a character string indicating the name of the column that specifies the batch ID of each sample. See Examples.
`col_order`	(optional) `NULL` or a character string indicating the name of the column that contains the injection order or temporal information (numeric values). This can explicitly ask the algorithm capture the technical variation introduced by injection order, which might be useful when your data have very obvious temporal drifts. If `NULL` (default), `train_samples` and `test_samples` should have **No** column contains injection order information.
`col_position`	(optional) `NULL` or a character string indicating the name of the column that contains the well position information (numeric values). This can explicitly ask the algorithm capture the technical variation introduced by well position, which might be useful when the well position has a great impact during data acquisition. If `NULL` (default), `train_samples` and `test_samples` should have **No** column contains well position information.
`targetVal_external`	(optional) a list generated by function `compute_targetVal`. See Details.
`targetVal_method`	a character string specifying how target values are to be computed. Can be `"mean"` (default) or `"median"`. Ignored if a list of external target values has been assigned to `targetVal_external`.
`targetVal_batchWise`	logical. If `TRUE`, the target values will be computed based on each batch, otherwise, based on the whole dataset. Setting `TRUE` might be useful if your dataset has very obvious batch effects, but this may also make the algorithm less robust. Default: `FALSE`. Ignored if a list of external target values has been assigned to `targetVal_external`.
`targetVal_removeOutlier`	logical. If `TRUE`, outliers will be removed before the computation. Outliers are determined with 1.5 * IQR (interquartile range) rule. We recommend turning this off when the target values are computed based on batches. Default: `!targetVal_batchWise`. Ignored if a list of external target values has been assigned to `targetVal_external`.

`selectVar_external`	(optional) a list generated by function `select_variable`. See Details.
`selectVar_corType`	a character string indicating correlation (`"cor"`, default) or partial correlation (`"pcor"`) is to be used. Can be abbreviated. Ignored if a list of selected variables has been assigned to `selectVar_external`. **Note**: computing partial correlations of a large dataset can be very time-consuming.
`selectVar_corMethod`	a character string indicating which correlation coefficient is to be computed. One of `"spearman"` (default) or `"pearson"`. Can be abbreviated. Ignored if a list of selected variables has been assigned to `selectVar_external`.
`selectVar_minNum`	an integer specifying the minimum number of the selected metabolite variables (injection order and well position are not regarded as metabolite variables). If `NULL`, no limited, but 1 at least. Default: `5`. Ignored if a list of selected variables has been assigned to `selectVar_external`.
`selectVar_maxNum`	an integer specifying the maximum number of the selected metabolite variables (injection order and well position are not regarded as metabolite variables). If `NULL`, no limited, but no more than the number of all available metabolite variables. Default: `10`. Ignored if a list of selected variables has been assigned to `selectVar_external`.
`selectVar_batchWise`	(advanced) logical. Specify whether the variable selection should be performed based on each batch. Default: `FALSE`. Ignored if a list of selected variables has been assigned to `selectVar_external`. **Note**: the support of batch-wise variable selection is provided for data requiring special processing (for example, data with strong batch effects). But in most case, batch-wise variable selection is not necessary. Setting `TRUE` can make the algorithm less robust.
`mtry_percent`	(advanced) a numeric vector indicating the percentages of selected variables randomly sampled as candidates at each split when training random forest models (base learners). **Note**: providing more arguments will include more base learners into the ensemble model, which will increase the processing time. Default: `seq(0.2, 0.8, 0.2)`.
`nodesize_percent`	(advanced) a numeric vector indicating the percentages of sample size used as the minimum sizes of the terminal nodes in random forest models (base learners). **Note**: providing more arguments will include more base learners into the ensemble model, which will increase the processing time. Default: `seq(0.2, 0.8, 0.2)`.
`...`	(advanced) optional arguments (except `mtry` and `nodesize`) to be passed to `randomForest` for model training. Arguments `mtry` and `nodesize` are determined by `mtry_percent` and `nodesize_percent`. See `randomForest` and Examples. **Note**: providing more arguments will include more base learners into the ensemble model, which will increase the processing time.
`parallel.cores`	an integer (== -1 or >= 1) specifying the number of cores for parallel computation. Setting `-1` to run with all cores. Default: `2`.

Details

TIGER can effectively process the datasets with its default setup. The following hyperparameters are provided to customise the algorithm and achieve the best possible performance. These hyperparameters are also practical for some special purposes (such as cross-kit adjustment, longitudinal dataset correction) or datasets requiring special processing (for example, data with very strong temporal drifts or batch effects). We recommend users to examine the normalised result with different metrics, such as RSD (relative standard deviation), MAPE (mean absolute percentage error) and PCA (principal component analysis), especially when using advanced options of TIGER.

Hyperparameters for target value computation

- `targetVal_external`

  TIGER by default captures and eliminate the technical variation within the input dataset, and the target values are automatically computed from `train_samples`. The target values can also be calculated from a reference dataset using function `compute_targetVal` and then passed to this function as an argument. This will enable TIGER to align `test_samples` with the reference dataset. In this case, `train_samples` is still the accompanying QC samples of `test_samples`. And argument `targetVal_external` accepts external target values (a list). If the list of external target values is provided, values in `targetVal_method`, `targetVal_batchWise` and `targetVal_removeOutlier` will be ignored.

- `targetVal_method`

  The target values can be the mean or median values of metabolite values. The target values of different kinds of QC samples are computed separately. `"mean"` is recommended here, but the optimal selection can differ for different datasets.

- `targetVal_batchWise`

  The target values can be computed from the whole dataset or from different batches. By default, the target values are computed based on the whole dataset. Computing based on batches (`targetVal_batchWise = TRUE`) is only recommended when the samples has very strong batch effects. For example, we set this as `TRUE` when normalising WaveICA's Amide dataset in our original paper.

- `targetVal_removeOutlier`

  If computing is based on the whole dataset (`targetVal_batchWise = TRUE`), users can remove the outliers in each metabolite by setting `targetVal_removeOutlier` as `TRUE`. This can weaken the impact of extreme values. If `targetVal_batchWise = FALSE`, it is generally not recommended to remove outliers, as we assume the input data have strong batch effects and contain extreme values—we hope TIGER can take these into account. Code for checking outliers is adapted from `boxplot.stats`.

Hyperparameters for variable selection

- `selectVar_external`:

  This argument accepts a list of selected variables generated by `select_variable`. This is helpful when you want to use the same selected variables to correct several datasets. You can also pass a self-defined list to this argument, as long as the self-defined list has similar data structure as the one generated by `select_variable`.

- `selectVar_corType` and `selectVar_corMethod`:

TIGER supports Pearson product-moment correlation ("pearson") and Spearman's rank correlation ("spearman") to compute correlation coefficients ("cor") or partial correlation coefficients ("por") for variable selection. See `cor` and `pcor` for further details.

- `selectVar_minNum` and `selectVar_maxNum`:

  For an objective metabolite to be corrected, the intersection of its top *t* highly-correlated metabolites calculated from training and test samples are selected to train the ensemble model. The highly-correlated metabolites are the ones with correlation coefficients greater than 0.5 (the objective metabolite itself will not be regarded as its highly-correlated metabolite). Arguments `selectVar_minNum` and `selectVar_maxNum` are used to avoid selecting too many or too few metabolites. Selecting too many metabolites can lower the process, sometimes even lower the accuracy.

- `selectVar_batchWise`:

  Advanced option designed for special cases. Setting it `TRUE` might be useful when your data have very obvious batch effects.

Hyperparameters for model construction

- `mtry_percent`, `nodesize_percent` and `...`:

  Advanced options to specify `mtry`, `nodesize` and other related arguments in `randomForest` for a customised ensemble learning architecture. See Examples.

Value

This function returns a data.frame with the same data structure as the input `test_samples`, but the metabolite values are the normalised/corrected ones. `NA` and zeros in the original `test_samples` will not be changed or normalised.

Reference

Han S. *et al.* TIGER: technical variation elimination for metabolomics data using ensemble learning architecture. *Briefings in Bioinformatics* (2022) bbab535. doi: 10.1093/bib/bbab535.

Examples

```
data(FF4_qc) # load demo dataset

# QC as training samples; QC1, QC2 and QC3 as test samples:
train_samples <- FF4_qc[FF4_qc$sampleType == "QC",]
test_samples  <- FF4_qc[FF4_qc$sampleType != "QC",]

# col_sampleID includes labels. You can assign names for different samples:
train_samples$sampleID <- "train"
test_samples$sampleID  <- "test"

# Use default setting and
# include injection order and well position into feature set:
test_norm_1 <- run_TIGER(test_samples = test_samples,
                         train_samples = train_samples,
                         col_sampleID  = "sampleID",     # input column name
                         col_sampleType = "sampleType",  # input column name
                         col_batchID = "plateID",        # input column name
                         col_order = "injectionOrder",   # input column name
                         col_position = "wellPosition",  # input column name
                         parallel.cores = 2)

# If the information of injection order and well position is not available,
# or you don't want to use them:
train_data <- train_samples[-c(4:5)]  # remove the two columns
test_data  <- test_samples[-c(4:5)]   # remove the two columns

test_norm_2 <- run_TIGER(test_samples = test_data,
                         train_samples = train_data,
                         col_sampleID  = "sampleID",
                         col_sampleType = "sampleType",
                         col_batchID = "plateID",
                         col_order = NULL,                # set NULL
                         col_position = NULL,             # set NULL
                         parallel.cores = 2)

# If use external target values and selected variables with
# customised settings:
target_val <- compute_targetVal(QC_num = train_samples[-c(1:5)],
                                sampleType = train_samples$sampleType,
                                batchID = train_samples$plateID,
                                targetVal_method = "median",
                                targetVal_batchWise = TRUE)

select_var <- select_variable(train_num = train_samples[-c(1:5)],
                              test_num = test_samples[-c(1:5)],
                              train_batchID = train_samples$plateID,
                              test_batchID = test_samples$plateID,
                              selectVar_corType = "pcor",
                              selectVar_corMethod = "spearman",
                              selectVar_minNum = 10,
                              selectVar_maxNum = 30,
                              selectVar_batchWise = TRUE)

test_norm_3 <- run_TIGER(test_samples = test_samples,
                         train_samples = train_samples,
                         col_sampleID  = "sampleID",
                         col_sampleType = "sampleType",
                         col_batchID = "plateID",
                         col_order = "injectionOrder",
                         col_position = "wellPosition",
                         targetVal_external = target_val,
                         selectVar_external = select_var,
                         parallel.cores = 2)

# The definitions of other hyperparameters correspond to
# randomForest::randomForest().
# If want to include more hyperparameters into model training,
# put hyperparameter values like this:
mtry_percent <- c(0.4, 0.8)
```

```
nodesize_percent <- c(0.4, 0.8)
replace <- c(TRUE, FALSE)
ntree <- c(100, 200, 300)

test_norm_4 <- run_TIGER(test_samples = test_data,
                         train_samples = train_data,
                         col_sampleID  = "sampleID",
                         col_sampleType = "sampleType",
                         col_batchID = "plateID",
                         mtry_percent = mtry_percent,
                         nodesize_percent = nodesize_percent,
                         replace = replace,
                         ntree = ntree,
                         parallel.cores = 2)

# test_norm_4 is corrected by the ensemble model consisted of base learners
# trained with (around) 24 different hyperparameter combinations:
expand.grid(mtry_percent, nodesize_percent, replace, ntree)

# Note: mtry and nodesize are calculated by mtry_percent and nodesize_percent,
#       duplicated hyperparameter combinations, if any, will be removed.
#       Thus, the total number of hyperparameter combinations can be less than 24.
#       This is determined by the shape of your input datasets.
```

`select_variable` *Select variables for ensemble learning architecture*

Description

This function provides an advanced option to select metabolite variables from external dataset(s). The selected variables (as a list) can be further passed to argument `selectVar_external` in function `run_TIGER` for a customised data correction.

Usage

```
select_variable(
  train_num,
  test_num = NULL,
  train_batchID = NULL,
  test_batchID = NULL,
  selectVar_corType = c("cor", "pcor"),
  selectVar_corMethod = c("spearman", "pearson"),
  selectVar_minNum = 5,
  selectVar_maxNum = 10,
  selectVar_batchWise = FALSE,
  coerce_numeric = FALSE
)
```

Arguments

`train_num`	a numeric data.frame **only** including the metabolite values of training samples (can be quality control samples). Information such as injection order or well position need to be excluded. Row: sample. Column: metabolite variable. See Examples.
`test_num`	an optional numeric data.frame including the metabolite values of test samples (can be subject samples). If provided, the column names of `test_num` should correspond to the column names of `train_num`. Row: sample. Column: metabolite variable. If `NULL`, the variables will be selected based on `train_num` only. See Examples.
`train_batchID`	`NULL` or a vector corresponding to `train_num` to specify the batch of each sample. Ignored if `selectVar_batchWise = FALSE`. See Examples.
`test_batchID`	`NULL` or a vector corresponding to `test_num` to specify the batch of each sample. Ignored if `selectVar_batchWise = FALSE`. See Examples.
`selectVar_corType`	a character string indicating correlation (`"cor"`, default) or partial correlation (`"pcor"`) is to be used. Can be abbreviated. See Details. **Note**: computing partial correlations of a large dataset can be very time-consuming.
`selectVar_corMethod`	a character string indicating which correlation coefficient is to be computed. One of `"spearman"` (default) or `"pearson"`. Can be abbreviated. See Details.
`selectVar_minNum`	an integer specifying the minimum number of the selected variables. If `NULL`, no limited, but 1 at least. See Details. Default: 5.
`selectVar_maxNum`	an integer specifying the maximum number of the selected variables. If `NULL`, no limited, but `ncol(train_num) - 1` at most. See Details. Default: 10.
`selectVar_batchWise`	(advanced) logical. Specify whether the variable selection should be performed based on each batch. Default: `FALSE`. **Note**: if `TRUE`, batch ID of each sample are required. The support of batch-wise variable selection is provided for data requiring special processing (for example, data with strong batch effects). But in most case, batch-wise variable selection is not necessary. Setting `TRUE` might make the algorithm less robust. See Details.
`coerce_numeric`	logical. If `TRUE`, values in `train_num` and `test_num` will be coerced to numeric before the computation. The columns cannot be coerced will be removed (with warnings). See Examples. Default: `FALSE`.

Details

See `run_TIGER`.

Value

If `selectVar_batchWise = FALSE`, the function returns a list of length one containing the selected variables computed on the whole dataset.

If `selectVar_batchWise = TRUE`, a list containing the selected variables computed on different batches is returned. The length of the returned list equals the number of batch specified by `test_batchID` and/or `train_batchID`.

Examples

```
data(FF4_qc) # load demo dataset

# QC as training samples; QC1, QC2 and QC3 as test samples:
train_samples <- FF4_qc[FF4_qc$sampleType == "QC",]
test_samples  <- FF4_qc[FF4_qc$sampleType != "QC",]

# Only numeric data of metabolite variables are allowed:
train_num = train_samples[-c(1:5)]
test_num  = test_samples[-c(1:5)]

# If the selection is performed on the whole dataset:
# based on training samples only:
selected_var_1 <- select_variable(train_num = train_num,
                                  test_num  = NULL,
                                  selectVar_batchWise = FALSE)

# also consider test samples:
selected_var_2 <- select_variable(train_num = train_num,
                                  test_num  = test_num,
                                  selectVar_batchWise = FALSE)

# If the selection is based on different batches:
# (In selectVar_batchWise, batch ID is required.)
selected_var_3 <- select_variable(train_num = train_num,
                                  test_num  = NULL,
                                  train_batchID = train_samples$plateID,
                                  test_batchID  = NULL,
                                  selectVar_batchWise = TRUE)

# If coerce_numeric = TRUE,
# columns cannot be coerced to numeric will be removed (with warnings):
# (In this example, columns of injection order and well position are excluded.
# Because we don't want to calculate the correlations between metabolites and
# injection order/well position.)
selected_var_4 <- select_variable(train_num = train_samples[-c(4,5)],
                                  train_batchID = train_samples$plateID,
                                  selectVar_batchWise = TRUE,
                                  coerce_numeric = TRUE)
identical(selected_var_3, selected_var_4)  # identical to selected_var_3

## Not run:

# will throw errors if input data have non-numeric columns
# and coerce_numeric = FALSE:

selected_var_5 <- select_variable(train_num = train_samples[-c(4,5)],
                                  coerce_numeric = FALSE)
```

```
## End(Not run)
```

Index