Package ‘notame’

October 11, 2025

Type Package

Title Workflow for non-targeted LC-MS metabolic profiling

Version 0.99.6

Description Provides functionality for untargeted LC-MS metabolomics research as specified in the associated protocol article in the 'Metabolomics Data Processing and Data Analysis—Current Best Practices' special issue of the Metabolites journal (2020). This includes tabular data preprocessing and quality control, uni- and multivariate analysis as well as quality control visualizations, feature-wise visualizations and results visualizations. Raw data preprocessing and functionality related to biological context, such as pathway analysis, is not included.

License MIT + file LICENSE

Encoding UTF-8

biocViews BiomedicalInformatics, Metabolomics, DataImport, MassSpectrometry, BatchEffect, MultipleComparison, Normalization, QualityControl, Visualization, Preprocessing

Depends R (>= 4.5.0), ggplot2, SummarizedExperiment

Imports BiocGenerics, BiocParallel, dplyr, futile.logger, methods, openxlsx, S4Vectors, scales, stringr, tidyr, utils

Suggests BiocStyle, fpc, igraph, knitr, missForest, notameViz, notameStats, pcaMethods, RUVSeq, testthat

URL `https://github.com/hanhineva-lab/notame`, `https://hanhineva-lab.github.io/notame/`

BugReports `https://github.com/hanhineva-lab/notame/issues`

RoxygenNote 7.3.3

VignetteBuilder knitr

Config/testthat/parallel true

git_url https://git.bioconductor.org/packages/notame

git_branch devel

git_last_commit 36de686

git_last_commit_date 2025-10-03

Repository Bioconductor 3.22

Date/Publication 2025-10-10

Author Anton Klåvus [aut, cph] (ORCID: <https://orcid.org/0000-0003-2612-0230>),
Jussi Paananen [aut, cph] (ORCID: <https://orcid.org/0000-0001-5100-4907>),
Oskari Timonen [aut, cph] (ORCID: <https://orcid.org/0000-0002-6317-6260>),
Atte Lihtamo [aut],
Vilhelm Suksi [aut, cre] (ORCID: <https://orcid.org/0009-0005-1108-518X>),
Retu Haikonen [aut] (ORCID: <https://orcid.org/0000-0003-0830-3850>),
Leo Lahti [aut] (ORCID: <https://orcid.org/0000-0001-5537-637X>),
Kati Hanhineva [aut] (ORCID: <https://orcid.org/0000-0001-6834-7375>),
Ville Koistinen [ctb] (ORCID: <https://orcid.org/0000-0003-1587-8361>),
Olli Kärkkäinen [ctb] (ORCID: <https://orcid.org/0000-0003-0825-4956>),
Artur Sannikov [ctb]

Maintainer Vilhelm Suksi <vksuks@utu.fi>

Contents

Index **32**

`notame-package` `notame` *package.*

Description

Provides tabular data preprocessing and data wrangling functionality for untargeted LC-MS metabolomics research.

Author(s)

Maintainer: Vilhelm Suksi `<vksuks@utu.fi>` (ORCID)

Authors:

- Anton Klåvus (ORCID) [copyright holder]
- Jussi Paananen (ORCID) [copyright holder]
- Oskari Timonen (ORCID) [copyright holder]
- Atte Lihtamo
- Retu Haikonen (ORCID)
- Leo Lahti (ORCID)
- Kati Hanhineva (ORCID)

Other contributors:

- Ville Koistinen (ORCID) [contributor]
- Olli Kärkkäinen (ORCID) [contributor]
- Artur Sannikov [contributor]

References

Klåvus et al. (2020). "notame": Workflow for Non-Targeted LC-MS Metabolic Profiling. Metabolites, 10: 135.

See Also

Useful links:

- `https://github.com/hanhineva-lab/notame`
- `https://hanhineva-lab.github.io/notame/`
- Report bugs at `https://github.com/hanhineva-lab/notame/issues`

`assess_quality` *Assess quality information of features*

Description

Assess features using the quality metrics defined in (Broadhurst 2018). The quality metrics are described in Details section of `flag_quality`

Usage

```
assess_quality(object, assay.type = NULL)
```

Arguments

`object`	a `SummarizedExperiment` object
`assay.type`	character, assay to be used in case of multiple assays

Value

A SummarizedExperiment object with quality metrics in feature data.

Examples

```
data(toy_notame_set)
ex_set <- assess_quality(toy_notame_set)
rowData(ex_set)
```

`citations` *Show citations*

Description

This function lists citations behind the notame functions that have been called during the session. All notame functions update the list automatically. The citations are taken from the call to '`citation("package")`, and complemented with a brief description of what the package was used for. NOTE: the citations might not point to the correct paper if the package authors have not supplied correct citation information for their package. The output is written to the current log file, if specified.

Usage

```
citations()
```

Value

None, the function is invoked for its side effect.

Examples

```
citations()
data(toy_notame_set)
ex_set <- flag_quality(toy_notame_set)
# Broadhurst et al.(2018) added to citations
citations()
```

`cluster_features` *Cluster correlated features originating from the same metabolite*

Description

Clusters features potentially originating from the same compound. Features with high Pearson correlation coefficient and small retention time difference are linked together. Then clusters are formed by setting a threshold for the relative degree that each node in a cluster needs to fulfil. Each cluster is named after the feature with the highest median peak area (median abundance). This is a wrapper around numerous functions that are based on the MATLAB code by David Broadhurst.

Usage

```
cluster_features(
  object,
  mz_col = NULL,
  rt_col = NULL,
  all_features = FALSE,
  rt_window = 1/60,
  corr_thresh = 0.9,
  d_thresh = 0.8,
  assay.type = NULL
)
```

Arguments

`object`	a `SummarizedExperiment` object
`mz_col`	the column name in feature data that holds mass-to-charge ratios
`rt_col`	the column name in feature data that holds retention times
`all_features`	logical, should all features be included in the clustering? If FALSE, as the default, flagged features are not included in clustering
`rt_window`	the retention time window for potential links NOTE: use the same unit as the retention time
`corr_thresh`	the correlation threshold required for potential links between features
`d_thresh`	the threshold for the relative degree required by each node
`assay.type`	character, assay to be used in case of multiple assays

Value

a SummarizedExperiment object, with median peak area (MPA), the cluster ID, the features in the cluster, and cluster size added to feature data.

Examples

```
data(toy_notame_set)
# The parameters are really weird because example data is imaginary
clustered <- cluster_features(toy_notame_set, rt_window = 1,
  corr_thresh = 0.5, d_thresh = 0.6)
```

`combined_data` *Retrieve both sample information and features*

Description

Retrieve both sample information and features

Usage

```
combined_data(object, assay.type = NULL)
```

Arguments

`object`	a `SummarizedExperiment` object
`assay.type`	character, assay to be used in case of multiple assays

Value

A data frame with sample information plus all features as columns, one row per sample.

Examples

```
data(toy_notame_set)
combined_data(toy_notame_set)
```

`compress_clusters` *Compress clusters of features to a single feature*

Description

This function compresses clusters found by cluster_features, keeping only the feature with the highest median peak area. The features that were discarded are recorded in feature data, under Cluster_features.

Usage

```
compress_clusters(object)
```

Arguments

`object`	a `SummarizedExperiment` object

Value

A SummarizedExperiment object with only one feature per cluster.

See Also

`cluster_features`

Examples

```
data(toy_notame_set)
clustered <- cluster_features(toy_notame_set,
  rt_window = 1, corr_thresh = 0.5, d_thresh = 0.6)
compressed <- compress_clusters(clustered)
```

`correct_drift` *Correct drift using cubic spline*

Description

A wrapper function for applying cubic spline drift correction and saving before and after plots.

Usage

```
correct_drift(
  object,
  log_transform = TRUE,
  spar = NULL,
  spar_lower = 0.5,
  spar_upper = 1.5,
  check_quality = FALSE,
  condition = "RSD_r < 0 & D_ratio_r < 0",
  file = NULL,
  width = 16,
  height = 8,
  color = "QC",
  shape = color,
  color_scale = getOption("notame.color_scale_dis"),
  shape_scale = scale_shape_manual(values = c(15, 16)),
  assay.type = NULL,
  name = NULL
)
```

Arguments

`object`	a `SummarizedExperiment` object
`log_transform`	logical, should drift correction be done on log-transformed values? See Details
`spar`	smoothing parameter as in `smooth.spline`
`spar_lower`, `spar_upper`	lower and upper limits for the smoothing parameter

`check_quality`	logical, whether quality should be monitored.
`condition`	a character specifying the condition used to decide whether drift correction works adequately, see Details
`file`	path to the PDF file where the plots should be saved
`width`, `height`	width and height of the plots in inches
`color`	character, name of the column used for coloring the points
`shape`	character, name of the column used for shape
`color_scale`, `shape_scale`	the color and shape scales as returned by a ggplot function
`assay.type`	character, assay to be used in case of multiple assays
`name`	character, name of the resultant assay

Details

If `log_transform` = `TRUE`, the correction will be done on log-transformed values. The correction formula depends on whether the correction is run on original values or log-transformed values. In log-space: $corrected = original + meanofQCs - predictionbycubicspline$. In original space: $corrected = original * predictionforfirstQC/predictionforcurrentpoint$. We recommend doing the correction in the log-space since the log-transfomred data better follows the assumptions of cubic spline regression. The drift correction in the original space also sometimes results in negative values, and results in rejection of the drift corrrection procedure. If `check_quality` = `TRUE`, the `condition` parameter should be a character giving a condition compatible with `filter`. The condition is applied on the **changes** in the quality metrics RSD, RSD_r, D_ratio and D_ratio_r. For example, the default is "RSD_r < 0 and D_ratio_r < 0", meaning that both RSD_r and D_ratio_r need to decrease in the drift correction, otherwise the drift corrected feature is discarded and the original is retained. By default, the column used for color is also used for shape.

Value

A SummarizedExperiment object as the one supplied, with drift corrected features.

See Also

`smooth.spline` for details about the regression

Examples

```
data(toy_notame_set)
corrected <- correct_drift(mark_nas(toy_notame_set[1:5, ], value = 0))
```

`drop_flagged` *Drop flagged features*

Description

Removes all features that have been flagged by quality control functions. Only features that do not have a flag (Flag == NA) are retained.

Usage

```
drop_flagged(object, all_features = FALSE)
```

Arguments

`object`	a `SummarizedExperiment` object
`all_features`	logical, should all features be retained? Mainly used by internal functions

Value

A SummarizedExperiment object without the previously flagged features.

Examples

```
data(toy_notame_set)
dim(toy_notame_set)
flagged <- flag_quality(toy_notame_set)
noflags <- drop_flagged(flagged)
dim(noflags)
```

`drop_qcs` *Drop QC samples*

Description

Drop QC samples

Usage

```
drop_qcs(object)
```

Arguments

`object`	a `SummarizedExperiment` object

Value

A SummarizedExperiment object as the one supplied, without QC samples.

Examples

```
data(toy_notame_set)
dim(toy_notame_set)
noqc <- drop_qcs(toy_notame_set)
dim(noqc)
```

`finish_log` *Finish a log*

Description

Logs the current date and time and session info, and switches logging off.

Usage

```
finish_log()
```

Value

None, the function is invoked for its side effect.

See Also

`init_log`, `log_text`

Examples

```
finish_log()
```

`fix_MSMS` *Transform the MS/MS output to publication ready*

Description

Change the MS/MS output from MS-DIAL format to publication-ready format. Original spectra is sorted according to abundance percentage and clarified. See the example below.

Usage

```
fix_MSMS(
  object,
  ms_ms_spectrum_col = "MS_MS_spectrum",
  peak_num = 10,
  min_abund = 5,
  deci_num = 3
)
```

Arguments

`object`	a `SummarizedExperiment` object
`ms_ms_spectrum_col`	name of column with original MS/MS spectra
`peak_num`	maximum number of peak that is kept (Recommended: 4-10)
`min_abund`	minimum relative abundance to be kept (Recommended: 1-5)
`deci_num`	maximum number of decimals to m/z value (Recommended: >2)

Details

Original MS/MS spectra from MS-DIAL: m/z:Raw Abundance

23.193:254 26.13899:5 27.50986:25 55.01603:82 70.1914:16 73.03017:941 73.07685:13 73.13951:120

Spectra after transformation: m/z (Abundance)

73.03 (100), 23.193 (27), 73.14 (12.8), 55.016 (8.7)

Value

A SummarizedExperiment object as the one supplied, with publication-ready MS/MS peak information.

Examples

```
data(toy_notame_set)
# Spectra before fixing
ex_set <- toy_notame_set
rowData(ex_set)$MS_MS_spectrum <- NA
rowData(ex_set)[1, ]$MS_MS_spectrum <-
  "28.769:53 44.933:42 52.106:89 69.518:140"
rowData(ex_set)$MS_MS_spectrum[
  !is.na(rowData(ex_set)$MS_MS_spectrum)]
# Fixing spectra with default settings
fixed_MSMS_peaks <- fix_MSMS(ex_set)
# Spectra after fixing
rowData(fixed_MSMS_peaks)$MS_MS_Spectrum_clean[
  !is.na(rowData(fixed_MSMS_peaks)$MS_MS_Spectrum_clean)]
```

`fix_object` *Fix object for functioning of notame*

Description

Attempts to create missing columns needed for notame in pheno and feature data. Optionally cleans the object and splits the object by mode.

Usage

```
fix_object(
  object,
  id_prefix = "ID_",
  id_column = NULL,
  split_by = NULL,
  name = NULL,
  clean = TRUE,
  split_data = FALSE,
  assay.type = NULL
)
```

Arguments

`object`	a `SummarizedExperiment` object
`id_prefix`	character, prefix for autogenerated sample IDs, see Details
`id_column`	character, column name for unique identification of samples
`split_by`	character vector, in the case where all the modes are in the same object, the column names of feature data used to separate the modes (usually Mode and Column)
`name`	in the case where object only contains one mode, the name of the mode, such as "Hilic_neg"
`clean`	boolean, whether to select best classes, reorder columns and consistently rename columns in pheno and feature
`split_data`	logical, whether to split data by analytical mode recorded in the "Split" column of feature data. If TRUE (the default), will return a list of objects, one per analytical mode. If FALSE, will return a single object.
`assay.type`	character, assay to be used in case of multiple assays

Details

Only specify one of `split_by` and `name`. The feature data will contain columns named "Split", used to separate features from different modes, and "Flag" for recording flagged features. Unless a column named "Feature_ID" is found in feature data, a feature ID will be generated based on the value of "Split", mass and retention time. The function will try to find columns for mass and retention time by looking at a few common alternatives, and throw an error if no matching column is found. Sample information needs to contain a row called "Injection_order", and the values need to be unique. In addition, a possible sample identifier row needs to be named "Sample_ID", or to be specified in `id_column`, and the values need to be unique, with an exception of QC samples: if there are any "QC" identifiers, they will be replaced with "QC_1", "QC_2" and so on. If a "Sample_ID" column is not found, it will be created using the `id_prefix` and injection order or by renaming `id_column`.

Value

A new SummarizedExperiment object with a single peak table. If split_data = TRUE, a list containing separate objects for analytical modes.

Examples

```
data(toy_notame_set)
ex_set <- toy_notame_set
rowData(ex_set)$Flag <- NULL
fixed <- fix_object(ex_set)
```

`flag` *Get and set the values in the flag column*

Description

Get and set the values in the flag column

Usage

```
flag(object)

flag(object) <- value
```

Arguments

`object`	a `SummarizedExperiment` object
`value`	character vector, values for flag column

Value

Character vector of feature flags.

For the endomorphism, an object with a modified flag column.

Examples

```
data(toy_notame_set)
# Get values in flag column of rowData
flag(toy_notame_set)

data(toy_notame_set)
# Flag a suspicious feature manually
flag(toy_notame_set)[1] <- "Contaminant, known from experience"
```

`flag_contaminants` *Flag contaminants based on blanks*

Description

Flags contaminant features by comparing the median values of blanks and biological samples. Biological sampels are defined as samples that are not marked as blanks and are not QCs. If the median of blanks > the median of biological samples times a set ratio, the feature is flagged as contaminant.

Usage

```
flag_contaminants(
  object,
  blank_col,
  blank_label,
  flag_thresh = 0.05,
  flag_label = "Contaminant",
  assay.type = NULL
)
```

Arguments

`object`	a `SummarizedExperiment` object
`blank_col`	character, the column name in pheno data with blank labels
`blank_label`	character, the label for blank samples in blank_col
`flag_thresh`	numeric, the ratio threshold for flagging contaminants. If the median of blanks > flag_thresh * median of biological samples, the feature gets flagged.
`flag_label`	character, the label used when flagging contaminants. Can be changed if sample processing contaminants and carryover contaminants are flagged separately.
`assay.type`	character, assay to be used in case of multiple assays

Value

A SummarizedExperiment object with contaminant features flagged.

Examples

```
data(toy_notame_set)
# Make a blank sample which has one (first) feature exceeding the threshold
## Abundance matrix
med <- median(assay(toy_notame_set)[1, toy_notame_set$QC != "QC"])
assay <- matrix(c(med * 0.05 + 1, rep(0, 79)), ncol = 1, nrow = 80,
                  dimnames = list(NULL, "Demo_51"))
assay <- cbind(assay(toy_notame_set), assay)
## Sample metadata
pheno_data <- colData(toy_notame_set)[1, ]
rownames(pheno_data) <- "Demo_51"
pheno_data$Sample_ID <- "Demo_51"
pheno_data$Injection_order <- 51
pheno_data[c("Subject_ID", "Group", "QC", "Time")] <- "Blank"
pheno_data <- rbind(colData(toy_notame_set), pheno_data)
## Feature metadata
feature_data <- rowData(toy_notame_set)

# Construct SummarizedExperiment object with blank sample
ex_set <- SummarizedExperiment(assays = assay,
                               colData = pheno_data,
                               rowData = feature_data)
# Flag contaminant(s)
contaminants_flagged <- flag_contaminants(ex_set, blank_col = "QC",
                                          blank_label = "Blank")
```

`flag_detection` *Flag features with low detection rate*

Description

Flags features with too high amount of missing values. There are two detection rate limits, both defined as the minimum proportion of samples that need to have a value (not NA) for the feature to be kept. '`qc_limit` is the detection rate limit for QC samples, '`group_limit` is the detection rate limit for the actual study groups. If the group limit is passed for AT LEAST ONE GROUP, then the feature is kept. Features with low detection rate in QCs are flagged as "Low_qc_detection", while low detection rate in the study groups is flagged as "Low_group_detection". The detection rates for all the groups are recorded in feature data.

Usage

```
flag_detection(
  object,
  qc_limit = 0.7,
  group_limit = 0.5,
  group = NULL,
  assay.type = NULL
)
```

Arguments

`object`	a `SummarizedExperiment` object
`qc_limit`	the detection rate limit for QC samples
`group_limit`	the detection rate limit for study groups
`group`	the columns name in sample information to use as the grouping variable
`assay.type`	character, assay to be used in case of multiple assays

Value

A SummarizedExperiment object with the features flagged.

Examples

```
data(toy_notame_set)
ex_set <- mark_nas(toy_notame_set, value = 0)
ex_set <- flag_detection(ex_set, group = "Group")
rowData(ex_set)
```

`flag_quality` *Flag low-quality features*

Description

Flags low-quality features using the quality metrics defined in (Broadhurst 2018). The metrics are described in more detain in Details. A condition for keeping the features is given as a character, which is passed to `filter`.

Usage

```
flag_quality
flag_quality(object, assay.type = NULL, condition =
  "(RSD_r < 0.2 & D_ratio_r < 0.4) |
  (RSD < 0.1 & RSD_r < 0.1 & D_ratio < 0.1)")
```

Arguments

`object`	a `SummarizedExperiment` object
`assay.type`	character, assay to be used in case of multiple assays
`condition`	character, condition for keeping the features, see Details

Details

The quality metrics measure two things: internal spread of the QCs, and spread of the QCs compared to the spread of the biological samples. Internal spread is measured with relative standard deviation (RSD), also known as coefficient of variation (CV).

$$RSD = sd(QC)/mean(QC)$$

Where $sd(QC)$ is the standard deviation of the QC samples and '$mean(QC)$ is the sample mean of the signal in the QC samples. RSD can also be replaced by a non-parametric, robust version based on the median and median absolute deviation (MAD):

$$RSD_r = 1.4826 * MAD(QC)/median(QC)$$

The spread of the QC samples compared to the biological samples is measured using a metric called D-ratio:

$$D_ratio = sd(QC)/sd(biological)$$

Or, as before, a non-parametric, robust alternative:

$$D_ratio_r = MAD(QC)/MAD(biolofical)$$

The default condition keeps features that pass either of the two following conditions:

$$RSD_r < 0.2 \& D_ratio_r < 0.4$$

$$RSD < 0.1 \& RSD_r < 0.1 \& D_ratio < 0.1$$

Value

a SummarizedExperiment object with the features flagged.

References

Broadhurst, David et al. Guidelines and considerations for the use of system suitability and quality control samples in mass spectrometry assays applied in untargeted clinical metabolomic studies. Metabolomics : Official journal of the Metabolomic Society vol. 14,6 (2018): 72. doi:10.1007/s11306-018-1367-3

Examples

```
data(toy_notame_set)
ex_set <- flag_quality(toy_notame_set)
rowData(ex_set)
# Custom condition
ex_set <- flag_quality(toy_notame_set,
  condition = "RSD_r < 0.3 & D_ratio_r < 0.6")
rowData(ex_set)
```

`flag_report` *A report of flagged features*

Description

Computes the number of features at each stage of flagging for each mode.

Usage

```
flag_report(object)
```

Arguments

`object`	a `SummarizedExperiment` object

Value

A data frame with the number of features at each stage of flagging.

Examples

```
data(toy_notame_set)
flagged <- toy_notame_set |>
  mark_nas(0) |>
  flag_detection(group = "Group") |>
  flag_quality()
flag_report(flagged)
```

`import_from_excel` *Read formatted Excel files*

Description

Reads data from an Excel file of the following format:

- Left side of the sheet contains information about the features, size features x feature info columns
- Top part contains sample information, size sample info variables x samples
- The middle contains the actual abundances, size features x samples

This function separates the three parts from the file, and returns them in a list.

Usage

```
import_from_excel(
  file,
  sheet = 1,
  id_column = NULL,
  corner_row = NULL,
  corner_column = NULL,
  id_prefix = "ID_",
  split_by = NULL,
  name = NULL,
  mz_limits = c(10, 2000),
  rt_limits = c(0, 20),
  skip_checks = FALSE
)
```

Arguments

`file`	path to the Excel file
`sheet`	the sheet number or name
`id_column`	character, column name for unique identification of samples
`corner_row`	integer, the bottom row of sample information, usually contains data file names and feature info column names. If set to NULL, will be detected automatically.
`corner_column`	integer or character, the corresponding column number or the column name (letter) in Excel. If set to NULL, will be detected automatically.
`id_prefix`	character, prefix for autogenerated sample IDs, see Details
`split_by`	character vector, in the case where all the modes are in the same Excel file, the column names of feature data used to separate the modes (usually Mode and Column)
`name`	in the case where the Excel file only contains one mode, the name of the mode, such as "Hilic_neg"
`mz_limits`	numeric vector of two, all m/z values should be in between these
`rt_limits`	numeric vector of two, all retention time values should be in between these
`skip_checks`	logical: skip checking and fixing of data integrity. Not recommended, but sometimes useful when you just want to read the data in as is and fix errors later. The data integrity checks are important for functioning of notame.

Details

If skip_checks = FALSE, `import_from_excel` attempts to modify the data as per `fix_object` and checks the data. If skip_checks = TRUE, parameters for `fix_object` are ignored.

Value

A list of three data frames:

- assay: the actual abundances, size features x samples
- pheno_data: sample information, size sample info variables x samples
- feature_data: information about the features, size features x feature info columns

Examples

```
data <- import_from_excel(
  file = system.file("extdata", "toy_notame_set.xlsx",
  package = "notame"), sheet = 1, corner_row = 11, corner_column = "H",
  split_by = c("Column", "Ion_mode"))
```

`impute_rf` *Impute missing values using random forest*

Description

Impute the missing values in the peak table of the object using a random forest. The estimated error in the imputation is logged. It is recommended to set the seed number for reproducibility (it is called random forest for a reason). This a wrapper around `missForest`. Use parallelize = "variables" to run in parallel for faster testing. NOTE: running in parallel prevents user from setting a seed number.

Usage

```
impute_rf(object, all_features = FALSE, assay.type = NULL, name = NULL, ...)
```

Arguments

`object`	a `SummarizedExperiment` object
`all_features`	logical, should all features be used? If FALSE (the default), flagged features are removed before imputation.
`assay.type`	character, assay to be used in case of multiple assays
`name`	character, name of the resultant assay in case of multiple assays
`...`	passed to `missForest`

Value

An object as the one supplied, with missing values imputed.

See Also

`missForest` for detail about the algorithm and the parameters

Examples

```
data(toy_notame_set)
missing <- mark_nas(toy_notame_set, 0)
set.seed(38)
imputed <- impute_rf(missing)
```

`impute_simple` *Simple imputation*

Description

Impute missing values using a simple imputation strategy. All missing values of a feature are imputed with the same value. It is possible to only impute features with a large number of missing values this way. This can be useful for using this function before random forest imputation to speed things up. The imputation strategies available are:

- a numeric value: impute all missing values in all features with the same value, e.g. 1
- "mean": impute missing values of a feature with the mean of observed values of that feature
- "median": impute missing values of a feature with the median of observed values of that feature
- "min": impute missing values of a feature with the minimum observed value of that feature
- "half_min": impute missing values of a feature with half the minimum observed value of that feature
- "small_random": impute missing values of a feature with random numbers between 0 and the minimum of that feature (uniform distribution, remember to set the seed number!).

Usage

```
impute_simple(object, value, na_limit = 0, assay.type = NULL, name = NULL)
```

Arguments

`object`	a `SummarizedExperiment` object
`value`	the value used for imputation, either a numeric or one of '"min", "half_min", "small_random", see above
`na_limit`	only impute features with the proportion of NAs over this limit. For example, if `na_limit = 0.5`, only features with at least half of the values missing are imputed.
`assay.type`	character, assay to be used in case of multiple assays
`name`	character, name of the resultant assay in case of multiple assays

Value

A SummarizedExperiment object with imputed peak table.

Examples

```
data(toy_notame_set)
missing <- mark_nas(toy_notame_set, 0)
imputed <- impute_simple(missing, value = "min")
```

`init_log` *Initialize log to a file*

Description

Initialize a log file with the current data and time. All major operations run after this will be logged to the specified file.

Usage

```
init_log(log_file)
```

Arguments

`log_file`	Path to the log file

Value

None, the function is invoked for its side effect.

See Also

`log_text`, `finish_log`

Examples

```
file_name <- "log.txt"
init_log(file_name)
# Print the contents of the file
scan(file_name, sep = "\n", what = "character")
```

`inverse_normalize` *Inverse-rank normalization*

Description

Applies inverse rank normalization to all features to approximate a normal distribution.

Usage

```
inverse_normalize(object, assay.type = NULL, name = NULL)
```

Arguments

`object`	a `SummarizedExperiment` object
`assay.type`	character, assay to be used in case of multiple assays
`name`	character, name of the resultant assay in case of multiple assays

Value

An object as the one supplied, with normalized features.

Examples

```
data(toy_notame_set)
normalized <- inverse_normalize(toy_notame_set)
```

`join_colData` *Join new columns to pheno data*

Description

Join a new data frame of information to pheno data of a SummarizedExperiment object.

Usage

```
join_colData(object, dframe)
```

Arguments

`object`	a `SummarizedExperiment` object
`dframe`	a data frame with the new information

Value

A SummarizedExperiment object with the new information added to colData(object).

Examples

```
data(toy_notame_set)
new_info <- data.frame(
  Sample_ID = colnames(toy_notame_set),
  BMI = stats::runif(ncol(toy_notame_set), 22, 26)
)
with_new_info <- join_colData(toy_notame_set, new_info)
colnames(colData(with_new_info))
```

`join_rowData` *Join new columns to feature data*

Description

Join a new data frame of information to feature data of a SummarizedExperiment object. The data frame needs to have a column "Feature_ID". This function is usually used internally by some of the functions in the package, but can be useful.

Usage

```
join_rowData(object, dframe)
```

Arguments

`object`	a `SummarizedExperiment`
`dframe`	a data frame with the new information

Value

A SummarizedExperiment object with the new information added to rowData(object).

Examples

```
data(toy_notame_set)
new_info <- data.frame(
  Feature_ID = rownames(toy_notame_set),
  Feature_number = seq_len(nrow(toy_notame_set))
)
with_new_info <- join_rowData(toy_notame_set, new_info)
colnames(rowData(with_new_info))
```

`log_text` *Log text to the current log file*

Description

The specified text is printed and written to the current log file. Does not overwrite the file. Also used internally by many functions in the package.

Usage

```
log_text(text)
```

Arguments

`text`	The text to be logged

Value

None, the function is invoked for its side effect.

See Also

`init_log`, `finish_log`

Examples

```
file_name <- "log.txt"
init_log(file_name)
log_text("Hello World!")
# Print the contents of the file
scan(file_name, sep = "\n", what = "character")
```

`mark_nas` *Mark specified values as missing*

Description

Replaces all values in the peak table that equal the specified value with NA. For example, vendor software might use 0 or 1 to signal a missing value, which is not understood by R.

Usage

```
mark_nas(object, value, assay.type = NULL, name = NULL)
```

Arguments

`object`	a `SummarizedExperiment` object
`value`	the value to be converted to NA
`assay.type`	character, assay to be used in case of multiple assays
`name`	character, name of the resultant assay in case of multiple assays

Value

SummarizedExperiment object as the one supplied, with missing values correctly set to NA.

Examples

```
data(toy_notame_set)
nas_marked <- mark_nas(toy_notame_set, value = 0)
```

`merge_notame_sets` *Merge SummarizedExperiment objects together*

Description

Merges two or more SummarizedExperiment objects together. Can be used to merge analytical modes or batches.

Usage

```
merge_notame_sets(..., merge = c("features", "samples"), assay.type = NULL)
```

Arguments

`...`	`SummarizedExperiment` objects or a list of objects
`merge`	what to merge? features is used for combining analytical modes, samples is used for batches
`assay.type`	character, assay to be used in case of multiple assays. The same assay needs to be present in all objects to be merged, and the resultant object contains this single assay.

Details

When merging samples, sample IDs that begin with "QC" or "Ref" are combined so that they have running numbers on them. This means that if both batches have samples called "QC_1", this will not result in an error, but the sample IDs will be adjusted so that they are unique.

Value

A merged SummarizedExperiment object.

Examples

```
# Merge analytical modes
data(hilic_neg_sample, hilic_pos_sample, rp_neg_sample, rp_pos_sample)
merged <- merge_notame_sets(
  hilic_neg_sample, hilic_pos_sample,
  rp_neg_sample, rp_pos_sample
)
# Merge batches
batch1 <- toy_notame_set[, toy_notame_set$Batch == 1]
batch2 <- toy_notame_set[, toy_notame_set$Batch == 2]
merged <- merge_notame_sets(batch1, batch2, merge = "samples")
```

`pca_bhattacharyya_dist`

Bhattacharyya distance between batches in PCA space

Description

Computes Bhattacharyya distance between all pairs of batches using `fpc:bhattacharyya.matrix` after projecting the samples into PCA space with `pca`.

Usage

```
pca_bhattacharyya_dist(
  object,
  batch,
  all_features = FALSE,
  center = TRUE,
  scale = "uv",
  nPcs = 3,
  assay.type = NULL,
  ...
)
```

Arguments

`object`	a `SummarizedExperiment` object
`batch`	column name of pheno data giving the batch labels
`all_features`	logical, should all features be used? If FALSE (the default), flagged features are removed before imputation.
`center`	logical, should the data be centered prior to PCA? (usually yes)
`scale`	scaling used, as in `prep`. Default is "uv" for unit variance
`nPcs`	the number of principal components to use
`assay.type`	character, assay to be used in case of multiple assays
`...`	other parameters to `pca`

Value

A matrix of Bhattacharyya distances between batches.

Examples

```
data(toy_notame_set)
# Batch correction
replicates <- list(which(toy_notame_set$QC == "QC"))
batch_corrected <- ruvs_qc(toy_notame_set, replicates = replicates)
# Evaluate batch correction
pca_bhattacharyya_dist(toy_notame_set, batch = "Batch")
pca_bhattacharyya_dist(batch_corrected, batch = "Batch")
```

`perform_repeatability` *Compute repeatability measures*

Description

Computes repeatability for each feature with the following formula:

$$\frac{\sigma^2_{between}}{\sigma^2_{between} + \sigma^2_{within}}$$

The repeatability ranges from 0 to 1. Higher repeatability depicts less variation between batches.

Usage

```
perform_repeatability(object, group, assay.type = NULL)
```

Arguments

`object`	a `SummarizedExperiment` object
`group`	column name of pheno data giving the group labels
`assay.type`	character, assay to be used in case of multiple assays

Value

A data frame with one row per feature with the repeatability measure.

Examples

```
data(toy_notame_set)
# Batch correction
replicates <- list(which(toy_notame_set$QC == "QC"))
batch_corrected <- ruvs_qc(toy_notame_set, replicates = replicates)
# Evaluate batch correction
rep_orig <- perform_repeatability(toy_notame_set, group = "Group")
mean(rep_orig$Repeatability, na.rm = TRUE)
rep_corr <- perform_repeatability(batch_corrected, group = "Group")
mean(rep_corr$Repeatability, na.rm = TRUE)
```

`pqn_normalization` *Probabilistic quotient normalization*

Description

Apply probabilistic quotient normalization (PQN) to the peak table of a SummarizedExperiment object. By default, reference is calculated from high-quality QC samples and the median of the reference is used for normalization. Check parameters for more options.

Usage

```
pqn_normalization(
  object,
  ref = c("qc", "all"),
  method = c("median", "mean"),
  all_features = FALSE,
  assay.type = NULL,
  name = NULL
)
```

Arguments

`object`	a `SummarizedExperiment` object
`ref`	character, the type of reference samples to use for normalization.
`method`	character, the method to use for calculating the reference sample.
`all_features`	logical, should all features be used for calculating the reference sample?
`assay.type`	character, assay to be used in case of multiple assays
`name`	character, name of the resultant assay in case of multiple assays

Value

A SummarizedExperiment object with altered feature abundances.

Examples

```
data(toy_notame_set)
pqn_set <- pqn_normalization(toy_notame_set)
```

`quality` *Extract quality information of features*

Description

Extract quality information of features

Usage

```
quality(object)
```

Arguments

`object`	a `SummarizedExperiment` object

Value

A data frame with quality metrics for each feature.

Examples

```
data(toy_notame_set)
ex_set <- assess_quality(toy_notame_set)
quality(ex_set)
```

`ruvs_qc` *Remove Unwanted Variation (RUV) between batches*

Description

An interface for `link[RUVSeq]{RUVs}` method

Usage

```
ruvs_qc(object, replicates, k = 3, assay.type = NULL, name = NULL, ...)
```

Arguments

`object`	a `SummarizedExperiment` object
`replicates`	list of numeric vectors, indexes of replicates
`k`	The number of factors of unwanted variation to be estimated from the data.
`assay.type`	character, assay to be used in case of multiple assays
`name`	character, name of the resultant assay in case of multiple assays
`...`	other parameters passed to `link[RUVSeq]{RUVs}`

Value

A SummarizedExperiment object with the normalized data.

Examples

```
data(toy_notame_set)
# Batch correction
replicates <- list(which(toy_notame_set$QC == "QC"))
batch_corrected <- ruvs_qc(toy_notame_set, replicates = replicates)
# Evaluate batch correction
pca_bhattacharyya_dist(toy_notame_set, batch = "Batch")
pca_bhattacharyya_dist(batch_corrected, batch = "Batch")
```

`toy_notame_set` *Toy data set*

Description

Contains imaginary data used in testing the package functions. The dataset has 50 samples and 80 features. This dataset includes multiple observations from same subjects, sampled at two timepoints in separate batches and divided to two groups. The analytical modes are also available as separate SummarizedExperiment objects. Note that across batches, the features don't have different feature ID's, m/z and retention time as would be the case with real-world data. In essence, the example data reflects that features were aligned perfectly between batches.

Usage

```
toy_notame_set

hilic_neg_sample

hilic_pos_sample

rp_neg_sample

rp_pos_sample
```

Format

An object of class `SummarizedExperiment` with 80 rows and 50 columns.

An object of class `SummarizedExperiment` with 20 rows and 50 columns.

An object of class `SummarizedExperiment` with 20 rows and 50 columns.

An object of class `SummarizedExperiment` with 20 rows and 50 columns.

An object of class `SummarizedExperiment` with 20 rows and 50 columns.

`write_to_excel` *Write results to Excel file*

Description

Writes all the data in a SummarizedExperiment object to an Excel spreadsheet. The format is similar to the one used to read data in, except for the fact that EVERYTHING NEEDS TO BE WRITTEN AS TEXT. To fix numeric values in Excel, choose any cell with a number, press Ctrl + A, then go to the dropdown menu in upper left corner and choose "Convert to Number". This will fix the file, but can take quite a while.

Usage

```
write_to_excel(object, file, ...)
```

Arguments

`object`	a `SummarizedExperiment` object
`file`	path to the file to write
`...`	Additional parameters passed to `write.xlsx`

Value

None, the function is invoked for its side effect.

Examples

```
data(toy_notame_set)
write_to_excel(toy_notame_set, file = "toy_notame_set.xlsx")
```

Index