Package 'scp'

March 26, 2024

Title Mass Spectrometry-Based Single-Cell Proteomics Data Analysis

Version 1.12.0

Description Utility functions for manipulating, processing, and analyzing mass spectrometry-based single-cell proteomics data. The package is an extension to the 'QFeatures' package and relies on 'SingleCellExpirement' to enable single-cell proteomics analyses. The package offers the user the functionality to process quantitative table (as generated by MaxQuant, Proteome Discoverer, and more) into data tables ready for downstream analysis and data visualization.

Depends R (>= 4.2.0), QFeatures (>= 1.3.5)

Imports methods, stats, utils, SingleCellExperiment, SummarizedExperiment, MultiAssayExperiment, MsCoreUtils, matrixStats, S4Vectors, dplyr, magrittr

Suggests scpdata, testthat, knitr, BiocStyle, rmarkdown, ggplot2, patchwork, impute, scater, sva, preprocessCore, vsn, uwot

License Artistic-2.0

Encoding UTF-8

VignetteBuilder knitr

biocViews GeneExpression, Proteomics, SingleCell, MassSpectrometry, Preprocessing, CellBasedAssays

BugReports https://github.com/UCLouvain-CBIO/scp/issues

URL https://UCLouvain-CBIO.github.io/scp

Roxygen list(markdown=TRUE)

RoxygenNote 7.2.3

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

git_branch RELEASE_3_18

git_last_commit 9872d67

git_last_commit_date 2023-10-24

Repository Bioconductor 3.18

Date/Publication 2024-03-25

Author Christophe Vanderaa [aut, cre]
(<https://orcid.org/0000-0001-7443-5427>),
Laurent Gatto [aut] (<https://orcid.org/0000-0002-1520-2268>)

Maintainer Christophe Vanderaa <christophe.vanderaa@uclouvain.be>

R topics documented:

`aggregateFeaturesOverAssays`

Aggregate features over multiple assays

Description

This function is a wrapper function around QFeatures::aggregateFeatures. It allows the user to provide multiple assays for which `aggregateFeatures` will be applied sequentially.

Usage

```
aggregateFeaturesOverAssays(object, i, fcol, name, fun, ...)
```

Arguments

`object`	A `QFeatures` object
`i`	A `numeric(1)` or `character(1)` indicating which assay to transfer the `colData` to.
`fcol`	The feature variables for each assays `i` defining how to summarise the QFeatures. If `fcol` has length 1, the variable name is assumed to be the same for all assays

name	A character() naming the new assay. name must have the same length as i. Note that the function will fail if of the names in name is already present.
fun	A function used for quantitative feature aggregation.
...	Additional parameters passed the fun.

Value

A QFeatures object

See Also

QFeatures::aggregateFeatures

Examples

```
data("scp1")
scp1 <- aggregateFeaturesOverAssays(scp1,
                                    i = 1:3,
                                    fcol = "peptide",
                                    name = paste0("peptides", 1:3),
                                    fun = colMeans,
                                    na.rm = TRUE)
scp1
```

---

computeSCR *Compute the sample over carrier ratio (SCR)*

Description

The function computes the ratio of the intensities of sample channels over the intentisty of the carrier channel for each feature. The ratios are averaged within the assay.

Usage

```
computeSCR(
  object,
  i,
  colvar,
  samplePattern,
  sampleFUN = "mean",
  carrierPattern,
  carrierFUN = sampleFUN,
  rowDataName = "SCR"
)
```

Arguments

object	A QFeatures object.
i	A character() or integer() indicating for which assay(s) the SCR needs to be computed.
colvar	A character(1) indicating the variable to take from colData(object) that gives the sample annotation.
samplePattern	A character(1) pattern that matches the sample encoding in colvar.
sampleFUN	A character(1) or function that provides the summarization function to use (eg mean, sum, media, max, ...). Only used when the pattern matches multiple samples. Default is mean. Note for custom function, na.rm = TRUE is passed to sampleFUN to ignore missing values, make sure to provide a function that accepts this argument.
carrierPattern	A character(1) pattern that matches the carrier encoding in colvar. Only one match per assay is allowed, otherwise only the first match is taken
carrierFUN	A character(1) or function that provides the summarization function to use (eg mean, sum, media, max, ...). Only used when the pattern matches multiple carriers. Default is the same function as sampleFUN. Note for custom function, na.rm = TRUE is passed to carrierFUN to ignore missing values, make sure to provide a function that accepts this argument.
rowDataName	A character(1) giving the name of the new variable in the rowData where the computed SCR will be stored. The name cannot already exist in any of the assay rowData.

Value

A QFeatures object for which the rowData of the given assay(s) is augmented with the mean SCR.

Examples

```
data("scp1")
scp1 <- computeSCR(scp1,
                   i = 1,
                   colvar = "SampleType",
                   carrierPattern = "Carrier",
                   samplePattern = "Blank|Macrophage|Monocyte",
                   sampleFUN = "mean",
                   rowDataName = "MeanSCR")
## Check results
rowData(scp1)[[1]][, "MeanSCR"]
```

`cumulativeSensitivityCurve` *Cumulative sensitivity curve*

Description

The cumulative sensitivity curve is used to evaluate if the sample size is sufficient to accurately estimate the total sensitivity. If it is not the case, an asymptotic regression model may provide a prediction of the total sensitivity if more samples would have been acquired.

Usage

```
cumulativeSensitivityCurve(
  object,
  i,
  by = NULL,
  batch = NULL,
  nsteps = 30,
  niters = 10
)

predictSensitivity(df, nSamples)
```

Arguments

`object`	An object of class QFeatures.
`i`	The index of the assay in `object`. The assay must contain an identification matrix, that is a matrix where an entry is `TRUE` if the value is observed and `FALSE` is the value is missing (see examples).
`by`	A vector of length equal to the number of columns in assay `i` that defines groups for a cumulative sensitivity curve will be computed separately. If missing, the sensitivity curve is computed for the completd dataset.
`batch`	A vector of length equal to the number of columns in assay `i` that defines the cell batches. All cells in a batch will be aggregated to a single sample.
`nsteps`	The number of equally spaced sample sizes to compute the sensitivity.
`niters`	The number of iteration to compute
`df`	The output from `cumulativeSensitivityCurve()`.
`nSamples`	A `numeric()` of samples sizes. If `Inf`, the prediction provides the extrapolated total sensitivity.

Details

As more samples are added to a dataset, the total number of distinct features increases. When sufficient number of samples are acquired, all peptides that are identifiable by the technology and

increasing the sample size no longer increases the set of identified features. The cumulative sensitivity curve depicts the relationship between sensitivity (number of distinct peptides in the data) and the sample size. More precisely, the curve is built by sampling cells in the data and count the number of distinct features found across the sampled cells. The sampling is repeated multiple times to account for the stochasticity of the approach. Datasets that have a sample size sufficiently large should have a cumulative sensitivity curve with a plateau.

The set of features present in a cell depends on the cell type. Therefore, we suggest to build the cumulative sensitivity curve for each cell type separately. This is possible when providing the by argument.

For multiplexed experiments, several cells are acquired in a run. In that case, when a features is identified in a cell, it is frequently also identified in all other cells of that run, and this will distort the cumulative sensitivity curve. Therefore, the function allows to compute the cumulative sensitivity curve at the batches level rather than at the cell level. This is possible when providing the `batch` argument.

Once the cumulative sensitivity curve is computed, the returned data can be visualized to explore the relationship between the sensitivity and the sample size. If enough samples are acquired, the curve should plateau at high numbers of samples. If it is not the case, the total sensitivity can be predicted using an asymptotic regression curve. To predict the total sensitivity, the model is extrapolated to infinite sample size. Therefore, the accuracy of the extrapolation will highly depend on the available data. The closer the curve is to the plateau, the more accurate the prediction.

Value

A `data.frame` with groups as many rows as pairs of cells and the following column(s):

- `jaccard`: the computed Jaccard index
- `by`: if `by` is not `NULL`, the group of the pair of cells for which the Jaccard index is computed.

Examples

```
## Simulate data
## 1000 features in 100 cells
library(SingleCellExperiment)
id <- matrix(FALSE, 1000, 1000)
id[sample(1:length(id), 5000)] <- TRUE
dimnames(id) <- list(
    paste0("feat", 1:1000),
    paste0("cell", 1:1000)
)
sce <- SingleCellExperiment(assays = List(id))
sim <- QFeatures(experiments = List(id = sce))
sim$batch <- rep(1:100, each = 10)
sim$SampleType <- rep(c("A", "B"), each = 500)
sim

## Compute the cumulative sensitivity curve, take batch and sample
## type into account
csc <- cumulativeSensitivityCurve(
    sim, "id", by = sim$SampleType,
    batch = sim$batch
```

```
)
predCSC <- predictSensitivity(csc, nSample = 1:50)

library(ggplot2)
ggplot(csc) +
    aes(x = SampleSize, y = Sensitivity, colour = by) +
    geom_point() +
    geom_line(data = predCSC)

## Extrapolate the total sensitivity
predictSensitivity(csc, nSamples = Inf)
## (real total sensitivity = 1000)
```

`divideByReference` *Divide assay columns by a reference column*

Description

The function divides the sample columns by a reference column. The sample and reference columns are defined based on the provided `colvar` variable and on regular expression matching.

Usage

```
divideByReference(object, i, colvar, samplePattern = ".", refPattern)
```

Arguments

`object`	A `QFeatures` object
`i`	A `numeric()` or `character()` vector indicating from which assays the `rowData` should be taken.
`colvar`	A `character(1)` indicating the variable to take from `colData(object)` that gives the sample annotation.
`samplePattern`	A `character(1)` pattern that matches the sample encoding in `colvar`. By default all samples are devided (using the regex wildcard `.`).
`refPattern`	A `character(1)` pattern that matches the carrier encoding in `colvar`. Only one match per assay is allowed, otherwise only the first match is taken

Details

The supplied assay(s) are replaced with the values computed after reference division.

Value

A `QFeatures` object

Examples

```
data("scp1")
scp1 <- divideByReference(scp1,
                          i = 1,
                          colvar = "SampleType",
                          samplePattern = "Macrophage",
                          refPattern = "Ref")
```

`jaccardIndex` *Compute the pairwise Jaccard index*

Description

The function computes the Jaccard index between all pairs of cells.

Usage

```
jaccardIndex(object, i, by = NULL)
```

Arguments

`object`	An object of class QFeatures.
`i`	The index of the assay in `object`. The assay must contain an identification matrix, that is a matrix where an entry is `TRUE` if the value is observed and `FALSE` is the value is missing (see examples).
`by`	A vector of length equal to the number of columns in assay `i` that defines groups for which the Jaccard index should be computed separately. If missing, the Jaccard indices are computed for all airs of cells in the dataset.

Value

A `data.frame` with as many rows as pairs of cells and the following column(s):

- `jaccard`: the computed Jaccard index
- `by`: if `by` is not `NULL`, the group of the pair of cells for which the Jaccard index is computed.

Examples

```
data("scp1")

## Define the identification matrix
peps <- scp1[["peptides"]]
assay(peps) <- ifelse(is.na(assay(peps)), FALSE, TRUE)
scp1 <- addAssay(scp1, peps, "id")

## Compute Jaccard indices
jaccardIndex(scp1, "id")
```

```
## Compute Jaccard indices by sample type
jaccardIndex(scp1, "id", scp1$SampleType)
```

`medianCVperCell` *Compute the median coefficient of variation (CV) per cell*

Description

The function computes for each cell the median CV and stores them accordingly in the `colData` of the `QFeatures` object. The CVs in each cell are computed from a group of features. The grouping is defined by a variable in the `rowData`. The function can be applied to one or more assays, as long as the samples (column names) are not duplicated. Also, the user can supply a minimal number of observations required to compute a CV to avoid that CVs computed on too few observations influence the distribution within a cell. The quantification matrix can be optionally normalized before computing the CVs. Multiple normalizations are possible.

Usage

```
medianCVperCell(
  object,
  i,
  groupBy,
  nobs = 5,
  na.rm = TRUE,
  colDataName = "MedianCV",
  norm = "none",
  ...
)
```

Arguments

`object`	A `QFeatures` object
`i`	A `numeric()` or `character()` vector indicating from which assays the `rowData` should be taken.
`groupBy`	A `character(1)` indicating the variable name in the `rowData` that contains the feature grouping.
`nobs`	An `integer(1)` indicating how many observations (features) should at least be considered for computing the CV. Since no CV can be computed for less than 2 observations, `nobs` should at least be 2.
`na.rm`	A `logical(1)` indicating whether missing data should be removed before computation.
`colDataName`	A `character(1)` giving the name of the new variable in the `colData` where the computed CVs will be stored. The name cannot already exist in the `colData`.

norm	A character() of normalization methods that will be sequentially applied to each feature (row) in each assay. Available methods and additional information about normalization can be found in MsCoreUtils::normalizeMethods. You can also specify norm = "SCoPE2" to reproduce the normalization performed before computing the CVs as suggested by Specht et al. norm = "none" will not normalize the data (default)
...	Additional arguments that are passed to the normalization method.

Details

A new column is added to the colData of the object. The samples (columns) that are not present in the selection i will get assigned an NA.

Value

A QFeatures object.

References

Specht, Harrison, Edward Emmott, Aleksandra A. Petelski, R. Gray Huffman, David H. Perlman, Marco Serra, Peter Kharchenko, Antonius Koller, and Nikolai Slavov. 2021. "Single-Cell Proteomic and Transcriptomic Analysis of Macrophage Heterogeneity Using SCoPE2." Genome Biology 22 (1): 50.

Examples

```
data("scp1")
scp1 <- filterFeatures(scp1, ~ !is.na(Proteins))
scp1 <- medianCVperCell(scp1,
                        i = 1:3,
                        groupBy = "Proteins",
                        nobs = 5,
                        na.rm = TRUE,
                        colDataName = "MedianCV",
                        norm = "div.median")
## Check results
hist(scp1$MedianCV)
```

---

mqScpData *Example MaxQuant/SCoPE2 output*

Description

A data.frame with 1088 observations and 139 variables, as produced by reading a MaxQuant output file with read.delim().

- Sequence: a character vector

- Length: a numeric vector
- Modifications: a character vector
- Modified.sequence: a character vector
- Deamidation..N..Probabilities: a character vector
- Oxidation..M..Probabilities: a character vector
- Deamidation..N..Score.Diffs: a character vector
- Oxidation..M..Score.Diffs: a character vector
- Acetyl..Protein.N.term.: a numeric vector
- Deamidation..N.: a numeric vector
- Oxidation..M.: a numeric vector
- Missed.cleavages: a numeric vector
- Proteins: a character vector
- Leading.proteins: a character vector
- protein: a character vector
- Gene.names: a character vector
- Protein.names: a character vector
- Type: a character vector
- Set: a character vector
- MS.MS.m.z: a numeric vector
- Charge: a numeric vector
- m.z: a numeric vector
- Mass: a numeric vector
- Resolution: a numeric vector
- Uncalibrated...Calibrated.m.z..ppm.: a numeric vector
- Uncalibrated...Calibrated.m.z..Da.: a numeric vector
- Mass.error..ppm.: a numeric vector
- Mass.error..Da.: a numeric vector
- Uncalibrated.mass.error..ppm.: a numeric vector
- Uncalibrated.mass.error..Da.: a numeric vector
- Max.intensity.m.z.0: a numeric vector
- Retention.time: a numeric vector
- Retention.length: a numeric vector
- Calibrated.retention.time: a numeric vector
- Calibrated.retention.time.start: a numeric vector
- Calibrated.retention.time.finish: a numeric vector
- Retention.time.calibration: a numeric vector
- Match.time.difference: a logical vector

- Match.m.z.difference: a logical vector
- Match.q.value: a logical vector
- Match.score: a logical vector
- Number.of.data.points: a numeric vector
- Number.of.scans: a numeric vector
- Number.of.isotopic.peaks: a numeric vector
- PIF: a numeric vector
- Fraction.of.total.spectrum: a numeric vector
- Base.peak.fraction: a numeric vector
- PEP: a numeric vector
- MS.MS.count: a numeric vector
- MS.MS.scan.number: a numeric vector
- Score: a numeric vector
- Delta.score: a numeric vector
- Combinatorics: a numeric vector
- Intensity: a numeric vector
- Reporter.intensity.corrected.0: a numeric vector
- Reporter.intensity.corrected.1: a numeric vector
- Reporter.intensity.corrected.2: a numeric vector
- Reporter.intensity.corrected.3: a numeric vector
- Reporter.intensity.corrected.4: a numeric vector
- Reporter.intensity.corrected.5: a numeric vector
- Reporter.intensity.corrected.6: a numeric vector
- Reporter.intensity.corrected.7: a numeric vector
- Reporter.intensity.corrected.8: a numeric vector
- Reporter.intensity.corrected.9: a numeric vector
- Reporter.intensity.corrected.10: a numeric vector
- RI1: a numeric vector
- RI2: a numeric vector
- RI3: a numeric vector
- RI4: a numeric vector
- RI5: a numeric vector
- RI6: a numeric vector
- RI7: a numeric vector
- RI8: a numeric vector
- RI9: a numeric vector
- RI10: a numeric vector

- RI11: a numeric vector
- Reporter.intensity.count.0: a numeric vector
- Reporter.intensity.count.1: a numeric vector
- Reporter.intensity.count.2: a numeric vector
- Reporter.intensity.count.3: a numeric vector
- Reporter.intensity.count.4: a numeric vector
- Reporter.intensity.count.5: a numeric vector
- Reporter.intensity.count.6: a numeric vector
- Reporter.intensity.count.7: a numeric vector
- Reporter.intensity.count.8: a numeric vector
- Reporter.intensity.count.9: a numeric vector
- Reporter.intensity.count.10: a numeric vector
- Reporter.PIF: a logical vector
- Reporter.fraction: a logical vector
- Reverse: a character vector
- Potential.contaminant: a logical vector
- id: a numeric vector
- Protein.group.IDs: a character vector
- Peptide.ID: a numeric vector
- Mod..peptide.ID: a numeric vector
- MS.MS.IDs: a character vector
- Best.MS.MS: a numeric vector
- AIF.MS.MS.IDs: a logical vector
- Deamidation..N..site.IDs: a numeric vector
- Oxidation..M..site.IDs: a logical vector
- remove: a logical vector
- dart_PEP: a numeric vector
- dart_qval: a numeric vector
- razor_protein_fdr: a numeric vector
- Deamidation..NQ..Probabilities: a logical vector
- Deamidation..NQ..Score.Diffs: a logical vector
- Deamidation..NQ.: a logical vector
- Reporter.intensity.corrected.11: a logical vector
- Reporter.intensity.corrected.12: a logical vector
- Reporter.intensity.corrected.13: a logical vector
- Reporter.intensity.corrected.14: a logical vector
- Reporter.intensity.corrected.15: a logical vector

- Reporter.intensity.corrected.16: a logical vector
- RI12: a logical vector
- RI13: a logical vector
- RI14: a logical vector
- RI15: a logical vector
- RI16: a logical vector
- Reporter.intensity.count.11: a logical vector
- Reporter.intensity.count.12: a logical vector
- Reporter.intensity.count.13: a logical vector
- Reporter.intensity.count.14: a logical vector
- Reporter.intensity.count.15: a logical vector
- Reporter.intensity.count.16: a logical vector
- Deamidation..NQ..site.IDs: a logical vector
- input_id: a logical vector
- rt_minus: a logical vector
- rt_plus: a logical vector
- mu: a logical vector
- muij: a logical vector
- sigmaij: a logical vector
- pep_new: a logical vector
- exp_id: a logical vector
- peptide_id: a logical vector
- stan_peptide_id: a logical vector
- exclude: a logical vector
- residual: a logical vector
- participated: a logical vector
- peptide: a character vector

Usage

```
data("mqScpData")
```

Format

An object of class `data.frame` with 1361 rows and 149 columns.

Details

The dataset is a subset of the SCoPE2 dataset (version 2, Specht et al. 2019, BioRXiv). The input file `evidence_unfiltered.csv` was downloaded from a Google Drive repository. The MaxQuant evidence file was loaded and the data was cleaned (renaming columns, removing duplicate fields,...). MS runs that were selected in the `scp1` dataset (see `?scp1`) were kept along with a blank run. The data is stored as a `data.frame`.

See Also

readSCP() for an example on how mqScpData is parsed into a QFeatures object.

---

normalizeSCP *Normalize single-cell proteomics (SCP) data*

Description

This function normalises an assay in a QFeatures according to the supplied method (see Details). The normalized data is added as a new assay

Usage

```
normalizeSCP(object, i, name = "normAssay", method, ...)
```

Arguments

object	An object of class QFeatures.
i	A numeric vector or a character vector giving the index or the name, respectively, of the assay(s) to be processed.
name	A character(1) naming the new assay name. Defaults is are normAssay.
method	character(1) defining the normalisation method to apply. See Details.‘
...	Additional parameters passed to MsCoreUtils::normalizeMethods().

Details

The method parameter in normalize can be one of "sum", "max", "center.mean", "center.median", "div.mean", "div.median", "diff.meda", "quantiles", "quantiles.robust" or "vsn". The MsCoreUtils::normalizeMethods() function returns a vector of available normalisation methods.

- For "sum" and "max", each feature's intensity is divided by the maximum or the sum of the feature respectively. These two methods are applied along the features (rows).
- "center.mean" and "center.median" center the respective sample (column) intensities by subtracting the respective column means or medians. "div.mean" and "div.median" divide by the column means or medians. These are equivalent to sweeping the column means (medians) along MARGIN = 2 with FUN = "-" (for "center.*") or FUN = "/" (for "div.*").
- "diff.median" centers all samples (columns) so that they all match the grand median by subtracting the respective columns medians differences to the grand median.
- Using "quantiles" or "quantiles.robust" applies (robust) quantile normalisation, as implemented in preprocessCore::normalize.quantiles() and preprocessCore::normalize.quantiles.robust() "vsn" uses the vsn::vsn2() function. Note that the latter also glog-transforms the intensities. See respective manuals for more details and function arguments.

For further details and examples about normalisation, see MsCoreUtils::normalize_matrix().

Value

A QFeatures object with an additional assay containing the normalized data.

See Also

QFeatures::normalize for more details about normalize

Examples

```
data("scp1")
scp1
normalizeSCP(scp1, i = "proteins", name = "normproteins",
             method = "center.mean")
```

---

pep2qvalue *Compute q-values*

Description

This function computes q-values from the posterior error probabilities (PEPs). The functions takes the PEPs from the given assay's rowData and adds a new variable to it that contains the computed q-values.

Usage

```
pep2qvalue(object, i, groupBy, PEP, rowDataName = "qvalue")
```

Arguments

object	A QFeatures object
i	A numeric() or character() vector indicating from which assays the rowData should be taken.
groupBy	A character(1) indicating the variable name in the rowData that contains the grouping variable, for instance to compute protein FDR. When groupBy is not missing, the best feature approach is used to compute the PEP per group, meaning that the smallest PEP is taken as the PEP of the group.
PEP	A character(1) indicating the variable names in the rowData that contains the PEPs. Since, PEPs are probabilities, the variable must be contained in (0, 1).
rowDataName	A character(1) giving the name of the new variable in the rowData where the computed FDRs will be stored. The name cannot already exist in any of the assay rowData.

Details

The q-value of a feature (PSM, peptide, protein) is the minimum FDR at which that feature will be selected upon filtering (Savitski et al.). On the other hand, the feature PEP is the probability that the feature is wrongly matched and hence can be seen as a local FDR (Kall et al.). While filtering on PEP is guaranteed to control for FDR, it is usually too conservative. Therefore, we provide this function to convert PEP to q-values.

We compute the q-value of a feature as the average of the PEPs associated to PSMs that have equal or greater identification confidence (so smaller PEP). See Kall et al. for a visual interpretation.

We also allow inference of q-values at higher level, for instance computing the protein q-values from PSM PEP. This can be performed by supplying the `groupBy` argument. In this case, we adopt the best feature strategy that will take the best (smallest) PEP for each group (Savitski et al.).

Value

A `QFeatures` object.

References

Käll, Lukas, John D. Storey, Michael J. MacCoss, and William Stafford Noble. 2008. "Posterior Error Probabilities and False Discovery Rates: Two Sides of the Same Coin." Journal of Proteome Research 7 (1): 40–44.

Savitski, Mikhail M., Mathias Wilhelm, Hannes Hahne, Bernhard Kuster, and Marcus Bantscheff. 2015. "A Scalable Approach for Protein False Discovery Rate Estimation in Large Proteomic Data Sets." Molecular & Cellular Proteomics: MCP 14 (9): 2394–2404.

Examples

```
data("scp1")
scp1 <- pep2qvalue(scp1,
                   i = 1,
                   groupBy = "protein",
                   PEP = "dart_PEP",
                   rowDataName = "qvalue_protein")
## Check results
rowData(scp1)[[1]][, c("dart_PEP", "qvalue_protein")]
```

---

readSCP *Read single-cell proteomics data as a QFeatures object from tabular data and metadata*

Description

Convert tabular quantitative MS data and metadata from a spreadsheet or a `data.frame` into a QFeatures object containing SingleCellExperiment objects.

Usage

```
readSCP(
  featureData,
  colData,
  batchCol,
  channelCol,
  suffix = NULL,
  sep = "",
  removeEmptyCols = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

featureData	File or object holding the identification and quantitative data. Can be either a character(1) with the path to a text-based spreadsheet (comma-separated values by default, but see ...) or an object that can be coerced to a data.frame. It is advised not to encode characters as factors.
colData	A data.frame or any object that can be coerced to a data.frame. colData is expected to contain all the sample meta information. Required fields are the acquisition batch (given by batchCol) and the acquisition channel within the batch (e.g. TMT channel, given by channelCol). Additional fields (e.g. sample type, acquisition date,...) are allowed and will be stored as sample meta data.
batchCol	A numeric(1) or character(1) pointing to the column of featureData and colData that contain the batch names. Make sure that the column name in both table are either identical and syntactically valid (if you supply a character) or have the same index (if you supply a numeric). Note that characters can be converted to syntactically valid names using make.names
channelCol	A numeric(1) or character(1) pointing to the column of colData that contains the column names of the quantitative data in featureData (see Example).
suffix	A character() giving the suffix of the column names in each assay. Sample/single-cell (column) names are automatically generated using: batch name + sep + suffix. Make sure suffix contains unique character elements. The length of the vector should equal the number of quantification channels. If NULL (default), the suffix is derived from the the names of the quantification columns in featureData.
sep	A character(1) that is inserted between the assay name and the suffix (see suffix argument for more details).
removeEmptyCols	A logical(1). If true, the function will remove in each batch the columns that contain only missing values.
verbose	A logical(1) indicating whether the progress of the data reading and formatting should be printed to the console. Default is TRUE.
...	Further arguments that can be passed on to read.csv except stringsAsFactors, which is always FALSE.

Value

An instance of class QFeatures. The expression data of each batch is stored in a separate assay as a SingleCellExperiment object.

Note

The `SingleCellExperiment` class is built on top of the `RangedSummarizedExperiment` class. This means that some column names are forbidden in the `rowData`. Avoid using the following names: `seqnames`, `ranges`, `strand`, `start`, `end`, `width`, `element`

Author(s)

Laurent Gatto, Christophe Vanderaa

Examples

```
## Load an example table containing MaxQuant output
data("mqScpData")

## Load the (user-generated) annotation table
data("sampleAnnotation")

## Format the tables into a QFeatures object
readSCP(featureData = mqScpData,
        colData = sampleAnnotation,
        batchCol = "Raw.file",
        channelCol = "Channel")
```

---

readSCPfromDIANN *Read DIA-NN output as a QFeatures objects for single-cell proteomics data*

Description

This function takes the output tables from DIA-NN and converts them into a QFeatures object using the scp framework.

Usage

```
readSCPfromDIANN(
  colData,
  reportData,
  extractedData = NULL,
  ecol = "MS1.Area",
  multiplexing = "none",
  ...
)
```

Arguments

colData	A data.frame or any object that can be coerced to a data.frame. colData is expected to contains all the sample annotations. We require the table to contain a column called File.Name that links to the File.Name in the DIA-NN report table. If multiplexing = "mTRAQ", we require a second column called Label that links the label to the sample (the labels identified by DIA-NN can be retrieved from Modified Sequence column in the report table).
reportData	A data.frame or any object that can be coerced to a data.frame that contains the data from the Report.tsv file generated by DIA-NN.
extractedData	A data.frame or any object that can be coerced to a data.frame that contains the data from the *_ms1_extracted.tsv file generated by DIA-NN. This argument is optional and is only applicable for mulitplixed experiments
ecol	A character(1) indicating which column in reportData contains the quantitative information.
multiplexing	A character(1) indicating the type of multiplexing used in the experiment. Provide "none" if the experiment is label-free (default). Available options are: "mTRAQ".
...	Further arguments passed to readSCP()

Value

An instance of class QFeatures. The expression data of each acquisition run is stored in a separate assay as a SingleCellExperiment object.

---

readSingleCellExperiment

Read SingleCellExperiment from tabular data

Description

Convert tabular data from a spreadsheet or a data.frame into a SingleCellExperiment object.

Usage

```
readSingleCellExperiment(table, ecol, fnames, ...)
```

Arguments

table	File or object holding the quantitative data. Can be either a character(1) with the path to a text-based spreadsheet (comma-separated values by default, but see ...) or an object that can be coerced to a data.frame. It is advised not to encode characters as factors.

ecol	A numeric indicating the indices of the columns to be used as assay values. Can also be a character indicating the names of the columns. Caution must be taken if the column names are composed of special characters like (or - that will be converted to a . by the read.csv function. If ecol does not match, the error message will display the column names as seen by the read.csv function.
fnames	An optional character(1) or numeric(1) indicating the column to be used as row names.
...	Further arguments that can be passed on to read.csv except stringsAsFactors, which is always FALSE.

Value

An instance of class SingleCellExperiment.

Note

The SingleCellExperiment class is built on top of the RangedSummarizedExperiment class. This means that some column names are forbidden in the rowData. Avoid using the following names: seqnames, ranges, strand, start, end, width, element

Author(s)

Laurent Gatto, Christophe Vanderaa

See Also

The code relies on QFeatures::readSummarizedExperiment.

Examples

```
## Load a data.frame with PSM-level data
data("mqScpData")

## Create the QFeatures object
sce <- readSingleCellExperiment(mqScpData,
                                grep("RI", colnames(mqScpData)))
```

---

reportMissingValues *Four metrics to report missing values*

Description

The function computes four metrics to report missing values in single-cell proteomics.

Usage

```
reportMissingValues(object, i, by = NULL)
```

Arguments

object	An object of class QFeatures.
i	The index of the assay in object. The assay must contain an identification matrix, that is a matrix where an entry is TRUE if the value is observed and FALSE is the value is missing (see examples).
by	A vector of length equal to the number of columns in assay i that defines groups for which the metrics should be computed separately. If missing, the metrics are computed for the complete assay.

Value

A data.frame with groups as rows and 5 columns:

- LocalSensitivityMean: the average number of features per cell.
- LocalSensitivitySd: the standard deviation of the local sensitivity.
- TotalSensitivity: the total number of features found in the dataset.
- Completeness: the proportion of values that are not missing in the data.
- NumberCells: the number of cells in the dataset.

Examples

```
data("scp1")

## Define the identification matrix
peps <- scp1[["peptides"]]
assay(peps) <- ifelse(is.na(assay(peps)), FALSE, TRUE)
scp1 <- addAssay(scp1, peps, "id")

## Report metrics
reportMissingValues(scp1, "id")
## Report metrics by sample type
reportMissingValues(scp1, "id", scp1$SampleType)

data
```

---

sampleAnnotation *Single cell sample annotation*

Description

A data frame with 48 observations on the following 6 variables.

- Set: a character vector
- Channel: a character vector
- SampleType: a character vector

- lcbatch: a character vector
- sortday: a character vector
- digest: a character vector

Usage

```
data("sampleAnnotation")
```

Format

An object of class `data.frame` with 64 rows and 6 columns.

Details

##' The dataset is a subset of the SCoPE2 dataset (version 2, Specht et al. 2019, BioRXiv). The input files `batch.csv` and `annotation.csv` were downloaded from a Google Drive repository. The two files were loaded and the columns names were adapted for consistency with `mqScpData` table (see `?mqScpData`). The two tables were filtered to contain only sets present in "mqScpData. The tables were then merged based on the run ID, hence merging the sample annotation and the batch a

See Also

`readSCP()` to see how this file is used.

---

scp1 *Single Cell QFeatures data*

Description

A small QFeatures object with SCoPE2 data. The object is composed of 5 assays, including 3 PSM-level assays, 1 peptide assay and 1 protein assay.

Usage

```
data("scp1")
```

Format

An object of class `QFeatures` of length 5.

Details

The dataset is a subset of the SCoPE2 dataset (version 2, Specht et al. 2019, BioRXiv). This dataset was converted to a `QFeatures` object where each assay in stored as a `SingleCellExperiment` object. One assay per chromatographic batch ("LCA9", "LCA10", "LCB3") was randomly sampled. For each assay, 100 proteins were randomly sampled. PSMs were then aggregated to peptides and joined in a single assay. Then peptides were aggregated to proteins.

Examples

```
data("scp1")
scp1
```

Index