Package 'scp'

December 5, 2025

Title Mass Spectrometry-Based Single-Cell Proteomics Data Analysis

Version 1.21.0

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 (>= 4.3.0), QFeatures (>= 1.19.1)

Imports IHW, ggplot2, ggrepel, matrixStats, metapod, methods, MsCoreUtils, MultiAssayExperiment, nipals, RColorBrewer, S4Vectors, SingleCellExperiment, SummarizedExperiment, stats, utils

Suggests BiocStyle, BiocGenerics, MsDataHub (>= 1.3.3), impute, knitr, patchwork, preprocessCore, rmarkdown, scater, scpdata, sva, testthat, vdiffr, 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 7.3.2

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

git_branch devel

git_last_commit c32536b

git_last_commit_date 2025-10-29

Repository Bioconductor 3.23

Date/Publication 2025-12-04

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

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

Contents

`addReducedDims` *Add scplainer Component Analysis Results*

Description

The function will add the component results computed by `scpComponentAnalysis()` to a `SingleCellExperiment`'s `reducedDims` slot, to all using the many `scater` functions, such as `scater::plotReducedDim()`, `scater::plotTSNE()`, ...

Usage

```
addReducedDims(sce, x)
```

Arguments

sce	An instance of class SingleCellExperiment.
x	A List of DataFrames containing principal components. This list is typically the bySample element produced by scpComponentAnalysis().

Value

A SingleCellExperiment with updated reducedDims.

Author(s)

Laurent Gatto and Christophe Vanderaa

Examples

```
library("scater")
data("leduc_minimal")
pcs <- scpComponentAnalysis(
   leduc_minimal, method = "ASCA",
   effects = "SampleType")$bySample

reducedDims(leduc_minimal)
leduc_minimal <- addReducedDims(leduc_minimal, pcs)
reducedDims(leduc_minimal)
plotReducedDim(leduc_minimal, dimred = "ASCA_SampleType",
               colour_by = "SampleType")
leduc_minimal <- runTSNE(leduc_minimal, dimred = "ASCA_SampleType")
plotTSNE(leduc_minimal, colour_by = "SampleType")
```

---

aggregateFeaturesOverAssays-deprecated

Aggregate features over multiple assays

Description

The aggregateFeaturesOverAssays function is deprecated and will be removed in a future release. Please use the aggregateFeatures method from the QFeatures package instead.

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(SummarizedExperiment)
id <- matrix(FALSE, 1000, 1000)
id[sample(1:length(id), 5000)] <- TRUE
dimnames(id) <- list(
    paste0("feat", 1:1000),
    paste0("cell", 1:1000)
)
sce <- SummarizedExperiment(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)
```

`leduc_minimal` *Minimally processed single-cell proteomics data set*

Description

A `SingleCellExperiment` object that has been minimally processed. The data set is published by Leduc et al. 2022 (see references) and retrieved using `scpdata::leduc2022_pSCoPE()`. The data processing was conducted with `QFeatures` and `scp`. Quality control was performed, followed by building the peptide data and log2-transformation. To limit the size of the data, only cells associated to the 3 first and 3 last MS acquisition runs were kept. For the same reason, 200 peptides were randomly sampled. Therefore, the data set consists of 200 peptides and 73 cells. Peptide annotations can be retrieved from the `rowData` and cell annotations can be retrieved from the `colData`.

Usage

```
data("leduc_minimal")
```

Format

An object of class `SingleCellExperiment` with 200 rows and 73 columns.

Quality control

Any zero value has been replaced by NA.

A peptide was removed from the data set if:

- it matched to a decoy or contaminant peptide
- it had an parental ion fraction below 60 \
- it had a DART-ID adjusted q-value superior to 1\
- it had an average sample to carrier ratio above 0.05

A cell was removed from the data set if:

- it had a median coefficient of variation superior to 0.6
- it had a log2 median intensity outside (6, 8)
- it contained less than 750 peptides

Building the peptide matrix

PSMs belonging to the same peptide were aggregating using the median value. Some peptides were mapped to a different protein depending on the MS acquisition run. To solve this issue, a majority vote was applied to assign a single protein to each peptide. Protein IDs were translated into gene symbols using the `ensembldb` package.

Author(s)

Christophe Vanderaa, Laurent Gatto

References

Leduc, Andrew, R. Gray Huffman, Joshua Cantlon, Saad Khan, and Nikolai Slavov. 2022. "Exploring Functional Protein Covariation across Single Cells Using nPOP." Genome Biology 23 (1): 261.

`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 tabular data*

Description

Function to import and convert tabular data from a spreadsheet or a `data.frame` into a `SingleCellExperiment` and `QFeatures` object.

Usage

```
readSCP(..., experimentsAsSce = FALSE)

readSCPfromDIANN(..., experimentsAsSce = FALSE)

readSingleCellExperiment(...)
```

Arguments

`...`	Parameters passed to `readSummarizedExperiment()`, `readQFeatures()` or `readQFeaturesFromDIANN(` See these respective manual pages for details.
`experimentsAsSce`	A `logical(1)` indicating whether the `QFeatures` object should be composed of `SingleCellExperiment` objects. By default: `FALSE`, the `QFeatures` object will be composed of `SummarizedExperiment` objects. Note that using `SingleCellExperiment` can impact the performance.

Value

An instance of class `SingleCellExperiment` or a `QFeatures`, composed of `SingleCellExperiment` or `SummarizedExperiment` objects.

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`

See Also

- The more general `QFeatures::readQFeatures()` function, which this function depends on.
- The more general `QFeatures::readQFeaturesFromDIANN()` function, for details and an example on how to read label-free and plexDIA (mTRAQ) data processed with DIA-NN.

- The QFeatures::readSummarizedExperiment() function, which readSingleCellExperiment() depends on.
- The SingleCellExperiment::SingleCellExperiment() class.

Examples

```
#####################################################
## Load a single acquisition as a SingleCellExperiment

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

## Create the QFeatures object
quantCols <- grep("Reporter.intensity.\\d", colnames(mqScpData))
sce <- readSingleCellExperiment(mqScpData, quantCols)
sce

#####################################################
## Load multiple acquisitions as a QFeatures

## 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(assayData = mqScpData,
        colData = sampleAnnotation,
        runCol = "Raw.file")
```

---

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). i may be numeric, character or logical, but it must select only one assay.

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) <- !is.na(assay(peps))
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
```

scpAnnotateResults *Annotate single-cell proteomics analysis output*

Description

The function takes as input a list of DFrame and a table with additional annotations. The annotation tables is automatically merged into all tables of the list by matching the specified columns (given by the arguments by and by2). This function is useful to add annotation to analysis results generated by scpVarianceAnalysis(), scpDifferentialAnalysis(), or scpComponentAnalysis(). The annotation table is typically the colData or rowData of the object used for modelling. In case of shared column names between the input tables and the annotation table, any annotation that is already present in the list of tables will be overwritten by the new annotations.

Usage

```
scpAnnotateResults(tableList, annotations, by, by2 = NULL)
```

Arguments

tableList	A list of tables, typically the output of scpVarianceAnalysis(), scpDifferentialAnalysis(), or the the bySample or byFeature elements returned by scpComponentAnalysis().
annotations	A table of class 'data.frame' or 'DFrame' containing the annotations to add. If no further arguments are provided, the table must have row names.
by	A character(1) providing the name of the column in the tables in tableList to use to match the rows of the annotation table.
by2	A character(1) providing the name of the column in the annotation table to use to match the rows of the tables in tableList. If NULL, it will be defined by by. The column pointed by by2 will be dropped in the output tables.

Author(s)

Christophe Vanderaa, Laurent Gatto

See Also

- ScpModel-VarianceAnalysis
- ScpModel-DifferentialAnalysis
- ScpModel-ComponentAnalysis

Examples

```
data("leduc_minimal")
var <- scpVarianceAnalysis(leduc_minimal)
colnames(var$Residuals)
## Add peptide annotations available from the rowData
var <- scpAnnotateResults(
```

```
    var, rowData(leduc_minimal), by = "feature", by2 = "Sequence"
)
colnames(var$Residuals)
```

`scplainer` *scplainer: linear models to understand mass spectrometry-based single-cell proteomics data*

Description

scplainer, standing for SCP-based Linear modelling Approach for Interpretable aNd Explorable Results, is a principled and standardised approach for extracting meaningful insights from SCP data. At its core, the approach performs statistical modelling using linear regression.

The workflow starts from a SingleCellExperiment object containing SCP data. The data is assumed to be **log-transformed**. We advise to perform cell and feature quality control to avoid that failed or outlying cells/feature distort the results. We also recommend starting at the precursor or the peptide-level, but the workflow also allows protein-level data. Similarly, the workflow is robust against for missing values, but it also allows for data where missing values are imputed.

To learn how to import your data, we suggest reading the vignette: `vignette("read_scp", package = "scp")`

To learn how to process your data, we suggest reading the vignette: `vignette("scp", package = "scp")`

Outline of the workflow

1. scpModel-Workflow: performs the data modelling and filtering using linear regression.
2. ScpModel-VarianceAnalysis: investigate the contribution of each model variable to the data
3. ScpModel-DifferentialAnalysis: assess the statistical significance of the differences observed between group of samples of interest.
4. ScpModel-ComponentAnalysis: visually explore the data captured by each model variable.

Once the data are modelled and explored, the filtered, normalised and batch-corrected data can be retrieved for further downstream analysis, such as clustering or trajectory inference.

You can find a demonstration of the *scplainer* workflow in a dedicated vignette: `vignette("scp_data_modelling", package = "scp")`

Author(s)

Christophe Vanderaa, Laurent Gatto

References

scplainer: using linear models to understand mass spectrometry-based single-cell proteomics data
Christophe Vanderaa, Laurent Gatto bioRxiv 2023.12.14.571792; doi: https://doi.org/10.1101/2023.12.14.571792.

ScpModel *Class to store the results of single-cell proteomics modelling*

Description

An ScpModel object must be always stored in the metadata() of an object that inherits from the SummarizedExperiment class. The ScpModel object should **never be accessed directly** by the user. Instead, we provide several setter function to retrieve information that may be useful to the user.The ScpModel class contains several slots:

- scpModelFormula: a formula object controlling which variables are to be modelled.
- scpModelInputIndex: a numeric(1), selecting the assay to use in the SummarizedExperiment object as input matrix. Note that this slot serves as a pointer, meaning that the quantitative data is not duplicated. Any change to the assay in the SummarizedExperiment will impact the estimation of the ScpModel object.
- scpModelFilterThreshold: A numeric(1) indicating the minimal n/p ratio required for a feature to be included in further model exploration. n/p is the number of measured values for a features divided by the number of coefficients to estimate. n/p cannot be smaller than 1 because this would lead to over-specified models.
- scpModelFitList: A List that contains the model results for each feature. Each element is a ScpModelFit object (see ScpModelFit)

Usage

```
scpModelFormula(object, name)

scpModelInput(object, name, filtered = TRUE)

scpModelFilterThreshold(object, name)

scpModelFilterNPRatio(object, name, filtered = TRUE)

scpModelResiduals(object, name, join = TRUE, filtered = TRUE)

scpModelEffects(object, name, join = TRUE, filtered = TRUE)

scpModelNames(object)

scpModelFilterThreshold(object, name) <- value
```

Arguments

object	An object that inherits from the SummarizedExperiment class.
name	A character(1) providing the name to use to store or retrieve the modelling results. When retrieving a model and name is missing, the name of the first model found in object is used.

filtered	A logical(1) indicating whether the output should return all features (FALSE) or the features that comply to the n/p ratio threshold (TRUE).
join	A logical(1) indicating whether the output should be combined in a single matrix (TRUE) or it should be returned as a list with one element for each feature (FALSE). When TRUE, any gaps across features will be filled with NA's.
value	An numeric(1), the new value for the n/p ratio threshold

Getters

Each slot has a getter function associated:

- scpModelNames(): returns a vector of names of ScpModel objects stored in the SummarizedExperiment object.
- scpModelFormula(): returns the formula slot of the ScpModel within an object that inherits from the SummarizedExperiment class.
- scpModelFilterThreshold(): returns the n/p ration threshold used for feature filtering.
- scpModelInput(): returns a matrix with the quantitative values used as input of the model. Hence, the matrix contains the data before modelling. If filtered = TRUE, the feature of the matrix are restricted to the features that satisfy the n/p ratio threshold.
- scpModelFilterNPRatio(): returns the computed n/p ratio for each feature. If filtered = TRUE, the function returns only the n/p of the features that satisfy the n/p ratio threshold.
- scpModelResiduals(): when join = FALSE, the function returns a list where each element corresponds to a feature and contains the estimated residuals. When join = TRUE (default), the function combines the list into a matrix with features in rows and cells in columns, and filling the gaps with NA. If filtered = TRUE, the feature of the matrix are restricted to the features that satisfy the n/p ratio threshold.
- scpModelEffects(): when join = FALSE, the function return a list where each element of the list corresponds to a feature. Each element contains another list with as many elements as variable in the model and each element contains the data effect vector for that vector. When join = TRUE (default), each element of the list is a matrix with features in rows and cells in columns where gaps are filled with NA. If filtered = TRUE, the feature of the matrix are restricted to the features that satisfy the n/p ratio threshold.

Setter:

- scpModelFilterThreshold<-(): the function changes the n/p ratio threshold used for filtering features.

Author(s)

Christophe Vanderaa, Laurent Gatto

See Also

- ScpModelFit for a description of the class that store modelling results
- ScpModel-Workflow that uses the class to store the estimated model.

Examples

```
data("leduc_minimal")

####---- Getters ----####

scpModelNames(leduc_minimal)

scpModelFormula(leduc_minimal)

dim(leduc_minimal)
dim(scpModelInput(leduc_minimal))
dim(scpModelInput(leduc_minimal, filtered = FALSE))

head(scpModelFilterNPRatio(leduc_minimal))

dim(scpModelResiduals(leduc_minimal))
dim(scpModelResiduals(leduc_minimal, filtered = FALSE))
scpModelResiduals(leduc_minimal, join = FALSE)

scpModelEffects(leduc_minimal)
dim(scpModelEffects(leduc_minimal)$Set)
dim(scpModelEffects(leduc_minimal, filtered = FALSE)$Set)
scpModelEffects(leduc_minimal, join = FALSE)[[1]]

scpModelFilterThreshold(leduc_minimal)

####---- Setter ----####

scpModelFilterThreshold(leduc_minimal) <- 2
scpModelFilterThreshold(leduc_minimal)
```

ScpModel-DataCorrection

Correct single-cell proteomics data

Description

The function uses the data modelling output to generate corrected data that can be used for downstream analysis. The input is expected to be a `SummarizedExperiment` object that contains an estimated `ScpModel`. There are two approaches:

- `scpKeepEffect()`: keep the effects of interests. The reconstructed data is the sum of the effect matrices for the variable of interest and the residuals. Note that the intercepts (baseline intensity of each feature) are not included by default, but they can be added when `intercept = TRUE`.
- `scpRemoveBatchEffect()`: remove any undesired effect. The batch corrected data is the input data minus the effect matrices that correspond to batch effect variables. Note that the intercepts (baseline intensity of each feature) are removed by default, but they can be kept when `intercept = FALSE`.

Despite the two approaches are conceptually different, they can lead to similar results if the effects that are used to reconstruct the data are the ones that are not removed when performing batch correction (see examples).

The function returns a new `SummarizedExperiment` that contains an assay with the batch corrected data. Note that the 'ScpModel' is erased in this new object.

Usage

```
scpKeepEffect(object, effects = NULL, intercept = FALSE, name)

scpRemoveBatchEffect(object, effects = NULL, intercept = TRUE, name)
```

Arguments

`object`	An object that inherits from the `SummarizedExperiment` class. It must contain an estimated `ScpModel` in its metadata
`effects`	A `character()` vector. For `scpKeepEffect()`, which model variable should be used to reconstruct the data. For `scpRemoveBatchEffect()`, which model variable should be removed from the data. When `NULL` (default), both functions return the model residuals.
`intercept`	A `logical(1)`. For `scpKeepEffect()`, should the intercepts be included when reconstructing the data? Defaults to `FALSE`, hence the intercepts are not included. For `scpRemoveBatchEffect()`, should the intercepts be removed from the data? Defaults to `TRUE`, hence the intercepts are removed from the data.
`name`	A `character(1)` providing the name to use to retrieve the model results. When retrieving a model and `name` is missing, the name of the first model found in `object` is used.

Author(s)

Christophe Vanderaa, Laurent Gatto

See Also

- ScpModel for functions to extract information from the `ScpModel` object
- ScpModel-Workflow to run a model on SCP data required for batch correction.

Examples

```
data("leduc_minimal")
scpModelFormula(leduc_minimal)

reconstructed <- scpKeepEffect(leduc_minimal, effects = "SampleType")
batchCorreced <- scpRemoveBatchEffect(
    leduc_minimal, effects = c("Channel", "Set", "MedianIntensity")
)
## The two approaches are identical
identical(reconstructed, batchCorreced)
```

`ScpModel-DifferentialAnalysis`
Differential abundance analysis for single-cell proteomics

Description

Differential abundance analysis assess the statistical significance of the differences observed between group of samples of interest. Differential abundance analysis is part of the *scplainer* workflow.

Usage

```
scpDifferentialAnalysis(object, coefficients = NULL, contrasts = NULL, name)

scpDifferentialAggregate(differentialList, fcol, ...)

scpVolcanoPlot(
  differentialList,
  fdrLine = 0.05,
  top = 10,
  by = "padj",
  decreasing = FALSE,
  textBy = "feature",
  pointParams = list(),
  labelParams = list()
)
```

Arguments

`object`	An object that inherits from the `SummarizedExperiment` class. It must contain an estimated `ScpModel` in its metadata.
`coefficients`	A `character()` vector with coefficient names to test. `coefficients` and `contrasts` cannot be both NULL.
`contrasts`	A `list()` where each element is a contrast to test. Each element must be a vector with 3 strings: 1. The name of a categorical variable to test; 2. The name of the reference group: 3. The name of the second group to contrast against the reference group. `coefficients` and `contrasts` cannot be both NULL.
`name`	A `character(1)` providing the name to use to retrieve the model results. When retrieving a model and `name` is missing, the name of the first model found in `object` is used.
`differentialList`	A list of tables returned by `scpDifferentialAnalysis()`.
`fcol`	A `character(1)` indicating the column to use for grouping features. Typically, this would be protein or gene names for grouping proteins.
`...`	Further arguments passed to `metapod::combineGroupedPValues()`.

fdrLine	A numeric(1) indicating the FDR threshold bar to show on the plot.
top	A numeric(1) indicating how many features should be labelled on the plot.
by	A character(1) used to order the features It indicates which variable should be considered when sorting the results. Can be one of: "Estimate", "SE", "Df", "tstatistic", "pvalue", "padj" or any other annotation added by the user.
decreasing	A logical(1) indicating whether the features should be ordered decreasingly (TRUE, default) or increasingly (FALSE) depending on the value provided by by.
textBy	A character(1) indicating the name of the column to use to label points.
pointParams	A list where each element is an argument that is provided to ggplot2::geom_point(). This is useful to change point size, transparency, or assign colour based on an annotation (see ggplot2::aes()).
labelParams	A list where each element is an argument that is provided to ggrepel::geom_label_repel(). This is useful to change label size, transparency, or assign colour based on an annotation (see ggplot2::aes()).

Running the differential abundance analysis

scpDifferentialAnalysis() performs statistical inference by means of a t-test on the estimatated parameters. There are 2 use cases:

1. **Statistical inference for differences between 2 groups**

You can **contrast** 2 groups of interest through the contrasts argument. Multiple contrasts, that is multiple pairwise group comparisons, can be performed. Therefore, contrasts must be provided as a list where each element describes the comparison to perform as a three-element character vector (see examples). The first element is the name of the annotation variable that contains the two groups to compare. This variable must be **categorical**. The second element is the name of the reference group. The third element is the name of the other group to compare against the reference.

1. **Statistical inference for numerical variables**

Numerical variables can be tested by providing the coefficient argument, that is the name of the numerical annotation variable.

The statistical tests in both use cases are conducted for each feature independently. The p-values are adjusted using IHW::ihw(), where each test is weighted using the feature intercept (that is the average feature intensity). The function returns a list of DataFrames with one table for each test contrast and/or coefficient. It provides the adjusted p-values and the estimates. For contrast, the estimates represent the estimated log fold changes between the groups. For coefficients, the estimates are the estimated slopes. Results are only provided for features for which contrasts or coefficients are estimable, that are features for which there is sufficient observations for inference.

Differential abundance at the protein level

scpDifferentialAggregate() combines the differential abundance analysis results for groups of features. This is useful, for example, to return protein-level results when data is modelled at the peptide level. The function heavily relies on the approaches implemented in metapod::combineGroupedPValues(). The p-values are combined into a single value using one of the following methods: Simes' method

(default), Fisher's method, Berger's method, Pearson's method, minimum Holm's approach, Stouffer's Z-score method, and Wilkinson's method. We refer to the `metapod` documentation for more details on the assumptions underlying each approach. The estimates are combined using the representative estimate, as defined by `metapod`. Which estimate is representative depends on the selected combination method. The function takes the list of tables generated by `scpDifferentialAnalysis()` and returns a new list of `DataFrames` with aggregated results. Note that we cannot meaningfully aggregate degrees of freedom. Those are hence removed from the aggregated result tables.

Volcano plots

`scpAnnotateResults()` adds annotations to the differential abundance analysis results. The annotations are added to all elements of the list returned by `()`. See the associated man page for more information.

`scpVolcanoPlot()` takes the list of tables generated by `scpDifferentialAnalysis()` and returns a `ggplot2` scatter plot. The plots show the adjusted p-values with respect to the estimate. A horizontal bar also highlights the significance threshold (defaults to 5%, `fdrLine`). The top (default 10) features with lowest p-values are labeled on the plot. You can control which features are labelled using the `top`, `by` and `decreasing` arguments. Finally, you can change the point and label aesthetics thanks to the `pointParams` and the `labelParams` arguments, respectively.

Author(s)

Christophe Vanderaa, Laurent Gatto

References

scplainer: using linear models to understand mass spectrometry-based single-cell proteomics data
Christophe Vanderaa, Laurent Gatto bioRxiv 2023.12.14.571792; doi: https://doi.org/10.1101/2023.12.14.571792.

See Also

This function is part of the *scplainer* workflow, which also consists of ScpModel-Workflow to run a model on SCP data upstream of analysis of variance, and ScpModel-VarianceAnalysis and ScpModel-ComponentAnalysis to explore the model results.

`scpAnnotateResults()` streamlines the annotation of the differential abundance results.

Examples

```
library("patchwork")
library("ggplot2")
data("leduc_minimal")
## Add n/p ratio information in rowData
rowData(leduc_minimal)$npRatio <-
    scpModelFilterNPRatio(leduc_minimal, filtered = FALSE)

####---- Run differential abundance analysis ----####

(res <- scpDifferentialAnalysis(
    leduc_minimal, coefficients =  "MedianIntensity",
    contrasts = list(c("SampleType", "Melanoma", "Monocyte"))
```

```
))
## IHW return a message because of the example data set has only few
## peptides, real dataset should not have that problem.

####---- Annotate results ----####

## Add peptide annotations available from the rowData
res <- scpAnnotateResults(
    res, rowData(leduc_minimal),
    by = "feature", by2 = "Sequence"
)

####---- Plot results ----####

scpVolcanoPlot(res, textBy = "gene") |>
    wrap_plots(guides = "collect")

## Modify point and label aesthetics
scpVolcanoPlot(
    res, textBy = "gene", top = 20,
    pointParams = list(aes(colour = npRatio), alpha = 0.5),
    labelParams = list(size = 2, max.overlaps = 20)) |>
    wrap_plots(guides = "collect")

####---- Aggregate results ----####

## Aggregate to protein-level results
byProteinDA <- scpDifferentialAggregate(
    res, fcol = "Leading.razor.protein.id"
)
scpVolcanoPlot(byProteinDA) |>
    wrap_plots(guides = "collect")
```

ScpModel-VarianceAnalysis

Analysis of variance for single-cell proteomics

Description

Analysis of variance investigates the contribution of each effects in capturing the variance in the data. Analysis of variance is part of the *scplainer* workflow.

Usage

```
scpVarianceAnalysis(object, name)

scpVarianceAggregate(varianceList, fcol)

scpVariancePlot(
```

```
  varianceList,
  effect = "Residuals",
  by = "percentExplainedVar",
  top = Inf,
  decreasing = TRUE,
  combined = TRUE,
  fcol = NULL,
  colourSeed = 1234
)
```

Arguments

`object`	An object that inherits from the `SummarizedExperiment` class. It must contain an estimated `ScpModel` in its metadata.
`name`	A `character(1)` providing the name to use to retrieve the model results. When retrieving a model and `name` is missing, the name of the first model found in `object` is used.
`varianceList`	A list of tables returned by `scpVarianceAnalysis()`.
`fcol`	A `character(1)` indicating the column to use for grouping features. Typically, this would be protein or gene names for grouping proteins.
`effect`	A `character(1)` used to filter theb results. It indicates which effect should be considered when sorting the results.
`by`	A `character(1)` used to filter the results. It indicates which variable should be considered when sorting the results. Can be one of: "SS", "df", or "percentExplainedVar".
`top`	A `numeric(1)` used to filter the results. It indicates how many features should be plotted. When `top = Inf` (default), all feature are considered.
`decreasing`	A `logical(1)` indicating whether the effects should be ordered decreasingly (`TRUE`, default) or increasingly (`FALSE`) depending on the value provided by `by`.
`combined`	A `logical(1)` indicating whether the results should be combined across all features. When `TRUE`, the barplot shows the explained variance for the complete dataset.
`colourSeed`	A `integer(1)` providing a seed that is used when randomly sampling colours for the effects. Change the number to generate another colour scheme.

Running the variance analysis

`scpVarianceAnalysis()` computes the amount of data (measured as the sums of squares) that is captured by each model variable, but also that is not modelled and hence captured in the residuals. The proportion of variance explained by each effect is the sums of squares for that effect divided by the sum of all sums of squares for each effect and residuals. This is computed for each feature separately. The function returns a list of `DataFrames` with one table for each effect.

`scpVarianceAggregate()` combines the analysis of variance results for groups of features. This is useful, for example, to return protein-level results when data is modelled at the peptide level. The function takes the list of tables generated by `scpVarianceAnalysis()` and returns a new list of `DataFrames` with aggregated results.

Exploring variance analysis results

`scpAnnotateResults()` adds annotations to the component analysis results. The annotations are added to all elements of the list returned by `scpComponentAnalysis()`. See the associated man page for more information.

`scpVariancePlot()` takes the list of tables generated by `scpVarianceAnalysis()` and returns a `ggplot2` bar plot. The bar plot shows the proportion of explained variance by each effect and the residual variance. By default, the function will combine the results over all features, showing the effect's contributions on the complete data set. When `combine = FALSE`, the results are shown for individual features, with additional arguments to control how many and which features are shown. Bars can also be grouped by `fcol`. This is particularly useful when exploring peptide level results, but grouping peptides that belong to the same protein (note that you should not use `scpVarianceAggregate()` in that case).

Author(s)

Christophe Vanderaa, Laurent Gatto

References

scplainer: using linear models to understand mass spectrometry-based single-cell proteomics data Christophe Vanderaa, Laurent Gatto bioRxiv 2023.12.14.571792; doi: https://doi.org/10.1101/2023.12.14.571792.

See Also

This function is part of the *scplainer* workflow, which also consists of ScpModel-Workflow to run a model on SCP data upstream of analysis of variance, and ScpModel-DifferentialAnalysis and ScpModel-ComponentAnalysis to explore the model results.

`scpAnnotateResults()` streamlines the annotation of the analysis of variance results.

Examples

```
data("leduc_minimal")

####---- Run analysis of variance ----####

(var <- scpVarianceAnalysis(leduc_minimal))

####---- Annotate results ----####

## Add peptide annotations available from the rowData
var <- scpAnnotateResults(
    var, rowData(leduc_minimal), by = "feature", by2 = "Sequence"
)

####---- Plot results ----####

## Plot the analysis of variance through the whole data
scpVariancePlot(var)

## Plot the analysis of variance for the top 20 peptides with highest
```

```
## percentage of variance explained by the cell type
scpVariancePlot(
    var, effect = "SampleType", top = 20, combined = FALSE
)

## Same but grouped by protein
scpVariancePlot(
    var, effect = "SampleType", top = 20, combined = FALSE, fcol = "gene"
)

####---- Aggregate results ----####

## Aggregate to protein-level results
varProtein <- scpVarianceAggregate(var, fcol = "gene")
scpVariancePlot(
    varProtein, effect = "SampleType", top = 20, combined = FALSE
)
```

`ScpModel-Workflow` *Modelling single-cell proteomics data*

Description

Function to estimate a linear model for each feature (peptide or protein) of a single-cell proteomics data set. This is the modelling step of the *scplainer* workflow.

Usage

```
scpModelWorkflow(object, formula, i = 1, name = "model", verbose = TRUE)

scpModelFilterPlot(object, name)
```

Arguments

`object`	An object that inherits from the `SummarizedExperiment` class.
`formula`	A `formula` object controlling which variables are to be modelled.
`i`	A `logical`, `numeric` or `character` indicating which assay of `object` to use as input for modelling. Only a single assay can be provided. Defaults to the first assays.
`name`	A `character(1)` providing the name to use to store or retrieve the modelling results. When retrieving a model and `name` is missing, the name of the first model found in `object` is used.
`verbose`	A `logical(1)` indicating whether to print progress to the console.

Input data

The main input is `object` that inherits from the `SummarizedExperiment` class. The quantitative data will be retrieve using `assay(object)`. If `object` contains multiple assays, you can specify which assay to take as input thanks to the argument `i`, the function will then assume `assay(object, i)` as quantification input .

The objective of modelling single-cell proteomics data is to estimate, for each feature (peptide or protein), the effect of known cell annotations on the measured intensities. These annotations may contain biological information such as the cell line, FACS-derived cell type, treatment, etc. We also highly recommend including technical information, such as the MS acquisition run information or the chemical label (in case of multiplexed experiments). These annotation must be available from `colData(object)`. `formula` specifies which annotations to use during modelling.

Data modelling workflow

The modelling worflow starts with generating a model matrix for each feature given the `colData(object)` and `formula`. The model matrix for peptide i, denoted X_i, is adapted to the pattern of missing values (see section below). Then, the functions fits the model matrix against the quantitative data. In other words, the function determines for each feature i (row in the input data) the contribution of each variable in the model. More formally, the general model definition is:

$$Y_i = \beta_i X_{(i)}^T + \epsilon_i$$

where Y is the feature by cell quantification matrix, β_i contains the estimated coefficients for feature i with as many coefficients as variables to estimate, $X_{(i)}^T$ is the model matrix generated for feature i, and ϵ is the feature by cell matrix with residuals.

The coefficients are estimated using penalized least squares regression. Next, the function computes the residual matrix and the effect matrices. An effect matrix contains the data that is captured by a given cell annotation. Formally, for each feature i:

$$\hat{M_i^f} = \hat{\beta_i^f} X_{(i)}^{fT}$$

where $\hat{M}^f$ is a cell by feature matrix containing the variables associated to annotation f, $\hat{\beta_i^f}$ are the estimated coefficients associated to annotation f and estimated for feature i, and $X_{(i)}^{fT}$ is the model matrix for peptide i containing only the variables to annotation f.

All the results are stored in an ScpModel object which is stored in the `object`'s metadata. Note that multiple models can be estimated for the same `object`. In that case, provide the `name` argument to store the results in a separate `ScpModel`.

Feature filtering

The proportion of missing values for each features is high in single-cell proteomics data. Many features can typically contain more coefficients to estimate than observed values. These features cannot be estimated and will be ignored during further steps. These features are identified by computing the ratio between the number of observed values and the number of coefficients to estimate. We call it the **n/p ratio**. Once the model is estimated, use `scpModelFilterPlot(object)` to explore the distribution of n/p ratios across the features. You can also extract the n/p ratio for each feature using

`scpModelFilterNPRatio(object)`. By default, any feature that has an n/p ratio lower than 1 is ignored. However, feature with an n/p ratio close to 1 may lead to unreliable outcome because there are not enough observed data. You could consider the n/p ratio as the average number of replicate per coefficient to estimate. Therefore, you may want to increase the n/p threshold. You can do so using `scpModelFilter(object) <- npThreshold`.

About missing values

The data modelling workflow is designed to take the presence of missing values into account. We highly recommend to **not impute** the data before modelling. Instead, the modelling approach will ignore missing values and will generate a model matrix using only the observed values for each feature. However, the model matrices for some features may contain highly correlated variables, leading to near singular designs. We include a small ridge penalty to reduce numerical instability associated to correlated variables.

Author(s)

Christophe Vanderaa, Laurent Gatto

References

scplainer: using linear models to understand mass spectrometry-based single-cell proteomics data Christophe Vanderaa, Laurent Gatto bioRxiv 2023.12.14.571792; doi: https://doi.org/10.1101/2023.12.14.571792.

See Also

This function is part of the *scplainer* workflow, which also consists of ScpModel-VarianceAnalysis, ScpModel-DifferentialAnalysis, ScpModel-ComponentAnalysis to explore the model results

ScpModel provides functions to extract information from the `ScpModel` object.

scpKeepEffect and scpRemoveBatchEffect perform batch correction for downstream analyses.

Examples

```
data("leduc_minimal")
leduc_minimal
## Overview of available cell annotations
colData(leduc_minimal)

####---- Model data ----####

f <- ~ 1 + ## intercept
    Channel + Set + ## batch variables
    MedianIntensity +## normalization
    SampleType ## biological variable
leduc_minimal <- scpModelWorkflow(leduc_minimal, formula = f)

####---- n/p feature filtering ----####

## Get n/p ratios
head(scpModelFilterNPRatio(leduc_minimal))
```

```
## Plot n/p ratios
scpModelFilterPlot(leduc_minimal)

## Change n/p ratio threshold
scpModelFilterThreshold(leduc_minimal) <- 2
scpModelFilterPlot(leduc_minimal)
```

---

scpModelComponentMethods

Component analysis for single cell proteomics

Description

Component analysis is a powerful tool for exploring data. The package implements the ANOVA-principal component analysis extended to linear models (APCA+) and derivatives (suggested by Thiel at al. 2017). This framework is based on principal component analysis (PCA) and allows exploring the data captured by each model variable individually. Component analysis is part of the *scplainer* workflow.

Usage

```
scpModelComponentMethods

scpComponentAnalysis(
  object,
  method = NULL,
  effects = NULL,
  pcaFUN = "auto",
  residuals = TRUE,
  unmodelled = TRUE,
  name,
  ...
)

scpComponentAggregate(componentList, fcol, fun = colMedians, ...)

scpComponentPlot(
  componentList,
  comp = 1:2,
  pointParams = list(),
  maxLevels = NULL
)

scpComponentBiplot(
  scoreList,
  eigenvectorList,
```

```
  comp = 1:2,
  pointParams = list(),
  arrowParams = list(arrow = arrow(length = unit(0.2, "cm"))),
  labelParams = list(size = 2, max.overlaps = 10),
  textBy = "feature",
  top = 10,
  maxLevels = NULL
)
```

Arguments

object	An object that inherits from the SummarizedExperiment class. It must contain an estimated ScpModel in its metadata.
method	A character() indicating which approach(es) to use for principal component analysis (PCA). Are allowed: "APCA" (default), "ASCA" and/or "ASCA.E" (multiple values are allowed). "ASCA", "APCA", "ASCA.E" are iterated through each desired effects.
effects	A character() indicating on which model variables the component analysis should be performed. Default to all modelled variables.
pcaFUN	A character(1) indicating which function to use to perform PCA. "nipals" will use nipals::nipals() while "svd" will use base::svd(). If "auto", the function uses "nipals" if the data contain missing values and "svd" otherwise.
residuals	A logical(1), if TRUE, PCA is performed on the residual matrix as well.
unmodelled	A logical(1), if TRUE, PCA is performed on the input matrix as well.
name	A character(1) providing the name to use to retrieve the model results. When retrieving a model and name is missing, the name of the first model found in object is used.
...	For scpComponentAnalysis(), further arguments passed to the PCA function. For scpComponentAggregate(), further arguments passed to QFeatures::aggregateFeatures().
componentList	A list of components analysis results. This is typically the bySample or byFeature element of the list returned by scpComponentAnalysis().
fcol	A character(1) providing the name of the column to use to group features.
fun	A function that summarises the values for each group. See QFeatures::aggregateFeatures() for a list of available functions.
comp	An integer(2) pointing to which components to fit. The values of comp are not allowed to exceed the number of computed components in componentList.
pointParams	A list where each element is an argument that is provided to ggplot2::geom_point(). This is useful to change point size, transparency, or assign colour based on an annotation (see ggplot2::aes()).
maxLevels	An integer(1) indicating how many colour levels should be shown on the legend when colours are derived from a discrete factor. If maxLevels = NULL, all levels are shown. This parameters is useful to colour points based on a factor with many levels that would otherwise overcrowd the legend.
scoreList	A list of components analysis results. This is typically the bySample element in the list returned by scpComponentAnalysis().

`eigenvectorList`
: A list of components analysis results. This is typically the `byFeature` element in the list returned by `scpComponentAnalysis()`.

`arrowParams`
: A `list` where each element is an argument that is provided to `ggplot2::geom_segment()`. This is useful to change arrow head style, line width, transparency, or assign colour based on an annotation (see `ggplot2::aes()`). Note that changing the 'x', 'y', 'xend', and 'yend' aesthetics is not allowed.

`labelParams`
: A `list` where each element is an argument that is provided to `ggrepel::geom_label_repel()`. This is useful to change label size, transparency, or assign colour based on an annotation (see `ggplot2::aes()`). Note that changing the 'x', 'y', 'xend', and 'yend' aesthetics is not allowed.

`textBy`
: A `character(1)` indicating the name of the column to use to label arrow heads.

`top`
: An `integer(1)` indicating how many arrows should be drawn. The arrows are sorted based on their size as determined by the euclidean distance in the principal component space.

Format

An object of class `character` of length 3.

PCA - notation and algorithms

Given A a m x n matrix, PCA can be summarised as the following decomposition:

$$AA^T/(n-1) = VLV^T$$

Where V is a m x k orthogonal matrix, that is $VV^T = I$, with k the number of components. V is called the matrix of eigenvectors. L is the k x k diagonal matrix of eigenvalues that contains the variance associated to each component, ordered from highest to lowest variance. The unscaled PC scores are given by $S = A^TV$.

There are 2 available algorithm to perform PCA:

- `nipals`: The non-linear iterative partial least squares (NIPALS) algorithm **can handle missing values** and approximates classical PCA, although it does not explicitly maximise the variance. This is implemented in `nipals::nipals()`.
- `svd`: The singular value decomposition (SVD) is used to perform an exact PCA, but it **cannot handle missing values**. This is implemented in `base::svd()`.

Which algorithm to use is controlled by the `pcaFUN` argument, by default (`"auto"`), the function automatically uses `svd` when there is no missing values and `nipals` when there is at least one missing value.

Component analysis methods

`scpComponentAnalysis()` performs a PCA on the modelling output. What modelling output the function will use depends on the `method`. The are 3 PCA approaches:

- `ASCA` performs a PCA on the effect matrix, that is $A = \hat{M}_f$ where f is one of the effects in the model. This PCA is useful to explore the modelled effects and the relationship between different levels of a factor.
- `ASCA.E`: perform PCA on the effect matrix, just like ASCA. The scores are then updated by projecting the effect matrix added to the residuals using the eigenvectors, that is $scores = (\hat{M}_f + \epsilon)^T V$. This PCA is useful to explore the modelled effects while blurring these effects with the unmodelled variability. Note however that for this approach, the scores are no longer guaranteed to be orthogonal and the eigenvalues are no longer meaningful. The percentage of variation should not be interpreted.
- `APCA` (default) performs PCA on the effect matrix plus the residuals, that is $A = \hat{M}_f + \epsilon$. This PCA is useful to explore the modelled effects in relation with the unmodelled variability that is remaining in the residuals.

Available methods are listed in `scpModelComponentMethods`. Note that for all three methods, a PCA on the residual matrix is also performed when `residuals = TRUE`, that is $A = \epsilon = Y - \hat{\beta}X^T$. A PCA on the residuals is useful to explore residual effects that are not captured by any effect in the model. Similarly, a PCA on the input data matrix, that is on the data before modelling is also performed when `unmodelled = TRUE`, that is $A = Y$.

`scpComponentAnalysis()` always returns a list with 2 elements. The first element, `bySample` is a list where each element contains the PC scores for the desired model variable(s). The second element, `byFeature` is a list where each element contains the eigenvectors for the desired model variable(s).

Exploring component analysis results

`scpAnnotateResults()` adds annotations to the component analysis results. The annotations are added to all elements of the list returned by `scpComponentAnalysis()`. See the associated man page for more information.

`scpComponentPlot()` takes one of the two elements of the list generated by `scpComponentAnalysis()` and returns a list of `ggplot2` scatter plots. Commonly, the first two components, that bear most of the variance, are explored for visualisation, but other components can be explored as well thanks to the `comp` argument. Each point represents either a sample or a feature, depending on the provided component analysis results (see examples). Change the point aesthetics by providing ggplot arguments in a list (see examples).

`scpComponentBiplot()` simultaneously explores the PC scores (sample-space) and the eigenvectors (feature-space). Scores are shown as points while eigenvectors are shown as arrows. Point aesthetics and arrow aesthetics can be controlled with the `pointParams` and the `arrowParams` arguments, respectively. Moreover, arrows are also labelled and label aesthetics can be controlled using `labelParams` and `textBy`. Plotting all eigenvectors as arrows leads to overcrowded plots. You can limit the plotting to the top longest arrows (default to the top 10) as defined by the distance on the two selected PCs.

`scpComponentAggregate()` offers functionality to aggregate the results from multiple features. This can be used to obtain, for example, component analysis results for proteins when modelling at the peptide level. The approach is inspired from scuttle::aggregateAcrossCells() and combines, for each group, multiple values for each component using `QFeatures::aggregateFeatures()`. By default, values are aggregated using the median, but `QFeatures` offers other methods as well. The annotation of the component results are automatically aggregated as well. See the `aggregateFeatures()` man page for more information on available methods and expected behavior.

Author(s)

Christophe Vanderaa, Laurent Gatto

References

Thiel, Michel, Baptiste Féraud, and Bernadette Govaerts. 2017. "ASCA+ and APCA+: Extensions of ASCA and APCA in the Analysis of Unbalanced Multifactorial Designs." Journal of Chemometrics 31 (6): e2895.

scplainer: using linear models to understand mass spectrometry-based single-cell proteomics data Christophe Vanderaa, Laurent Gatto bioRxiv 2023.12.14.571792; doi: https://doi.org/10.1101/2023.12.14.571792.

See Also

This function is part of the *scplainer* workflow, which also consists of ScpModel-Workflow to run a model on SCP data upstream of analysis of variance, and ScpModel-DifferentialAnalysis and ScpModel-VarianceAnalysis to explore the model results.

Other useful functions:

- The `nipals::nipals()` function and package for detailed information about the algorithm and associated parameters.
- The `ggplot2::ggplot()` function and associated tutorials to manipulate and save the visualisation output
- `scpAnnotateResults()` to annotate component analysis results.

Examples

```
library("patchwork")
library("ggplot2")
data("leduc_minimal")
leduc_minimal$cell <- rownames(colData(leduc_minimal))

####---- Run component analysis ----####

(pcs <- scpComponentAnalysis(
    leduc_minimal, method = "ASCA", effects = "SampleType",
    pcaFUN = "auto", residuals = FALSE, unmodelled = FALSE
))

####---- Annotate results ----####

## Add cell annotation available from the colData
bySamplePCs <- scpAnnotateResults(
    pcs$bySample, colData(leduc_minimal), by = "cell"
)

## Add peptide annotations available from the rowData
byFeaturePCs <- scpAnnotateResults(
    pcs$byFeature, rowData(leduc_minimal),
    by = "feature", by2 = "Sequence"
)
```

```
####---- Plot results ----####

## Plot result in cell-space, ie each dot is a cell
scpComponentPlot(
    bySamplePCs,
    pointParams = list( ## ggplot arguments
        aes(colour = SampleType, shape = lcbatch),
        alpha = 0.6
    )
) |>
    wrap_plots(guides = "collect")

## Plot result in peptide-space, ie each dot is a peptide
scpComponentPlot(
    byFeaturePCs,
    pointParams = list(colour = "dodgerblue", alpha = 0.6)
) |>
    wrap_plots(guides = "collect")

## Plot both
scpComponentBiplot(
    bySamplePCs, byFeaturePCs,
    pointParams = list(aes(colour = SampleType), alpha = 0.6),
    labelParams = list(max.overlaps = 20),
    textBy = "gene"
) |>
    wrap_plots(guides = "collect")

####---- Aggregate results ----####

## Aggregate to protein-level results
byProteinPCs <- scpComponentAggregate(
    byFeaturePCs, fcol = "Leading.razor.protein.id"
)

## Plot result in protein-space, ie each dot is a protein
scpComponentPlot(
    byProteinPCs,
    pointParams = list(colour = "firebrick", alpha = 0.6)
) |>
    wrap_plots(guides = "collect")
```

`ScpModelFit` *Class to store the components of an estimated model for a feature*

Description

An `ScpModelFit` object is expected to be stored as a list element in the `scpModelFitList` of an `ScpModel` object. The `ScpModelFit` object should **never be accessed directly** by the user. Refer to

the ScpModel for a list of function to access the information in an ScpModelFit. The ScpModelFit class contains several slots that contain the model output for a feature:

- n: an integer, the number of observations for the feature
- p: an integer, the number of coefficient to estimate
- coefficients: a numeric vector with the estimated coefficients
- residuals: a numeric vector with the estimated residuals
- effects: a List with the
- df: an integer providing the number of degrees of freedom of the model estimation
- var: a numeric vector with the residual variance of the model estimation
- uvcov: the unscaled variance covariance matrix
- levels: a named List where each elements corresponds to a categorical model variable and contains a vector with the possible categories.

Author(s)

Christophe Vanderaa, Laurent Gatto

See Also

ScpModel for a description of the class that relies on ScpModelFit

Examples

```
new("ScpModelFit") ## this should never be used by the user
```

Index