Package 'MsFeatures'

September 19, 2021

Title Functionality for Mass Spectrometry Features

Version 1.0.0

Description The MsFeature package defines functionality for Mass Spectrometry features. This includes functions to group (LC-MS) features based on some of their properties, such as retention time (coeluting features), or correlation of signals across samples. This packge hence allows to group features, and its results can be used as an input for the `QFeatures` package which allows to aggregate abundance levels of features within each group. This package defines concepts and functions for base and common data types, implementations for more specific data types are expected to be implemented in the respective packages (such as e.g. `xcms`). All functionality of this package is implemented in a modular way which allows combination of different grouping approaches and enables its re-use in other R packages.

Depends R (>= 4.1)

Imports methods, ProtGenerics (>= 1.23.5), MsCoreUtils, SummarizedExperiment, stats

Suggests testthat, roxygen2, BiocStyle, pheatmap, knitr, rmarkdown

License Artistic-2.0

Encoding UTF-8

LazyData no

BugReports https://github.com/RforMassSpectrometry/MsFeatures/issues

URL https://github.com/RforMassSpectrometry/MsFeatures

biocViews Infrastructure, MassSpectrometry, Metabolomics

RoxygenNote 7.1.1

Roxygen list(markdown=TRUE)

VignetteBuilder knitr

Collate 'corRows.R' 'grouping-functions.R' 'hidden_aliases.R' 'AllGenerics.R' 'AbundanceSimilarityParam.R' 'SimilarRtimeParam.R' 'featureGroups.R' 'se.R'

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

git_branch RELEASE_3_13

git_last_commit 7c06b89

git_last_commit_date 2021-05-19

Date/Publication 2021-09-19

Author Johannes Rainer [aut, cre] (<https://orcid.org/0000-0002-6977-7147>)

Maintainer Johannes Rainer <Johannes.Rainer@eurac.edu>

R topics documented:

`corRows` *Correlate rows of a numeric matrix*

Description

`corRows` is a simple function to perform a pairwise correlation between **rows** of a numeric matrix by calling `stats::cor()` on the transposed input matrix `x`.

Usage

```
corRows(
  x,
  y = NULL,
  use = "pairwise.complete.obs",
  method = c("pearson", "kendall", "spearman"),
  ...
)
```

Arguments

x	numeric matrix.
y	not supported (ignored).
use	see information for parameter use in stats::cor(). Defaults to use = "pairwise.complete.obs".
method	see information for parameter method in stats::cor().
...	additional parameters (ignored).

Value

matrix with correlation coefficients between rows in x.

Author(s)

Johannes Rainer

Examples

```
## Define a simple numeric matrix
x <- rbind(
    c(12, 34, 231, 234, 9, 5, 7),
    c(900, 900, 800, 10, 12, 9, 4),
    c(25, 70, 400, 409, 15, 8, 4),
    c(12, 13, 14, 15, 16, 17, 18),
    c(14, 36, 240, 239, 12, 7, 8)
    )

corRows(x)

corRows(x, method = "spearman")
```

---

featureGroups *Get or set feature group definitions from an object*

Description

featureGroups and featureGroups<- allow to extract or set the feature definitions from the input object. The implementations for SummarizedExperiment() get or set the content of a column named "feature_group" in the object's rowData.

This method should be implemented for all other object for which a groupFeatures() method is defined.

Usage

```
featureGroups(object, ...)

featureGroups(object) <- value

## S4 method for signature 'SummarizedExperiment'
featureGroups(object)

## S4 replacement method for signature 'SummarizedExperiment'
featureGroups(object) <- value
```

Arguments

object	the input object. In the MsFeatures package this method is implemented for SummarizedExperiment.
...	ignored.
value	the new value for the *feature groups* variable.

Value

a character with the group assignment of the features. Has to have the same length as there are features in object.

Author(s)

Johannes Rainer

Examples

```
## Load the test SummarizedExperiment
library(SummarizedExperiment)
data(se)

## No column "feature_group" present in the object, this NA is returned
featureGroups(se)

## Add a column "feature_group" to the `rowData` of the object
rowData(se)$feature_group <- seq_len(nrow(rowData(se)))

featureGroups(se)
```

---

groupClosest *Group values with differences below threshold*

Description

Group values with a difference between them being smaller than a user defined threshold. This function uses the `groupSimilarityMatrix()` function to create groups with smallest differences between its members. Differences between **all** members of one group are below the user defined threshold `maxDiff`. This is a more stringent grouping than what `groupConsecutive()` performs leading thus to smaller groups (with smaller differences between its members).

Usage

```
groupClosest(x, maxDiff = 1)
```

Arguments

`x`	`numeric` of values that should be grouped.
`maxDiff`	`numeric(1)` defining the threshold for difference between values in `x` to be grouped into the same group.

Value

`integer` with the group assignment (values grouped together have the same return value).

Author(s)

Johannes Rainer

See Also

Other grouping operations: `groupConsecutive()`, `groupSimilarityMatrix()`

Examples

```
x <- c(1.1, 1.9, 2.2)
groupClosest(x)
## Although the difference between the 1st and 2nd element would be smaller
## than the threshold, they are not grouped because the difference between
## the 2nd and 3rd element is even smaller. The first element is also not
## put into the same group, because it has a difference > diffRt to the 3rd
## element.

x <- c(1.1, 1.5, 1.7, 2.3, 2.7, 4.3, 4.4, 4.9, 5.2, 5.4, 5.8, 6, 7,
    9, 9.5, 15)

groupClosest(x)
```

`groupConsecutive` *Grouping of sorted values into sets with smallest differences*

Description

`groupConsecutive` groups **sorted** values in `x` for which the difference is smaller than `maxDiff`. As a result, the mean difference between the groups will always be larger than `maxDiff`, but difference between individual values within the same group (e.g. between the first and last) can be larger `maxDiff`.

In detail, from the sorted `x`, the function starts from the smallest value defining the first group as the one containing all values in `x` with a difference to this first value which is <= maxDiff. The next group is the defined based on the next larger value that is not part of the first group and includes all values with a difference <= maxDiff to this value. For values fulfilling this criteria but being already part of a previous group, the differences to the mean value of the current group and to the mean of previous groups are compared and values are assigned to the group to which they have the smallest difference.

Example: values 1.1, 1.9, 2.2 should be grouped with a `maxDiff = 1`. The first group is defined to include all values for which the difference to the first value (`1.1`) is smaller `maxDiff`. Thus, the first group is defined to contain values 1.1 and 1.9. Then the next group is defined based on the next larger value not part of any group, `2.2`. This group contains values `1.9` and `2.2` with the value `1.9` being already assigned to the first group. The difference between this value `1.9` and the mean of the current group (mean(c(1.9, 2.2)) is then compared to the difference of `1.9` to the mean value of the group `1.9` is already part of (which is `mean(c(1.1,1.9))`). Since the difference to the second group is smaller, `1.9` is removed from the first group and assigned to the second one.

Usage

```
groupConsecutive(x, maxDiff = 1)
```

Arguments

`x`	`numeric` of values that should be grouped.
`maxDiff`	`numeric(1)` defining the threshold for difference between values in `x` to be grouped into the same group.

Value

`integer` with the group assignment (values grouped together have the same return value).

Note

The difference between consecutive (ordered) values within a defined group is always <= maxDiff, but the difference between e.g. the first and the last of the (ordered) values can be larger than `maxDiff`. See `groupClosest()` for a more stringent grouping function.

Author(s)

Johannes Rainer

See Also

Other grouping operations: `groupClosest()`, `groupSimilarityMatrix()`

Examples

```
## The example described above
x <- c(1.1, 1.9, 2.2)
groupConsecutive(x)

x <- c(1.1, 1.5, 1.7, 2.3, 2.7, 4.3, 4.4, 4.9, 5.2, 5.4, 5.8, 6, 7,
    9, 9.5, 15)

groupConsecutive(x)
## value 5.2 was initially grouped with 4.3 (because their difference is
## smaller 1, but then re-grouped together with 5.4 because the difference
## between 5.4 (the next value outside the group of 4.3) and 5.2 is smaller
## than its difference to the mean value of the group for value 4.3

## Example for a case in which values are NOT grouped into the same group
## even if the difference between them is <= maxDiff
a <- c(4.9, 5.2, 5.4)
groupConsecutive(a, maxDiff = 0.3)
```

---

groupFeatures *General Feature Grouping Concept*

Description

This documentation describes the general concepts of feature grouping, which can be achieved by the different approaches described further below.

The main function for the stepwise feature grouping is `groupFeatures`. The selection of the actual grouping algorithm (along with the definition of its parameters) is done by passing the respective *parameter* object, along with the object containing the input data and optional additional arguments, to the `groupFeatures` method.

Usage

```
groupFeatures(object, param, ...)
```

Arguments

`object`	input data object on which (with which data) the feature grouping should be performed.
`param`	parameter object which type defines the selection of the grouping algorithm.
`...`	additional arguments to be passed to the grouping algorithm.

Value

Depending on the implementation and the input object. Generally the input object with grouping results added. See respective help pages for more information.

Single-step Feature Grouping

Each feature grouping algorithm can be applied individually as a single-step approach, e.g. by grouping features only on a single feature property, such as the retention time. Additional feature grouping approaches might also be implemented that consider combination of different MS feature properties in a single clustering process.

Stepwise Feature Grouping Refinement

Stepwise feature grouping evaluates a single property of MS features (such as their retention time or abundances) at a time to define the feature groups. Each subsequent grouping step *builds* on the previous one by eventually sub-grouping each feature group, if needed. Thus, feature groups get refined in each step. As an example, grouping of features based on a similar retention time would loosely group features from all compounds eluting at about the same time from a e.g. liquid chromatography run. This obviously would also group features representing ions from different co-eluting compounds. Thus, calling `groupFeatures` on the previous feature grouping result with a different parameter object would *refine* these initial feature groups, splitting them based on another property of the features (such as correlation of feature abundances across samples).

The advantage of the stepwise approach is that results can be evaluated after each grouping step and parameters adapted if needed. Also, it provides flexibility by allowing to change the order of grouping approaches, or skip individual steps if not suitable for the available data or the experimental setup.

The major disadvantage is that a wrong group assignment in one of the initial steps can not be *corrected* for in later steps.

Author(s)

Johannes Rainer

See Also

`featureGroups()` for the function to extract (defined) feature groups from a result object.

Examples

```
## For examples please refer to the help pages of the `SimilarRtimeParam` or
## `AbundanceSimilarityParam` objects.
NULL
```

groupFeatures-similar-abundance
Group features based on abundance similarities across samples

Description

Group features based on similar abundances (i.e. *feature values*) across samples. Parameter `subset` allows to define a sub set of samples on which the similarity calculation should be performed. It might for example be better to exclude QC samples from the analysis because feature values are supposed to be constant in these samples.

The function first calculates a nxn similarity matrix with n being the number of features and subsequently groups features for which the similarity is higher than the user provided threshold. Parameter `simFun` allows to specify the function to calculate the pairwise similarities on the feature values (eventually transformed by the function specified with parameter `transform`). `simFun` defaults to a function that uses `cor` to calculate similarities between rows in `object` but any function that calculates similarities between rows and that returns a (symmetric) numeric similarity matrix can be used.

If `object` is a `SummarizedExperiment()`: if a column `"feature_group"` is found in `SummarizedExperiment::colData()` feature groups defined in that column are further sub-grouped with this method. See `groupFeatures()` for the general concept of this feature grouping.

Parameter `groupFun` allows to specify the function to group the features based on the similarity function. It defaults to `groupSimilarityMatrix`. See `groupSimilarityMatrix()` for details.

Additional settings for the `groupFun` and `simFun` functions can be passed to the **parameter object** with the `...` in the `AbundanceSimilarityParam` constructor function. Other additional parameters specific for the type of `object` can be passed *via* `...` in the `groupFeatures` call.

Usage

```
AbundanceSimilarityParam(
  threshold = 0.9,
  simFun = corRows,
  groupFun = groupSimilarityMatrix,
  subset = integer(),
  transform = identity,
  ...
)

## S4 method for signature 'matrix,AbundanceSimilarityParam'
groupFeatures(object, param, ...)

## S4 method for signature 'SummarizedExperiment,AbundanceSimilarityParam'
groupFeatures(object, param, i = 1L, ...)
```

Arguments

threshold	numeric(1) defining the (similarity) threshold to be used for the feature grouping. This parameter is passed to the groupFun function.
simFun	function to be used to calculate (pairwise) similarities (between **rows**). Defaults to simFun = corRows. See description or corRows() for more details.
groupFun	function to group features based on the calculated similarity matrix. Defaults to groupFun = groupSimilarityMatrix. See groupSimilarityMatrix() for details.
subset	integer or logical defining a subset of samples (at least 2) on which the similarity calculation should be performed. By default the calculation is performed on all samples.
transform	function to be used to transform feature abundances prior to the similarity calculation. Defaults to transform = identity. Alternatively, values could e.g. transformed into log2 scale with transform = log2.
...	for AbundanceSimilarityParam: optional parameters to be passed along to simFun and groupFun. For groupFeatures: optional parameters for the extraction/definition of the feature values from object.
object	object containing the feature abundances on which features should be grouped.
param	AbundanceSimilarityParam defining the settings for the grouping based on feature values.
i	for object being a SummarizedExperiment(): integer(1) or character(1) specifying either the index or name of the the *assay* in object that contains the feature values that should be used. Use assayNames() on object to list all available assays.

Value

for object being a SummarizedExperiment: a SummarizedExperiment with the grouping results added to a column "feature_group" in the object's rowData. For object being a matrix: an integer of length equal to the number of rows with the group identifiers.

Author(s)

Johannes Rainer

See Also

groupFeatures() for the general concept of feature grouping.

featureGroups() for the function to extract defined feature groups from a SummarizedExperiment.

Other feature grouping methods: groupFeatures-similar-rtime

Examples

```
## Define a simple numeric matrix on which we want to group the rows
x <- rbind(
    c(12, 34, 231, 234, 9, 5, 7),
```

```
    c(900, 900, 800, 10, 12, 9, 4),
    c(25, 70, 400, 409, 15, 8, 4),
    c(12, 13, 14, 15, 16, 17, 18),
    c(14, 36, 240, 239, 12, 7, 8),
    c(100, 103, 80, 2, 3, 1, 1)
    )

## Group rows based on similarity calculated with Pearson's correlation
## on the actual data values (without transforming them).
res <- groupFeatures(x, AbundanceSimilarityParam())
res

## Use Spearman's rho to correlate rows of the log2 transformed x matrix
res <- groupFeatures(x, AbundanceSimilarityParam(method = "spearman",
    transform = log2))
res

## Perform the grouping on a SummarizedExperiment
library(SummarizedExperiment)
data(se)

## Group features based on log2 transformed feature values in the first
## assay of the SummarizedExperiment
res <- groupFeatures(se, param = AbundanceSimilarityParam(threshold = 0.7,
    transform = log2))

featureGroups(res)

## Perform feature grouping only on a subset of rows/features:
featureGroups(res) <- NA_character_
featureGroups(res)[40:80] <- "FG"
res <- groupFeatures(res, AbundanceSimilarityParam(transform = log2))
featureGroups(res)
```

`groupFeatures-similar-rtime` *Group features based on approximate retention times*

Description

Group features based on similar retention time. This method is supposed to be used as an initial *crude* grouping of LC-MS features based on the median retention time of all their chromatographic peaks. All features with a difference in their retention time which is <= parameter `diffRt` of the parameter object are grouped together.

If `object` is a `SummarizedExperiment()`: if a column `"feature_group"` is found in `SummarizedExperiment::colData()` feature groups defined in that column are further sub-grouped with this method. See `groupFeatures()` for the general concept of this feature grouping. Also, it might be required to specify the column in the object's `rowData` containing the retention times with the `rtime` parameter (which defaults to `rtime = "rtime"`.

Parameter groupFun allows to specify the function that should be used for the actual grouping. Two possible choices are:

- groupFun = groupClosest (the default): this method creates groups of features with smallest differences in retention times between the individual group members. All differences between group members are < diffRt (in contrast to the other grouping functions listed below). See groupSimilarityMatrix() (which is used for the actual grouping on pairwise retention time differences) for more details.
- groupFun = groupConsecutive: the groupConsecutive() function groups values together if their difference is smaller than diffRt. This function iterates over the sorted retention times starting the grouping from the lowest value. If the difference of a feature to more than one group is smaller diffRt it is assigned to the group to which its retention time is closest (most similar) to the mean retention time of that group. This leads to smaller group sizes. Be aware that with this grouping differences in retention times between individual features within a group could still be larger diffRt. See groupConsecutive() for details and examples.
- groupFun = MsCoreUtils::group: this function consecutively groups elements together if their difference in retention time is smaller than diffRt. If two features are grouped into one group, also all other features with a retention time within the defined window to any of the two features are also included into the feature group. This grouping is recursively expanded which can lead, depending on diffRt, to very large feature groups spanning a large retention time window. See MsCoreUtils::group() for details.

Other grouping functions might be added in future. Alternatively it is also possible to provide a custom function for the grouping operation.

Usage

```
SimilarRtimeParam(diffRt = 1, groupFun = groupClosest)

## S4 method for signature 'numeric,SimilarRtimeParam'
groupFeatures(object, param, ...)

## S4 method for signature 'SummarizedExperiment,SimilarRtimeParam'
groupFeatures(object, param, rtime = "rtime", ...)
```

Arguments

diffRt	numeric(1) defining the retention time window within which features should be grouped. All features with a rtime difference smaller or equal than diffRt are grouped.
groupFun	function that can be used to group values. Defaults to groupFun = groupClosest. See description for details and alternatives.
object	input object that provides the retention times that should be grouped. The MsFeatures package defines a method for object being either a numeric or a SummarizedExperiment.
param	SimilarRtimeParam object with the settings for the method.
...	additional parameters passed to the groupFun function.
rtime	for object being a SummarizedExperiment(): character(1) specifying the column in rowData(object) that contains the retention time values.

Value

Depending on parameter `object`:

- for `object` being a `numeric`: returns a `factor` defining the feature groups.
- for `object` being `SummarizedExperiment`: returns the input object with the feature group definition added to a column `"feature_group"` in the result object's `rowData`.

Author(s)

Johannes Rainer

See Also

`groupFeatures()` for the general concept of feature grouping.

`featureGroups()` for the function to extract defined feature groups from a `SummarizedExperiment`.

Other feature grouping methods: `groupFeatures-similar-abundance`

Examples

```
## Simple grouping of a numeric vector.
##
## Define a numeric vector that could represent retention times of features
x <- c(2, 3, 4, 5, 10, 11, 12, 14, 15)

## Group the values using a `group` function. This will create larger
## groups.
groupFeatures(x, param = SimilarRtimeParam(2, MsCoreUtils::group))

## Group values using the default `groupClosest` function. This creates
## smaller groups in which all elements have a difference smaller than the
## defined `diffRt` with each other.
groupFeatures(x, param = SimilarRtimeParam(2, groupClosest))

## Grouping on a SummarizedExperiment
##
## load the test SummarizedExperiment object
library(SummarizedExperiment)
data(se)

## No feature groups defined yet
featureGroups(se)

## Determine the column that contains retention times
rowData(se)

## Column "rtmed" contains the (median) retention time for each feature
## Group features that are eluting within 10 seconds
res <- groupFeatures(se, SimilarRtimeParam(10), rtime = "rtmed")

featureGroups(res)
```

```
## Evaluating differences between retention times within each feature group
rts <- split(rowData(res)$rtmed, featureGroups(res))
lapply(rts, function(z) abs(diff(z)) <= 10)

## One feature group ("FG.053") has elements with a difference larger 10:
rts[["FG.053"]]
abs(diff(rts[["FG.053"]]))

## But the difference between the **sorted** retention times is < 10:
abs(diff(sort(rts[["FG.053"]])))

## Feature grouping with pre-defined feature groups: groupFeatures will
## sub-group the pre-defined feature groups, features with the feature group
## being `NA` are skipped. Below we perform the feature grouping only on
## features 40 to 70
fgs <- rep(NA, nrow(rowData(se)))
fgs[40:70] <- "FG"
featureGroups(se) <- fgs
res <- groupFeatures(se, SimilarRtimeParam(10), rtime = "rtmed")
featureGroups(res)
```

`groupSimilarityMatrix` *Group rows of a diagonal matrix using a threshold*

Description

This function groups elements (rows or columns) of a diagonal matrix, such as a pairwise correlation matrix or similarity matrix, with a value >= threshold. This creates clusters of elements in which **all** elements have a value >= threshold with **any** other element in that cluster. On a correlation matrix (such as created with `cor`) it will generate small clusters of highly correlated elements. Note however that single elements in one cluster could also have a correlation >= threshold to another element in another cluster. The average similarity to its own cluster will however be higher to that of the other.

Usage

```
groupSimilarityMatrix(x, threshold = 0.9, full = TRUE, ...)
```

Arguments

`x`	symmetrix `numeric matrix`.
`threshold`	`numeric(1)` above which rows in `x` should be grouped.
`full`	`logical(1)` whether the full matrix should be considered, or just the upper triangular matrix (including the diagonal).
`...`	ignored.

Details

The algorithm is defined as follows:

- all pairs of values in x which are >= threshold are identified and sorted decreasingly.
- starting with the pair with the highest correlation, groups are defined:
- if none of the two is in a group, both are put into the same new group.
- if one of the two is already in a group, the other is put into the same group if **all** correlations of it to that group are >= threshold (and are not NA).
- if both are already in the same group nothing is done.
- if both are in different groups: an element is put into the group of the other if a) all correlations of it to members of the other's group are not NA and >= threshold **and** b) the average correlation to the other group is larger than the average correlation to its own group.

This ensures that groups are defined in which all elements have a correlation >= threshold with each other and the correlation between members of the same group is maximized.

Value

integer same length than nrow(x), grouped elements (rows) defined by the same value.

Author(s)

Johannes Rainer

See Also

Other grouping operations: groupClosest(), groupConsecutive()

Examples

```
x <- rbind(
    c(1, 0.9, 0.6, 0.8, 0.5),
    c(0.9, 1, 0.7, 0.92, 0.8),
    c(0.6, 0.7, 1, 0.91, 0.7),
    c(0.8, 0.92, 0.91, 1, 0.9),
    c(0.5, 0.8, 0.7, 0.9, 1)
    )

groupSimilarityMatrix(x, threshold = 0.9)

groupSimilarityMatrix(x, threshold = 0.1)

## Add also a correlation between 3 and 2
x[2, 3] <- 0.9
x[3, 2] <- 0.9
x
groupSimilarityMatrix(x, threshold = 0.9)

## Add a higher correlation between 4 and 5
x[4, 5] <- 0.99
```

```
x[5, 4] <- 0.99
x
groupSimilarityMatrix(x, threshold = 0.9)

## Increase correlation between 2 and 3
x[2, 3] <- 0.92
x[3, 2] <- 0.92
x
groupSimilarityMatrix(x, threshold = 0.9) ## Don't break previous cluster!
```

---

se *Quantified LC-MS preprocessing result test data*

Description

The se variable is a SummarizedExperiment() object representing the results from a xcms-based pre-processing of an LC-MS untargeted metabolomics data set. The raw data files are provided in the faahKO package. The pre-processing of this data set is described in detail in the *xcms* vignette of the xcms package. This object was created from the XCMSnExp result object with the quantify method.

Examples

```
## Load the data
data(se)

library(SummarizedExperiment)

## Access row (feature) data
rowData(se)
```

Index