Package 'patRoon'

October 15, 2020

Type Package

Title Workflows for Mass-Spectrometry Based Non-Target Analysis

Version 1.0.2

Description Provides an easy-to-use interface to a mass spectrometry based non-target analysis workflow. Various (open-source) tools are combined which provide algorithms for extraction and grouping of features, extraction of MS and MS/MS data, automatic formula and compound annotation and grouping related features to components. In addition, various tools are provided for e.g. data preparation and cleanup, plotting results and automatic reporting.

License GPL-3

LazyData TRUE

RoxygenNote 7.1.0

URL https://github.com/rickhelmus/patRoon

BugReports https://github.com/rickhelmus/patRoon/issues

Encoding UTF-8

Depends R (¿= 3.5.0)

SystemRequirements GNU make

Imports methods,
checkmate (¿= 1.8.5),
Rcpp,
VennDiagram,
UpSetR,
stats,
utils,
parallel,
grid,
graphics,
RColorBrewer,
data.table,
withr,
digest,

enviPick,
XML,
DBI,
RSQLite,
fst,
processx,
tools,
MSnbase,
xcms,
cluster,
fastcluster,
gplots,
heatmaply,
dynamicTreeCut,
dendextend,
igraph,
visNetwork,
rJava,
rcdk,
fingerprint,
mzR,
circlize,
miniUI,
plotly,
rhandsontable,
rstudioapi,
htmlwidgets,
shiny,
templates,
CAMERA,
RAMClustR,
enviPat,
nontarget,
knitr,
rmarkdown,
flexdashboard,
DT,
magrittr,
kableExtra,
R.utils,
magick,
glue,
ggplot2,
ggrepel,
cowplot,
jsonlite,
Rdpack,
rsm

Suggests RDCOMClient,
metfRag,
testthat,
rlang,
vdiffr,
patRoonData,
devtools,
covr,
DiagrammeR,
DiagrammeRsvg,
rsvg,
pkgload

LinkingTo Rcpp

RdMacros Rdpack

Collate 'generics.R'
'cache.R'
'main.R'
'workflow-step.R'
'features.R'
'feature_groups.R'
'EIC-tool.R'
'RcppExports.R'
'utils-adduct.R'
'adduct.R'
'components.R'
'components-camera.R'
'components-intclust.R'
'components-nontarget.R'
'components-ramclustr.R'
'compounds.R'
'compounds-cluster.R'
'formulas.R'
'mspeaklists.R'
'compounds-metfrag.R'
'utils-sirius.R'
'compounds-sirius.R'
'convert.R'
'deprecated.R'
'utils-IPO.R'
'doe-optimizer.R'
'feature_groups-bruker.R'
'feature_groups-comparison.R'
'feature_groups-envimass.R'
'feature_groups-filter.R'
'feature_groups-openms.R'
'feature_groups-optimize.R'
'feature_groups-optimize-openms.R'
'feature_groups-optimize-xcms.R'

'feature_groups-optimize-xcms3.R'
'feature_groups-screening.R'
'feature_groups-tasq.R'
'feature_groups-xcms.R'
'feature_groups-xcms3.R'
'utils-bruker.R'
'features-bruker.R'
'features-envipick.R'
'features-openms.R'
'features-optimize.R'
'features-optimize-envipick.R'
'features-optimize-openms.R'
'features-optimize-xcms.R'
'features-optimize-xcms3.R'
'features-tasq.R'
'features-xcms.R'
'features-xcms3.R'
'formulas-bruker.R'
'formulas-genform.R'
'formulas-sirius.R'
'mspeaklists-bruker.R'
'utils-mzr.R'
'mspeaklists-mzr.R'
'multi-process.R'
'project-tool.R'
'report.R'
'utils-checkmate.R'
'utils-compounds.R'
'utils-exported.R'
'utils-formulas.R'
'utils-mol.R'
'utils-mspeaklists.R'
'utils-optimize.R'
'utils-plot.R'
'utils-xcms.R'
'utils.R'
'zzz.R'

VignetteBuilder knitr

R topics documented:

`patRoon-package` *Workflow solutions for mass-spectrometry based non-target analysis.*

Description

Provides an easy-to-use interface to a mass spectrometry based non-target analysis workflow. Various (open-source) tools are combined which provide algorithms for extraction and grouping of features, extraction of MS and MS/MS data, automatic formula and compound annotation and grouping related features to components. In addition, various tools are provided for e.g. data preparation and cleanup, plotting results and automatic reporting.

Package options

The following package options (see `options`) can be set:

- patRoon.cache.mode: A character setting the current caching mode: "save" and "load" will only save/load results to/from the cache, "both" (default) will do both and "none" to completely disable caching. This option can be changed anytime, which might be useful, for instance, to temporarily disable cached results before running a function.
- patRoon.cache.fileName: a character specifying the name of the cache file (default is 'cache.sqlite').
- patRoon.maxProcAmount: The maximum number of processes that should be initiated in parallel. A good starting point is the number of physical cores, which is the default as detected by detectCores.
- patRoon.path.pwiz: The path in which the ProteoWizard binaries are installed. If unset an attempt is made to find this directory from the Windows registry and 'PATH' environment variable.
- patRoon.path.GenForm: The path to the GenForm executable. If not set (the default) the internal GenForm binary is used. Only set if you want to override the executable.
- patRoon.path.MetFragCL: The complete file path to the MetFrag CL 'jar' file that *must* be set when using generateCompoundsMetfrag. Example: "C:/MetFrag2.4.2-CL.jar".
- patRoon.path.MetFragCompTox: The complete file path to the CompTox database 'csv' file. See generateCompounds for more details.
- patRoon.path.MetFragPubChemLite: The complete file path to the PubChem database 'csv' file. See generateCompounds for more details.
- patRoon.path.SIRIUS: The directory in which SIRIUS is installed. Unless the binaries can be located via the 'PATH' environment variable, this *must* be set when using generateFormulasSIRIUS or generateCompoundsSIRIUS. Example: "C:/sirius-win64-3.5.1".
- patRoon.path.OpenMS: The path in which the OpenMS binaries are installed. Usually the location is added to the 'PATH' environment variable when OpenMS is installed, in which case this option can be left empty.
- patRoon.path.pngquant: The path of the pngquant binary that is used when optimizing '.png' plots generated by reportHTML (with optimizePng set to TRUE). If the binary can be located through the 'PATH' environment variable this option can remain empty. Note that some of the functionality of reportHTML only locates the binary through the 'PATH' environment variable, hence, it is recommended to set up 'PATH' instead.
- patRoon.path.obabel: The path in which the OpenBabel binaries are installed. Usually the location is added to the 'PATH' environment variable when OpenBabel is installed, in which case this option can be left empty.

Author(s)

Maintainer: Rick Helmus <r.helmus@uva.nl> (ORCID)

Other contributors:

- Olaf Brock (ORCID) [contributor]
- Vittorio Albergamo (ORCID) [contributor]
- Andrea Brunner (ORCID) [contributor]
- Emma Schymanski (ORCID) [contributor]

See Also

Useful links:

- https://github.com/rickhelmus/patRoon
- Report bugs at https://github.com/rickhelmus/patRoon/issues

---

`adduct-class` *Generic adduct class*

Description

Objects from this class are used to specify adduct information in an algorithm independent way.

Usage

```
adduct(...)

## S4 method for signature 'adduct'
show(object)

## S4 method for signature 'adduct'
as.character(x, format = "generic")
```

Arguments

`x, object`	An `adduct` object.
`format`	A `character` that specifies the source format. `"generic"` is an internally used generic format that supports full textual conversion. Examples: `"[M+H]+"`, `"[2M+H]+"`, `"[M+3H]3+"`. `"sirius"` Is the format used by `SIRIUS`. It is similar to `generic` but does not allow multiple charges/molecules. See the SIRIUS manual for more details. `"genform"` and `"metfrag"` support fixed types of adducts which can be obtained with the `GenFormAdducts` and `MetFragAdducts` functions, respectively.
`...`	Any of `add`, `sub`, `molMult` and/or `charge`. See Slots.

Methods (by generic)

- `show`: Shows summary information for this object.
- `as.character`: Converts an `adduct` object to a specified `character` format.

Slots

`add,sub` A `character` with one or more formulas to add/subtract.

`molMult` How many times the original molecule is present in this molecule (*e.g.* for a dimer this would be '2'). Default is '1'.

`charge` The final charge of the adduct (default '1').

See Also

`as.adduct` for easy creation of `adduct` objects and adduct utilities for other adduct functionality.

Examples

```
adduct("H") # [M+H]+
adduct(sub = "H", charge = -1) # [M-H]-
adduct(add = "K", sub = "H2", charge = -1) # [M+K-H2]+
adduct(add = "H3", charge = 3) # [M+H3]3+
adduct(add = "H", molMult = 2) # [2M+H]+

as.character(adduct("H")) # returns "[M+H]+"
```

---

adduct-utils *Adduct utilities*

Description

Several utility functions to work with adducts.

Usage

```
GenFormAdducts()

MetFragAdducts()

as.adduct(x, format = "generic", isPositive = NULL)

calculateIonFormula(formula, adduct)

calculateNeutralFormula(formula, adduct)
```

Arguments

`x`	The object that should be converted. Should be a `character` string, a `numeric MetFrag` adduct identifier (`adduct_mode` column obtained with `MetFragAdducts`) or an `adduct` object (in which case no conversion occurs).

format — A character that specifies the source format.

"generic" is an internally used generic format that supports full textual conversion. Examples: "[M+H]+", "[2M+H]+", "[M+3H]3+".

"sirius" Is the format used by SIRIUS. It is similar to generic but does not allow multiple charges/molecules. See the SIRIUS manual for more details.

"genform" and "metfrag" support fixed types of adducts which can be obtained with the GenFormAdducts and MetFragAdducts functions, respectively.

isPositive — A logical that specifies whether the adduct should be positive. Should only be set when format="metfrag" and x is a numeric identifier.

formula — A character vector with formulae to convert.

adduct — An adduct object (or something that can be converted to it with as.adduct). Examples: "[M-H]-", "[M+Na]+".

Details

GenFormAdducts returns a table with information on adducts supported by GenForm.

MetFragAdducts returns a table with information on adducts supported by MetFrag.

as.adduct Converts an object in to an adduct object.

calculateIonFormula Converts one or more neutral formulae to adduct ions.

calculateNeutralFormula Converts one or more adduct ions to neutral formulae.

Examples

```
as.adduct("[M+H]+")
as.adduct("[M+H2]2+")
as.adduct("[2M+H]+")
as.adduct("[M-H]-")
as.adduct("+H", format = "genform")
as.adduct(1, isPositive = TRUE, format = "metfrag") # MetFrag adduct ID 1 --> returns [M+H]+

calculateIonFormula("C2H4O", "[M+H]+") # C2H5O
calculateNeutralFormula("C2H5O", "[M+H]+") # C2H4O
```

---

analysis-information *Analysis information*

Description

Required information for analyses that should be processed and utilities to automatically generate this information.

Usage

```
generateAnalysisInfo(
  paths,
  groups = "",
  blanks = "",
  concs = NULL,
  formats = MSFileFormats()
)

generateAnalysisInfoFromEnviMass(path)
```

Arguments

`paths`	A character vector containing one or more file paths that should be used for finding the analyses.
`groups, blanks`	An (optional) character vector containing replicate groups and references, respectively (will be recycled). If `groups` is an empty character string (`""`) the analysis name will be set as replicate group.
`concs`	An optional numeric vector containing concentration values for each analysis. Can be `NA` if unknown. If the length of `concs` is less than the number of analyses the remainders will be set to `NA`. Set to `NULL` to not include concentration data.
`formats`	A character vector of analyses file types. Valid values are: `Bruker`, `mzXML` and `mzML`.
`path`	The path of the enviMass project.

Details

Several properties need to be known about analyses that should be processed during various workflow steps such as finding features, averaging intensities of feature groups and blank subtraction. This information should be made available with an 'analysis info' object, which is a `data.frame` containing the following columns:

- `path` the full path to the analysis.
- `analysis` the filename **without** extension. Must be **unique**, even if the `path` is different.
- `group` name of *replicate group*. A replicate group is used to group analyses together that are replicates of each other. Thus, the `group` column for all analyses considered to be belonging to the same replicate group should have an equal (but unique) value. Used for *e.g.* avaraging and `filter`.
- `blank`: all analyses within this replicate group are used by `filter` for blank subtraction. Multiple entries can be entered by separation with a comma.
- `conc` a numeric value specifying the 'concentration' of the analysis. This can be actually any kind of quantitative value such as exposure time, dilution factor or anything else which may be used to form a linear relationship.

Most functionality requires the data files to be in either the '`.mzXML`' or '`.mzML`' format. Functionality that utilizes Bruker DataAnalysis (*e.g.* `findFeaturesBruker`) may only work with data files in the proprietary '`.d`' format. Therefore, when tools with varying requirements are mixed (a common scenario), the data files also need to be present in the required formats. To deal with this situation the data files with varying formats should all be placed in the same path that was used to specify the location of the analysis.

Whether analysis data files are present in multiple formats or not, each analysis should only be entered *once*. The `analysis` column is used as a basename to automatically find back the data file with the required format, hence, analysis names should be specified as the file name without its file extension.

The `group` column is *mandatory* and needs to be filled in for each analysis. The `blank` column should also be present, however, these may contain empty character strings (`""`) for analyses where no blank subtraction should occur. The `conc` column is only required when obtaining regression information is desired with the `as.data.table` method.

`generateAnalysisInfo` is an utility function that automatically generates an analysis information object. It will collect all datafiles from given file paths and convert the filenames into valid analysis names (*i.e.* without extensions such as '`.d`' and '`.mzML`'). Duplicate analyses, which may appear when datafiles with different file extension ('`.d`', '`.mzXML`' and/or '`.mzML`') are present, will be automatically removed.

`generateAnalysisInfoFromEnviMass` loads analysis information from an **enviMass** project. Note: this funtionality has only been tested with older versions of **enviMass**.

`bruker-utils` *Bruker DataAnalysis utilities*

Description

Miscellaneous utility functions which interface with Bruker DataAnalysis

Usage

```
showDataAnalysis()

setDAMethod(anaInfo, method, close = TRUE)

revertDAAnalyses(anaInfo, close = TRUE, save = close)

recalibrarateDAFiles(anaInfo, close = TRUE, save = close)

getDACalibrationError(anaInfo)

addDAEIC(
  analysis,
  path,
  mz,
  mzWindow = 0.005,
```

```
  ctype = "EIC",
  mtype = "MS",
  polarity = "both",
  bgsubtr = FALSE,
  fragpath = "",
  name = NULL,
  hideDA = TRUE,
  close = FALSE,
  save = close
)

addAllDAEICs(
  fGroups,
  mzWindow = 0.005,
  ctype = "EIC",
  bgsubtr = FALSE,
  name = TRUE,
  onlyPresent = TRUE,
  hideDA = TRUE,
  close = FALSE,
  save = close
)
```

Arguments

anaInfo	Analysis info table
method	The full path of the DataAnalysis method.
close, save	If TRUE then Bruker files are closed and saved after processing with DataAnalysis, respectively. Setting close=TRUE prevents that many analyses might be opened simultaneously in DataAnalysis, which otherwise may use excessive memory or become slow. By default save is TRUE when close is TRUE, which is likely what you want as otherwise any processed data is lost.
analysis	Analysis name (without file extension).
path	path of the analysis.
mz	m/z (Da) value used for the chromatographic trace (if applicable).
mzWindow	m/z window (in Da) used for the chromatographic trace (if applicable).
ctype	Type of the chromatographic trace. Valid options are: "EIC" (extracted ion chromatogram), "TIC" (total ion chromatogram, only for addDAEIC) and "BPC" (Base Peak Chromatogram).
mtype	MS filter for chromatographic trace. Valid values are: "all", "MS", "MSMS", "allMSMS" and "BBCID".
polarity	Polarity filter for chromatographic trace. Valid values: "both", "positive" and "negative".
bgsubtr	If TRUE then background subtraction ('Spectral' algorithm) will be performed.

`fragpath`	Precursor *m/z* used for MS/MS traces (`""` for none).
`name`	For `addDAEIC`: the name for the chromatographic trace. For `addAllEICs`: `TRUE` to automatically set EIC names. Set to `NULL` for none.
`hideDA`	Hides DataAnalysis while adding the chromatographic trace (faster).
`fGroups`	The `featureGroups` object for which EICs should be made.
`onlyPresent`	If `TRUE` then EICs are only generated for analyses where the feature was detected.

Details

These functions communicate directly with Bruker DataAnalysis to provide various functionality, such as calibrating and exporting data and adding chromatographic traces. For this the **RDCOMClient** package is required to be installed.

`showDataAnalysis` makes a hidden DataAnalysis window visible again. Most functions using DataAnalysis will hide the window during processing for efficiency reasons. If the window remains hidden (*e.g.* because there was an error) this function can be used to make it visible again. This function can also be used to start DataAnalysis if it is not running yet.

`setDAMethod` Sets a given DataAnalysis method ('`.m`' file) to a set of analyses. **NOTE**: as a workaround for a bug in DataAnalysis, this function will save(!), close and re-open any analyses that are already open prior to setting the new method. The `close` argument only controls whether the file should be closed after setting the method (files are always saved).

`revertDAAnalyses` Reverts a given set of analyses to their unprocessed raw state.

`recalibrarateDAFiles` Performs automatic mass recalibration of a given set of analyses. The current method settings for each analyses will be used.

`getDACalibrationError` is used to obtain the standard deviation of the current mass calibration (in ppm).

`addDAEIC` adds an Extracted Ion Chromatogram (EIC) or other chromatographic trace to a given analysis which can be used directly with DataAnalysis.

`addAllDAEICs` adds Extracted Ion Chromatograms (EICs) for all features within a `featureGroups` object.

Value

`getDACalibrationError` returns a `data.frame` with a column of all analyses (named `analysis`) and their mass error (named `error`).

See Also

`analysis-information`

clearCache *Clearing of cached data*

Description

Remove (part of) the cache database used to store (intermediate) processing results.

Usage

```
clearCache(what = NULL, file = NULL)
```

Arguments

what	This argument describes what should be done. When what = NULL this function will list which tables are present along with an indication of their size (database rows). If what = "all" then the complete file will be removed. Otherwise, what should be a character string (a regular expression) that is used to match the table names that should be removed.
file	The cache file. If NULL then the value of the patRoon.cache.fileName option is used.

Details

This function will either remove one or more tables within the cache sqlite database or simply wipe the whole cache file. Removing tables will VACUUM the database, which may take some time for large cache files.

component-generation *Grouping feature groups in components*

Description

Functionality to automatically group related feature groups (*e.g.* isotopes, adducts and homologues) to assist and simplify compound annotation.

Usage

```
## S4 method for signature 'featureGroups'
generateComponents(fGroups, algorithm, ...)

generateComponentsCAMERA(
  fGroups,
  ionization,
  onlyIsotopes = FALSE,
  minSize = 2,
  relMinReplicates = 0.5,
```

```
  extraOpts = NULL
)

generateComponentsIntClust(
  fGroups,
  method = "complete",
  metric = "euclidean",
  normFunc = max,
  average = TRUE,
  maxTreeHeight = 1,
  deepSplit = TRUE,
  minModuleSize = 1
)

generateComponentsNontarget(
  fGroups,
  ionization,
  rtRange = c(-120, 120),
  mzRange = c(5, 120),
  elements = c("C", "H", "O"),
  rtDev = 30,
  absMzDev = 0.002,
  absMzDevLink = absMzDev * 2,
  extraOpts = NULL,
  traceHack = all(R.Version()[c("major", "minor")] >= c(3, 4))
)

generateComponentsRAMClustR(
  fGroups,
  st = NULL,
  sr = NULL,
  maxt = 12,
  hmax = 0.3,
  normalize = "TIC",
  ionization,
  absMzDev = 0.002,
  relMzDev = 5,
  minSize = 2,
  relMinReplicates = 0.5,
  RCExperimentVals = list(design = list(platform = "LC-MS"), instrument =
    list(ionization = ionization, MSlevs = 1)),
  extraOptsRC = NULL,
  extraOptsFM = NULL
)
```

Arguments

fGroups	featureGroups object for which components should be generated.
algorithm	A character string describing the algorithm that should be used: "ramclustr",

"camera", "nontarget", "intclust"

`...` Any parameters to be passed to the selected component generation algorithm.

`ionization` Which ionization polarity was used to generate the data: should be "positive" or "negative".

`onlyIsotopes` Logical value. If TRUE only isotopes are considered when generating components (faster). Corresponds to quick argument of CAMERA::annotate.

`minSize` The minimum size of a component. Smaller components than this size will be removed. For RAMClustR: sets the minModuleSize argument to ramclustR. See note below.

`relMinReplicates`
Feature groups within a component are only kept when they contain data for at least this (relative) amount of replicate analyses. For instance, '0.5' means that at least half of the replicates should contain data for a particular feature group in a component. In this calculation replicates that are fully absent within a component are not taken in to account. See note below.

`extraOpts` Named character vector with extra arguments directly passed to homol.search (generateComponentsNontarget) or CAMERA::annotate (generateComponentsCAMERA). Set to NULL to ignore.

`method` Clustering method that should be applied (passed to hclust).

`metric` Distance metric used to calculate the distance matrix (passed to daisy).

`normFunc` Function that should be used for normalization of data. Intensitity values of a feature group will be divided by the result of this function when it is called with all intensity values of that feature group. For example, when max is used normalized intensities will be between zero and one.

`average` If TRUE then all intensitity data will be averaged for each replicate group.

`maxTreeHeight, deepSplit, minModuleSize`
Arguments used by cutreeDynamicTree.

`rtRange` A numeric vector containing the minimum and maximum retention time (in seconds) between homologues. Series are always considered from low to high m/z, thus, a negative minimum retention time allows detection of homologous series with increasing m/z and decreasing retention times. These values set the minrt and maxrt arguments of homol.search.

`mzRange` A numeric vector specifying the minimum and maximum m/z increment of a homologous series. Sets the minmz and maxmz arguments of homol.search.

`elements` A character vector with elements to be considered for detection of repeating units. Sets the elements argument of homol.search function.

`rtDev, absMzDev, relMzDev`
Maximum deviation for retention time or absolute/relative m/z.

For generateComponentsRAMClustR: Sets the mzabs.error and ppm.error arguments to do.findmain.

For generateComponentsNontarget: Sets the rttol and mztol arguments of homol.search.

absMzDevLink — Maximum absolute m/z deviation when linking series. This should usually be a bit higher than absMzDev to ensure proper linkage.

traceHack — Currently homol.search does not work with R '>3.3.3'. This flag, which is enabled by default on these R versions, implements a (messy) workaround (more details here).

st, sr, maxt, hmax, normalize — Arguments to tune the behaviour of feature group clustering. See their documentation from ramclustR. When st is NULL it will be automatically calculated as the half of the median for all chromatographic peak widths.

RCExperimentVals — A named list containing two more lists: design and instrument. These are used to construct the ExpDes argument passed to ramclustR.

extraOptsRC, extraOptsFM — Named list with further arguments to be passed to ramclustR and do.findmain. Set to NULL to ignore.

Details

Several algorithms are provided to group feature groups that are related in some (chemical) way to each other. These components generally include adducts, isotopes, in-source fragments and homologues. The linking of this data is generally useful to provide more information for compound annotation and reduce the data size and thus complexity.

generateComponents is a generic function that will generate components using one of the supported algorithms. The actual functionality is provided by algorithm specific functions such as generateComponentsRAMClustR and generateComponentsNontarget. While these functions may be called directly, generateComponents provides a generic interface and is therefore usually preferred.

generateComponentsCAMERA provides an interface to CAMERA which is used to generate components from known adducts, isotopes and in-source fragments. The specified featureGroups object is automatically converted to an xcmsSet object using getXCMSSet.

generateComponentsIntClust generates components based on intensity profiles of feature groups. Hierarchical clustering is performed on normalized (and optionally replicate averaged) intensity data and the resulting dendrogram is automatically cut with cutreeDynamicTree. The distance matrix is calculated with daisy and clustering is performed with hclust. The clustering of the resulting components can be further visualized and modified using the methods defined for componentsIntClust.

generateComponentsNontarget uses the nontarget R package to generate components by unsupervised detection of homologous series. In the first step the homol.search function is used to detect all homologues within each replicate group (analyses within each replicate group are averaged prior to detection). Then, homologous series across replicate groups are merged in case of full overlap or when merging of partial overlapping series causes no conflicts.

generateComponentsRAMClustR uses RAMClustR to generate components from feature groups which follow similar chromatographic retention profiles, but are not necessarily restricted to known rules (*e.g.* adducts or isotopes). This method uses the ramclustR functions for generating the components, whereas do.findmain is used for annotation.

Value

A components (derived) object containing all generated components.

Note

For generateComponentsCAMERA and generateComponentsRAMClustR: the minSize and relMinReplicates arguments provide additional filtering functionality not provided by **CAMERA** or **RAMClustR** (except minSize). Note that these filters are enabled by default, hence, final results may be different than what CAMERA/RAMClustR normally would return.

References

Kuhl, C., Tautenhahn, R., Boettcher, C., Larson, T. R. and Neumann, S. CAMERA: an integrated strategy for compound spectra extraction and annotation of liquid chromatography/mass spectrometry data sets. Analytical Chemistry, 84:283-289 (2012)

Martin Loos (2016). nontarget: Detecting Isotope, Adduct and Homologue Relations in LC-MS Data. R package version 1.9.

Loos, M., Gerber, C., Corona, F., Hollender, J., Singer, H. (2015). Accelerated isotope fine structure calculation using pruned transition trees, Analytical Chemistry 87(11), 5738-5744.

Broeckling, Heuberger CD;, Prince AL;, Ingelsson JA;, Prenni E;, E. J (2013). "Assigning precursor-product ion relationships in indiscriminant MS/MS data from non-targeted metabolite profiling studies." *Analytical Chemistry*, **9**, 33–43.

Broeckling CD, Afsar FA, Neumann S, Ben-Hur A, Prenni JE (2014). "RAMClust: A Novel Feature Clustering Method Enables Spectral-Matching-Based Annotation for Metabolomics Data." *Analytical Chemistry*, **86 (14)**, 6812–6817.

---

components-class *Component class*

Description

Contains data for feature groups that are related in some way. These *components* commonly include adducts, isotopes and homologues.

Usage

```
## S4 method for signature 'components'
componentTable(obj)

## S4 method for signature 'components'
componentInfo(obj)

## S4 method for signature 'components'
```

```
groupNames(obj)

## S4 method for signature 'components'
length(x)

## S4 method for signature 'components'
names(x)

## S4 method for signature 'components'
show(object)

## S4 method for signature 'components,ANY,ANY,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'components,ANY,ANY'
x[[i, j]]

## S4 method for signature 'components'
x$name

## S4 method for signature 'components'
as.data.table(x)

## S4 method for signature 'components'
filter(
  obj,
  size = NULL,
  adducts = NULL,
  isotopes = NULL,
  rtIncrement = NULL,
  mzIncrement = NULL,
  negate = FALSE
)

## S4 method for signature 'components'
findFGroup(obj, fGroup)

## S4 method for signature 'components'
plotSpec(
  obj,
  index,
  markFGroup = NULL,
  useGGPlot2 = FALSE,
  xlim = NULL,
  ylim = NULL,
  ...
)
```

```
## S4 method for signature 'components'
plotEIC(obj, index, fGroups, rtWindow = 5, ...)

## S4 method for signature 'components'
consensus(obj, ...)
```

Arguments

obj, object, x	The component object.
i, j	A numeric or character value which is used to select components/feature groups by their index or name, respectively (for the order/names see names()/groupNames()). For [: Can also be logical to perform logical selection (similar to regular vectors). If missing all components/feature groups are selected. For [[: should be a scalar value. j is optional.
...	For plotEIC: Further (optional) arguments passed to the plotEIC method for the featureGroups class. Note that the colourBy, showPeakArea, showFGroupRect and topMost arguments cannot be set as these are set by this method. For plotSpec: Further arguments passed to plot. For consensus: components objects that should be used to generate the consensus.
drop	ignored.
name	The component name (partially matched).
size	Should be a two sized vector with the minimum/maximum size of a component. Set to NULL to ignore.
adducts	Remove any feature groups within components that do not match given adduct rules. If adducts is a logical then only results are kept when an adduct is assigned (adducts=TRUE) or not assigned (adducts=FALSE). Otherwise, if adducts contains one or more adduct objects (or something that can be converted to it with as.adduct) then only results are kept that match the given adducts. Set to NULL to ignore this filter.
isotopes	Only keep results that match a given isotope rule. If isotopes is a logical then only results are kept with (isotopes=TRUE) or without (isotopes=FALSE) isotope assignment. Otherwise isotopes should be a numeric vector with isotope identifiers to keep (*e.g.* '0' for monoisotopic results, '1' for 'M+1' results etc.). Set to NULL to ignore this filter.
rtIncrement, mzIncrement	Should be a two sized vector with the minimum/maximum retention or mz increment of a homologous series. Set to NULL to ignore.
negate	If TRUE then filters are applied in opposite manner.
fGroup	The name (thus a character) of the feature group that should be searched for.

index	The index of the component. Can be a numeric index or a character with its name.
markFGroup	If specified (*i.e.* not NULL) this argument can be used to mark a feature group in the plotted spectrum. The value should be a character with the name of the feature group. Setting this to NULL will not mark any peak.
useGGPlot2	If TRUE then **ggplot2** is used for plotting, otherwise base plot used. For plotSpec, ggplot2 allows nicely repelled text for annotation. However, base plot is generally faster.
xlim, ylim	Sets the plot size limits used by plot. Set to NULL for automatic plot sizing.
fGroups	The featureGroups object that was used to generate the components.
rtWindow	Retention window: see the plotEIC method for the featureGroups class.

Details

components objects are obtained from component generators.

Value

The subset operator ("[") and filter method return the data subset in an object from the componentsReduced class. This object does not contain any algorithm specific data and as such, algorithm specific methods (*e.g.* treeCut) will not work on this object. The reason for this is that it is often very difficult or impossible to subset the algorithmic data.

consensus returns a components object that is produced by merging multiple specified components objects.

Methods (by generic)

- componentTable: Accessor method for the components slot of a components class. Each component is stored as a data.table.
- componentInfo: Accessor method for the componentInfo slot of a components class.
- groupNames: returns a character vector with the names of the feature groups for which data is present in this object.
- length: Obtain total number of components.
- names: Obtain the names of all components.
- show: Show summary information for this object.
- [: Subset on components/feature groups.
- [[: Extracts a component table, optionally filtered by a feature group.
- $: Extracts a component table by component name.
- as.data.table: Returns all component data in a table.
- filter: Provides rule based filtering for components.
- findFGroup: Returns the component id(s) to which a feature group belongs.
- plotSpec: Plot a *pseudo* mass spectrum for a single component.

- plotEIC: Plot an extracted ion chromatogram (EIC) for all feature groups within a single component.
- consensus: Generates a consensus from multiple components objects. At this point results are simply combined and no attempt is made to merge similar components.

Slots

components List of all components in this object. Use the componentTable method for access.

componentInfo A data.table containing general information for each component. Use the componentInfo method for access.

S4 class hierarchy

- workflowStep
  - components
    - componentsReduced
    - componentsCamera
    - componentsIntClust
    - componentsNT
    - componentsRC

Note

filter Applies only those filters for which a component has data available. For instance, filtering by adduct will only filter any results within a component if that component contains adduct information.

See Also

component-generation, componentsNT and componentsIntClust

---

componentsIntClust-class

Components based on clustered intensity profiles.

Description

This class is derived from components and is used to store hierarchical clustering information from intensity profiles of feature groups.

Usage

```
## S4 method for signature 'componentsIntClust'
clusters(obj)

## S4 method for signature 'componentsIntClust'
cutClusters(obj)

## S4 method for signature 'componentsIntClust'
clusterProperties(obj)

## S4 method for signature 'componentsIntClust'
treeCut(obj, k = NULL, h = NULL)

## S4 method for signature 'componentsIntClust'
treeCutDynamic(obj, maxTreeHeight, deepSplit, minModuleSize)

## S4 method for signature 'componentsIntClust'
plotHeatMap(
  obj,
  interactive = FALSE,
  col = NULL,
  margins = c(6, 2),
  cexCol = 1,
  ...
)

## S4 method for signature 'componentsIntClust'
plotInt(obj, index, pch = 20, type = "b", lty = 3, col = NULL, ...)

## S4 method for signature 'componentsIntClust,ANY'
plot(
  x,
  pal = "Paired",
  numericLabels = TRUE,
  colourBranches = length(x) < 50,
  showLegend = length(x) < 20,
  ...
)

## S4 method for signature 'componentsIntClust'
plotSilhouettes(obj, kSeq, pch = 16, type = "b", ...)
```

Arguments

k, h	Desired number of clusters or tree height to be used for cutting the dendrogram, respectively. One or the other must be specified. Analogous to cutree.

`maxTreeHeight, deepSplit, minModuleSize`
: Arguments used by `cutreeDynamicTree`.

`interactive`
: If `TRUE` an interactive heatmap will be drawn (with `heatmaply`).

`col`
: The colour used for plotting. Set to `NULL` for automatic colours.

`margins, cexCol`
: Passed to `heatmap.2`

`...`
: Further options passed to `heatmap.2` / `heatmaply` (`plotHeatMap`), `plot.dendrogram` (`plot`) or `plot` (`plotInt`).

`index`
: Numeric component/cluster index.

`pch, type`
: Passed to `plot`.

`lty`
: Passed to `lines`.

`x, obj`
: A `componentsIntClust` object.

`pal`
: Colour palette to be used from **RColorBrewer**.

`numericLabels`
: Set to `TRUE` to label with numeric indices instead of (long) feature group names.

`colourBranches`
: Whether branches from cut clusters (and their labels) should be coloured. Might be slow with large numbers of clusters, hence, the default is only `TRUE` when this is not the case.

`showLegend`
: If `TRUE` and `colourBranches` is also `TRUE` then a legend will be shown which outlines cluster numbers and their colours. By default `TRUE` for small amount of clusters to avoid overflowing the plot.

`kSeq`
: An integer vector containing the sequence that should be used for average silhouette width calculation.

Details

Objects from this class are generated by `generateComponentsIntClust`

Value

`plotHeatMap` returns the same as `heatmap.2` or `heatmaply`.

Methods (by generic)

- `clusters`: Accessor method to the `clust` slot, which was generated by `hclust`.
- `cutClusters`: Accessor method to the `cutClusters` slot. Returns a vector with cluster membership for each candidate (format as `cutree`).
- `clusterProperties`: Returns a list with properties on how the clustering was performed.
- `treeCut`: Manually (re-)cut the dendrogram.
- `treeCutDynamic`: Automatically (re-)cut the dendrogram using the `cutreeDynamicTree` function from **dynamicTreeCut**.
- `plotHeatMap`: draws a heatmap using the `heatmap.2` or `heatmaply` function.

- `plotInt`: makes a plot for all (normalized) intensity profiles of the feature groups within a given cluster.
- `plot`: generates a dendrogram from a given cluster object and optionally highlights resulting branches when the cluster is cut.
- `plotSilhouettes`: Plots the average silhouette width when the clusters are cut by a sequence of k numbers. The k value with the highest value (marked in the plot) may be considered as the optimal number of clusters.

Slots

`clusterm` Numeric matrix with normalized feature group intensities that was used for clustering.

`distm` Distance matrix that was used for clustering (obtained with `daisy`).

`clust` Object returned by `hclust`.

`cutClusters` A `list` with assigned clusters (same format as what `cutree` returns).

`gInfo` The `groupInfo` of the feature groups object that was used.

`properties` A list containing general properties and parameters used for clustering.

Note

The intensity values for components (used by `plotSpec`) are set to a dummy value (1) as no single intensity value exists for this kind of components.

References

Schollee JE, Bourgin M, von Gunten U, McArdell CS, Hollender J (2018-oct). "Non-target screening to trace ozonation transformation products in a wastewater treatment train including different post-treatments." *Water Research*, **142**, 267–278. doi: 10.1016/j.watres.2018.05.045.

See Also

`components` and component-generation

`componentsNT-class` *Components class for homologous series.*

Description

This class is derived from `components` and is used to store results from unsupervised homolog detection with the **nontarget** package.

Usage

```
## S4 method for signature 'componentsNT'
plotGraph(obj, onlyLinked = TRUE)
```

Arguments

obj	The componentsRC object to plot.
onlyLinked	If TRUE then only series with links are plotted.

Details

Objects from this class are generated by generateComponentsNontarget

Value

plotGraph returns the result of visNetwork.

Methods (by generic)

- plotGraph: Plots an interactive network graph for linked homologous series (*i.e.* series with (partial) overlap which could not be merged). The resulting graph can be browsed interactively and allows quick inspection of series which may be related. The graph is constructed with the **igraph** package and rendered with **visNetwork**.

Slots

homol A list with homol objects for each replicate group as returned by homol.search

References

Martin Loos (2016). nontarget: Detecting Isotope, Adduct and Homologue Relations in LC-MS Data. R package version 1.9.

Loos, M., Gerber, C., Corona, F., Hollender, J., Singer, H. (2015). Accelerated isotope fine structure calculation using pruned transition trees, Analytical Chemistry 87(11), 5738-5744.

Csardi G, Nepusz T: The igraph software package for complex network research, InterJournal, Complex Systems 1695. 2006. https://igraph.org

Almende B.V., Benoit Thieurmel and Titouan Robert (2019). visNetwork: Network Visualization using 'vis.js' Library. R package version 2.0.9. http://datastorm-open.github.io/visNetwork/

See Also

components and component-generation

compound-generation *Automatic compound identification*

Description

Functionality to automatically identify chemical compounds from feature groups.

Usage

```
## S4 method for signature 'featureGroups'
generateCompounds(fGroups, MSPeakLists, algorithm, ...)

generateCompoundsMetfrag(
  fGroups,
  MSPeakLists,
  method = "CL",
  logPath = file.path("log", "metfrag"),
  timeout = 300,
  timeoutRetries = 2,
  errorRetries = 2,
  topMost = 100,
  dbRelMzDev = 5,
  fragRelMzDev = 5,
  fragAbsMzDev = 0.002,
  adduct,
  database = "pubchem",
  extendedPubChem = "auto",
  chemSpiderToken = "",
  scoreTypes = compoundScorings("metfrag", database, onlyDefault = TRUE)$name,
  scoreWeights = 1,
  preProcessingFilters = c("UnconnectedCompoundFilter", "IsotopeFilter"),
  postProcessingFilters = c("InChIKeyFilter"),
  maxCandidatesToStop = 2500,
  identifiers = NULL,
  extraOpts = NULL,
  maxProcAmount = getOption("patRoon.maxProcAmount")
)

generateCompoundsSIRIUS(
  fGroups,
  MSPeakLists,
  relMzDev = 5,
  adduct = "[M+H]+",
  elements = "CHNOP",
  profile = "qtof",
  formulaDatabase = NULL,
  fingerIDDatabase = "pubchem",
```

```
  noise = NULL,
  errorRetries = 2,
  cores = NULL,
  topMost = 100,
  topMostFormulas = 5,
  extraOptsGeneral = NULL,
  extraOptsFormula = NULL,
  verbose = TRUE,
  SIRBatchSize = 0,
  logPath = file.path("log", "sirius_compounds"),
  maxProcAmount = getOption("patRoon.maxProcAmount")
)

compoundScorings(
  algorithm = NULL,
  database = NULL,
  includeSuspectLists = TRUE,
  onlyDefault = FALSE,
  includeNoDB = TRUE
)
```

Arguments

fGroups	featureGroups object for which compounds should be identified. This should be the same or a subset of the object that was used to create the specified MSPeakLists. In the case of a subset only the remaining feature groups in the subset are considered.
MSPeakLists	A MSPeakLists object that was generated for the supplied fGroups.
algorithm	A character string describing the algorithm that should be used: "metfrag", "sirius"
...	Any parameters to be passed to the selected compound generation algorithm.
method	Which method should be used for MetFrag execution: "CL" for MetFragCL and "R" for MetFragR. The former might be faster.
logPath	Destination directory for log files with output from executed commands. Will be created if non-existant. Set to NULL to disable logging.
timeout	Maximum time (in seconds) before a metFrag query for a feature group is stopped. Also see timeoutRetries argument.
timeoutRetries	Maximum number of retries after reaching a timeout before completely skipping the metFrag query for a feature group. Also see timeout argument.
errorRetries	Maximum number of retries after an error occurred. This may be useful to handle e.g. connection errors.
topMost	Only keep this number of candidates (per feature group) with highest score. Set to NULL to always keep all candidates, however, please note that this may result in significant usage of CPU/RAM resources for large numbers of candidates.

`dbRelMzDev` Relative mass deviation (in ppm) for database search. Sets the '`DatabaseSearchRelativeMassDevi` option.

`fragRelMzDev` Relative mass deviation (in ppm) for fragment matching. Sets the '`FragmentPeakMatchRelativeMa` option.

`fragAbsMzDev` Absolute mass deviation (in Da) for fragment matching. Sets the '`FragmentPeakMatchAbsoluteMas` option.

`adduct` An `adduct` object (or something that can be converted to it with `as.adduct`). Examples: `"[M-H]-"`, `"[M+Na]+"`.

`database` Compound database to use. Valid values are: `"pubchem"`, `"chemspider"`, `"for-ident"`, `"comptox"`, `"pubchemlite"`, `"kegg"`, `"sdf"`, `"psv"` and `"csv"`. See section below for more information. Sets the `MetFragDatabaseType` option.

`extendedPubChem`

If `database="pubchem"`: whether to use the *extended* database that includes information for compound scoring (*i.e.* number of patents/PubMed references). Note that downloading candidates from this database might take extra time. Valid values are: `FALSE` (never use it), `TRUE` (always use it) or `"auto"` (default, use if specified scorings demand it).

`chemSpiderToken`

A character string with the ChemSpider security token that should be set when the ChemSpider database is used. Sets the '`ChemSpiderToken`' option.

`scoreTypes` A character vector defining the scoring types. See the Scorings section below for more information. Note that both generic and `MetFrag` specific names are accepted (*i.e.* `name` and `metfrag` columns returned by `compoundScorings`). When a local database is used, the name should match what is given there (`e.g` column names when `database=csv`). Note that MetFrag may still report other scoring data, however, these are not used for ranking. Sets the '`MetFragScoreTypes`' option.

`scoreWeights` Numeric vector containing weights of the used scoring types. Order is the same as set in `scoreTypes`. Values are recycled if necessary. Sets the '`MetFragScoreWeights`' option.

`preProcessingFilters, postProcessingFilters`

A character vector defining pre/post filters applied before/after fragmentation and scoring (*e.g.* `"UnconnectedCompoundFilter"`, `"IsotopeFilter"`, `"ElementExclusionFilter"`). Some methods require further options to be set. For all filters and more information refer to the Candidate Filters section on the MetFragR homepage. Sets the '`MetFragPreProcessingCandidateFilter`' and `MetFragPostProcessingCandidateFilter` options.

`maxCandidatesToStop`

If more than this number of candidate structures are found then processing will be aborted and no results this feature group will be reported. Low values increase the chance of missing data, whereas too high values will use too much computer resources and signficantly slowdown the process. Sets the '`MaxCandidateLimitToStop`' option.

`identifiers` A `list` containing for each feature group a character vector with database identifiers that should be used to find candidates for a feature group (the list should be named by feature group names). If `NULL` all relevant candidates will be retrieved from the specified database. An example usage scenario is to obtain the list of candidate identifiers from a `compounds` object obtained with `generateCompoundsSIRIUS` using the `identifiers` method. This way, only those candidates will be searched by MetFrag that were generated by SIRIUS+CSI:FingerID. Sets the '`PrecursorCompoundIDs`' option.

`extraOpts` For `MetFrag`: A named `list` containing further settings to be passed to `run.metfrag`. See the MetFragR and MetFrag CL homepages for all available options.

For `SIRIUS`: a `character` vector with any extra commandline parameters for formula prediction. See the SIRIUS manual for more details.

Set to `NULL` to ignore.

`maxProcAmount` Maximum number of processes to run for parallelization. Usually a number close to the amount of physical cores yields most efficient results.

`relMzDev` Maximum relative deviation between the measured and candidate formula m/z values (in ppm). Sets the '`--ppm-max`' commandline option.

`elements` Elements to be considered for formulae calculation. This will heavily affects the number of candidates! Always try to work with a minimal set by excluding elements you don't expect. The minimum/maximum number of elements can also be specified, for example: a value of `"C[5]H[10-15]O"` will only consider formulae with up to five carbon atoms, between ten and fifteen hydrogen atoms and any amount of oxygen atoms. Sets the '`--elements`' commandline option.

`profile` Name of the configuration profile, for example: '`"qtof"`', '`"orbitrap"`', '`"fticr"`'. Sets the '`--profile`' commandline option.

`formulaDatabase` If not `NULL`, use a database for retrieval of formula candidates. Possible values are: '`"pubchem"`', '`"bio"`', '`"kegg"`', '`"hmdb"`'. Sets the '`--database`' commandline option.

`fingerIDDatabase` Database specifically used for `CSI:FingerID`. If `NULL`, the value of the `formulaDatabase` parameter will be used or `"pubchem"` when that is also `NULL`. Sets the '`--fingerid-db`' option.

`noise` Median intensity of the noise (`NULL` ignores this parameter). Sets the '`--noise`' commandline option.

`cores` The number of cores `SIRIUS` will use. If `NULL` then the default of all cores will be used.

`topMostFormulas` Do not return more than this number of candidate formulae. Note that only compounds for these formulae will be searched. Sets the '`--candidates`' commandline option.

`extraOptsGeneral, extraOptsFormula`
: a `character` vector with any extra commandline parameters for `SIRIUS`. For `SIRIUS` versions `<4.4` there is no distinction between general and formula options. Otherwise commandline options specified in `extraOptsGeneral` are added prior to the `formula` command, while options specified in `extraOptsFormula` are added in afterwards. See the `SIRIUS` manual for more details. Set to `NULL` to ignore.

`verbose`
: If `TRUE` then more output is shown in the terminal.

`SIRBatchSize`
: The maximum number of calculations done by `SIRIUS`. If this number is less than the amount of features to be calculated then calculations will be evenly split over multiple `SIRIUS` calls (which may be run in parallel if `maxProcAmount>1`). If `SIRBatchSize=0` then all feature calculations are performed from a single `SIRIUS` exection, which is often the fastest.

`includeSuspectLists, onlyDefault, includeNoDB`
: A logical specifying whether scoring terms releated to suspect lists, default scoring terms and non-database specific scoring terms should be included in the output, respectively.

Details

Several algorithms are provided to automatically identify compounds for given feature groups. To this end, each measured masses for all feature groups are searched within online database(s) (*e.g.* PubChem) to retrieve a list of potential candidate chemical compounds. Depending on the algorithm and its parameters, further scoring of candidates is then performed using, for instance, matching of measured and theoretical isotopic patterns, presence within other data sources such as patent databases and similarity of measured and in-silico predicted MS/MS fragments. Note that this process is often quite time consuming, especially for large feature group sets. Therefore, this is often one of the last steps within the workflow and not performed before feature groups have been prioritized.

`generateCompounds` is a generic function that will generate compounds using one of the supported algorithms. The actual functionality is provided by algorithm specific functions such as `generateCompoundsMetfrag` and `generateCompoundsSIRIUS`. While these functions may be called directly, `generateCompounds` provides a generic interface and is therefore usually preferred.

`generateCompoundsMetfrag` uses the **metfRag** package or `MetFrag CL` for compound identification (see `http://ipb-halle.github.io/MetFrag/`). Several online compound databases such as PubChem and ChemSpider may be chosen for retrieval of candidate structures. In addition, many options exist to score and filter resulting data, and it is highly suggested to optimize these to improve results. While MS/MS data is not mandatory, it will usually greatly improve candidate scoring. The `MetFrag` options `PeakList`, `IonizedPrecursorMass` and `ExperimentalRetentionTimeValue` (in minutes) fields are automatically set from feature data.

`generateCompoundsSIRIUS` uses SIRIUS in combination with CSI:FingerID for compound identification. Similar to `generateFormulasSIRIUS`, candidate formulae are generated with SIRIUS. These results are then feed to CSI:FingerID to acquire candidate structures. This method requires the availability of MS/MS data, and feature groups without it will be ignored.

compoundScorings displays an overview of scorings may be applied to rank candidate compounds (see Scorings section below).

Value

generateCompoundsMetFrag returns a compoundsMF object.

generateCompoundsSIRIUS returns a compounds object.

compoundScorings returns a data.frame with information on which scoring terms are used, what their algorithm specific name is and other information such as to which database they apply and short remarks.

Scorings

Each algorithm implements their own scoring system. Their names have been simplified and harmonized where possible and are used for reporting and in the case MetFrag is used to specify how compounds should be scored (scoreTypes argument). The compoundScorings function can be used to get an overview of both the algorithm specific and generic scoring names. For instance, the table below shows all scorings for MetFrag: (some columns are omitted)

name	metfrag	database
score	Score	
fragScore	FragmenterScore	
metFusionScore	OfflineMetFusionScore	
individualMoNAScore	OfflineIndividualMoNAScore	
numberPatents	PubChemNumberPatents	pubchem
numberPatents	Patent_Count	pubchemli
pubMedReferences	PubChemNumberPubMedReferences	pubchem
pubMedReferences	ChemSpiderNumberPubMedReferences	chemspide
pubMedReferences	NUMBER_OF_PUBMED_ARTICLES	comptox
pubMedReferences	PubMed_Count	pubchemli
extReferenceCount	ChemSpiderNumberExternalReferences	chemspide
dataSourceCount	ChemSpiderDataSourceCount	chemspide
referenceCount	ChemSpiderReferenceCount	chemspide
RSCCount	ChemSpiderRSCCount	chemspide
smartsInclusionScore	SmartsSubstructureInclusionScore	
smartsExclusionScore	SmartsSubstructureExclusionScore	
suspectListScore	SuspectListScore	
retentionTimeScore	RetentionTimeScore	
CPDATCount	CPDAT_COUNT	comptox
TOXCASTActive	TOXCAST_PERCENT_ACTIVE	comptox
dataSources	DATA_SOURCES	comptox
pubChemDataSources	PUBCHEM_DATA_SOURCES	comptox
EXPOCASTPredExpo	EXPOCAST_MEDIAN_EXPOSURE_PREDICTION_MG/KG-BW/DAY	comptox
ECOTOX	ECOTOX	comptox
NORMANSUSDAT	NORMANSUSDAT	comptox
MASSBANKEU	MASSBANKEU	comptox
TOX21SL	TOX21SL	comptox
TOXCAST	TOXCAST	comptox

KEMIMARKET	KEMIMARKET	comptox
MZCLOUD	MZCLOUD	comptox
pubMedNeuro	PubMedNeuro	comptox
CIGARETTES	CIGARETTES	comptox
INDOORCT16	INDOORCT16	comptox
SRM2585DUST	SRM2585DUST	comptox
SLTCHEMDB	SLTCHEMDB	comptox
THSMOKE	THSMOKE	comptox
ITNANTIBIOTIC	ITNANTIBIOTIC	comptox
STOFFIDENT	STOFFIDENT	comptox
KEMIMARKET_EXPO	KEMIMARKET_EXPO	comptox
KEMIMARKET_HAZ	KEMIMARKET_HAZ	comptox
REACH2017	REACH2017	comptox
KEMIWW_WDUIndex	KEMIWW_WDUIndex	comptox
KEMIWW_StpSE	KEMIWW_StpSE	comptox
KEMIWW_SEHitsOverDL	KEMIWW_SEHitsOverDL	comptox
ZINC15PHARMA	ZINC15PHARMA	comptox
PFASMASTER	PFASMASTER	comptox
peakFingerprintScore	AutomatedPeakFingerprintAnnotationScore	
lossFingerprintScore	AutomatedLossFingerprintAnnotationScore	
agroChemInfo	AgroChemInfo	pubchemli
bioPathway	BioPathway	pubchemli
drugMedicInfo	DrugMedicInfo	pubchemli
foodRelated	FoodRelated	pubchemli
pharmacoInfo	PharmacoInfo	pubchemli
safetyInfo	SafetyInfo	pubchemli
toxicityInfo	ToxicityInfo	pubchemli
knownUse	KnownUse	pubchemli
annoTypeCount	FPSum	pubchemli
annoTypeCount	AnnoTypeCount	pubchemli

In addition, the `compoundScorings` function is also useful to programatically generate a set of scorings to be used by `MetFrag`. For instance, the following can be given to the `scoreTypes` argument to use all default scorings for PubChem: `compoundScorings("metfrag","pubchem",onlyDefault=TRUE)$r`

For all `MetFrag` scoring types refer to the Candidate Scores section on the MetFragR homepage.

Usage of MetFrag databases

When `database="chemspider"` setting the `chemSpiderToken` argument is mandatory.

When a local database is set (*i.e.* `sdf`, `psv`, `csv`, `comptox`, `pubchemlite`) the file location of the database should be set in the `LocalDatabasePath` value via the `extraOpts` argument or using the `patRoon.path.MetFragCompTox`/`patRoon.path.MetFragPubChemLite` option (only when `database="comptox"` or `database="pubchemlite"`).

Examples: options(patRoon.path.MetFragCompTox = "C:/CompTox_17March2019_SelectMetaData.csv") extraOpts = list(LocalDatabasePath = "C:/myDB.csv").

For `database="comptox"` the files can be obtained from here. Furthermore, the files with additions for https://zenodo.org/record/3364464#.XnjM-XLvKUk and https://zenodo.org/record/3472781#.XnjMA metadata are also supported. Note that only recent `MetFrag` versions (¿= '`2.4.5`') support these libraries.

Note

For annotations performed with `SIRIUS` it is often the fastest to keep the default `SIRBatchSize=0`. In this case, the `maxProcAmount` argument will be ignored and all `SIRIUS` output will be printed to the terminal (unless `verbose=FALSE`).

References

Ruttkies C, Schymanski EL, Wolf S, Hollender J, Neumann S (2016-jan). "MetFrag relaunched: incorporating strategies beyond in silico fragmentation." *Journal of Cheminformatics*, **8**. doi: 10.1186/s1332101601159.

Duhrkop K, Fleischauer M, Ludwig M, Aksenov AA, Melnik AV, Meusel M, Dorrestein PC, Rousu J, Bocker S (2019-mar). "SIRIUS 4: a rapid tool for turning tandem mass spectra into metabolite structure information." *Nature Methods*, **16**, 299–302. doi: 10.1038/s41592-01903448.

Duhrkop K, Bocker S (2015). "Fragmentation Trees Reloaded." In *Research in Computational Molecular Biology*, 65–79. ISBN 978-3-319-16706-0.

Duhrkop K, Shen H, Meusel M, Rousu J, Bocker S (2015-sep). "Searching molecular structure databases with tandem mass spectra using CSI:FingerID." *Proceedings of the National Academy of Sciences*, **112**, 12580–12585. doi: 10.1073/pnas.1509788112.

Bocker S, Letzel MC, Liptak Z, Pervukhin A (2008-nov). "SIRIUS: decomposing isotope patterns for metabolite identification." *Bioinformatics*, **25**, 218–224. doi: 10.1093/bioinformatics/btn603.

See Also

`compounds-class`

`compounds-class` *Compound lists class*

Description

Contains data of generated chemical compounds for given feature groups.

Usage

```
## S4 method for signature 'compounds'
compoundTable(obj)

## S4 method for signature 'compounds'
algorithm(obj)

## S4 method for signature 'compounds'
groupNames(obj)

## S4 method for signature 'compounds'
length(x)

## S4 method for signature 'compounds'
show(object)

## S4 method for signature 'compounds,ANY,missing,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'compounds,ANY,missing'
x[[i, j]]

## S4 method for signature 'compounds'
x$name

## S4 method for signature 'compounds'
as.data.table(
  x,
  fGroups = NULL,
  fragments = FALSE,
  normalizeScores = "none",
  excludeNormScores = c("score", "individualMoNAScore")
)

## S4 method for signature 'compounds'
identifiers(compounds)

## S4 method for signature 'compounds'
filter(
  obj,
  minExplainedPeaks = NULL,
  minScore = NULL,
  minFragScore = NULL,
  minFormulaScore = NULL,
  scoreLimits = NULL,
  elements = NULL,
  fragElements = NULL,
  lossElements = NULL,
```

```
  topMost = NULL,
  negate = FALSE
)

## S4 method for signature 'compounds'
addFormulaScoring(
  compounds,
  formulas,
  updateScore = FALSE,
  formulaScoreWeight = 1
)

## S4 method for signature 'compounds'
getMCS(obj, index, groupName)

## S4 method for signature 'compounds'
plotStructure(
  obj,
  index,
  groupName,
  width = 500,
  height = 500,
  useGGPlot2 = FALSE
)

## S4 method for signature 'compounds'
plotScores(
  obj,
  index,
  groupName,
  normalizeScores = "max",
  excludeNormScores = c("score", "individualMoNAScore"),
  onlyUsed = TRUE,
  useGGPlot2 = FALSE
)

## S4 method for signature 'compounds'
annotatedPeakList(
  obj,
  index,
  groupName,
  MSPeakLists,
  formulas = NULL,
  onlyAnnotated = FALSE
)

## S4 method for signature 'compounds'
plotSpec(
```

```
  obj,
  index,
  groupName,
  MSPeakLists,
  formulas = NULL,
  plotStruct = TRUE,
  title = NULL,
  useGGPlot2 = FALSE,
  xlim = NULL,
  ylim = NULL,
  ...
)

## S4 method for signature 'compounds'
plotVenn(obj, ..., labels = NULL, vennArgs = NULL)

## S4 method for signature 'compounds'
plotUpSet(
  obj,
  ...,
  labels = NULL,
  nsets = length(list(...)) + 1,
  nintersects = NA,
  upsetArgs = NULL
)

## S4 method for signature 'compounds'
consensus(
  obj,
  ...,
  absMinAbundance = NULL,
  relMinAbundance = NULL,
  uniqueFrom = NULL,
  uniqueOuter = FALSE,
  minMaxNormalization = FALSE,
  rankWeights = 1,
  labels = NULL
)
```

Arguments

`obj, object, x, compounds`
: The `compound` object.

`i`
: A numeric or character value which is used to select feature groups by their index or name, respectively (for the order/names see `groupNames()`).

  For `[`: Can also be logical to perform logical selection (similar to regular vectors). If missing all feature groups are selected.

For `[[`: should be a scalar value.

`...` For `plotSpec`: Further arguments passed to `plot`.
Others: Any further (and unique) `compounds` objects.

`drop, j` ignored.

`name` The feature group name (partially matched).

`fGroups` The `featureGroups` object that was used to generate this object. If not `NULL` it is used to add feature group information (retention and m/z values).

`fragments` If `TRUE` then information on annotated fragments will be included.

`normalizeScores`
A `character` that specifies how normalization of compound scorings occurs. Either `"none"` (no normalization), `"max"` (normalize to max value) or `"minmax"` (perform min-max normalization). Note that normalization of negative scores (e.g. output by `SIRIUS`) is always performed as min-max. Furthermore, currently normalization for `compounds` takes the original min/max scoring values into account when candidates were generated. Thus, for `compounds` scoring, normalization is not affected when candidate results were removed after they were generated (*e.g.* by use of `filter`).

`excludeNormScores`
A `character` vector specifying any compound scoring names that should *not* be normalized. Set to `NULL` to normalize all scorings. Note that whether any normalization occurs is set by the `excludeNormScores` argument.
For `compounds`: By default `score` and `individualMoNAScore` are set to mimic the behavior of the `MetFrag` web interface.

`minExplainedPeaks, minScore, minFragScore, minFormulaScore`
Minimum number of explained peaks, overall score, in-silico fragmentation score and formula score, respectively. Set to `NULL` to ignore. The `scoreLimits` argument allows for more advanced score filtering.

`scoreLimits` Filter results by their scores. Should be a named `list` that contains two-sized numeric vectors with the minimum/maximum value of a score (use `-Inf`/`Inf` for no limits). The names of each element should follow the values returned by `compoundScorings()$name`. For instance, `scoreLimits=list(numberPatents=c(1` specifies that `numberPatents` should be at least '`10`'. For more details of scorings see `compoundScorings`. Note that a result without a specified scoring is never removed. Set to `NULL` to skip this filter.

`elements` Only retain candidate formulae (neutral form) that match a given elemental restriction. The format of `elements` is a `character` string with elements that should be present where each element is followed by a valid amount or a range thereof. If no number is specified then '`1`' is assumed. For instance, `elements="C1-10H2-20O0-2P"`, specifies that '`1-10`', '`2-20`', '`0-2`' and '`1`' carbon, hydrogen, oxygen and phosphorus atoms should be present, respectively. When `length(elements)>1` formulas are tested to follow at least one of the given elemental restrictions. For instance, `elements=c("P","S")` specifies that either one phosphorus or one sulphur atom should be present. Set to `NULL` to ignore this filter.

`fragElements, lossElements`
: Specifies elemental restrictions for fragment or neutral loss formulae (charged form). Candidates are retained if at least one of the fragment formulae follow (or not follow if `negate=TRUE`) the given restrictions. See `elements` for the used format.

`topMost`
: Only keep a maximum of `topMost` candidates with highest score (or least highest if `negate=TRUE`). Set to `NULL` to ignore.

`negate`
: If `TRUE` then filters are applied in opposite manner.

`formulas`
: The `formulas` object that should be used for scoring/annotation. For `plotSpec`: set to `NULL` to ignore.

`updateScore`
: If set to `TRUE` then the `score` column is updated by adding the normalized '`formulaScore`' (weighted by '`formulaScoreWeight`'). Currently, this **only** makes sense for `MetFrag` results!

`formulaScoreWeight`
: Weight used to update scoring (see `updateScore` parameter).

`index`
: The numeric index of the candidate structure. Multiple indices (*i.e.* vector with length ¿=2) may be specified for `plotStructure` and are mandatory for `getMCS`. Alternatively, '`-1`' may be specified to these methods to select all candidates. When multiple indices are specified for `plotStructure`, their maximum common substructure will be drawn.

`groupName`
: The name of the feature group to which the candidate belongs.

`width, height`
: The dimensions (in pixels) of the raster image that should be plotted.

`useGGPlot2`
: If `TRUE` then **ggplot2** is used for plotting, otherwise base plot used. For `plotSpec`, `ggplot2` allows nicely repelled text for annotation. However, base plot is generally faster.

`onlyUsed`
: If `TRUE` then only scorings are plotted that actually have been used to rank data (see the `scoreTypes` argument to `generateCompoundsMetfrag` for more details).

`MSPeakLists`
: The `MSPeakLists` object that was used to generate the candidate

`onlyAnnotated`
: Set to `TRUE` to filter out any peaks that could not be annotated.

`plotStruct`
: If `TRUE` then the candidate structure is drawn in the spectrum.

`title`
: The title of the plot. If `NULL` a title will be automatically made.

`xlim, ylim`
: Sets the plot size limits used by `plot`. Set to `NULL` for automatic plot sizing.

`labels`
: A `character` with names to use for labelling. If `NULL` labels are automatically generated.

`vennArgs`
: A `list` with further arguments passed to **VennDiagram** plotting functions. Set to `NULL` to ignore.

`nsets`
: See `upset`.

`nintersects`
: See `upset`.

`upsetArgs`
: A list with any further arguments to be passed to `upset`. Set to `NULL` to ignore.

absMinAbundance, relMinAbundance
: Minimum absolute or relative ('0-1') abundance across objects for a result to be kept. For instance, relMinAbundance=0.5 means that a result should be present in at least half of the number of compared objects. Set to 'NULL' to ignore and keep all results. Limits cannot be set when uniqueFrom is not NULL.

uniqueFrom
: Set this argument to only retain compounds that are unique within one or more of the objects for which the consensus is made. Selection is done by setting the value of uniqueFrom to a logical (values are re-cycled), numeric (select by index) or a character (as obtained with algorithm(obj)). For logical and numeric values the order corresponds to the order of the objects given for the consensus. Set to NULL to ignore.

uniqueOuter
: If uniqueFrom is not NULL and if uniqueOuter=TRUE: only retain data that are also unique between objects specified in uniqueFrom.

minMaxNormalization
: Set to TRUE to apply min-max normalization of (merged) scoring columns. FALSE will apply normalization to the maximum value. Scorings with negative values will always be min-max normalized.

rankWeights
: A numeric vector with weights of to calulcate the mean ranking score for each candidate. The value will be re-cycled if necessary, hence, the default value of '1' means equal weights for all considered objects.

Details

compounds objects are obtained from compound generators.

Value

plotSpec and plotStructure will return a ggplot object if useGGPlot2 is TRUE.

compoundTable returns a list containing for each feature group a data.table with an overview of all candidate compounds and other data such as candidate scoring, matched MS/MS fragments, etc.

filter returns a filtered compounds object.

addFormulaScoring returns a compounds object updated with formula scoring.

getMCS returns an **rcdk** molecule object (IAtomContainer).

plotVenn (invisibly) returns a list with the following fields:

- gList the gList object that was returned by the utilized **VennDiagram** plotting function.
- areas The total area for each plotted group.
- intersectionCounts The number of intersections between groups.

The order for the areas and intersectionCounts fields is the same as the parameter order from the used plotting function (see *e.g.* draw.pairwise.venn and draw.triple.venn).

consensus returns a compounds object that is produced by merging multiple specified compounds objects.

Methods (by generic)

- compoundTable: Accessor method to obtain generated compounds.
- algorithm: Accessor method for the algorithm (a character string) used to generate compounds.
- groupNames: returns a character vector with the names of the feature groups for which data is present in this object.
- length: Obtain total number of candidate compounds.
- show: Show summary information for this object.
- [: Subset on feature groups.
- [[: Extract a compound table for a feature group.
- $: Extract a compound table for a feature group.
- as.data.table: Returns all MS peak list data in a table.
- identifiers: Returns a list containing for each feature group a character vector with database identifiers for all candidate compounds. The list is named by feature group names, and is typically used with the identifiers option of generateCompoundsMetfrag.
- filter: Provides rule based filtering for generated compounds. Useful to eliminate unlikely candidates and speed up further processing.
- addFormulaScoring: Adds formula ranking data from a formulas object as an extra compound candidate scoring (formulaScore column). The formula score for each compound candidate is between '0-1', where *zero* means no match with any formula candidates, and *one* means that the compound candidate's formula is the highest ranked.
- getMCS: Calculates the maximum common substructure (MCS) for two or more candidate structures for a feature group. This method uses the get.mcs function from **rcdk**.
- plotStructure: Plots a structure of a candidate compound using the **rcdk** package. If multiple candidates are specified (*i.e.* by specifying a vector for index) then the maximum common substructure (MCS) of the selected candidates is drawn.
- plotScores: Plots a barplot with scoring of a candidate compound.
- annotatedPeakList: Returns an MS/MS peak list annotated with data from a given candidate compound for a feature group.
- plotSpec: Plots an annotated spectrum for a given candidate compound for a feature group.
- plotVenn: plots a Venn diagram (using **VennDiagram**) outlining unique and shared compound candidates of up to five different compounds objects. Comparison is made on InChIKey1.
- plotUpSet: plots an UpSet diagram (using the upset function) outlining unique and shared compound candidates between different compounds objects. Comparison is made on InChIKey1.
- consensus: Generates a consensus of results from multiple objects. In order to rank the consensus candidates, first each of the candidates are scored based on their original ranking (the scores are normalized and the highest ranked candidate gets value '1'). The (weighted) mean is then calculated for all scorings of each candidate to derive the final ranking (if an object lacks the candidate its score will be '0'). The original rankings for each object is stored in the rank columns.

Slots

`compounds` Lists of all generated compounds. Use the `compounds` method for access.

`scoreTypes` A `character` with all the score types that were used when generating the compounds.

`scoreRanges` The original min/max values of all scorings when candidate results were generated. This is used for normalization.

S4 class hierarchy

- `workflowStep`
  - `compounds`
    - `compoundsConsensus`
    - `compoundsMF`

Source

Subscripting of formulae for plots generated by `plotSpec` is based on the `chemistry2expression` function from the ReSOLUTION package.

References

Guha, R. (2007). 'Chemical Informatics Functionality in R'. Journal of Statistical Software 6(18)

Conway JR, Lex A, Gehlenborg N (2017). "UpSetR: an R package for the visualization of intersecting sets and their properties." *Bioinformatics*, **33**, 2938–2940. doi: 10.1093/bioinformatics/btx364.

Lex A, Gehlenborg N, Strobelt H, Vuillemot R, Pfister H (2014-dec). "UpSet: Visualization of Intersecting Sets." *IEEE Transactions on Visualization and Computer Graphics*, **20**, 1983–1992. doi: 10.1109/tvcg.2014.2346248.

`compounds-cluster` *Hierarchical clustering of compounds*

Description

Perform hierarchical clustering of structure candidates based on chemical similarity and obtain overall structural information based on the maximum common structure (MCS).

Usage

```
## S4 method for signature 'compounds'
makeHCluster(
  obj,
  method,
  fpType = "extended",
```

```
  fpSimMethod = "tanimoto",
  maxTreeHeight = 1,
  deepSplit = TRUE,
  minModuleSize = 1
)
```

Arguments

`obj`	The `compounds` object to be clustered.
`method`	The clustering method passed to `hclust`.
`fpType`	The type of structural fingerprint that should be calculated. See the `type` argument of the `get.fingerprint` function of **rcdk**.
`fpSimMethod`	The similarity method (i.e. not dissimilarity!) to be used for generating the distance matrix. See the `method` argument of the `fp.sim.matrix` function of the **fingerprint** package.
`maxTreeHeight, deepSplit, minModuleSize`	Arguments used by `cutreeDynamicTree`.

Details

Often many possible chemical structure candidates are found for each feature group when performing compound identification. Therefore, it may be useful to obtain an overview of their general structural properties. One strategy is to perform hierarchical clustering based on their chemical (dis)similarity, for instance, using the Tanimoto score. The resulting clusters can then be characterized by evaluating their *maximum common substructure* (MCS).

`makeHCluster` performs hierarchical clustering of all structure candidates for each feature group within a `compounds` object. The resulting dendrograms are automatically cut using the `cutreeDynamicTree` function from the **dynamicTreeCut** package. The returned `compoundsCluster` object can then be used, for instance, for plotting dendrograms and MCS structures and manually re-cutting specific clusters.

Value

`makeHCluster` returns an `compoundsCluster` object.

Source

The methodology applied here has been largely derived from '`chemclust.R`' from the **metfRag** package and the package vignette of **rcdk**.

See Also

compoundsCluster

compoundsCluster-class *Compounds cluster class*

Description

Objects from this class are used to store hierarchical clustering data of candidate structures within compounds objects.

Usage

```
## S4 method for signature 'compoundsCluster'
clusters(obj)

## S4 method for signature 'compoundsCluster'
cutClusters(obj)

## S4 method for signature 'compoundsCluster'
clusterProperties(obj)

## S4 method for signature 'compoundsCluster'
groupNames(obj)

## S4 method for signature 'compoundsCluster'
length(x)

## S4 method for signature 'compoundsCluster'
lengths(x, use.names = TRUE)

## S4 method for signature 'compoundsCluster'
show(object)

## S4 method for signature 'compoundsCluster,ANY,missing,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'compoundsCluster'
treeCut(obj, k = NULL, h = NULL, groupName)

## S4 method for signature 'compoundsCluster'
treeCutDynamic(obj, maxTreeHeight, deepSplit, minModuleSize, groupName)

## S4 method for signature 'compoundsCluster,ANY'
plot(
  x,
  groupName,
  pal = "Paired",
  colourBranches = lengths(x)[groupName] < 50,
```

```
  showLegend = lengths(x)[groupName] < 20,
  ...
)

## S4 method for signature 'compoundsCluster'
getMCS(obj, groupName, cluster)

## S4 method for signature 'compoundsCluster'
plotStructure(
  obj,
  groupName,
  cluster,
  width = 500,
  height = 500,
  withTitle = TRUE
)

## S4 method for signature 'compoundsCluster'
plotSilhouettes(obj, kSeq, groupName, pch = 16, type = "b", ...)
```

Arguments

`obj, x, object`	A `compoundsCluster` object.
`use.names`	A logical value specifying whether the returned vector should be named with the feature group names.
`i`	A numeric or character value which is used to select feature groups by their index or name, respectively (for the order/names see `groupNames()`). Can also be logical to perform logical selection (similar to regular vectors). If missing all feature groups are selected.
`...`	Further arguments passed directly to the plotting function (`plot` or `plot.dendrogram`).
`drop, j`	ignored.
`k, h`	Desired number of clusters or tree height to be used for cutting the dendrogram, respecitively. One or the other must be specified. Analogous to `cutree`.
`groupName`	A character specifying the feature group name.
`maxTreeHeight, deepSplit, minModuleSize`	Arguments used by `cutreeDynamicTree`.
`pal`	Colour palette to be used from **RColorBrewer**.
`colourBranches`	Whether branches from cut clusters (and their labels) should be coloured. Might be slow with large numbers of clusters, hence, the default is only `TRUE` when this is not the case.
`showLegend`	If `TRUE` and `colourBranches` is also `TRUE` then a legend will be shown which outlines cluster numbers and their colours. By default `TRUE` for small amount of clusters to avoid overflowing the plot.
`cluster`	A numeric value specifying the cluster.

width, height	The dimensions (in pixels) of the raster image that should be plotted.
withTitle	A logical value specifying whether a title should be added.
kSeq	An integer vector containing the sequence that should be used for average silhouette width calculation.
pch, type	Passed to plot.

Details

Objects from this type are returned by the compounds method for makeHCluster.

Value

cutTree and cutTreeDynamic return the modified compoundsCluster object.

getMCS returns an **rcdk** molecule object (IAtomContainer).

Methods (by generic)

- clusters: Accessor method to the clusters slot. Returns a list that contains for each feature group an object as returned by hclust.
- cutClusters: Accessor method to the cutClusters slot. Returns a list that contains for each feature group a vector with cluster membership for each candidate (format as cutree).
- clusterProperties: Returns a list with properties on how the clustering was performed.
- groupNames: returns a character vector with the names of the feature groups for which data is present in this object.
- length: Returns the total number of clusters.
- lengths: Returns a vector with the number of clusters per feature group.
- show: Show summary information for this object.
- [: Subset on feature groups.
- treeCut: Manually (re-)cut a dendrogram that was generated for a feature group.
- treeCutDynamic: Automatically (re-)cut a dendrogram that was generated for a feature group using the cutreeDynamicTree function from **dynamicTreeCut**.
- plot: Plot the dendrogram for clustered compounds of a feature group. Clusters are highlighted using **dendextend**.
- getMCS: Calculates the maximum common substructure (MCS) for all candidate structures within a specified cluster. This method uses the get.mcs function from **rcdk**.
- plotStructure: Plots the maximum common substructure (MCS) for all candidate structures within a specified cluster.
- plotSilhouettes: Plots the average silhouette width when the clusters are cut by a sequence of k numbers. The k value with the highest value (marked in the plot) may be considered as the optimal number of clusters.

Slots

`clusters` A `list` with `hclust` objects for each feature group.

`dists` A `list` with distance matrices for each feature group.

`SMILES` A `list` containing a vector with `SMILES` for all candidate structures per feature group.

`cutClusters` A `list` with assigned clusters for all candidates per feature group (same format as what `cutree` returns).

`properties` A list containing general properties and parameters used for clustering.

---

compoundsMF-class *Compounds list class for MetFrag results.*

Description

This class is derived from `compounds` and contains additional specific MetFrag data.

Usage

```
## S4 method for signature 'compoundsMF'
settings(compoundsMF)
```

Arguments

`compoundsMF` A `compoundsMF` object.

Details

Objects from this class are generated by `generateCompoundsMetfrag`

Methods (by generic)

- `settings`: Accessor method for the `settings` slot.

Slots

`settings` A list with all general configuration settings passed to MetFrag. Feature specific items (*e.g.* spectra and precursor masses) are not contained in this list.

S4 class hierarchy

- `compounds`
  - `compoundsMF`

References

Ruttkies C, Schymanski EL, Wolf S, Hollender J, Neumann S (2016-jan). "MetFrag relaunched: incorporating strategies beyond in silico fragmentation." *Journal of Cheminformatics*, **8**. doi: 10.1186/s1332101601159.

See Also

compounds and compound-generation

---

convertMSFiles *MS data conversion*

Description

Conversion of MS analysis files between several open and closed data formats.

Usage

```
MSFileFormats(algorithm = "pwiz", vendor = FALSE)

convertMSFiles(
  files = NULL,
  outPath = NULL,
  dirs = TRUE,
  anaInfo = NULL,
  from = NULL,
  to = "mzML",
  overWrite = FALSE,
  algorithm = "pwiz",
  centroid = algorithm != "openms",
  filters = NULL,
  extraOpts = NULL,
  PWizBatchSize = 1,
  logPath = file.path("log", "convert"),
  maxProcAmount = getOption("patRoon.maxProcAmount")
)
```

Arguments

algorithm	Either "pwiz" (implemented by msConvert of ProteoWizard), "openms" (implemented by FileConverter of OpenMS) or "bruker" (implemented by DataAnalysis).
vendor	If TRUE only vendor formats are returned.
files, dirs	The files argument should be a character vector with input files. If files contains directories and dirs=TRUE then files from these directories are also considered. An alternative method to specify input files is by the anaInfo argument. If the latter is specified files may be NULL.
outPath	A character vector specifying directories that should be used for the output. Will be re-cycled if necessary. If NULL, output directories will be kept the same as the input directories.
anaInfo	An analysis info table used to retrieve input files. Either this argument or files (or both) should be set (*i.e.* not NULL).

from	Input format (see below). These are used to find analyses when dirs=TRUE or anaInfo is set.
to	Output format: "mzXML" or "mzML".
overWrite	Should existing destination file be overwritten (TRUE) or not (FALSE)?
centroid	Set to TRUE to enable centroiding (not supported if algorithm="openms"). In addition, when algorithm="pwiz" the value may be "vendor" to perform centroiding with the vendor algorithm or "cwt" to use ProteoWizard's wavelet algorithm.
filters	When algorithm="pwiz": a character vector specifying one or more filters. The elements of the specified vector are directly passed to the --filter option (see here)
extraOpts	A character vector specifying any extra commandline parameters passed to msConvert or FileConverter. Set to NULL to ignore. For options: see FileConverter and msConvert.
PWizBatchSize	When algorithm="pwiz": the number of analyses to process by a single call to msConvert. Usually a value of one is most efficient. Set to zero to run all analyses all at once from a single call.
logPath	Destination directory for log files with output from executed commands. Will be created if non-existant. Set to NULL to disable logging.
maxProcAmount	Maximum number of processes to run for parallelization. Usually a number close to the amount of physical cores yields most efficient results.

Details

MSFileFormats returns a character with all supported input formats (see below).

convertMSFiles converts the data format of an analysis to another. It uses tools from ProteoWizard (msConvert command), OpenMS (FileConverter command) or Bruker DataAnalysis to perform the conversion. Supported input and output formats include 'mzXML', '.mzML' and several vendor formats, depending on which algorithm is used.

Conversion formats

Possible output formats (to argument) are mzXML and mzML.

Possible input formats (from argument) depend on the algorithm that was chosen and may include:

- thermo: Thermo '.RAW' files (only algorithm="pwiz").
- bruker: Bruker '.d', '.yep', '.baf' and '.fid' files (only algorithm="pwiz" or algorithm="bruker").
- agilent: Agilent '.d' files (only algorithm="pwiz").
- ab: AB Sciex '.wiff' files (only algorithm="pwiz").
- waters Waters '.RAW' files (only algorithm="pwiz").
- mzXML/mzML: Open format '.mzXML'/'.mzML' files (only algorithm="pwiz" or algorithm="openms").

Note that the actual supported file formats of ProteoWizard depend on how it was installed (see here).

References

Rost HL, Sachsenberg T, Aiche S, Bielow C, Weisser H, Aicheler F, Andreotti S, Ehrlich H, Gutenbrunner P, Kenar E, Liang X, Nahnsen S, Nilse L, Pfeuffer J, Rosenberger G, Rurik M, Schmitt U, Veit J, Walzer M, Wojnar D, Wolski WE, Schilling O, Choudhary JS, Malmstrom L, Aebersold R, Reinert K, Kohlbacher O (2016-sep). "OpenMS: a flexible open-source software platform for mass spectrometry data analysis." *Nature Methods*, **13**, 741–748. doi: 10.1038/nmeth.3959.

Chambers MC, Maclean B, Burke R, Amodei D, Ruderman DL, Neumann S, Gatto L, Fischer B, Pratt B, Egertson J, Hoff K, Kessner D, Tasman N, Shulman N, Frewen B, Baker TA, Brusniak M, Paulse C, Creasy D, Flashner L, Kani K, Moulding C, Seymour SL, Nuwaysir LM, Lefebvre B, Kuhlmann F, Roark J, Rainer P, Detlev S, Hemenway T, Huhmer A, Langridge J, Connolly B, Chadick T, Holly K, Eckels J, Deutsch EW, Moritz RL, Katz JE, Agus DB, MacCoss M, Tabb DL, Mallick P (2012-oct). "A cross-platform toolkit for mass spectrometry and proteomics." *Nature Biotechnology*, **30**, 918–920. doi: 10.1038/nbt.2377.

Examples

```
## Not run:
# Use FileConverter of OpenMS to convert between open mzXML/mzML format
convertMSFiles("standard-1.mzXML", to = "mzML", algorithm = "openms")

# Convert all Thermo .RAW files in the analyses/raw directory to mzML and
# store the files in analyses/mzml. During conversion files are centroided by
# the peakPicking filter and only MS 1 data is kept.
convertMSFiles("analyses/raw", "analyses/mzml", dirs = TRUE, from = "thermo",
               centroid = "vendor", filters = "msLevel 1")

## End(Not run)
```

---

feature-filtering *Filtering of grouped features*

Description

Basic rule based filtering of feature groups.

Usage

```
## S4 method for signature 'featureGroups'
filter(
  obj,
  absMinIntensity = NULL,
  relMinIntensity = NULL,
  preAbsMinIntensity = NULL,
  preRelMinIntensity = NULL,
```

```
  absMinAnalyses = NULL,
  relMinAnalyses = NULL,
  absMinReplicates = NULL,
  relMinReplicates = NULL,
  absMinFeatures = NULL,
  relMinFeatures = NULL,
  absMinReplicateAbundance = NULL,
  relMinReplicateAbundance = NULL,
  maxReplicateIntRSD = NULL,
  blankThreshold = NULL,
  retentionRange = NULL,
  mzRange = NULL,
  mzDefectRange = NULL,
  chromWidthRange = NULL,
  rGroups = NULL,
  removeBlanks = FALSE,
  negate = FALSE
)

## S4 method for signature 'featureGroups'
replicateGroupSubtract(fGroups, rGroups, threshold = 0)
```

Arguments

`absMinIntensity, relMinIntensity`
: Minimum absolute/relative intensity for features to be kept. The relative intensity is determined from the feature with highest intensity (of all features from all groups). Set to '`0`' or `NULL` to skip this step.

`preAbsMinIntensity, preRelMinIntensity`
: As `absMinIntensity/relMinIntensity`, but applied *before* any other filters. This is typically used to speed-up subsequent filter steps. However, care must be taken that a sufficiently low value is choosen that is not expected to affect subsequent filtering steps. See below why this may be important.

`absMinAnalyses, relMinAnalyses`
: Feature groups are only kept when they contain data for at least this (absolute or relative) amount of analyses. Set to `NULL` to ignore.

`absMinReplicates, relMinReplicates`
: Feature groups are only kept when they contain data for at least this (absolute or relative) amount of replicates. Set to `NULL` to ignore.

`absMinFeatures, relMinFeatures`
: Analyses are only kept when they contain at least this (absolute or relative) amount of features. Set to `NULL` to ignore.

`absMinReplicateAbundance, relMinReplicateAbundance`
: Minimum absolute/relative abundance that a grouped feature should be present within a replicate group. If this mimimum is not met all features within the replicate group are removed. Set to `NULL` to skip this step.

`maxReplicateIntRSD`
: Maximum relative standard deviation (RSD) of intensity values for features within a replicate group. If the RSD is above this value all features within the replicate group are removed. Set to `NULL` to ignore.

`blankThreshold`
: Feature groups that are also present in blank analyses (see analysis info) are filtered out unless their relative intensity is above this threshold. For instance, a value of '`5`' means that only features with an intensity five times higher than that of the blank are kept. The relative intensity values between blanks and non-blanks are determined from the mean of all non-zero blank intensities. Set to `NULL` to skip this step.

`retentionRange, mzRange, mzDefectRange, chromWidthRange`
: Range of retention time (in seconds), *m/z*, mass defect (defined as the decimal part of *m/z* values) or chromatographic peak width (in seconds), respectively. Features outside this range will be removed. Should be a numeric vector with length of two containing the min/max values. The maximum can be `Inf` to specify no maximum range. Set to `NULL` to skip this step.

`rGroups`
: A character vector of replicate groups that should be kept (`filter`) or subtracted from (`replicateGroupSubtract`).

`removeBlanks`
: Set to `TRUE` to remove all analyses that belong to replicate groups that are specified as a blank in the analysis-information. This is useful to simplify the analyses in the specified `featureGroups` object after blank subtraction. When both `blankThreshold` and this argument are set, blank subtraction is performed prior to removing any analyses.

`negate`
: If set to `TRUE` then filtering operations are performed in opposite manner.

`fGroups, obj`
: `featureGroups` object to which the filter is applied.

`threshold`
: Minimum relative threshold (compared to mean intensity of replicate group being subtracted) for a feature group to be *not* removed. When '`0`' a feature group is always removed when present in the given replicate groups.

Details

`filter` performs common rule based filtering of feature groups such as blank subtraction, minimum intensity and minimum replicate abundance. Removing of features occurs by zeroing their intensity values. Furthermore, feature groups that are left completely empty (*i.e.* all intensities are zero) will be automatically removed.

`replicateGroupSubtract` removes feature groups present in a given set of replicate groups (unless intensities are above a given threshold). The replicate groups that are subtracted will be removed.

Value

A filtered `featureGroups` object. Feature groups that are filtered away have their intensity set to zero. In case a feature group is not present in any of the analyses anymore it will be removed completely.

Filter order

When multiple arguments are specified to `filter`, multiple filters are applied in sequence. Since some of these filters may affect each other, choosing their order correctly may be important for effective data filtering. For instance, when an intensity filter removes features from blank analyses, a subsequent blank filter may not adequately perform blank subtraction. Similarly, when intensity and blank filters are executed after the replicate abundance filter it may be necessary to ensure minimum replicate abundance again as the intensity and blank filters may have removed some features within a replicate group.

With this in mind, filters (if specified) occur in the following order:

1. Pre-Intensity filters (*i.e.* `preAbsMinIntensity` and `preRelMinIntensity`).
2. Chromatography and mass filters (*i.e* `retentionRange`, `mzRange`, `mzDefectRange` and `chromWidthRange`).
3. Replicate abundance filters (*i.e.* `absMinReplicateAbundance`, `relMinReplicateAbundance` and `maxReplicateIntRSD`).
4. Blank filter (*i.e.* blankThreshold).
5. Intensity filters (*i.e.* `absMinIntensity` and `relMinIntensity`).
6. Replicate abundance filters (2nd time, only if previous filters affected results).
7. General abundance filters (*i.e.* `absMinAnalyses`, `relMinAnalyses`, `absMinReplicates`, `relMinReplicates`, `absMinFeatures` and `relMinFeatures`).
8. Replicate group filter (*i.e.* `rGroups`) and blank analyses removal (*i.e.* if `removeBlanks=TRUE`).

If another filtering order is desired then `filter` should be called multiple times with only one filter argument at a time.

See Also

featureGroups-class

feature-grouping

`feature-finding` *Finding features*

Description

Functions and classes for collection of features.

Usage

```
findFeatures(analysisInfo, algorithm, ..., verbose = TRUE)

importFeatures(analysisInfo, type, ...)

findFeaturesBruker(
```

```
  analysisInfo,
  doFMF = "auto",
  startRange = 0,
  endRange = 0,
  save = TRUE,
  close = save,
  verbose = TRUE
)

findFeaturesEnviPick(analysisInfo, ..., verbose = TRUE)

importFeaturesEnviMass(analysisInfo, enviProjPath)

findFeaturesOpenMS(
  analysisInfo,
  noiseThrInt = 1000,
  chromSNR = 3,
  chromFWHM = 5,
  mzPPM = 10,
  reEstimateMTSD = TRUE,
  traceTermCriterion = "sample_rate",
  traceTermOutliers = 5,
  minSampleRate = 0.5,
  minTraceLength = 3,
  maxTraceLength = -1,
  widthFiltering = "fixed",
  minFWHM = 3,
  maxFWHM = 60,
  traceSNRFiltering = FALSE,
  localRTRange = 10,
  localMZRange = 6.5,
  isotopeFilteringModel = "metabolites (5% RMS)",
  MZScoring13C = FALSE,
  useSmoothedInts = TRUE,
  extraOpts = NULL,
  intSearchRTWindow = 3,
  logPath = file.path("log", "openms"),
  maxProcAmount = getOption("patRoon.maxProcAmount"),
  verbose = TRUE
)

findFeaturesXCMS(analysisInfo, method = "centWave", ..., verbose = TRUE)

importFeaturesXCMS(xs, analysisInfo)

findFeaturesXCMS3(analysisInfo, param = xcms::CentWaveParam(), verbose = TRUE)

importFeaturesXCMS3(xdata, analysisInfo)
```

Arguments

`analysisInfo`	Analysis info table.
`algorithm`	A character string describing the algorithm that should be used: `"bruker"`, `"openms"`, `"xcms"`, `"xcms3"`, `"envipick"`
`...`	further parameters passed to `xcmsSet` (`findFeaturesXCMS`), `enviPickwrap` (`featurefinderEnviPick`) or to selected feature finding or importing algorithms (`findFeatures` and `importFeatures`).
`verbose`	If set to `FALSE` then no text output is shown.
`type`	What type of data should be imported: `"xcms"`, `"xcms3"` or `"envimass"`.
`doFMF`	Run the 'Find Molecular Features' algorithm before loading compounds. Valid options are: `"auto"` (run FMF automatically if current results indicate it is necessary) and `"force"` (run FMF *always*, even if cached results exist). Note that checks done if `doFMF="auto"` are fairly simplistic, hence set `doFMF="force"` if feature data needs to be updated.
`startRange, endRange`	Start/End retention range (seconds) from which to collect features. A 0 (zero) for `endRange` marks the end of the analysis.
`close, save`	If `TRUE` then Bruker files are closed and saved after processing with DataAnalysis, respectively. Setting `close=TRUE` prevents that many analyses might be opened simultaneously in DataAnalysis, which otherwise may use excessive memory or become slow. By default `save` is `TRUE` when `close` is `TRUE`, which is likely what you want as otherwise any processed data is lost.
`enviProjPath`	The path of the enviMass project.
`noiseThrInt`	Noise intensity threshold. Sets `algorithm:common:noise_threshold_int` option.
`chromSNR`	Minimum S/N of a mass trace. Sets `algorithm:common:chrom_peak_snr` option.
`chromFWHM`	Expected chromatographic peak width (in seconds). Sets `algorithm:common:chrom_fwhm` option.
`mzPPM`	Allowed mass deviation (ppm) for trace detection. Sets `algorithm:mtd:mass_error_ppm`.
`reEstimateMTSD`	If `TRUE` then enables dynamic re-estimation of m/z variance during mass trace collection stage. Sets `algorithm:mtd:reestimate_mt_sd`.
`traceTermCriterion, traceTermOutliers, minSampleRate`	Termination criterion for the extension of mass traces. See FeatureFinderMetabo. Sets the `algorithm:mtd:trace_termination_criterion`, `algorithm:mtd:trace_termination_outliers` and `algorithm:mtd:min_sample_rate` options, respectively.
`minTraceLength, maxTraceLength`	Minimum/Maximum length of mass trace (seconds). Set negative value for maxlength to disable maximum. Sets `algorithm:mtd:min_trace_length` and `algorithm:mtd:min_trace_length`, respectively.

`widthFiltering, minFWHM, maxFWHM`
: Enable filtering of unlikely peak widths. See FeatureFinderMetabo. Sets `algorithm:epd:width_filtering`, `algorithm:epd:min_fwhm` and `algorithm:epd:max_fwhm`, respectively.

`traceSNRFiltering`
: If `TRUE` then apply post-filtering by signal-to-noise ratio after smoothing. Sets the `algorithm:epd:masstrace_snr_filtering` option.

`localRTRange, localMZRange`
: Retention/MZ range where to look for coeluting/isotopic mass traces. Sets the `algorithm:ffm:local_rt_range` and `algorithm:ffm:local_mz_range` options, respectively.

`isotopeFilteringModel`
: Remove/score candidate assemblies based on isotope intensities. See FeatureFinderMetabo. Sets the `algorithm:ffm:isotope_filtering_model` option.

`MZScoring13C`
: Use the 13C isotope as the expected shift for isotope mass traces. See FeatureFinderMetabo. Sets `algorithm:ffm:mz_scoring_13C`.

`useSmoothedInts`
: If `TRUE` then use LOWESS intensities instead of raw intensities. Sets the `algorithm:ffm:use_smoothed_intensities` option.

`extraOpts`
: Named `list` containing extra options that will be passed to `FeatureFinderMetabo`. Any options specified here will override any of the above. Example: `extraOpts=list("-algorithm:common:noise_threshold_int"=1000)` (corresponds to setting `noiseThrInt=1000`). Set to `NULL` to ignore.

`intSearchRTWindow`
: Retention time window (in seconds, +/- feature retention time) that is used to find the closest data point to the retention time to obtain the intensity of a feature (this is needed since OpenMS does not provide this data).

`logPath`
: Destination directory for log files with output from executed commands. Will be created if non-existant. Set to `NULL` to disable logging.

`maxProcAmount`
: Maximum number of processes to run for parallelization. Usually a number close to the amount of physical cores yields most efficient results.

`method`
: The method setting used by XCMS peak finding, see `xcms::findPeaks`

`xs`
: An `xcmsSet` object.

`param`
: The method parameters used by XCMS peak finding, see `xcms::findChromPeaks`

`xdata`
: An `XCMSnExp` object.

Details

Several functions exist to collect features (*i.e.* retention and MS information that represent potential compounds) from a set of analyses. All 'feature finders' return an object derived from the `features` base class. The next step in a general workflow is to group and align these features across analyses by feature groupers. Note that some feature finders have a plethora of options which sometimes may have a large effect on the quality of results.

Fine-tuning parameters is therefore important, and the optimum is largely dependent upon applied analysis methodology and instrumentation.

`findFeatures` is a generic function that will find features using one of the supported algorithms. The actual functionality is provided by algorithm specific functions such as `findFeaturesOpenMS` and `findFeaturesBruker`. While these functions may be called directly, `findFeatures` provides a generic interface and is therefore usually preferred.

`importFeatures` is a generic function to import feature groups produced by other software. The actual functionality is provided by specific functions such as `importFeaturesXCMS` and `importFeaturesEnviMass`.

`findFeaturesBruker` uses the 'Find Molecular Features' (FMF) algorithm of Bruker DataAnalysis vendor software to find features. The resulting 'compounds' are then transferred from DataAnalysis and stored as features.

`findFeaturesEnviPick` uses the `enviPickwrap`. function from the **enviPick** R package to extract features.

`importFeaturesEnviMass` imports features from a project generated by the **enviMass** package. NOTE: this functionality has only been tested with older versions of **enviMass**.

`findFeaturesOpenMS` uses the FeatureFinderMetabo TOPP tool (see `http://www.openms.de`).

`findFeaturesXCMS` uses the `xcmsSet` function from the **xcms** package to find features.

`importFeaturesXCMS` converts features from an existing `xcmsSet` object (obtained with the **xcms** package) to a new `features` object.

`findFeaturesXCMS3` uses the new `xcms3` interface from the **xcms** package to find features.

`importFeaturesXCMS3` converts features from an existing `XCMSnExp` object (obtained with the **xcms** package) to a new `features` object.

Value

An object of a class which is derived from `features`.

Note

The file format of analyses for `findFeaturesXCMS` and `findFeaturesXCMS3` must be `mzML` or `mzXML`.

`findFeaturesBruker` only works with Bruker data files (`.d` extension) and requires Bruker DataAnalysis and the **RDCOMClient** package to be installed. Furthermore, DataAnalysis combines multiple related masses in a feature (*e.g.* isotopes, adducts) but does not report the actual (monoisotopic) mass of the feature. Therefore, it is simply assumed that the feature mass equals that of the highest intensity mass peak.

If any errors related to `DCOM` appear it might be necessary to terminate DataAnalysis (note that DataAnalysis might still be running as a background process). The `ProcessCleaner` application installed with DataAnalayis can be used for this.

`findFeaturesEnviPick` Requires analysis files to be in the `mzXML` format.

The file format of analyses for `findFeaturesOpenMS` must be '`mzML`'. This functionality has been tested with OpenMS version ¿= 2.0. Please make sure it is installed and its binaries are added to the PATH environment variable or the `patRoon.path.OpenMS` option is set.

References

Rost HL, Sachsenberg T, Aiche S, Bielow C, Weisser H, Aicheler F, Andreotti S, Ehrlich H, Gutenbrunner P, Kenar E, Liang X, Nahnsen S, Nilse L, Pfeuffer J, Rosenberger G, Rurik M, Schmitt U, Veit J, Walzer M, Wojnar D, Wolski WE, Schilling O, Choudhary JS, Malmstrom L, Aebersold R, Reinert K, Kohlbacher O (2016-sep). "OpenMS: a flexible open-source software platform for mass spectrometry data analysis." *Nature Methods*, **13**, 741–748. doi: 10.1038/nmeth.3959.

pugixml (via Rcpp) is used to process OpenMS XML output.

Dirk Eddelbuettel and Romain Francois (2011). Rcpp: Seamless R and C++ Integration. Journal of Statistical Software, 40(8), 1-18. URL http://www.jstatsoft.org/v40/i08/.

Eddelbuettel, Dirk (2013) Seamless R and C++ Integration with Rcpp. Springer, New York. ISBN 978-1-4614-6867-7.

Dirk Eddelbuettel and James Joseph Balamuta (2017). Extending R with C++: A Brief Introduction to Rcpp. PeerJ Preprints 5:e3188v1. URL https://doi.org/10.7287/peerj.preprints.3188v1.

Smith, C.A. and Want, E.J. and O'Maille, G. and Abagyan,R. and Siuzdak, G.: XCMS: Processing mass spectrometry data for metabolite profiling using nonlinear peak alignment, matching and identification, Analytical Chemistry, 78:779-787 (2006)

Ralf Tautenhahn, Christoph Boettcher, Steffen Neumann: Highly sensitive feature detection for high resolution LC/MS BMC Bioinformatics, 9:504 (2008)

H. Paul Benton, Elizabeth J. Want and Timothy M. D. Ebbels Correction of mass calibration gaps in liquid chromatography-mass spectrometry metabolomics data Bioinformatics, 26:2488 (2010)

Smith, C.A. and Want, E.J. and O'Maille, G. and Abagyan,R. and Siuzdak, G.: XCMS: Processing mass spectrometry data for metabolite profiling using nonlinear peak alignment, matching and identification, Analytical Chemistry, 78:779-787 (2006)

Ralf Tautenhahn, Christoph Boettcher, Steffen Neumann: Highly sensitive feature detection for high resolution LC/MS BMC Bioinformatics, 9:504 (2008)

H. Paul Benton, Elizabeth J. Want and Timothy M. D. Ebbels Correction of mass calibration gaps in liquid chromatography-mass spectrometry metabolomics data Bioinformatics, 26:2488 (2010)

See Also

`features-class` and `analysis-information`

`feature-grouping` *Grouping of features*

Description

Functions and classes for grouping of features across analyses.

Usage

```
## S4 method for signature 'features'
groupFeatures(feat, algorithm, ..., verbose = TRUE)

importFeatureGroups(path, type, ...)

importFeatureGroupsBrukerPA(
  path,
  feat,
  rtWindow = 12,
  mzWindow = 0.005,
  intWindow = 5,
  warn = TRUE
)

importFeatureGroupsEnviMass(path, feat, positive)

groupFeaturesOpenMS(
  feat,
  rtalign = TRUE,
  QT = FALSE,
  maxAlignRT = 30,
  maxAlignMZ = 0.005,
  maxGroupRT = 12,
  maxGroupMZ = 0.005,
  extraOptsRT = NULL,
  extraOptsGroup = NULL,
  verbose = TRUE
)

groupFeaturesXCMS(
  feat,
  rtalign = TRUE,
  exportedData = TRUE,
  groupArgs = list(mzwid = 0.015),
  retcorArgs = list(method = "obiwarp"),
  verbose = TRUE
)

importFeatureGroupsXCMS(xs, analysisInfo)

groupFeaturesXCMS3(
  feat,
  rtalign = TRUE,
```

```
  groupParam = xcms::PeakDensityParam(sampleGroups = analysisInfo(feat)$group),
  retAlignParam = xcms::ObiwarpParam(),
  verbose = TRUE
)

importFeatureGroupsXCMS3(xdata, analysisInfo)
```

Arguments

feat	The features to be grouped. importFeatureGroupsBrukerPA and importFeatureGroupsEnviMass only support features generated by findFeaturesBruker and importFeaturesEnviMass, respectively.
algorithm	A character string describing the algorithm that should be used: "openms", "xcms", "xcms3"
...	Any parameters to be passed to the selected grouping/importing algorithm.
verbose	if FALSE then no text output will be shown.
path	The path that should be used for importing. For importFeatureGroupsBrukerPA an exported 'bucket table' '.txt' file from Bruker ProfileAnalysis, for importFeatureGroupsBrukerTASQ an exported global result table (converted to '.csv') and for importFeatureGroupsEnviMass the path of the enviMass project.
type	Which file type should be imported or exported: "brukerpa" (Bruker ProfileAnalysis), "brukertasq" (Bruker TASQ), envimass (**enviMass**, only import) or "mzmine" (MZMine, only export).
rtWindow, mzWindow, intWindow	Search window values for retention time (seconds), *m/z* (Da) and intensity used to find back features within feature groups from PA (+/- the retention/mass/intensity value of a feature).
warn	Warn about missing or duplicate features when relating them back from grouped features.
positive	Whether data from positive (TRUE) or negative (FALSE) should be loaded.
rtalign	Enable retention time alignment.
QT	If enabled, use FeatureLinkerUnlabeledQT instead of FeatureLinkerUnlabeled for feature grouping.
maxAlignRT, maxAlignMZ	Used for retention alignment. Maximum retention time or m/z difference (seconds/Dalton) for feature pairing. Sets -algorithm:pairfinder:distance_RT:max_difference and -algorithm:pairfinder:distance_MZ:max_difference otpions, respectively.
maxGroupRT, maxGroupMZ	as maxAlignRT and maxAlignMZ, but for grouping of features. Sets -algorithm:distance_RT:max_d and -algorithm:distance_MZ:max_difference options, respectively.
extraOptsRT, extraOptsGroup	Named list containing extra options that will be passed to MapAlignerPoseClustering or FeatureLinkerUnlabeledQT/FeatureLinkerUnlabeled, respectively. Any

options specified here will override any of the above. Example: `extraOptsGroup=list("-algorith` (corresponds to setting `maxGroupRT=12`). Set to `NULL` to ignore.

`exportedData` Set to `TRUE` if analyses were exported as `mzXML` or `mzML` files.

`groupArgs, retcorArgs`
named `character vector` that can contain extra parameters to be used by `xcms::group` and `xcms::retcor`, respectively.

`xs` An `xcmsSet` object.

`analysisInfo` A `data.frame` with analysis info.

`groupParam, retAlignParam`
parameter object that is directly passed to `xcms::groupChromPeaks` and `xcms::adjustRtime`, respectively.

`xdata` An `XCMSnExp` object.

Details

After features have been found the logical next step is to align and group them across analyses. This process is necessary to allow comparison of features between multiple analyses, which otherwise would be difficult due to small deviations in retention and mass data. Thus, algorithms of 'feature groupers' are used to collect features with similar retention and mass data. In addition, advanced retention time alignment algorithms exist to enhance grouping of features even with relative large retention time deviations (*e.g.* possibly observed from analyses collected over a long period). Like finding of features, various algorithms are supported which may have many parameters that can be fine-tuned. This fine-tuning is likely to be necessary, since optimal settings often depend on applied methodology and instrumentation.

`groupFeatures` is a generic function that will group features using one of the supported algorithms. The actual functionality is provided by algorithm specific functions such as `groupFeaturesOpenMS` and `groupFeaturesXCMS3`. While these functions may be called directly, `groupFeatures` provides a generic interface and is therefore usually preferred.

`importFeatureGroups` is a generic function to import feature groups produced by other software. The actual functionality is provided by specific functions such as `importFeatureGroupsBrukerPA` and `importFeatureGroupsEnviMass`.

`importFeatureGroupsBrukerPA` imports grouped features generated with Bruker ProfileAnalysis (PA). To do so, a 'bucket table' should be generated using PA and exported as '`.txt`' file. Please note that this function only supports features generated by `findFeaturesBruker` and it is **crucial** that DataAnalysis files remain unchanged when features are collected and the bucket table is generated. Furthermore, please note that PA does not retain information about originating features for generated buckets. For this reason, this function tries to find back the original features and care must be taken to correctly specify search parameters (`rtWindow`, `mzWindow`, `intWindow`).

`importFeatureGroupsEnviMass` imports grouped features ('profiles') generated with **enviMass**. Note that this function *only* imports 'raw' profiles, *not* any results from further componentization steps performed in **enviMass**. Furthermore, this functionality has only been tested with older versions of **enviMass**. Finally, please note that this function only supports features imported by `importFeaturesEnviMass` (obviously, the same project should be used for both importing functions).

`groupFeaturesOpenMS` uses the OpenMS tools for grouping of features (see `http://www.openms.de`). Retention times may be aligned by the MapAlignerPoseClustering TOPP tool. Grouping is achieved by either the FeatureLinkerUnlabeled or FeatureLinkerUnlabeledQT TOPP tools.

`groupFeaturesXCMS` uses the **xcms** package for grouping of features. Grouping of features and alignment of their retention times are performed with the `xcms::group` and `xcms::retcor` functions, respectively. Both functions have an extensive list of parameters to modify their behaviour and may therefore be used to potentially optimize results.

`importFeatureGroupsXCMS` converts grouped features from an `xcmsSet` object (from the **xcms** package).

`groupFeaturesXCMS3` uses the new interface from the **xcms** package for grouping of features. Grouping of features and alignment of their retention times are performed with the `xcms::groupChromPeaks` and `xcms::adjustRtime` functions, respectively. Both of these functions support an extensive amount of parameters that modify their behaviour and may therefore require optimization.

`importFeatureGroupsXCMS3` converts grouped features from an `XCMSnExp` object (from the **xcms** package).

Value

An object of a class which is derived from `featureGroups`.

References

Rost HL, Sachsenberg T, Aiche S, Bielow C, Weisser H, Aicheler F, Andreotti S, Ehrlich H, Gutenbrunner P, Kenar E, Liang X, Nahnsen S, Nilse L, Pfeuffer J, Rosenberger G, Rurik M, Schmitt U, Veit J, Walzer M, Wojnar D, Wolski WE, Schilling O, Choudhary JS, Malmstrom L, Aebersold R, Reinert K, Kohlbacher O (2016-sep). "OpenMS: a flexible open-source software platform for mass spectrometry data analysis." *Nature Methods*, **13**, 741–748. doi: 10.1038/nmeth.3959.

pugixml (via Rcpp) is used to process OpenMS XML output.

Dirk Eddelbuettel and Romain Francois (2011). Rcpp: Seamless R and C++ Integration. Journal of Statistical Software, 40(8), 1-18. URL http://www.jstatsoft.org/v40/i08/.

Eddelbuettel, Dirk (2013) Seamless R and C++ Integration with Rcpp. Springer, New York. ISBN 978-1-4614-6867-7.

Dirk Eddelbuettel and James Joseph Balamuta (2017). Extending R with C++: A Brief Introduction to Rcpp. PeerJ Preprints 5:e3188v1. URL https://doi.org/10.7287/peerj.preprints.3188v1.

Smith, C.A. and Want, E.J. and O'Maille, G. and Abagyan,R. and Siuzdak, G.: XCMS: Processing mass spectrometry data for metabolite profiling using nonlinear peak alignment, matching and identification, Analytical Chemistry, 78:779-787 (2006)

Ralf Tautenhahn, Christoph Boettcher, Steffen Neumann: Highly sensitive feature detection for high resolution LC/MS BMC Bioinformatics, 9:504 (2008)

H. Paul Benton, Elizabeth J. Want and Timothy M. D. Ebbels Correction of mass calibration gaps in liquid chromatography-mass spectrometry metabolomics data Bioinformatics, 26:2488 (2010)

Smith, C.A. and Want, E.J. and O'Maille, G. and Abagyan,R. and Siuzdak, G.: XCMS: Processing mass spectrometry data for metabolite profiling using nonlinear peak alignment, matching and identification, Analytical Chemistry, 78:779-787 (2006)

Ralf Tautenhahn, Christoph Boettcher, Steffen Neumann: Highly sensitive feature detection for high resolution LC/MS BMC Bioinformatics, 9:504 (2008)

H. Paul Benton, Elizabeth J. Want and Timothy M. D. Ebbels Correction of mass calibration gaps in liquid chromatography-mass spectrometry metabolomics data Bioinformatics, 26:2488 (2010)

See Also

featureGroups-class

`feature-optimization` *Optimization of feature finding and grouping parameters*

Description

Automatic optimization of feature finding and grouping parameters through Design of Experiments (DoE).

Usage

```
optimizeFeatureGrouping(
  features,
  algorithm,
  ...,
  templateParams = list(),
  paramRanges = list(),
  maxIterations = 50,
  maxModelDeviation = 0.1
)

generateFGroupsOptPSet(algorithm, ...)

getDefFGroupsOptParamRanges(algorithm)

optimizeFeatureFinding(
  anaInfo,
  algorithm,
  ...,
```

```
  templateParams = list(),
  paramRanges = list(),
  isoIdent = if (algorithm == "openms") "OpenMS" else "IPO",
  checkPeakShape = "none",
  CAMERAOpts = list(),
  maxIterations = 50,
  maxModelDeviation = 0.1
)

generateFeatureOptPSet(algorithm, ...)

getDefFeaturesOptParamRanges(algorithm, method = "centWave")
```

Arguments

features	A features object with the features that should be used to optimize grouping.
algorithm	The algorithm used for finding or grouping features (see findFeatures and groupFeatures).
...	One or more lists with parameter sets (see below) (for optimizeFeatureFinding and optimizeFeatureGrouping). Alternatively, named arguments that set (and possibly override) the parameters that should be returned from generateFeatureOptPSet or generateFGroupsOptPSet.
templateParams	Template parameter set (see below).
paramRanges	A list with vectors containing absolute parameter ranges (minimum/maximum) that constrain numeric parameters choosen during experiments. See the getDefFeaturesOptParamRanges and getDefFGroupsOptParamRanges functions for defaults. Values should be Inf when no limit should be used.
maxIterations	Maximum number of iterations that may be performed to find optimimum values. Used to restrict neededless long optimization procedures. In IPO this was fixed to '50'.
maxModelDeviation	See the Potential suboptimal results by optimization model section below.
anaInfo	Analysis info table (passed to findFeatures).
isoIdent	Sets the algorithm used to identify isotopes. Valid values are: "IPO", "CAMERA" and "OpenMS". The latter can only be used when OpenMS is used to find features, and is highly recommended in this situation.
checkPeakShape	Additional peak shape checking of isotopes. Only used if isoIdent="IPO". Valid values: "none", "borderIntensity", "sinusCurve" or "normalDistr".
CAMERAOpts	A list with additional arguments passed to CAMERA::findIsotopes when isoIdent="CAMERA".
method	Method used by XCMS to find features (only if algorithm="xcms").

Details

Many different parameters exist that may affect the output quality of feature finding and grouping. To avoid time consuming manual experimentation, functionality is provided to largely automate the optimization process. The methodology, which uses design of experiments (DoE), is based on the excellent Isotopologue Parameter Optimization (IPO) R package. The functionality of this package is directly integrated in patRoon. Some functionality was added or changed, however, the principle algorithm workings are nearly identical.

Compared to IPO, the following functionality was added or changed:

- The code was made more generic in order to include support for other feature finding/grouping algorithms (*e.g.* OpenMS, enviPick, XCMS3).
- The methodology of `FeatureFinderMetabo` (OpenMS) may be used to find isotopes.
- The `maxModelDeviation` parameter was added to potentially avoid suboptimal results (issue discussed here).
- The use of multiple 'parameter sets' (discussed below) which, for instance, allow optimizing qualitative paremeters more easily (see examples).
- More consistent optimization code for feature finding/grouping.
- More consistent output using S4 classes (*i.e.* `optimizationResult` class).
- Experiments are not (yet) executed in parallel (although feature finding or grouping may be if the algorithm supports it).

Value

The `optimizeFeatureFinding` and `optimizeFeatureGrouping` return their results in a `optimizationResult` object.

Parameter sets

Which parameters should be optimized is determined by a *parameter set*. A set is defined by a named `list` containing the minimum and maximum starting range for each parameter that should be tested. For instance, the set `list(chromFWHM = c(5,10),mzPPM = c(5,15))` specifies that the `chromFWHM` and `mzPPM` parameters (used by OpenMS feature finding) should be optimized within a range of '`5`'-'`10`' and '`5`'-'`15`', respectively. Note that this range may be increased or decreased after a DoE iteration in order to find a better optimum. The absolute limits are controlled by the `paramRanges` function argument.

Multiple parameter sets may be specified (*i.e.* through the ... function argument). In this situation, the optimization algorithm is repeated for each set, and the final optimum is determined from the parameter set with the best response. The `templateParams` function argument may be useful in this case to define a template for each parameter set. Actual parameter sets are then constructed by joining each parameter set with the set specified for `templateParams`. When a parameter is defined in both a regular and template set, the parameter in the regular set takes precedence.

Parameters that should not be optimized but still need to be set for the feature finding/grouping functions should also be defined in a (template) parameter set. Which parameters should be optimized is determined whether its value is specified as a vector range or

a single fixed value. For instance, when a set is defined as `list(chromFWHM = c(5,10),mzPPM = 5)`, only the `chromFWHM` parameter is optimized, whereas `mzPPM` is kept constant at '5'.

Using multiple parameter sets with differing fixed values allows optimization of qualitative values (see examples below).

The parameters specified in parameter sets are directly passed through the `findFeatures` or `groupFeatures` functions. Hence, grouping and retention time alignment parameters used by XCMS should (still) be set through the `groupArgs` and `retcorArgs` parameters.

NOTE: For XCMS3, which normally uses parameter classes for settings its options, the parameters must be defined in a named list like any other algorithm. The set parameters are then used to automatically constructor of the right parameter class object (e.g. `CentWaveParam`, `ObiwarpParam`). For grouping/alignment sets, these parameters need to be specified in nested lists called `groupParams` and `retAlignParams`, respectively (similar to `groupArgs`/`retcorArgs` for `algorithm="xcms"`). Finally, the underlying XCMS method to be used should be defined in the parameter set (*i.e.* by setting the `method` field for feature parameter sets and the `groupMethod` and `retAlignMethod` for grouping/aligning parameter sets). See the examples below for more details.

NOTE: Similar to IPO, the `peakwidth` and `prefilter` parameters for XCMS feature finding should be split in two different values:

- The minimum and maximum ranges for `peakwidth` are optimized by setting `min_peakwidth` and `max_peakwidth`, respectively.
- The `k` and `I` parameters contained in `prefilter` are split in `prefilter` and `value_of_prefilter`, respectively.

Functions

The `optimizeFeatureFinding` and `optimizeFeatureGrouping` are the functions to be used to optimize parameters for feature finding and grouping, respectively. These functions are analogous to `optimizeXcmsSet` and `optimizeRetGroup` from **IPO**.

The `generateFeatureOptPSet` and `generateFGroupsOptPSet` functions may be used to generate a parameter set for feature finding and grouping, respectively. Some algorithm dependent default parameter optimization ranges will be returned. These functions are analogous to `getDefaultXcmsSetStartingParams` and `getDefaultRetGroupStartingParams` from **IPO**. However, unlike their IPO counterparts, these functions will not output default fixed values. The `generateFGroupsOptPSet` will only generate defaults for density grouping if `algorithm="xcms"`.

The `getDefFeaturesOptParamRanges` and `getDefFGroupsOptParamRanges` return the default absolute optimization parameter ranges for feature finding and grouping, respectively. These functions are useful if you want to set the `paramRanges` function argument.

Potential suboptimal results by optimization model

After each experiment iteration an optimimum parameter set is found by generating a model containing the tested parameters and their responses. Sometimes the actual response from the parameters derived from the model is actually signficantly lower than expected. When the response is lower than the maximum reponse found during the experiment, the parameters belonging to this experimental maximum may be choosen instead.

The `maxModelDeviation` argument sets the maximum deviation in response between the modelled and experimental maxima. The value is relative: '`0`' means that experimental values will always be favored when leading to improved responses, whereas `1` will effectively disable this procedure (and return to 'regular' IPO behaviour).

Source

The code and methodology is a direct adaptation from the IPO R package.

References

Libiseller G, Dvorzak M, Kleb U, Gander E, Eisenberg T, Madeo F, Neumann S, Trausinger G, Sinner F, Pieber T, Magnes C (2015-apr). "IPO: a tool for automated optimization of XCMS parameters." *BMC Bioinformatics*, **16**. doi: 10.1186/s1285901505628.

Examples

```
# example data from patRoonData package
dataDir <- patRoonData::exampleDataPath()
anaInfo <- generateAnalysisInfo(dataDir)
anaInfo <- anaInfo[1:2, ] # only focus on first two analyses (e.g. training set)

# optimize mzPPM and chromFWHM parameters
ftOpt <- optimizeFeatureFinding(anaInfo, "openms", list(mzPPM = c(5, 10), chromFWHM = c(4, 8)))

# optimize chromFWHM and isotopeFilteringModel (a qualitative parameter)
ftOpt2 <- optimizeFeatureFinding(anaInfo, "openms",
                                 list(isotopeFilteringModel = "metabolites (5% RMS)"),
                                 list(isotopeFilteringModel = "metabolites (2% RMS)"),
                                 templateParams = list(chromFWHM = c(4, 8)))

# perform grouping optimization with optimized features object
fgOpt <- optimizeFeatureGrouping(optimizedObject(ftOpt), "xcms",
                                 list(groupArgs = list(bw = c(22, 28)),
                                      retcorArgs = list(method = "obiwarp")))

# same, but using the XCMS3 interface
fgOpt2 <- optimizeFeatureGrouping(optimizedObject(ftOpt), "xcms3",
                          list(groupMethod = "density", groupParams = list(bw = c(22, 28)),
                                       retAlignMethod = "obiwarp"))


# plot contour of first parameter set/DoE iteration
plot(ftOpt, paramSet = 1, DoEIteration = 1, type = "contour")

# generate parameter set with some predefined and custom parameters to be
# optimized.
pSet <- generateFeatureOptPSet("openms", chromSNR = c(3, 9),
                               useSmoothedInts = FALSE)
```

`featureGroups-class` *Base class for grouped features.*

Description

This class holds all the information for grouped features.

Usage

```
## S4 method for signature 'featureGroups'
names(x)

## S4 method for signature 'featureGroups'
analyses(obj)

## S4 method for signature 'featureGroups'
replicateGroups(obj)

## S4 method for signature 'featureGroups'
groupNames(obj)

## S4 method for signature 'featureGroups'
length(x)

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

## S4 method for signature 'featureGroups'
groups(object, areas = FALSE)

## S4 method for signature 'featureGroups'
analysisInfo(obj)

## S4 method for signature 'featureGroups'
groupInfo(fGroups)

## S4 method for signature 'featureGroups'
featureTable(obj)

## S4 method for signature 'featureGroups'
getFeatures(obj)

## S4 method for signature 'featureGroups'
groupFeatIndex(fGroups)

## S4 method for signature 'featureGroups,ANY,ANY,missing'
x[i, j, ..., rGroups, drop = TRUE]
```

```
## S4 method for signature 'featureGroups,ANY,ANY'
x[[i, j]]

## S4 method for signature 'featureGroups'
x$name

## S4 method for signature 'featureGroups'
export(fGroups, type, out)

## S4 method for signature 'featureGroups'
as.data.table(
  x,
  average = FALSE,
  areas = FALSE,
  features = FALSE,
  regression = FALSE
)

## S4 method for signature 'featureGroups,ANY'
plot(
  x,
  colourBy = c("none", "rGroups", "fGroups"),
  onlyUnique = FALSE,
  retMin = FALSE,
  showLegend = TRUE,
  col = NULL,
  pch = NULL,
  ...
)

## S4 method for signature 'featureGroups'
plotInt(obj, average = FALSE, pch = 20, type = "b", lty = 3, col = NULL, ...)

## S4 method for signature 'featureGroups'
plotChord(
  obj,
  addSelfLinks = FALSE,
  addRetMzPlots = TRUE,
  average = FALSE,
  outerGroups = NULL,
  addIntraOuterGroupLinks = FALSE,
  ...
)

## S4 method for signature 'featureGroups'
plotEIC(
  obj,
```

```
  rtWindow = 30,
  mzWindow = 0.005,
  retMin = FALSE,
  topMost = NULL,
  EICs = NULL,
  showPeakArea = FALSE,
  showFGroupRect = TRUE,
  title = NULL,
  colourBy = c("none", "rGroups", "fGroups"),
  showLegend = TRUE,
  onlyPresent = TRUE,
  annotate = c("none", "ret", "mz"),
  showProgress = FALSE,
  xlim = NULL,
  ylim = NULL,
  ...
)

## S4 method for signature 'featureGroups'
plotVenn(obj, which = NULL, ...)

## S4 method for signature 'featureGroups'
plotUpSet(obj, which = NULL, nsets = length(which), nintersects = NA, ...)

## S4 method for signature 'featureGroups'
unique(x, which, relativeTo = NULL, outer = FALSE)

## S4 method for signature 'featureGroups'
overlap(fGroups, which, exclusive = FALSE)
```

Arguments

`areas` If set to `TRUE` then areas are considered instead of peak intensities.

`fGroups, obj, x, object`
`featureGroups` object to be accessed.

`i, j` A numeric or character value which is used to select analyses/feature groups by their index or name, respectively (for the order/names see `analyses()/names()`).

For `[`: Can also be logical to perform logical selection (similar to regular vectors). If missing all analyses/feature groups are selected.

For `[[`: should be a scalar value. If `j` is not specified, `i` selects by feature groups instead.

`...` Ignored for `"["` operator or passed to `plot` (`plot` and `plotEIC`), `lines` (`plotInt`), **VennDiagram** plotting functions (`plotVenn`), `chordDiagram` (`plotChord`) or `upset` (`plotUpSet`).

`rGroups` An optional `character` vector: if specified only keep results for the given replicate groups (equivalent to the `rGroups` argument to `filter`).

`drop` ignored.

`name` The feature group name (partially matched).

`out` The destination file for the exported data.

`average` Average data within replicate groups.

`features` If `TRUE` then feature specific data will be added. If `average=TRUE` this data will be averaged for each feature group.

`regression` Set to `TRUE` to add regression data for each feature group. For this a linear model is created (intensity/area [depending on `areas` argument] *vs* concentration). The model concentrations (e.g. of a set of standards) is derived from the `conc` column of the analysis information. From this model the intercept, slope and R2 is added to the output. In addition, when `features=TRUE`, concentrations for each feature are added. Note that no regression information is added when no `conc` column is present in the analysis information or when less than two concentrations are specified (*i.e.* the minimum amount).

`colourBy` Sets the automatic colour selection: `"none"` for a single colour or `"rGroups"`/`"fGroups"` for a distinct colour per replicate/feature group.

`onlyUnique` If `TRUE` and `colourBy="rGroups"` then only feature groups that are unique to a replicate group are plotted.

`retMin` Plot retention time in minutes (instead of seconds).

`showLegend` If `TRUE` a legend will be shown with either replicate groups (`colourBy == "rGroups"`) or feature groups (`colourBy == "fGroups"`, only for `plotEIC`). If `colourBy` is `"none"` no legend will be shown.

`col` Colour(s) used. If `col=NULL` then colours are automatically generated.

`pch, type, lty` Common plotting parameters passed to *e.g.* `plot`. For `plot`: if `pch=NULL` then values are automatically assigned.

`addSelfLinks` If `TRUE` then 'self-links' are added which represent non-shared data.

`addRetMzPlots` Set to `TRUE` to enable *m/z vs* retention time scatter plots.

`outerGroups` Character vector of names to be used as outer groups. The values in the specified vector should be named by analysis names (`average` set to `FALSE`) or replicate group names (`average` set to `TRUE`), for instance: `c(analysis1 = "group1",analysis2 = "group1",analysis3 = "group2")`. Set to `NULL` to disable outer groups.

`addIntraOuterGroupLinks` If `TRUE` then links will be added within outer groups.

`rtWindow` Retention time (in seconds) that will be subtracted/added to respectively the minimum and maximum retention time of the plotted feature groups. Thus, setting this value to a positive value will 'zoom out' on the retention time axis.

`mzWindow` The *m/z* value (in Da) which will be subtracted/added to a feature group *m/z* value to determine the width of its EIC.

topMost — Only plot EICs from features within this number of top most intense analyses. If NULL then all analyses are used for plotted.

EICs — Internal parameter for now and should be kept at NULL (default).

showPeakArea — Set to TRUE to display integrated chromatographic peak ranges by filling (shading) their areas.

showFGroupRect — Set to TRUE to mark the full retention/intensity range of all features within a feature group by drawing a rectangle around it.

title — Character string used for title of the plot. If NULL a title will be automatically generated.

onlyPresent — If TRUE then EICs will only be generated for analyses in which a particular feature group was detected. Disabling this option might be useful to see if any features were 'missed'.

annotate — If set to "ret" and/or "mz" then retention and/or m/z values will be drawn for each plotted feature group.

showProgress — if set to TRUE then a text progressbar will be displayed when all EICs are being plot. Set to "none" to disable any annotation.

xlim, ylim — Sets the plot size limits used by plot. Set to NULL for automatic plot sizing.

which — A character vector with replicate groups used for comparison. For plotting functions: set to NULL for all replicate groups.

nsets, nintersects — See upset.

relativeTo — A character vector with replicate groups that should be used for unique comparison. If NULL then all replicate groups are used for comparison. Replicate groups specified in which are ignored.

outer — If TRUE then only feature groups are kept which do not overlap between the specified replicate groups for the which parameter.

exclusive — If TRUE then all feature groups are removed that are not unique to the given replicate groups.

Details

The featureGroup class is the workhorse of **patRoon**: almost all functionality operate on its instantiated objects. The class holds all information from grouped features (obtained from features). This class itself is virtual, hence, objects are not created directly from it. Instead, 'feature groupers' such as groupFeaturesXCMS return a featureGroups derived object after performing the actual grouping of features across analyses.

Value

plotVenn (invisibly) returns a list with the following fields:

- gList the gList object that was returned by the utilized **VennDiagram** plotting function.
- areas The total area for each plotted group.

- `intersectionCounts` The number of intersections between groups.

The order for the `areas` and `intersectionCounts` fields is the same as the parameter order from the used plotting function (see *e.g.* `draw.pairwise.venn` and `draw.triple.venn`).

Methods (by generic)

- `names`: Obtain feature group names.
- `analyses`: returns a `character` vector with the names of the analyses for which data is present in this object.
- `replicateGroups`: returns a `character` vector with the names of the replicate groups for which data is present in this object.
- `groupNames`: Same as `names`. Provided for consistency to other classes.
- `length`: Obtain number of feature groups.
- `show`: Shows summary information for this object.
- `groups`: Accessor for `groups` slot.
- `analysisInfo`: Obtain analysisInfo (see analysisInfo slot in `features`).
- `groupInfo`: Accessor for `groupInfo` slot.
- `featureTable`: Obtain feature information (see `features`).
- `getFeatures`: Accessor for `features` slot.
- `groupFeatIndex`: Accessor for `ftindex` slot.
- `[`: Subset on analyses/feature groups.
- `[[`: Extract intensity values.
- `$`: Extract intensity values for a feature group.
- `export`: Exports feature groups to a '`.csv`' file that is readable to Bruker ProfileAnalysis (a 'bucket table'), Bruker TASQ (an analyte database) or that is suitable as input for the Targeted peak
detection functionality of MZmine.
- `as.data.table`: Obtain a summary table (a `data.table`) with retention, *m/z*, intensity and optionally other feature data.
- `plot`: Generates an *m/z vs* retention time plot for all featue groups. Optionally highlights unique/overlapping presence amongst replicate groups.
- `plotInt`: Generates a line plot for the (averaged) intensity of feature groups within all analyses
- `plotChord`: Generates a chord diagram which can be used to visualize shared presence of feature groups between analyses or replicate groups. In addition, analyses/replicates sharing similar properties (*e.g.* location, age, type) may be grouped to enhance visualization between these 'outer groups'.
- `plotEIC`: Plots extracted ion chromatograms (EICs) of feature groups.
- `plotVenn`: plots a Venn diagram (using **VennDiagram**) outlining unique and shared feature groups between up to five replicate groups.
- `plotUpSet`: plots an UpSet diagram (using the `upset` function) outlining unique and shared feature groups between given replicate groups.

- `unique`: Obtain a subset with unique feature groups present in one or more specified replicate group(s).
- `overlap`: Obtain a subset with feature groups that overlap between a set of specified replicate group(s).

Slots

`groups` Matrix (`data.table`) with intensities for each feature group (columns) per analysis (rows). Access with `groups` method.

`analysisInfo,features` Analysis info and `features` class associated with this object. Access with `analysisInfo` and `featureTable` methods, respectively.

`groupInfo` `data.frame` with retention time (`rts` column, in seconds) and *m/z* (`mzs` column) for each feature group. Access with `groupInfo` method.

`ftindex` Matrix (`data.table`) with feature indices for each feature group (columns) per analysis (rows). Each index corresponds to the row within the feature table of the analysis (see `featureTable`).

S4 class hierarchy

- `workflowStep`
  - `featureGroups`
    - `featureGroupsBruker`
    - `featureGroupsConsensus`
    - `featureGroupsEnviMass`
    - `featureGroupsOpenMS`
    - `featureGroupsScreening`
    - `featureGroupsBrukerTASQ`
    - `featureGroupsXCMS`
    - `featureGroupsXCMS3`

References

Gu, Z. (2014) circlize implements and enhances circular visualization in R. Bioinformatics.

Conway JR, Lex A, Gehlenborg N (2017). "UpSetR: an R package for the visualization of intersecting sets and their properties." *Bioinformatics*, **33**, 2938–2940. doi: 10.1093/bioinformatics/btx364.

Lex A, Gehlenborg N, Strobelt H, Vuillemot R, Pfister H (2014-dec). "UpSet: Visualization of Intersecting Sets." *IEEE Transactions on Visualization and Computer Graphics*, **20**, 1983–1992. doi: 10.1109/tvcg.2014.2346248.

`featureGroups-compare` *Comparing feature groups*

Description

Functionality to compare feature groups and make a consensus.

Usage

```
## S4 method for signature 'featureGroups'
comparison(..., groupAlgo, groupArgs = list(rtalign = FALSE))

## S4 method for signature 'featureGroupsComparison,ANY'
plot(x, retMin = TRUE, ...)

## S4 method for signature 'featureGroupsComparison'
plotVenn(obj, which = NULL, ...)

## S4 method for signature 'featureGroupsComparison'
plotUpSet(obj, which = NULL, ...)

## S4 method for signature 'featureGroupsComparison'
plotChord(obj, addSelfLinks = FALSE, addRetMzPlots = TRUE, ...)

## S4 method for signature 'featureGroupsComparison'
consensus(
  obj,
  absMinAbundance = NULL,
  relMinAbundance = NULL,
  uniqueFrom = NULL,
  uniqueOuter = FALSE
)
```

Arguments

`...`	For `comparison`: `featureGroups` objects that should be compared. If the arguments are named (*e.g.* `myGroups = fGroups`) then these are used for labelling, otherwise objects are automatically labelled by their `algorithm`. For `plot`, `plotVenn`, `plotChord`: further options passed to `plot`, **VennDiagram** plotting functions (*e.g.* `draw.pairwise.venn`) and `chordDiagram` respectively. For `plotUpSet`: any further arguments passed to the `plotUpSet` method defined for `featureGroups`.
`groupAlgo`	The `feature grouping algorithm` that should be used for grouping *pseudo* features (see details). Valid values are: `"xcms"` or `"openms"`. Note: `xcms3` is not (yet) supported.

groupArgs	A list containing further parameters for feature grouping.
x, obj	The featureGroupsComparison object.
retMin	If TRUE retention times are plotted as minutes (seconds otherwise).
which	A character vector specifying one or more labels of compared feature groups. For plotVenn: if NULL then all compared groups are used.
addSelfLinks	If TRUE then 'self-links' are added which represent non-shared data.
addRetMzPlots	Set to TRUE to enable *m/z vs* retention time scatter plots.
absMinAbundance, relMinAbundance	Minimum absolute or relative ('0-1') abundance across objects for a result to be kept. For instance, relMinAbundance=0.5 means that a result should be present in at least half of the number of compared objects. Set to 'NULL' to ignore and keep all results. Limits cannot be set when uniqueFrom is not NULL.
uniqueFrom	Set this argument to only retain feature groups that are unique within one or more of the objects for which the consensus is made. Selection is done by setting the value of uniqueFrom to a logical (values are recycled), numeric (select by index) or a character (as obtained with algorithm(obj)). For logical and numeric values the order corresponds to the order of the objects given for the consensus. Set to NULL to ignore.
uniqueOuter	If uniqueFrom is not NULL and if uniqueOuter=TRUE: only retain data that are also unique between objects specified in uniqueFrom.

Details

Feature groups objects originating from differing feature finding and/or grouping algorithms (or their parameters) may be compared to assess their output and generate a consensus.

The comparison method generates a featureGroupsComparison object from given feature groups objects, which in turn may be used for (visually) comparing presence of feature groups and generating a consensus. Internally, this function will collapse each feature groups object to *pseudo* features objects by averaging their retention times, *m/z* values and intensities, where each original feature groups object becomes an 'analysis'. All *pseudo* features are then grouped using regular feature grouping algorithms so that a comparison can be made.

plot generates an *m/z vs* retention time plot.

plotVenn plots a Venn diagram outlining unique and shared feature groups between up to five compared feature groups.

plotUpSet plots an UpSet diagram outlining unique and shared feature groups.

plotChord plots a chord diagram to visualize the distribution of feature groups.

consensus combines all compared feature groups and averages their retention, *m/z* and intensity data.

Value

comparison returns a featureGroupsComparison object.

plotVenn (invisibly) returns a list with the following fields:

- `gList` the `gList` object that was returned by the utilized **VennDiagram** plotting function.
- `areas` The total area for each plotted group.
- `intersectionCounts` The number of intersections between groups.

The order for the `areas` and `intersectionCounts` fields is the same as the parameter order from the used plotting function (see *e.g.* `draw.pairwise.venn` and `draw.triple.venn`).

`consensus` returns a `featureGroups` object with a consensus from the compared feature groups.

`featureGroupsComparison-class`

Feature groups comparison class

Description

This class is used for comparing different `featureGroups` objects.

Usage

```
## S4 method for signature 'featureGroupsComparison'
names(x)

## S4 method for signature 'featureGroupsComparison'
length(x)

## S4 method for signature 'featureGroupsComparison,ANY,missing,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'featureGroupsComparison,ANY,missing'
x[[i, j]]

## S4 method for signature 'featureGroupsComparison'
x$name
```

Arguments

`x`	A `featureGroupsComparison` object.
`i`	A numeric or character value which is used to select labels by their index or name, respectively (for the order/names see `names()`). For `[`: Can also be logical to perform logical selection (similar to regular vectors). If missing all labels are selected. For `[[`: should be a scalar value.
`...`	Ignored.
`drop, j`	ignored.
`name`	The label name (partially matched).

Details

Objects from this class are returned by comparison.

Methods (by generic)

- names: Obtain the labels that were given to each compared feature group.
- length: Number of feature groups objects that were compared.
- [: Subset on labels that were assigned to compared feature groups.
- [[: Extract a featureGroups object by its label.
- $: Extract a compound table for a feature group.

Slots

fGroupsList A list of featureGroups object that were compared

comparedFGroups A *pseudo* featureGroups object containing grouped feature groups.

---

features-class *Base features class*

Description

Holds information for all features present within a set of analysis.

Usage

```
## S4 method for signature 'features'
length(x)

## S4 method for signature 'features'
show(object)

## S4 method for signature 'features'
featureTable(obj)

## S4 method for signature 'features'
analysisInfo(obj)

## S4 method for signature 'features'
analyses(obj)

## S4 method for signature 'features'
replicateGroups(obj)

## S4 method for signature 'features'
as.data.table(x)
```

```
## S4 method for signature 'features'
filter(
  obj,
  absMinIntensity = NULL,
  relMinIntensity = NULL,
  retentionRange = NULL,
  mzRange = NULL,
  mzDefectRange = NULL,
  chromWidthRange = NULL,
  negate = FALSE
)

## S4 method for signature 'features,ANY,missing,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'features,ANY,missing'
x[[i]]

## S4 method for signature 'features'
x$name

## S4 method for signature 'featuresXCMS,ANY,missing,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'featuresXCMS'
filter(obj, ...)

## S4 method for signature 'featuresXCMS3,ANY,missing,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'featuresXCMS3'
filter(obj, ...)
```

Arguments

`obj, x, object`	`features` object to be accessed
`absMinIntensity, relMinIntensity`	Minimum absolute/relative intensity for features to be kept. The relative intensity is determined from the feature with highest intensity (within the same analysis). Set to '`0`' or `NULL` to skip this step.
`retentionRange, mzRange, mzDefectRange, chromWidthRange`	Range of retention time (in seconds), *m/z*, mass defect (defined as the decimal part of *m/z* values) or chromatographic peak width (in seconds), respectively. Features outside this range will be removed. Should be a numeric vector with length of two containing the min/max values. The maximum can be `Inf` to specify no maximum range. Set to `NULL` to skip this step.
`negate`	If set to `TRUE` then filtering operations are performed in opposite manner.

i	A numeric or character value which is used to select analyses by their index or name, respectively (for the order/names see analyses()). For [: Can also be logical to perform logical selection (similar to regular vectors). If missing all analyses are selected. For [[: should be a scalar value.
...	Ignored.
drop, j	ignored.
name	The analysis name (partially matched).

Details

This class provides a way to store intensity, retention times, *m/z* and other data for all features in a set of analyses. The class is virtual and derived objects are created by 'feature finders' such as findFeaturesOpenMS, findFeaturesXCMS and findFeaturesBruker.

Value

featureTable: A list containing a data.table for each analysis with feature data

analysisInfo: A data.frame containing a column with analysis name (analysis), its path (path), and other columns such as replicate group name (group) and blank reference (blank).

Methods (by generic)

- length: Obtain total number of features.
- show: Shows summary information for this object.
- featureTable: Get table with feature information
- analysisInfo: Get analysis information
- analyses: returns a character vector with the names of the analyses for which data is present in this object.
- replicateGroups: returns a character vector with the names of the replicate groups for which data is present in this object.
- as.data.table: Returns all feature data in a table.
- filter: Performs common rule based filtering of features. Note that this (and much more) functionality is also provided by the filter method defined for featureGroups. However, filtering a features object may be useful to avoid grouping large amounts of features.
- [: Subset on analyses.
- [[: Extract a feature table for an analysis.
- $: Extract a feature table for an analysis.

Slots

features List of features per analysis file. Use the featureTable method for access.

analysisInfo Analysis group information. Use the analysisInfo method for access.

S4 class hierarchy

- workflowStep
  - features
    - featuresFromFeatGroups
    - featuresConsensus
    - featuresBruker
    - featuresEnviPick
    - featuresOpenMS
    - featuresBrukerTASQ
    - featuresXCMS
    - featuresXCMS3

See Also

feature-finding

---

formula-generation *Automatic chemical formula generation*

Description

Functionality to automatically calculate chemical formulae for all feature groups.

Usage

```
## S4 method for signature 'featureGroups'
generateFormulas(fGroups, algorithm, ...)

generateFormulasDA(
  fGroups,
  precursorMzSearchWindow = 0.002,
  MSMode = "both",
  adduct,
  featThreshold = 0.75,
  save = TRUE,
  close = save
)

generateFormulasGenForm(
  fGroups,
  MSPeakLists,
  relMzDev = 5,
  adduct = "[M+H]+",
  elements = "CHNOP",
  hetero = TRUE,
```

```
  oc = FALSE,
  extraOpts = NULL,
  calculateFeatures = TRUE,
  featThreshold = 0.75,
  MSMode = "both",
  isolatePrec = TRUE,
  timeout = 120,
  topMost = 50,
  maxProcAmount = getOption("patRoon.maxProcAmount"),
  batchSize = 8
)

generateFormulasSIRIUS(
  fGroups,
  MSPeakLists,
  relMzDev = 5,
  adduct = "[M+H]+",
  elements = "CHNOP",
  profile = "qtof",
  database = NULL,
  noise = NULL,
  cores = NULL,
  topMost = 100,
  extraOptsGeneral = NULL,
  extraOptsFormula = NULL,
  calculateFeatures = TRUE,
  featThreshold = 0.75,
  verbose = TRUE,
  SIRBatchSize = 0,
  logPath = file.path("log", "sirius_formulas"),
  maxProcAmount = getOption("patRoon.maxProcAmount")
)

formulaScorings()
```

Arguments

fGroups	featureGroups object for which formulae should be generated. This should be the same or a subset of the object that was used to create the specified MSPeakLists (only relevant for algorithms using MSPeakLists). In the case of a subset only the remaining feature groups in the subset are considered.
algorithm	A character string describing the algorithm that should be used: "bruker", "genform", "sirius"
...	Any parameters to be passed to the selected formula generation algorithm.
precursorMzSearchWindow	Search window for m/z values (+/- the feature m/z) used to find back feature data of precursor/parent ions from MS/MS spectra (this data is

not readily available from `SmartFormula3D` results).

`MSMode` Whether formulae should be generated only from MS data (`"ms"`), MS/MS data (`"msms"`) or both (`"both"`).

For `GenForm` selecting `"both"` will fall back to formula calculation with only MS data in case no MS/MS data is available.

For calulation with Bruker DataAnalysis selecting `"both"` will calculate formulae from MS data *and* MS/MS data and combines the results (duplicated formulae are removed). This is useful when poor MS/MS data would exclude proper candidates.

`adduct` An `adduct` object (or something that can be converted to it with `as.adduct`). Examples: `"[M-H]-"`, `"[M+Na]+"`.

`featThreshold` If `calculateFeatures=TRUE`: minimum presence ('`0-1`') of a formula in all features before it is considered as a candidate for a feature group. For instance, `featThreshold` dictates that a formula should be present in at least 75 feature group.

`close, save` If `TRUE` then Bruker files are closed and saved after processing with DataAnalysis, respectively. Setting `close=TRUE` prevents that many analyses might be opened simultaneously in DataAnalysis, which otherwise may use excessive memory or become slow. By default `save` is `TRUE` when `close` is `TRUE`, which is likely what you want as otherwise any processed data is lost.

`MSPeakLists` An `MSPeakLists` object that was generated for the supplied `fGroups`.

`relMzDev` Maximum relative deviation between the measured and candidate formula m/z values (in ppm). Sets the '`ppm`' and '`--ppm-max`' commandline options for `GenForm` and `SIRIUS`, respectively.

`elements` Elements to be considered for formulae calculation. This will heavily affects the number of candidates! Always try to work with a minimal set by excluding elements you don't expect. For `generateFormulasSIRIUS`, the minimum/maximum number of elements can also be specified, for example: a value of `"C[5]H[10-15]O"` will only consider formulae with up to five carbon atoms, between ten and fifteen hydrogen atoms and any amount of oxygen atoms. Sets the '`el`' and '`--elements`' commandline options for `GenForm` and `SIRIUS`, respectively.

`hetero` Only consider formulae with at least one hetero atom. Sets the '`het`' commandline option.

`oc` Only consider organic formulae (*i.e.* with at least one carbon atom). Sets the '`oc`' commandline option.

`extraOpts` An optional character vector with any other commandline options that will be passed to `GenForm` or `SIRIUS`. See the GenForm options section/SIRIUS manual for all available commandline options.

`calculateFeatures`

If `TRUE` fomulae are first calculated for all features prior to feature group assignment (see details).

`isolatePrec` Settings used for isolation of precursor mass peaks and their isotopes. This isolation is highly important for accurate isotope scoring of candidates, as

non-relevant mass peaks will dramatically decrease the score. The value of `isolatePrec` should either be a `list` with parameters (see the `filter` method for `MSPeakLists` for more details), `TRUE` for default parameters (the `z` parameter is automatically deduced from the `adduct` argument) or `FALSE` for no isolation (*e.g.* when you already performed isolation with the `filter` method).

`timeout` Maximum time (in seconds) that a `GenForm` command is allowed to execute. If this time is exceeded a warning is emitted and the command is terminated. See the notes section for more information on the need of timeouts.

`topMost` Only keep this number of candidates (per feature group) with highest score. For `SIRIUS`: Sets the '`--candidates`' commandline option.

`maxProcAmount` Maximum number of processes to run for parallelization. Usually a number close to the amount of physical cores yields most efficient results.

`batchSize` Maximum number of `GenForm` commands that should be run sequentially in each parallel process. Combining commands with short runtimes (such as `GenForm`) can significantly increase parallel performance. For more information see `executeMultiProcess`.

`profile` Name of the configuration profile, for example: '`"qtof"`', '`"orbitrap"`', '`"fticr"`'. Sets the '`--profile`' commandline option.

`database` If not `NULL`, use a database for retrieval of formula candidates. Possible values are: '`"pubchem"`', '`"bio"`', '`"kegg"`', '`"hmdb"`'. Sets the '`--database`' commandline option.

`noise` Median intensity of the noise (`NULL` ignores this parameter). Sets the '`--noise`' commandline option.

`cores` The number of cores `SIRIUS` will use. If `NULL` then the default of all cores will be used.

`extraOptsGeneral, extraOptsFormula`
a `character` vector with any extra commandline parameters for `SIRIUS`. For `SIRIUS` versions `<4.4` there is no distinction between general and formula options. Otherwise commandline options specified in `extraOptsGeneral` are added prior to the `formula` command, while options specified in `extraOptsFormula` are added in afterwards. See the `SIRIUS` manual for more details. Set to `NULL` to ignore.

`verbose` If `TRUE` then more output is shown in the terminal.

`SIRBatchSize` The maximum number of calculations done by `SIRIUS`. If this number is less than the amount of features to be calculated then calculations will be evenly split over multiple `SIRIUS` calls (which may be run in parallel if `maxProcAmount>1`). If `SIRBatchSize=0` then all feature calculations are performed from a single `SIRIUS` exection, which is often the fastest.

`logPath` Destination directory for log files with output from executed commands. Will be created if non-existant. Set to `NULL` to disable logging.

Details

Several algorithms are provided to automatically generate formulae for given feature groups. All tools use the accurate mass of a feature to back-calculate candidate formulae. Depending

on the algorithm and data availability, other data such as isotopic pattern and MS/MS fragments may be used to further improve formula assignment and ranking.

When DataAnalysis is used for formula generation or `calculateFeatures=TRUE` formulae are first calculated for each feature. The results are then combined for final assignment of candidate formulae for each feature group. If a formula was found in multiple features within the group, the reported scorings and mass errors are averaged and other numeric values are those from the feature in the analysis of the `"analysis"` column. The calculation of formulae on 'feature level' might result in a more thorough formula search and better removal of outliers (controlled by `featThreshold` argument). In contrast, when calculations occur on 'feature group level' (*i.e.* `calculateFeatures=FALSE`), formulae are directly assigned to each feature group (by using group averaged peak MS lists), which significantly reduces processing time is, especially with many analyses. Note that in both situations subsequent algorithms that use formula data (*e.g.* `addFormulaScoring` and reporting functions) only use formula data that was eventually assigned to feature groups. Furthermore, please note that calculation of formulae with DataAnalysis always occurs on 'feature level'.

`generateFormulas` is a generic function that will generate formulae using one of the supported algorithms. The actual functionality is provided by algorithm specific functions such as `generateFormulasDA` and `generateFormulasGenForm`. While these functions may be called directly, `generateFormulas` provides a generic interface and is therefore usually preferred.

`generateFormulasDA` uses Bruker DataAnalysis to generate chemical formulae. This method supports scoring based on overlap between measured and theoretical isotopic patterns (both MS and MS/MS data) and the presence of 'fitting' MS/MS fragments. The method will iterate through all features (or "Compounds" in DataAnalysis terms) and call `SmartFormula` (and `SmartFormula3D` if MS/MS data is available) to generate all formulae. Parameters affecting formula calculation have to be set in advance within the DataAnalysis method for each analysis (*e.g.* by `setDAMethod`). This method requires that features were obtained with `findFeaturesBruker`. Unlike other algorithms, there is no prior need to generate `MS peak lists`.

`generateFormulasGenForm` uses GenForm to generate chemical formulae. When MS/MS data is available it will be used to score candidate formulae by presence of 'fitting' fragments.

`generateFormulasSIRIUS` uses SIRIUS to generate chemical formulae. Similarity of measured and theoretical isotopic patterns will be used for scoring candidates. Note that `SIRIUS` requires availability of MS/MS data.

`formulaScorings` returns a `data.frame` with information on which scoring terms are used and what their algorithm specific name is.

Value

A `formulas` object containing all generated formulae.

Scorings

Each algorithm implements their own scoring system. Their names have been harmonized where possible. An overview is obtained with the `formulaScorings` function:

name	genform	sirius	bruker	description

combMatch	comb_match	-	-	MS and MS/MS combined match value
frag_mSigma	-	-	mSigma (SmartFormula3D)	Deviation of isotopic pattern of fragment
frag_score	-	-	Score (SmartFormula3D)	MS/MS fragment score
isoScore	MS_match	isoScore	-	How well the isotopic pattern matches
mSigma	-	-	mSigma	Deviation of the isotopic pattern
MSMSScore	MSMS_match	treeScore	-	How well MS/MS data matches
score	-	score	Score	Overall MS formula score

GenForm options

Below is a list of options (generated by running `GenForm` without commandline options) which can be set by the `extraOpts` parameter.

```
Formula calculation from MS and MS/MS data as described in
Meringer et al (2011) MATCH Commun Math Comput Chem 65: 259-290
Usage: GenForm ms=<filename> [msms=<filename>] [out=<filename>]
        [exist[=mv]] [m=<number>] [ion=-e|+e|-H|+H|+Na] [cha=<number>]
        [ppm=<number>] [msmv=ndp|nsse|nsae] [acc=<number>] [rej=<number>]
        [thms=<number>] [thmsms=<number>] [thcomb=<number>]
       [sort[=ppm|msmv|msmsmv|combmv]] [el=<elements> [oc]] [ff=<fuzzy formula>]
        [vsp[=<even|odd>]] [vsm2mv[=<value>]] [vsm2ap2[=<value>]] [hcf]
        [wm[=lin|sqrt|log]] [wi[=lin|sqrt|log]] [exp=<number>] [oei]
        [dbeexc=<number>] [ivsm2mv=<number>] [vsm2ap2=<number>]
        [oms[=<filename>]] [omsms[=<filename>]] [oclean[=<filename>]]
        [analyze [loss] [intens]] [dbe] [cm] [pc] [sc]
Explanation:
        ms      : filename of MS data (*.txt)
        msms    : filename of MS/MS data (*.txt)
        out     : output generated formulas
        exist   : allow only molecular formulas for that at least one
                  structural formula exists;overrides vsp, vsm2mv, vsm2ap2;
                  argument mv enables multiple valencies for P and S
        m       : experimental molecular mass (default: mass of MS basepeak)
        ion     : type of ion measured (default: M+H)
        ppm     : accuracy of measurement in parts per million (default: 5)
        msmv    : MS match value based on normalized dot product, normalized
                  sum of squared or absolute errors (default: nsae)
        acc     : allowed deviation for full acceptance of MS/MS peak in ppm
                  (default: 2)
        rej     : allowed deviation for total rejection of MS/MS peak in ppm
                  (default: 4)
        thms    : threshold for the MS match value
        thmsms  : threshold for the MS/MS match value
        thcomb  : threshold for the combined match value
        sort    : sort generated formulas according to mass deviation in ppm,
                  MS match value, MS/MS match value or combined match value
        el      : used chemical elements (default: CHBrClFINOPSSi)
        oc      : only organic compounds, i.e. with at least one C atom
        ff      : overwrites el and oc and uses fuzzy formula for limits of
```

```
          element multiplicities
het     : formulas must have at least one hetero atom
vsp     : valency sum parity (even for graphical formulas)
vsm2mv  : lower bound for valency sum - 2 * maximum valency
          (>=0 for graphical formulas)
vsm2ap2 : lower bound for valency sum - 2 * number of atoms + 2
          (>=0 for graphical connected formulas)
hcf     : apply Heuerding-Clerc filter
wm      : m/z weighting for MS/MS match value
wi      : intensity weighting for MS/MS match value
exp     : exponent used, when wi is set to log
oei     : allow odd electron ions for explaining MS/MS peaks
dbeexc  : excess of double bond equivalent for ions
ivsm2mv : lower bound for valency sum - 2 * maximum valency
          for fragment ions
ivsm2ap2: lower bound for valency sum - 2 * number of atoms + 2
          for fragment ions
oms     : write scaled MS peaks to output
omsms   : write weighted MS/MS peaks to output
oclean  : write explained MS/MS peaks to output
analyze : write explanations for MS/MS peaks to output
loss    : for analyzing MS/MS peaks write losses instead of fragments
intens  : write intensities of MS/MS peaks to output
dbe     : write double bond equivalents to output
cm      : write calculated ion masses to output
pc      : output match values in percent
sc      : strip calculated isotope distributions
noref   : hide the reference information
```

Note

If any errors related to `DCOM` appear it might be necessary to terminate DataAnalysis (note that DataAnalysis might still be running as a background process). The `ProcessCleaner` application installed with DataAnalayis can be used for this.

`generateFormulasGenForm` always sets the '`exist`' and '`oei`' `GenForm` commandline options.

Formula calculation with `GenForm` may produce an excessive number of candidates for high m/z values (*e.g.* above 600) and/or many elemental combinations (set by `elements`). In this scenario formula calculation may need a very long time. Timeouts are used to avoid excessive computational times by terminating long running commands (set by the `timeout` argument).

For annotations performed with `SIRIUS` it is often the fastest to keep the default `SIRBatchSize=0`. In this case, the `maxProcAmount` argument will be ignored and all `SIRIUS` output will be printed to the terminal (unless `verbose=FALSE`).

References

Meringer M, Reinker S, Zhang J, Muller A (2011). "MS/MS Data Improves Automated Determination of Molecular Formulas by Mass Spectrometry." *MATCH Commun. Math. Comput. Chem.*, **65**, 259–290.

Duhrkop K, Fleischauer M, Ludwig M, Aksenov AA, Melnik AV, Meusel M, Dorrestein PC, Rousu J, Bocker S (2019-mar). "SIRIUS 4: a rapid tool for turning tandem mass spectra into metabolite structure information." *Nature Methods*, **16**, 299–302. doi: 10.1038/s41592-01903448.

Duhrkop K, Bocker S (2015). "Fragmentation Trees Reloaded." In *Research in Computational Molecular Biology*, 65–79. ISBN 978-3-319-16706-0.

Duhrkop K, Shen H, Meusel M, Rousu J, Bocker S (2015-sep). "Searching molecular structure databases with tandem mass spectra using CSI:FingerID." *Proceedings of the National Academy of Sciences*, **112**, 12580–12585. doi: 10.1073/pnas.1509788112.

Bocker S, Letzel MC, Liptak Z, Pervukhin A (2008-nov). "SIRIUS: decomposing isotope patterns for metabolite identification." *Bioinformatics*, **25**, 218–224. doi: 10.1093/bioinformatics/btn603.

See Also

formulas-class. The GenForm manual (also known as MOLGEN-MSMS).

`formulas-class` *Formula lists class*

Description

Contains data of generated chemical formulae for given feature groups.

Usage

```
## S4 method for signature 'formulas'
formulaTable(obj, features = FALSE)

## S4 method for signature 'formulas'
algorithm(obj)

## S4 method for signature 'formulas'
analyses(obj)

## S4 method for signature 'formulas'
groupNames(obj)

## S4 method for signature 'formulas'
length(x)

## S4 method for signature 'formulas'
show(object)

## S4 method for signature 'formulas,ANY,missing,missing'
```

```
x[i, j, ..., drop = TRUE]

## S4 method for signature 'formulas,ANY,ANY'
x[[i, j]]

## S4 method for signature 'formulas'
x$name

## S4 method for signature 'formulas'
as.data.table(
  x,
  fGroups = NULL,
  average = FALSE,
  countElements = NULL,
  countFragElements = NULL,
  OM = FALSE,
  maxFormulas = NULL,
  maxFragFormulas = NULL,
  normalizeScores = "none",
  excludeNormScores = NULL
)

## S4 method for signature 'formulas'
filter(
  obj,
  minExplainedPeaks = NULL,
  elements = NULL,
  fragElements = NULL,
  lossElements = NULL,
  topMost = NULL,
  scoreLimits = NULL,
  OM = FALSE,
  negate = FALSE
)

## S4 method for signature 'formulas'
annotatedPeakList(
  obj,
  precursor,
  groupName,
  analysis = NULL,
  MSPeakLists,
  onlyAnnotated = FALSE
)

## S4 method for signature 'formulas'
plotScores(
  obj,
```

```
  precursor,
  groupName,
  analysis = NULL,
  normalizeScores = "max",
  excludeNormScores = NULL,
  useGGPlot2 = FALSE
)

## S4 method for signature 'formulas'
plotSpec(
  obj,
  precursor,
  groupName,
  analysis = NULL,
  MSPeakLists,
  title = NULL,
  useGGPlot2 = FALSE,
  xlim = NULL,
  ylim = NULL,
  ...
)

## S4 method for signature 'formulas'
plotVenn(obj, ..., labels = NULL, vennArgs = NULL)

## S4 method for signature 'formulas'
plotUpSet(
  obj,
  ...,
  labels = NULL,
  nsets = length(list(...)) + 1,
  nintersects = NA,
  upsetArgs = NULL
)

## S4 method for signature 'formulas'
consensus(
  obj,
  ...,
  absMinAbundance = NULL,
  relMinAbundance = NULL,
  uniqueFrom = NULL,
  uniqueOuter = FALSE,
  rankWeights = 1,
  labels = NULL
)
```

Arguments

`obj, x, object, formulas`
: The `formulas` object.

`features`
: If `TRUE` returns formula data for features, otherwise for feature groups.

`i, j`
: A numeric or character value which is used to select analyses/feature groups by their index or name, respectively (for the order/names see `analyses()/groupNames()`).

  For `[`: Can also be logical to perform logical selection (similar to regular vectors). If missing all analyses/feature groups are selected.

  For `[[`: should be a scalar value. If `j` is not specified, `i` selects by feature groups instead.

`...`
: For `plotSpec`: Further arguments passed to `plot`.

  Others: Any further (and unique) `formulas` objects.

`drop`
: ignored.

`name`
: The feature group name (partially matched).

`fGroups`
: The `featureGroups` object that was used to generate this object. If not `NULL` it is used to add feature group information (retention and m/z values).

`average`
: If set to `TRUE` an 'average formula' is generated for each feature group by combining all elements from all candidates and averaging their amounts. This obviously leads to non-existing formulae, however, this data may be useful to deal with multiple candidate formulae per feature group when performing elemental characterization.

`countElements, countFragElements`
: A `character` vector with elements that should be counted for each MS(/MS) formula candidate. For instance, `c("C","H")` adds columns for both carbon and hydrogen amounts of each formula. Note that the neutral formula (`neutral_formula` column) is used to count elements of non-fragmented formulae, whereas the charged formula of fragments (`frag_formula` column) is used for fragments. Set to `NULL` to not count any elements.

`OM`
: For `as.data.table`: if set to `TRUE` several columns with information relevant for organic matter (OM) characterization will be added (e.g. elemental ratios, classification). This will also make sure that `countElements` contains at least C, H, N, O, P and S.

  For `filter`: If `TRUE` then several filters are applied to exclude unlikely formula candidates present in organic matter (OM). See Source section for details.

`maxFormulas, maxFragFormulas`
: Maximum amount of unique candidate formulae (or fragment formulae) per feature group. Set to `NULL` to ignore.

`normalizeScores`
: A `character` that specifies how normalization of compound scorings occurs. Either `"none"` (no normalization), `"max"` (normalize to max value)

or "minmax" (perform min-max normalization). Note that normalization of negative scores (e.g. output by SIRIUS) is always performed as min-max. Furthermore, currently normalization for compounds takes the original min/max scoring values into account when candidates were generated. Thus, for compounds scoring, normalization is not affected when candidate results were removed after they were generated (*e.g.* by use of filter).

excludeNormScores
: A character vector specifying any compound scoring names that should *not* be normalized. Set to NULL to normalize all scorings. Note that whether any normalization occurs is set by the excludeNormScores argument.

  For compounds: By default score and individualMoNAScore are set to mimic the behavior of the MetFrag web interface.

minExplainedPeaks
: Minimum number of fragment peaks that are explained. Setting this to '1' will remove any MS only formula results. Set to NULL to ignore.

elements
: Only retain candidate formulae (neutral form) that match a given elemental restriction. The format of elements is a character string with elements that should be present where each element is followed by a valid amount or a range thereof. If no number is specified then '1' is assumed. For instance, elements="C1-10H2-20O0-2P", specifies that '1-10', '2-20', '0-2' and '1' carbon, hydrogen, oxygen and phosphorus atoms should be present, respectively. When length(elements)>1 formulas are tested to follow at least one of the given elemental restrictions. For instance, elements=c("P","S") specifies that either one phosphorus or one sulphur atom should be present. Set to NULL to ignore this filter.

fragElements, lossElements
: Specifies elemental restrictions for fragment or neutral loss formulae (charged form). Candidates are retained if at least one of the fragment formulae follow (or not follow if negate=TRUE) the given restrictions. See elements for the used format.

topMost
: Retain no more than this amount of best ranked (or worst ranked if negate=TRUE) candidates for each feature group.

scoreLimits
: Filter results by their scores. Should be a named list that contains two-sized numeric vectors with the minimum/maximum value of a score (use -Inf/Inf for no limits). The names of each element should follow the values returned by formulaScorings()$name. For instance, scoreLimits=list(isoScore=c(0.5,Inf specifies that the isotopic match score should be at least '0.5'. More details of scorings can be obtained with formulaScorings. Note that a result without a specified scoring is never removed. Set to NULL to skip this filter.

negate
: If TRUE then filters are applied in opposite manner.

precursor
: The formula of the precursor (in ionic form, *i.e.* as detected by the MS).

groupName
: The name of the feature group to which the candidate belongs.

analysis
: A character specifying the analysis for which the annotated spectrum should be plotted. If NULL then annotation results for the complete feature group will be plotted.

`MSPeakLists` The `MSPeakLists` object that was used to generate the candidate

`onlyAnnotated` Set to `TRUE` to filter out any peaks that could not be annotated.

`useGGPlot2` If `TRUE` then **ggplot2** is used for plotting, otherwise base plot used. For `plotSpec`, `ggplot2` allows nicely repelled text for annotation. However, base plot is generally faster.

`title` The title of the plot. Set to `NULL` for an automatically generated title.

`xlim, ylim` Sets the plot size limits used by `plot`. Set to `NULL` for automatic plot sizing.

`labels` A `character` with names to use for labelling. If `NULL` labels are automatically generated.

`vennArgs` A `list` with further arguments passed to **VennDiagram** plotting functions. Set to `NULL` to ignore.

`nsets, nintersects`
See `upset`.

`upsetArgs` A list with any further arguments to be passed to `upset`. Set to `NULL` to ignore.

`absMinAbundance, relMinAbundance`
Minimum absolute or relative ('`0-1`') abundance across objects for a result to be kept. For instance, `relMinAbundance=0.5` means that a result should be present in at least half of the number of compared objects. Set to '`NULL`' to ignore and keep all results. Limits cannot be set when `uniqueFrom` is not `NULL`.

`uniqueFrom` Set this argument to only retain formulas that are unique within one or more of the objects for which the consensus is made. Selection is done by setting the value of `uniqueFrom` to a `logical` (values are re-cycled), `numeric` (select by index) or a `character` (as obtained with `algorithm(obj)`). For `logical` and `numeric` values the order corresponds to the order of the objects given for the consensus. Set to `NULL` to ignore.

`uniqueOuter` If `uniqueFrom` is not `NULL` and if `uniqueOuter=TRUE`: only retain data that are also unique between objects specified in `uniqueFrom`.

`rankWeights` A numeric vector with weights of to calulcate the mean ranking score for each candidate. The value will be re-cycled if necessary, hence, the default value of '`1`' means equal weights for all considered objects.

Details

`formulas` objects are obtained from formula generators.

Value

`formulaTable` returns a `list` containing for each feature group (or feature if `features=TRUE`) a `data.table` with an overview of all generated formulae and other data such as candidate scoring and MS/MS fragments.

`as.data.table` returns a `data.table`.

`filter` returns a filtered `formulas` object.

plotSpec will return a ggplot object if useGGPlot2 is TRUE.

plotVenn (invisibly) returns a list with the following fields:

- gList the gList object that was returned by the utilized **VennDiagram** plotting function.
- areas The total area for each plotted group.
- intersectionCounts The number of intersections between groups.

The order for the areas and intersectionCounts fields is the same as the parameter order from the used plotting function (see *e.g.* draw.pairwise.venn and draw.triple.venn).

consensus returns a formulas object that is produced by merging results from multiple formulas objects.

Methods (by generic)

- formulaTable: Accessor method to obtain generated formulae.
- algorithm: Accessor method for the algorithm (a character string) used to generate formulae.
- analyses: returns a character vector with the names of the analyses for which data is present in this object.
- groupNames: returns a character vector with the names of the feature groups for which data is present in this object.
- length: Obtain total number of formulae entries.
- show: Show summary information for this object.
- [: Subset on feature groups.
- [[: Extract a formula table. If both arguments (i and j) are specified, the feature specific formula table belonging to the analysis (i)/feature group (j) is returned. Otherwise the formula table for the feature group specified by j is returned.
- $: Extract a formula table for a feature group.
- as.data.table: Generates a table with all candidate formulae for each feature group and other information such as element counts.
- filter: Performs rule based filtering on formula results.
- annotatedPeakList: Returns an MS/MS peak list annotated with data from a given candidate formula.
- plotScores: Plots a barplot with scoring of a candidate compound.
- plotSpec: Plots an annotated spectrum for a given candidate formula of a feature or feature group.
- plotVenn: plots a Venn diagram (using **VennDiagram**) outlining unique and shared formula candidates of up to five different formulas objects.
- plotUpSet: plots an UpSet diagram (using the upset function) outlining unique and shared formula candidates between different formula objects.

- `consensus`: Generates a consensus of results from multiple objects. In order to rank the consensus candidates, first each of the candidates are scored based on their original ranking (the scores are normalized and the highest ranked candidate gets value '`1`'). The (weighted) mean is then calculated for all scorings of each candidate to derive the final ranking (if an object lacks the candidate its score will be '`0`'). The original rankings for each object is stored in the `rank` columns.

Slots

`formulas,featureFormulas` Lists of all generated formulae. Use the `formulaTable` method for access.

`scoreRanges` The original min/max values of all scorings when candidate results were generated. This is used for normalization.

Source

Calculation of the aromaticity index (AI) and related double bond equivalents (DBE_AI) is performed as described in Koch 2015. Formula classification is performed by the rules described in Abdulla 2013. Filtering of OM related molecules is performed as described in Koch 2006 and Kujawinski 2006. (see references).

Subscripting of formulae for plots generated by `plotSpec` is based on the `chemistry2expression` function from the ReSOLUTION package.

S4 class hierarchy

- `workflowStep`
  - `formulas`

Note

`filter` does not modify any formula results for features (if present).

References

Koch BP, Dittmar T (2015-dec). "From mass to structure: an aromaticity index for high-resolution mass data of natural organic matter." *Rapid Communications in Mass Spectrometry*, **30**, 250–250. doi: 10.1002/rcm.7433.

Abdulla HA, Sleighter RL, Hatcher PG (2013-apr). "Two Dimensional Correlation Analysis of Fourier Transform Ion Cyclotron Resonance Mass Spectra of Dissolved Organic Matter: A New Graphical Analysis of Trends." *Analytical Chemistry*, **85**, 3895–3902. doi: 10.1021/ac303221j.

Koch BP, Dittmar T (2006-feb). "From mass to structure: an aromaticity index for high-resolution mass data of natural organic matter." *Rapid Communications in Mass Spectrometry*, **20**, 926–932. doi: 10.1002/rcm.2386.

Kujawinski EB, Behn MD (2006-jul). "Automated Analysis of Electrospray Ionization

Fourier Transform Ion Cyclotron Resonance Mass Spectra of Natural Organic Matter." *Analytical Chemistry*, **78**, 4363–4373. doi: 10.1021/ac0600306.

Conway JR, Lex A, Gehlenborg N (2017). "UpSetR: an R package for the visualization of intersecting sets and their properties." *Bioinformatics*, **33**, 2938–2940. doi: 10.1093/bioinformatics/btx364.

Lex A, Gehlenborg N, Strobelt H, Vuillemot R, Pfister H (2014-dec). "UpSet: Visualization of Intersecting Sets." *IEEE Transactions on Visualization and Computer Graphics*, **20**, 1983–1992. doi: 10.1109/tvcg.2014.2346248.

`generics` *Miscellaneous generics*

Description

Various (S4) generic functions providing a common interface for common tasks such as plotting and filtering data. The actual functionality and function arguments are often specific for the implemented methods, for this reason, please refer to the linked method documentation for each generic.

Usage

```
algorithm(obj)

analysisInfo(obj)

analyses(obj)

annotatedPeakList(obj, ...)

clusterProperties(obj)

clusters(obj)

cutClusters(obj)

consensus(obj, ...)

featureTable(obj)

filter(obj, ...)

getFeatures(obj)

getMCS(obj, ...)

groupNames(obj)
```

```
plotChord(obj, addSelfLinks = FALSE, addRetMzPlots = TRUE, ...)

plotEIC(obj, ...)

plotInt(obj, ...)

plotScores(obj, ...)

plotSilhouettes(obj, kSeq, ...)

plotSpec(obj, ...)

plotStructure(obj, ...)

plotVenn(obj, ...)

plotUpSet(obj, ...)

replicateGroups(obj)

treeCut(obj, k = NULL, h = NULL, ...)

treeCutDynamic(
  obj,
  maxTreeHeight = 1,
  deepSplit = TRUE,
  minModuleSize = 1,
  ...
)
```

Arguments

obj	The object the generic should be applied to.
...	Any further method specific arguments. See method documentation for details.
addSelfLinks	If TRUE then 'self-links' are added which represent non-shared data.
addRetMzPlots	Set to TRUE to enable *m/z vs* retention time scatter plots.
kSeq	An integer vector containing the sequence that should be used for average silhouette width calculation.
k, h	Desired numbers of clusters. See cutree.
maxTreeHeight, deepSplit, minModuleSize	Arguments used by cutreeDynamicTree.

Details

algorithm returns the algorithm that was used to generate the object.

- Methods are defined for: compounds; formulas; optimizationResult; workflowStep.

analysisInfo returns the analysis information from an object.

- Methods are defined for: featureGroups; features.

analyses returns a character vector with the analyses for which data is present in this object.

- Methods are defined for: featureGroups; features; formulas; MSPeakLists.

annotatedPeakList returns an annotated MS peak list.

- Methods are defined for: compounds; formulas.

clusterProperties Obtain a list with properties of the generated cluster(s).

- Methods are defined for: componentsIntClust; compoundsCluster.

clusters Obtain clustering object(s).

- Methods are defined for: componentsIntClust; compoundsCluster.

cutClusters Returns assigned cluster indices of a cut cluster.

- Methods are defined for: componentsIntClust; compoundsCluster.

consensus combines and merges data from various algorithms to generate a consensus.

- Methods are defined for: components; compounds; featureGroupsComparison; formulas.

featureTable returns feature information.

- Methods are defined for: featureGroups; features.

filter provides various functionality to do post-filtering of data.

- Methods are defined for: components; compounds; featureGroups; features; featuresXCMS; featuresXCMS3; formulas; MSPeakLists.

getFeatures returns the object's features object.

- Methods are defined for: featureGroups.

getMCS Calculcates the maximum common substructure.

- Methods are defined for: compounds; compoundsCluster.

groupNames returns a character vector with the names of the feature groups for which data is present in this object.

- Methods are defined for: components; compounds; compoundsCluster; featureGroups; formulas; MSPeakLists.

`plotChord` plots a Chord diagram to assess overlapping data.

- Methods are defined for: `featureGroups`; `featureGroupsComparison`.

`plotEIC` plots extracted ion chromatogram(s).

- Methods are defined for: `components`; `featureGroups`.

`plotInt` plots the intensity of all contained features.

- Methods are defined for: `componentsIntClust`; `featureGroups`.

`plotScores` plots candidate scorings.

- Methods are defined for: `compounds`; `formulas`.

`plotSilhouettes` plots silhouette widths to evaluate the desired cluster size.

- Methods are defined for: `componentsIntClust`; `compoundsCluster`.

`plotSpec` plots a (annotated) spectrum.

- Methods are defined for: `components`; `compounds`; `formulas`; `MSPeakLists`.

`plotStructure` plots a chemical structure.

- Methods are defined for: `compounds`; `compoundsCluster`.

`plotVenn` plots a Venn diagram to assess unique and overlapping data.

- Methods are defined for: `compounds`; `featureGroups`; `featureGroupsComparison`; `formulas`.

`plotUpSet` plots an UpSet diagram to assess unique and overlapping data.

- Methods are defined for: `compounds`; `featureGroups`; `featureGroupsComparison`; `formulas`.

`replicateGroups` returns a `character` vector with the analyses for which data is present in this object.

- Methods are defined for: `featureGroups`; `features`.

`treeCut` Manually cut a cluster.

- Methods are defined for: `componentsIntClust`; `compoundsCluster`.

`treeCutDynamic` Automatically cut a cluster.

- Methods are defined for: `componentsIntClust`; `compoundsCluster`.

Other generics

Below are methods that are defined for existing generics (*e.g.* defined in base). Please see method specific documentation for more details.

[Subsets data within an object.

- Methods are defined for: components,ANY,ANY,missing; compounds,ANY,missing,missing; compoundsCluster,ANY,missing,missing; featureGroups,ANY,ANY,missing; featureGroupsComparison,ANY, features,ANY,missing,missing; featuresXCMS,ANY,missing,missing; featuresXCMS3,ANY,missing,missing formulas,ANY,missing,missing; MSPeakLists,ANY,ANY,missing.

[[Extract data from an object.

- Methods are defined for: components,ANY,ANY; compounds,ANY,missing; featureGroups,ANY,ANY; featureGroupsComparison,ANY,missing; features,ANY,missing; formulas,ANY,ANY; MSPeakLists,ANY,ANY.

$ Extract data from an object.

- Methods are defined for: components; compounds; featureGroups; featureGroupsComparison; features; formulas; MSPeakLists.

as.data.table Converts an object to a table (data.table).

- Methods are defined for: components; compounds; featureGroups; features; formulas; MSPeakLists; workflowStep.

as.data.frame Converts an object to a table (data.frame).

- Methods are defined for: workflowStep.

length Returns the length of an object.

- Methods are defined for: components; compounds; compoundsCluster; featureGroups; featureGroupsComparison; features; formulas; MSPeakLists; optimizationResult.

lengths Returns the lengths of elements within this object.

- Methods are defined for: compoundsCluster; optimizationResult.

names Return names for this object.

- Methods are defined for: components; featureGroups; featureGroupsComparison.

plot Generates a plot for an object.

- Methods are defined for: componentsIntClust,ANY; compoundsCluster,ANY; featureGroups,ANY; featureGroupsComparison,ANY; optimizationResult,ANY.

show Prints information about this object.

- Methods are defined for: adduct; components; compounds; compoundsCluster; featureGroups; features; formulas; MSPeakLists; optimizationResult; workflowStep.

`getXCMSSet` *Conversion to xcmsSet objects*

Description

Converts a `features` or `featureGroups` object to an `xcmsSet` object.

Usage

```
getXCMSSet(obj, verbose = TRUE, ...)

## S4 method for signature 'features'
getXCMSSet(obj, verbose, exportedData)

## S4 method for signature 'featuresOpenMS'
getXCMSSet(obj, verbose = TRUE, ...)

## S4 method for signature 'featuresXCMS'
getXCMSSet(obj, verbose = TRUE, ...)

## S4 method for signature 'featureGroups'
getXCMSSet(obj, verbose, exportedData)

## S4 method for signature 'featureGroupsXCMS'
getXCMSSet(obj, verbose, exportedData)
```

Arguments

`obj`	The object that should be converted.
`verbose`	If `FALSE` then no text output is shown.
`exportedData, ...`	Set to `TRUE` if analyses were exported as `mzXML` or `mzML` files (ignored for `featuresOpenMS` and `featuresXCMS` methods).

`GUI-utils` *Interactive GUI utilities*

Description

Interactive utilities using **shiny** to provide a graphical user interface (GUI).

Usage

```
## S4 method for signature 'featureGroups'
checkChromatograms(fGroups, mzWindow = 0.005, enabledFGroups = NULL)

newProject(destPath = NULL)

## S4 method for signature 'featureGroups,MSPeakLists,compounds'
compoundViewer(fGroups, MSPeakLists, compounds)
```

Arguments

fGroups	A featureGroups object.
mzWindow	Default *m/z* window to be used for creating extracted ion chromatograms (EICs).
enabledFGroups	A logical vector that states for each feature group whether it should be kept (TRUE) or not (FALSE). The order is the same as the fGroups parameter. If NULL then all feature groups are considered to be kept.
destPath	Set destination path value to this value (useful for debugging). Set to NULL for a default value.
MSPeakLists	A MSPeakLists object.
compounds	A compounds object.

Details

checkChromatograms is used to review chromatographic information for feature groups. This is especially useful to get a visual impression of the quality of detected features. In addition, this function may be used to remove unwanted (*e.g.* outlier) features. Better performance is often obtained when an external browser is used to use this Shiny application. Furthermore, when a large featureGroups object is used it is recommended to limit the number of analyses/feature groups by subsetting the object.

The newProject function is used to quickly generate a processing R script. This tool allows the user to quickly select the targeted analyses, workflow steps and configuring some of their common parameters. This function requires to be run within a RStudio session. The resulting script is either added to the current open file or to a new file. The analysis information will be written to a '.csv' file so that it can easily be modified afterwards.

The compoundViewer method is used to view compound identification results. It will display available candidate information such as scorings and identifiers, MS/MS spectra with explained peaks and chemical structures.

Value

checkChromatograms returns a logical vector for all feature groups that were selected to be kept (TRUE) or not (FALSE). This result can be passed to the enabledFGroups parameter for subsequent calls to checkChromatograms in order to restore the keep/not keep state from a previous call. To actually remove unwanted feature groups the object should be subset by the subsetting ([) operator to which the return value should be passed as the second parameter.

`MSPeakLists-class` *Class containing MS Peak Lists*

Description

Contains all MS (and MS/MS where available) peak lists for a `featureGroups` object.

Usage

```
## S4 method for signature 'MSPeakLists'
peakLists(obj)

## S4 method for signature 'MSPeakLists'
averagedPeakLists(obj)

## S4 method for signature 'MSPeakLists'
analyses(obj)

## S4 method for signature 'MSPeakLists'
groupNames(obj)

## S4 method for signature 'MSPeakLists'
length(x)

## S4 method for signature 'MSPeakLists'
show(object)

## S4 method for signature 'MSPeakLists,ANY,ANY,missing'
x[i, j, ..., reAverage = TRUE, drop = TRUE]

## S4 method for signature 'MSPeakLists,ANY,ANY'
x[[i, j]]

## S4 method for signature 'MSPeakLists'
x$name

## S4 method for signature 'MSPeakLists'
as.data.table(x, fGroups = NULL, averaged = TRUE)

## S4 method for signature 'MSPeakLists'
filter(
  obj,
  absMSIntThr = NULL,
  absMSMSIntThr = NULL,
  relMSIntThr = NULL,
  relMSMSIntThr = NULL,
  topMSPeaks = NULL,
```

```
  topMSMSPeaks = NULL,
  isolatePrec = NULL,
  deIsotopeMS = FALSE,
  deIsotopeMSMS = FALSE,
  withMSMS = FALSE,
  retainPrecursorMSMS = TRUE,
  negate = FALSE
)

## S4 method for signature 'MSPeakLists'
plotSpec(
  obj,
  groupName,
  analysis = NULL,
  MSLevel = 1,
  title = NULL,
  useGGPlot2 = FALSE,
  xlim = NULL,
  ylim = NULL,
  ...
)

getDefIsolatePrecParams(...)
```

Arguments

`obj, x, object`	The `MSPeakLists` object to access.
`i, j`	A numeric or character value which is used to select analyses/feature groups by their index or name, respectively (for the order/names see `analyses()/groupNames()`). For `[`: Can also be logical to perform logical selection (similar to regular vectors). If missing all analyses/feature groups are selected. For `[[`: should be a scalar value. If `j` is not specified, `i` selects by feature groups instead.
`...`	Further arguments passed to `plot`.
`reAverage`	Set to `TRUE` to regenerate averaged MS peak lists after subsetting analyses.
`drop`	ignored.
`name`	The feature group name (partially matched).
`fGroups`	The `featureGroups` object that was used to generate this object. If not `NULL` it is used to add feature group information (retention and m/z values).
`averaged`	If `TRUE` then feature group averaged peak list data is used.
`absMSIntThr, absMSMSIntThr, relMSIntThr, relMSMSIntThr`	Absolute/relative intensity threshold for MS or MS/MS peak lists. `NULL` for none.

`topMSPeaks, topMSMSPeaks`
: Only consider this amount of MS or MS/MS peaks with highest intensity. `NULL` to consider all.

`isolatePrec`
: If not `NULL` then value should be a `list` with parameters used for isolating the precursor and its isotopes in MS peak lists (see Isolating precursor data). Alternatively, `TRUE` to apply the filter with default settings (as given with `getDefIsolatePrecParams`).

`deIsotopeMS, deIsotopeMSMS`
: Remove any isotopic peaks in MS or MS/MS peak lists. This may improve data processing steps which do not assume the presence of isotopic peaks (e.g. MetFrag for MS/MS). Note that `getMzRPeakLists` does not (yet) support flagging of isotopes.

`withMSMS`
: If set to `TRUE` then only results will be retained for which MS/MS data is available. if `negate=TRUE` then only results *without* MS/MS data will be retained.

`retainPrecursorMSMS`
: If `TRUE` then precursor peaks will never be filtered out from MS/MS peak lists (note that precursors are never removed from MS peak lists). The `negate` argument does not affect this setting.

`negate`
: If `TRUE` then filters are applied in opposite manner.

`groupName`
: The name of the feature group for which a plot should be made.

`analysis`
: The name of the analysis for which a plot should be made. If `NULL` then data from the feature group averaged peak list is used.

`MSLevel`
: The MS level: '1' for regular MS, '2' for MSMS.

`title`
: The title of the plot. If `NULL` a title will be automatically made.

`useGGPlot2`
: If `TRUE` then **ggplot2** is used for plotting, otherwise base plot used. For `plotSpec`, `ggplot2` allows nicely repelled text for annotation. However, base plot is generally faster.

`xlim, ylim`
: Sets the plot size limits used by `plot`. Set to `NULL` for automatic plot sizing.

Details

Objects for this class are returned by MS peak lists generators.

The `getDefIsolatePrecParams` is used to create a parameter list for isolating the precursor and its isotopes (see Isolating precursor data).

Value

`peakLists` returns a nested list containing MS (and MS/MS where available) peak lists per feature group and per analysis. The format is: `[[analysis]][[featureGroupName]][[MSType]][[PeakLists]]` where `MSType` is either `"MS"` or `"MSMS"` and `PeakLists` a `data.table` containing all m/z values (`mz` column) and their intensities (`intensity` column). In addition, the peak list tables may contain a `cmp` column which contains an unique alphabetical identifier to which isotopic cluster (or "compound") a mass belongs (only supported by MS peak lists generated by Bruker tools at the moment).

`averagedPeakLists` returns a nested list of feature group averaged peak lists in a similar format as `peakLists`.

`plotSpec` will return a `ggplot object` if `useGGPlot2` is `TRUE`.

Methods (by generic)

- `peakLists`: Accessor method to obtain the MS peak lists.
- `averagedPeakLists`: Accessor method to obtain the feature group averaged MS peak lists.
- `analyses`: returns a `character` vector with the names of the analyses for which data is present in this object.
- `groupNames`: returns a `character` vector with the names of the feature groups for which data is present in this object.
- `length`: Obtain total number of m/z values.
- `show`: Shows summary information for this object.
- `[`: Subset on analyses/feature groups.
- `[[`: Extract a list with MS and MS/MS (if available) peak lists. If the second argument (`j`) is not specified the averaged peak lists for the group specified by the first argument (`i`) will be returned.
- `$`: Extract group averaged MS peaklists for a feature group.
- `as.data.table`: Returns all MS peak list data in a table.
- `filter`: provides post filtering of generated MS peak lists, which may further enhance quality of subsequent workflow steps (*e.g.* formulae calculation and compounds identification) and/or speed up these processes.
- `plotSpec`: Plots a spectrum using MS or MS/MS peak lists for a given feature group.

Slots

`peakLists` Contains a list of all MS (and MS/MS) peak lists. Use the `peakLists` method for access.

`metadata` Metadata for all spectra used to generate peak lists. Follows the format of the `peakLists` slot.

`averagedPeakLists` A `list` with averaged MS (and MS/MS) peak lists for each feature group.

`avgPeakListArgs` A `list` with arguments used to generate feature group averaged MS(/MS) peak lists.

`origFGNames` A `character` with the original input feature group names.

Isolating precursor data

Formula calculation typically relies on evaluating the measured isotopic pattern from the precursor to score candidates. Some algorithms (currently only `GenForm`) penalize candidates if mass peaks are present in MS1 spectra that do not contribute to the isotopic pattern. Since these spectra are typically very 'noisy' due to background and co-eluting

ions, an additional filtering step may be recommended prior to formula calculation. During this precursor isolation step all mass peaks are removed that are (1) not the precursor and (2) not likely to be an isotopologue of the precursor. To determine potential isotopic peaks the following parameters are used:

- `maxIsotopes` The maximum number of isotopes to consider. For instance, a value of '`5`' means that `M+0` (*i.e.* the monoisotopic peak) till `M+5` is considered. All mass peaks outside this range are removed.
- `mzDefectRange` A two-sized `vector` specifying the minimum (can be negative) and maximum *m/z* defect deviation compared to the precursor *m/z* defect. When chlorinated, brominated or other compounds with strong *m/z* defect in their isotopologues are to be considered a higher range may be desired. On the other hand, for natural compounds this range may be tightened. Note that the search range is propegated with increasing distance from the precursor, *e.g.* the search range is doubled for `M+2`, tripled for `M+3` etc.
- `intRange` A two-sized `vector` specifying the minimum and maximum relative intensity range compared to the precursor. For instance, `c(0.001,2)` removes all peaks that have an intensity below 0.1% or above 200% of that of the precursor.
- `z` The `z` value (*i.e.* absolute charge) to be considerd. For instance, a value of `2` would look for `M+0.5`, `M+1` etc. Note that the `mzDefectRange` is adjusted accordingly (*e.g.* halved if `z=2`).
- `maxGap` The maximum number of missing adjacent isotopic peaks ('gaps'). If the (rounded) *m/z* difference to the previous peak exceeds this value then this and all next peaks will be removed. Similar to `z`, the maximum gap is automatically adjusted for `charge`.

These parameters should be in a `list` that is passed to the `isolatePrec` argument to `filter`. The default values can be obtained with the `getDefIsolatePrecParams` function:

```
maxIsotopes=5; mzDefectRange=c(-0.01,0.01); intRange=c(0.001,2); z=1; maxGap=2
```

S4 class hierarchy

- `workflowStep`
  - `MSPeakLists`

`MSPeakLists-generation`

Generation of MS Peak Lists

Description

Functionality to generate MS peak lists.

Usage

```
## S4 method for signature 'featureGroups'
generateMSPeakLists(fGroups, algorithm, ...)

generateMSPeakListsDA(
  fGroups,
  bgsubtr = TRUE,
  maxMSRtWindow = 5,
  minMSIntensity = 500,
  minMSMSIntensity = 500,
  clear = TRUE,
  close = TRUE,
  save = close,
  MSMSType = "MSMS",
  avgFGroupParams = getDefAvgPListParams()
)

generateMSPeakListsDAFMF(
  fGroups,
  minMSIntensity = 500,
  minMSMSIntensity = 500,
  close = TRUE,
  save = close,
  avgFGroupParams = getDefAvgPListParams()
)

generateMSPeakListsMzR(
  fGroups,
  maxMSRtWindow = 5,
  precursorMzWindow = 4,
  topMost = NULL,
  avgFeatParams = getDefAvgPListParams(),
  avgFGroupParams = getDefAvgPListParams()
)

getDefAvgPListParams(...)
```

Arguments

fGroups	The featureGroups object from which MS peak lists should be extracted.
algorithm	A character string describing the algorithm that should be used: "bruker", "brukerfmf", "mzr"
...	For generateMSPeakLists: Any parameters to be passed to the selected MS peak lists generation algorithm. For getDefAvgPListParams: Optional named arguments that override defaults.
bgsubtr	If TRUE background will be subtracted using the 'spectral' algorithm.

`maxMSRtWindow` Maximum chromatographic peak window used for spectrum averaging (in seconds, +/- retention time). If `NULL` all spectra from a feature will be taken into account. Lower to decrease processing time.

`minMSIntensity, minMSMSIntensity`
Minimum intensity for peak lists obtained with DataAnalysis. Highly recommended to set '`>0`' as DA tends to report many very low intensity peaks.

`clear` Remove any existing chromatogram traces/mass spectra prior to making new ones.

`close, save` If `TRUE` then Bruker files are closed and saved after processing with DataAnalysis, respectively. Setting `close=TRUE` prevents that many analyses might be opened simultaneously in DataAnalysis, which otherwise may use excessive memory or become slow. By default `save` is `TRUE` when `close` is `TRUE`, which is likely what you want as otherwise any processed data is lost.

`MSMSType` The type of MS/MS experiment performed: `"MSMS"` for MRM/AutoMSMS or `"BBCID"` for broadband CID.

`precursorMzWindow`
The *m/z* window (in Da) to find MS/MS spectra of a precursor. This is typically used for Data-Dependent like MS/MS data and should correspond to the isolation *m/z* window (*i.e.* +/- the precursor *m/z*) that was used to collect the data. For Data-Independent MS/MS experiments, where precursor ions are not isolated prior to fragmentation (*e.g.* bbCID, MSe, all-ion, ...) the value should be `NULL`.

`topMost` Only extract MS peak lists from a maximum of `topMost` analyses with highest intensity. If `NULL` all analyses will be used.

`avgFeatParams, avgFGroupParams`
A `list` with parameters used for averaging of peak lists for individual features and feature groups, respectively (see below).

Details

Formula calculation and identification tools rely on mass spectra that belong to features of interest. For processing, MS (and MS/MS) spectra are typically reduced to a table with a column containing measured *m/z* values and a column containing their intensities. These 'MS peak lists' can then be used for formula generation and compound generation.

MS and MS/MS peak lists are first generated for all features (or a subset, if the `topMost` argument is set). During this step multiple spectra over the feature elution profile are averaged. Subsequently, peak lists will be generated for each feature group by averaging peak lists of the features within the group. Functionality that uses peak lists will either use data from individual features or from group averaged peak lists. For instance, the former may be used by formulae calculation, while compound identification and plotting functionality typically uses group averaged peak lists.

Several functions exist to automatically extract MS peak lists for feature groups.

`generateMSPeakLists` is a generic function that will generate MS peak lists using one of the supported algorithms. The actual functionality is provided by algorithm specific functions

such as generateMSPeakListsMzR and generateMSPeakListsDA. While these functions may be called directly, generateMSPeakLists provides a generic interface and is therefore usually preferred.

generateMSPeakListsDA uses Bruker DataAnalysis to generate MS peak lists. Naturally, this only works with analyses in the Bruker data format ('.d'). This function leverages DataAnalysis functionality to support averaging of spectra, background subtraction and identification of isotopes. In order to obtain mass spectra TICs will be added of the MS and relevant MS/MS signals.

generateMSPeakListsDAFMF is similar to generateMSPeakListsDA, but uses compounds that were generated by the Find Molecular Features (FMF) algorithm to extract MS peak lists. This is generally much faster than generateMSPeakListsDA, however, it only works when features were obtained using the findFeaturesBruker function. Since all MS spectra are generated in advance by Bruker DataAnalysis, no further parameters exist to customize its operation.

generateMSPeakListsMzR uses the **mzR** package to extract MS peak lists. For this analyses should be either in '.mzXML' or '.mzML' format. This function averages multiple spectra over a chromatgraphic peak to improve accuracy.

The getDefAvgPListParams is used to create a parameter list for peak list averaging (discussed below).

Value

A MSPeakLists object that can be used for formulae calculation and compound identification.

Peak list averaging parameters

The parameters set used for averaging peak lists are set by the avgFeatParams and avgFGroupParams arguments. This should be a named list with the following values:

- clusterMzWindow m/z window (in Da) used for clustering m/z values when spectra are averaged. For method="hclust" this corresponds to the cluster height, while for method="distance" this value is used to find nearby masses (+/- window). Too small windows will prevent clustering m/z values (thus erroneously treating equal masses along spectra as different), whereas too big windows may cluster unrelated m/z values from different or even the same spectrum together.
- topMost Only retain this maximum number of MS peaks when generating averaged spectra. Lowering this number may exclude more irrelevant (noisy) MS peaks and decrease processing time, whereas higher values may avoid excluding lower intense MS peaks that may still be of interest.
- minIntensityPre MS peaks with intensities below this value will be removed (applied prior to selection by topMost) before averaging.
- minIntensityPost MS peaks with intensities below this value will be removed after averaging.
- avgFun Function that is used to calculate average m/z values.

- `method` Method used for producing averaged MS spectra. Valid values are `"hclust"`, used for hierarchical clustering (using the **fastcluster** package), and `"distance"`, to use the between peak distance. The latter method may reduces processing time and memory requirements, at the potential cost of reduced accuracy.
- `pruneMissingPrecursorMS` For MS data only: if `TRUE` then peak lists without a precursor peak are removed. Note that even when this is set to `FALSE`, functionality that relies on MS (not MS/MS) peak lists (*e.g.* formulae calulcation) will still skip calculation if a precursor is not found.
- `retainPrecursorMSMS` For MS/MS data only: if `TRUE` then always retain the precursor mass peak even if is not amongst the `topMost` peaks. Note that MS precursor mass peaks are always kept. Furthermore, note that precursor peaks in both MS and MS/MS data may still be removed by intensity thresholds (this is unlike the `filter` method function).

Note that when Bruker algorithms are used these parameters only control generation of feature groups averaged peak lists: how peak lists for features are generated is controlled by DataAnalysis.

The `getDefAvgPListParams` function can be used to generate a default parameter list. The current defaults are:

`clusterMzWindow=0.005`; `topMost=50`; `minIntensityPre=500`; `minIntensityPost=500`; `avgFun=mean`; `method="hclust"`; `pruneMissingPrecursorMS=TRUE`; `retainPrecursorMSMS=TRUE`

Source

Averaging of mass spectra algorithms used by are based on the msProcess R package (now archived on CRAN).

Note

`generateMSPeakListsDA` requires that the '`Component`' column is active (Method–¿Parameters–¿Layouts–¿Mass List Layout) in order to add isotopologue information.

If any errors related to `DCOM` appear it might be necessary to terminate DataAnalysis (note that DataAnalysis might still be running as a background process). The `ProcessCleaner` application installed with DataAnalayis can be used for this.

References

A cross-platform toolkit for mass spectrometry and proteomics Chambers, Matthew C. and Maclean, Brendan and Burke, Robert and Amodei, Dario and Ruderman, Daniel L. and Neumann, Steffen and Gatto, Laurent and Fischer, Bernd and Pratt, Brian and Egertson, Jarrett and Hoff, Katherine and Kessner, Darren and Tasman, Natalie and Shulman, Nicholas and Frewen, Barbara and Baker, Tahmina A. and Brusniak, Mi-Youn and Paulse, Christopher and Creasy, David and Flashner, Lisa and Kani, Kian and Moulding, Chris and Seymour, Sean L. and Nuwaysir, Lydia M. and Lefebvre, Brent and Kuhlmann, Frank and Roark, Joe and Rainer, Paape and Detlev, Suckau and Hemenway, Tina and Huhmer, Andreas and Langridge, James and Connolly, Brian and Chadick, Trey and Holly, Krisztina and Eckels, Josh and Deutsch, Eric W. and Moritz, Robert L. and Katz, Jonathan E. and

Agus, David B. and MacCoss, Michael and Tabb, David L. and Mallick, Parag Nat Biotechnol. 2012 NOct;30(10):918-920.

Mol Cell Proteomics. 2010 Aug 17. mzML - a Community Standard for Mass Spectrometry Data. Martens L, Chambers M, Sturm M, Kessner D, Levander F, Shofstahl J, Tang WH, Rompp A, Neumann S, Pizarro AD, Montecchi-Palazzi L, Tasman N, Coleman M, Reisinger F, Souda P, Hermjakob H, Binz PA, Deutsch EW.

Nat Biotechnol. 2004 Nov;22(11):1459-66. A common open representation of mass spectrometry data and its application to proteomics research. Pedrioli PG, Eng JK, Hubley R, Vogelzang M, Deutsch EW, Raught B, Pratt B, Nilsson E, Angeletti RH, Apweiler R, Cheung K, Costello CE, Hermjakob H, Huang S, Julian RK, Kapp E, McComb ME, Oliver SG, Omenn G, Paton NW, Simpson R, Smith R, Taylor CF, Zhu W, Aebersold R.

Mol Syst Biol. 2005;1:2005.0017. Epub 2005 Aug 2. A uniform proteomics MS/MS analysis platform utilizing open XML file formats. Keller A, Eng J, Zhang N, Li XJ, Aebersold R.

Bioinformatics. 2008 Nov 1;24(21):2534-6. Epub 2008 Jul 7. ProteoWizard: open source software for rapid proteomics tools development. Kessner D, Chambers M, Burke R, Agus D, Mallick P.

Daniel Müllner (2013). fastcluster: Fast Hierarchical, Agglomerative Clustering Routines for R and Python. Journal of Statistical Software, 53(9), 1-18. URL http://www.jstatsoft.org/v53/i09/.

See Also

MSPeakLists-class

`optimizationResult-class`

Class containing optimization results.

Description

Objects from this class contain optimization results resulting from design of experiment (DoE).

Usage

```
## S4 method for signature 'optimizationResult'
algorithm(obj)

## S4 method for signature 'optimizationResult'
length(x)

## S4 method for signature 'optimizationResult'
lengths(x, use.names = FALSE)

## S4 method for signature 'optimizationResult'
show(object)

## S4 method for signature 'optimizationResult,ANY'
plot(
  x,
  paramSet,
  DoEIteration,
  paramsToPlot = NULL,
  maxCols = NULL,
  type = "contour",
  image = TRUE,
  contours = "colors",
  ...
)

## S4 method for signature 'optimizationResult'
optimizedParameters(object, paramSet = NULL, DoEIteration = NULL)

## S4 method for signature 'optimizationResult'
optimizedObject(object, paramSet = NULL)

## S4 method for signature 'optimizationResult'
scores(object, paramSet = NULL, DoEIteration = NULL)

## S4 method for signature 'optimizationResult'
experimentInfo(object, paramSet, DoEIteration)
```

Arguments

`obj, x, object`	An `optimizationResult` object.
`use.names`	Ignored.
`paramSet`	Numeric index of the parameter set (*i.e.* the first parameter set gets index '1'). For some methods optional: if `NULL` the best will be selected.
`DoEIteration`	Numeric index specifying the DoE iteration within the specified `paramSet`. For some methods optional: if `NULL` the best will be selected.
`paramsToPlot`	Which parameters relations should be plot. If `NULL` all will be plot. Alternatively, a `list` containing one or more `character` vectors specifying each

	two parameters that should be plotted. Finally, if only one pair should be plotted, can be a `character` vector specifying both parameters.
`maxCols`	Multiple parameter pairs are plotted in a grid. The maximum number of columns can be set with this argument. Set to `NULL` for no limit.
`type`	The type of plots to be generated: `"contour"`, `"image"` or `"persp"`. The equally named functions will be called for plotting.
`image`	Passed to `contour` (if `type="contour"`).
`contours`	Passed to `persp` (if `type="persp"`).
`...`	Further arguments passed to `contour`, `image` or `persp` (depending on `type`).

Details

Objects from this class are returned by `optimizeFeatureFinding` and `optimizeFeatureGrouping`.

Methods (by generic)

- `algorithm`: Returns the algorithm that was used for finding features.
- `length`: Obtain total number of experimental design iterations performed.
- `lengths`: Obtain number of experimental design iterations performed for each parameter set.
- `show`: Shows summary information for this object.
- `plot`: Generates response plots for all or a selected set of parameters.
- `optimizedParameters`: Returns parameter set yielding optimal results. The `paramSet` and `DoEIteration` arguments can be `NULL`.
- `optimizedObject`: Returns the object (*i.e.* a `features` or `featureGroups` object) that was generated with optimized parameters. The `paramSet` argument can be `NULL`.
- `scores`: Returns optimization scores. The `paramSet` and `DoEIteration` arguments can be `NULL`.
- `experimentInfo`: Returns a `list` with optimization information from an DoE iteration.

Slots

`algorithm` A character specifying the algorithm that was optimized.

`paramSets` A `list` with detailed results from each parameter set that was tested.

`bestParamSet` Numeric index of the parameter set yielding the best response.

Examples

```
## Not run:
# ftOpt is an optimization object.

# plot contour of all parameter pairs from the first parameter set/iteration.
plot(ftOpt, paramSet = 1, DoEIteration = 1)

# as above, but only plot two parameter pairs
```

```
plot(ftOpt, paramSet = 1, DoEIteration = 1,
     paramsToPlot = list(c("mzPPM", "chromFWHM"), c("chromFWHM", "chromSNR")))

# plot 3d perspective plots
plot(ftOpt, paramSet = 1, DoEIteration = 1, type = "persp")

## End(Not run)
```

`reporting` *Report feature group data*

Description

Functionality to report data produced by most workflow steps such as features, feature groups, calculated chemical formulae and tentatively identified compounds.

Usage

```
## S4 method for signature 'featureGroups'
reportCSV(
  fGroups,
  path = "report",
  reportFGroupsAsRows = TRUE,
  reportFGroupsAnalysisInfo = TRUE,
  reportFGroupsRetMz = TRUE,
  reportFeatures = FALSE,
  formulas = NULL,
  formulasNormalizeScores = "max",
  formulasExclNormScores = NULL,
  compounds = NULL,
  compoundsNormalizeScores = "max",
  compoundsExclNormScores = c("score", "individualMoNAScore"),
  compsCluster = NULL,
  components = NULL,
  retMin = TRUE,
  clearPath = FALSE
)

## S4 method for signature 'featureGroups'
reportPDF(
  fGroups,
  path = "report",
  reportFGroups = TRUE,
  formulas = NULL,
  formulasTopMost = 5,
  formulasNormalizeScores = "max",
  formulasExclNormScores = NULL,
```

```
  reportFormulaSpectra = TRUE,
  compounds = NULL,
  compoundsNormalizeScores = "max",
  compoundsExclNormScores = c("score", "individualMoNAScore"),
  compoundsOnlyUsedScorings = TRUE,
  compoundsTopMost = 5,
  compsCluster = NULL,
  components = NULL,
  MSPeakLists = NULL,
  retMin = TRUE,
  EICGrid = c(2, 1),
  EICRtWindow = 20,
  EICMzWindow = 0.005,
  EICTopMost = NULL,
  EICOnlyPresent = TRUE,
  clearPath = FALSE
)

## S4 method for signature 'featureGroups'
reportHTML(
  fGroups,
  path = "report",
  reportPlots = c("chord", "venn", "upset", "eics", "formulas"),
  formulas = NULL,
  formulasTopMost = 5,
  formulasNormalizeScores = "max",
  formulasExclNormScores = NULL,
  compounds = NULL,
  compoundsNormalizeScores = "max",
  compoundsExclNormScores = c("score", "individualMoNAScore"),
  compoundsOnlyUsedScorings = TRUE,
  compoundsTopMost = 5,
  compsCluster = NULL,
  includeMFWebLinks = "compounds",
  components = NULL,
  interactiveHeat = FALSE,
  MSPeakLists = NULL,
  retMin = TRUE,
  EICRtWindow = 20,
  EICMzWindow = 0.005,
  EICTopMost = NULL,
  EICOnlyPresent = TRUE,
  selfContained = TRUE,
  optimizePng = FALSE,
  maxProcAmount = getOption("patRoon.maxProcAmount"),
  clearPath = FALSE,
  openReport = TRUE,
  noDate = FALSE
```

)

Arguments

`fGroups`	The `featureGroups` object that should be used for reporting data.
`path`	The destination file path for files generated during reporting. Will be generated if needed.
`reportFGroupsAsRows`	Report feature groups as rows (instead of columns) within the resulting '`.csv`' file.
`reportFGroupsAnalysisInfo`	Include analyses information (reference and replicate groups) in the reported feature groups table '`.csv`' file.
`reportFGroupsRetMz`	Include feature group information (retention time and m/z) within the reported feature groups table '`.csv`' file.
`reportFeatures`	If set to `TRUE` then for each analysis a '`.csv`' file will be generated with information about its detected features.
`formulas, compounds, compsCluster, components`	Further objects (`formulas`, `compounds`, `compoundsCluster`, `components`) that should be reported. Specify `NULL` to skip reporting a particular object.
`compoundsNormalizeScores, formulasNormalizeScores`	A `character` that specifies how normalization of compound scorings occurs. Either `"none"` (no normalization), `"max"` (normalize to max value) or `"minmax"` (perform min-max normalization). Note that normalization of negative scores (e.g. output by `SIRIUS`) is always performed as min-max. Furthermore, currently normalization for `compounds` takes the original min/max scoring values into account when candidates were generated. Thus, for `compounds` scoring, normalization is not affected when candidate results were removed after they were generated (*e.g.* by use of `filter`).
`compoundsExclNormScores, formulasExclNormScores`	A `character` vector specifying any compound scoring names that should *not* be normalized. Set to `NULL` to normalize all scorings. Note that whether any normalization occurs is set by the `compoundsExclNormScores,formulasExclNormScor` argument. For `compounds`: By default `score` and `individualMoNAScore` are set to mimic the behavior of the `MetFrag` web interface.
`retMin`	If `TRUE` then report retention times in minutes (otehrwise seconds).
`clearPath`	If `TRUE` then the destination path will be (recursively) removed prior to reporting.
`reportFGroups`	If `TRUE` then feature group data will be reported.
`formulasTopMost, compoundsTopMost`	Only this amount of top ranked candidate formulae/compounds are reported. Lower values may significantly speed up reporting. Set to `NULL` to ignore.

`reportFormulaSpectra`
: If `TRUE` then explained MS/MS spectra (if available) for candidate formulae will be reported. Specifying `formulas` and setting this argument to `FALSE` still allows further annotation of compound MS/MS spectra.

`compoundsOnlyUsedScorings`
: If `TRUE` then only scorings are plotted that actually have been used to rank data (see the `scoreTypes` argument to `generateCompoundsMetfrag` for more details).

`MSPeakLists`
: A `MSPeakLists` object that is *mandatory* when spectra for formulae and/or compounds will be reported.

`EICGrid`
: An integer vector in the form `c(columns,rows)` that is used to determine the plotting grid when reporting EICs in PDF files.

`EICRtWindow, EICMzWindow, EICTopMost, EICOnlyPresent`
: Plotting parameters passed to `plotEIC` (*i.e.* `rtWindow`, `mzWindow`, `topMost` and `onlyPresent` arguments).

`reportPlots`
: A character vector specifying what should be plotted. Valid options are: `"chord"`, `"venn"`, `"upset"` (plot a chord, Venn and UpSet diagram, respectively), `"eics"` (plot EICs for individual feature groups) and `"formulas"` (plot annotated formula spectra). Set to `"none"` to plot none of these.

`includeMFWebLinks`
: A `character` specifying to which feature groups a web link should be added in the annotation page to MetFragWeb. Options are: `"compounds"` (only to those with compounds results), `"MSMS"` (only to those with MSMS peak lists) or `"none"`.

`interactiveHeat`
: If `TRUE` an interactive heatmap HTML widget will be generated to display hierarchical clustering results. Set to `FALSE` for a 'regular' static plot.

`selfContained`
: If `TRUE` the output will be a standalone HTML file which contains all graphics and script dependencies. When `FALSE`, the latter will be placed in an additional directory ('`report_files`') which should remain present when viewing the output file. Especially on Windows, a non-self contained output might be desirable when reporting large amounts of data to prevent `pandoc` from running out of memory.

`optimizePng`
: If `TRUE` then `pngquant` is used to reduce the size of generated graphics. A signficant reduction in disk space usage may be seen, however, at the cost additional processing time.

`maxProcAmount`
: Maximum amount of `pngquant` commands to run in parallel. Higher numbers will decrease processing time, with an optimum usually close to the amount of CPU cores.

`openReport`
: If set to `TRUE` then the output report file will be opened with the system browser.

`noDate`
: If `TRUE` then the current date is not added to the report. This is mainly used for testing and its main purpose is to guarentees equal report files when 'reportHTML()' is called multiple times with equal arguments.

Details

These functions are usually called at the very end of the workflow. It is used to report various data on features and feature groups. In addition, these functions may be used for reporting formulae and/or compounds that were generated for the specified feature groups. Data can be reported in tabular form (*i.e.* '`.csv`' files) by `reportCSV` or graphically by `reportPDF` and `reportHTML`. The latter functions will plot for instance chromatograms and annotated mass spectra, which are useful to get a graphical overview of results.

All functions have a wide variety of arguments that influence the reporting process. Nevertheless, most parameters are optional and only required to be given for fine tuning. In addition, only those objects (*e.g.* formulae, compounds, clustering) that are desired to be reported need to be specified.

`reportCSV` generates tabular data (*i.e.* '`.csv`' files) for given data to be reported. This may also be useful to allow import by other tools for post processing.

`reportPDF` will report graphical data (*e.g.* chromatograms and mass spectra) within PDF files. Compared to `reportHTML` this function may be faster and yield smaller report files, however, its functionality is a bit more basic and generated data is more 'scattered' around.

`reportHTML` will report graphical data (*e.g.* chromatograms and mass spectra) and summary information in an easy browsable `HTML` file using rmarkdown, flexdashboard and knitr.

Note

Any formulae and compounds for feature groups which are not present within `fGroups` (*i.e.* because it has been subset afterwards) will not be reported.

References

Creating MetFrag landing page URLs based on code from MetFamily R package.

Yihui Xie (2015) Dynamic Documents with R and knitr. 2nd edition. Chapman and Hall/CRC. ISBN 978-1498716963

Yihui Xie (2014) knitr: A Comprehensive Tool for Reproducible Research in R. In Victoria Stodden, Friedrich Leisch and Roger D. Peng, editors, Implementing Reproducible Computational Research. Chapman and Hall/CRC. ISBN 978-1466561595

`suspect-screening` *Target and suspect screening*

Description

Utilities to screen for analytes with known or suspected identity.

Usage

```
screenSuspects(
  obj,
  suspects,
  rtWindow = 12,
  mzWindow = 0.005,
  adduct = NULL,
  skipInvalid = TRUE
)

## S4 method for signature 'features'
screenSuspects(
  obj,
  suspects,
  rtWindow = 12,
  mzWindow = 0.005,
  adduct = NULL,
  skipInvalid = TRUE
)

## S4 method for signature 'featureGroups'
screenSuspects(
  obj,
  suspects,
  rtWindow = 12,
  mzWindow = 0.005,
  adduct = NULL,
  skipInvalid = TRUE
)

## S4 method for signature 'featureGroups'
groupFeaturesScreening(fGroups, scr)

importFeatureGroupsBrukerTASQ(path, analysisInfo, clusterRTWindow = 12)
```

Arguments

obj	The object that should be screened (*i.e.* features or featureGroups).
suspects	A data.frame that must contain a "name" column (the analyte name) and *at least* a "mz", "neutralMass", "formula", "SMILES", "InChI" column (with the ionized m/z value, the neutral monoisotopic mass or the chemical formula/SMILES/InChI character string for the molecule, respectively). If an ion mass needs to be calculated (*i.e.* no valid data is available in the mz column) then data is tried to be used from the columns in the aforementioned order. Furthermore, if ion masses need to be calculated then the adduct must be specified either with the adduct function argument or by an "adduct" column containing a character that can be converted with as.adduct (*e.g.* "[M+H]+"). In addition, a column "rt"

can be included to specify the retention time (if unspecified no retention times are checked).

`rtWindow, mzWindow`
: The retention time window (in seconds) and m/z window that will be used for matching a suspect (+/- feature data).

`adduct`
: An `adduct` object (or something that can be converted to it with `as.adduct`). Examples: `"[M-H]-"`, `"[M+Na]+"`. Can be `NULL` if either a `"mz"` or `"adduct"` column is present in the `suspects` argument.

`skipInvalid`
: If set to `TRUE` then suspects with invalid data (*e.g.* missing names or other missing data) will be ignored with a warning. Similarly, any suspects for which mass calculation failed (when no `mz` column is present in the suspect list), for instance, due to invalid `SMILES`, will be ignored with a warning.

`fGroups`
: The `featureGroups` object that should be transformed (and was used to obtain the screening results).

`scr`
: The screening results table returned by `screenSuspects`.

`path`
: The file path to an Excel export of the Global results table from TASQ, converted to '`.csv`' format.

`analysisInfo`
: A table with analysis information.

`clusterRTWindow`
: This retention time window (in seconds) is used to group hits across analyses together. See also the details section.

Details

Besides 'full non-target analysis', where compounds may be identitified with little to no prior knowledge, a common strategy is to screen for compounds with known or suspected identity. This may be a generally favourable approach if possible, as it can significantly reduce the load on data interpretation.

`screenSuspects` will screen a set of suspects (provided as a `data.frame`) within an `features` or `featureGroups` object.

`groupFeaturesScreening` uses results from `screenSuspects` to transform an existing `featureGroups` object by (1) renaming any matched feature groups by the respective name of the suspect and (2) filtering out any feature groups that were not matched. A common workflow is to first obtain and group features (with *e.g.* `findFeatures` and `groupFeatures`), screen them with `screenSuspects`, convert the `featureGroups` object that was used for screening with this method function and continue any further workflow steps such as compound identification as with 'regular' `featureGroups`.

`importFeatureGroupsBrukerTASQ` will convert screening results from Bruker TASQ to a `featureGroups` object. The feature groups across analyses are formed based on the name of suspects and their closeness in retention time. The latter is necessary because TASQ does not necessarily perform checks on retention times and may therefore assign a suspect to peaks with different retention times across analyses (or within a single analysis). Hence, suspects with equal names are hierarchically clustered on their retention times (using **fastcluster**) to form the feature groups. The cut-off value for this is specified by the `clusterRTWindow` argument. The input for this function is obtained by generating an Excel

export of the 'global' results and subsequently converting the file to '.csv' format. Similar to groupFeaturesScreening, this method will return an object that is suitable for any further workflow processes.

Value

screenSuspects will return a table (a data.table) with detected suspects and details such as retention and m/z values. If a suspect is matched on multiple features/feature groups then each hit is reported as a separate row.

groupFeaturesScreening returns a modified featureGroups object in which those feature groups that were not matched by any suspects are removed and others renamed by the respective suspect name. In case of duplicate suspect results, feature group names are made unique with make.unique.

importFeatureGroupsBrukerTASQ returns a new featureGroups object containing converted screening results from Bruker TASQ.

Note

Both groupFeaturesScreening and importFeatureGroupsBrukerTASQ use names from targets/suspects as feature group names, therefore, it is important that these are file-compatible names when reporting data 'csv' or 'pdf' data.

For screenSuspects in some cases you may need to install OpenBabel (*e.g.* when only InChI data is available for mass calculation).

Please note that groupFeaturesScreening method can only transform the featureGroups object that was used to obtain the given screening results.

importFeatureGroupsBrukerTASQ will use estimated min/max values for retention times and dummy min/max m/z values for conversion to features, since this information is not (readily) available. Hence, when plotting, for instance, extracted ion chromatograms (with plotEIC) the integrated chromatographic peak range shown is incorrect.

References

Daniel Müllner (2013). fastcluster: Fast Hierarchical, Agglomerative Clustering Routines for R and Python. Journal of Statistical Software, 53(9), 1-18. URL http://www.jstatsoft.org/v53/i09/.

---

verifyDependencies — *Verifies if all dependencies are installed properly and instructs the user if this is not the case.*

Description

Verifies if all dependencies are installed properly and instructs the user if this is not the case.

Usage

```
verifyDependencies()
```

workflowStep-class *(Virtual) Base class for all workflow objects.*

Description

All workflow objects (*e.g.* featureGroups, compounds, etc) are derived from this class. Objects from this class are never created directly.

Usage

```
## S4 method for signature 'workflowStep'
algorithm(obj)

## S4 method for signature 'workflowStep'
as.data.table(x, keep.rownames = FALSE, ...)

## S4 method for signature 'workflowStep'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S4 method for signature 'workflowStep'
show(object)
```

Arguments

obj, x, object	An object (derived from) this class.
keep.rownames	Ignored.
...	Method specific arguments. Please see the documentation of the derived classes.
row.names, optional	Ignored.

Methods (by generic)

- algorithm: Returns the algorithm that was used to generate an object.
- as.data.table: Summarizes the data in this object and returns this as a data.table.
- as.data.frame: This method simply calls as.data.table and converts the result to a classic a data.frame.
- show: Shows summary information for this object.

Slots

algorithm The algorithm that was used to generate this object. Use the algorithm method for access.

S4 class hierarchy

- workflowStep
  - features
    - featuresFromFeatGroups
    - featuresConsensus
    - featuresBruker
    - featuresEnviPick
    - featuresOpenMS
    - featuresBrukerTASQ
    - featuresXCMS
    - featuresXCMS3
  - featureGroups
    - featureGroupsBruker
    - featureGroupsConsensus
    - featureGroupsEnviMass
    - featureGroupsOpenMS
    - featureGroupsScreening
    - featureGroupsBrukerTASQ
    - featureGroupsXCMS
    - featureGroupsXCMS3
  - components
    - componentsReduced
    - componentsCamera
    - componentsIntClust
    - componentsNT
    - componentsRC
  - compounds
    - compoundsConsensus
    - compoundsMF
  - formulas
  - MSPeakLists