Package 'patRoon'

December 5, 2021

Type Package

Title Workflows for Mass-Spectrometry Based Non-Target Analysis

Version 2.0.0

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 7.1.2

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,
DBI,
RSQLite (>= 2.2.4),
fst,
processx,
tools,
MSnbase,
Biobase,
BiocParallel,
xcms,
cluster,
fastcluster,
gplots,
heatmaply,
dynamicTreeCut,
dendextend,
igraph,
visNetwork,
rJava,
rcdk,
fingerprint,
mzR,
circlize,
miniUI,
rhandsontable,
rstudioapi,
htmlwidgets,
shiny,
shinyjs,
CAMERA,
enviPat,
nontarget,
knitr,
rmarkdown,
flexdashboard,
DT,
magrittr,
kableExtra,
R.utils,
magick,
glue,
jsonlite,
Rdpack,
rsm,
future,
future.apply,
fs,
yaml,

keys,
backports

Suggests RDCOMClient,
metfRag,
RAMClustR,
KPIC,
MetaClean,
MetaCleanData,
testthat,
rlang,
vdiffr,
patRoonData,
devtools,
covr,
DiagrammeR,
DiagrammeRsvg,
rsvg,
pkgload

LinkingTo Rcpp

RdMacros Rdpack

Collate 'RcppExports.R'
'generics.R'
'cache.R'
'main.R'
'workflow-step.R'
'TP.R'
'TP-biotransformer.R'
'TP-library.R'
'features.R'
'workflow-step-set.R'
'features-set.R'
'feature_groups.R'
'feature_groups-set.R'
'TP-logic.R'
'utils.R'
'utils-adduct.R'
'adduct.R'
'check_ui.R'
'utils-components.R'
'components.R'
'check_components.R'
'check_features.R'
'components-camera.R'
'components-features.R'
'components-cliquems.R'
'components-clust.R'
'components-intclust.R'

'components-set.R'
'components-nontarget.R'
'components-nontarget-set.R'
'components-openms.R'
'components-ramclustr.R'
'components-specclust.R'
'feature_annotations.R'
'formulas.R'
'mspeaklists.R'
'compounds.R'
'utils-compounds.R'
'utils-screening.R'
'feature_groups-screening.R'
'feature_groups-screening-set.R'
'components-tps.R'
'compounds-cluster.R'
'compounds-metfrag.R'
'utils-feat_annotations-set.R'
'compounds-set.R'
'utils-sirius.R'
'compounds-sirius.R'
'convert.R'
'defunct.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-kpic2.R'
'feature_groups-openms.R'
'feature_groups-optimize.R'
'feature_groups-optimize-kpic2.R'
'feature_groups-optimize-openms.R'
'feature_groups-optimize-xcms.R'
'feature_groups-optimize-xcms3.R'
'feature_groups-sirius.R'
'feature_groups-tasq.R'
'feature_groups-xcms.R'
'feature_groups-xcms3.R'
'utils-bruker.R'
'features-bruker.R'
'features-envipick.R'
'features-kpic2.R'
'features-openms.R'
'features-optimize.R'
'features-optimize-envipick.R'

'features-optimize-kpic2.R'
'features-optimize-openms.R'
'features-optimize-xcms.R'
'features-optimize-xcms3.R'
'features-safd.R'
'features-sirius.R'
'features-tasq.R'
'features-xcms.R'
'features-xcms3.R'
'formulas-bruker.R'
'formulas-genform.R'
'formulas-set.R'
'formulas-sirius.R'
'mspeaklists-bruker.R'
'utils-mzr.R'
'mspeaklists-mzr.R'
'mspeaklists-set.R'
'multi-process-classic.R'
'multi-process-future.R'
'multi-process.R'
'project-tool.R'
'report.R'
'utils-TPs.R'
'utils-checkmate.R'
'utils-exported.R'
'utils-feat_annotations.R'
'utils-features.R'
'utils-formulas.R'
'utils-mol.R'
'utils-mspeaklists.R'
'utils-optimize.R'
'utils-plot.R'
'utils-progress.R'
'utils-sets.R'
'utils-xcms.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.MP.maxProcs`: 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`. This option is only used when 'patRoon.MP.method="classic"'.
- `patRoon.MP.method`: Either `"classic"` or `"future"`. The former is the default and uses **processx** to execute multiple commands in parallel. When `"future"` the `future.apply` package is used for parallelization, which is especially useful for *e.g.* cluster computing.
- `patRoon.MP.futureSched`: Sets the `future.scheduling` function argument for `future_lapply`. Only used if 'patRoon.MP.method="future"'.
- `patRoon.MP.logPath`: The path used for logging of output from commands executed by multiprocess. Set to `FALSE` to disable logging.
- `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.
- patRoon.path.BiotransFormer The full file path to the biotransformer '.jar' command line utility. This needs to be set when generateTPsBioTransformer is used. For more details see https://bitbucket.org/djoumbou/biotransformer/src/master.

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]
- Bas van de Velde (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", err = TRUE)
```

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. "openms" is the format used by the MetaboliteAdductDecharger tool. "cliquems" is the format used by **cliqueMS**.
err	If TRUE then an error will be thrown if conversion fails, otherwise returns NA.
...	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, charge = 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. `"openms"` is the format used by the `MetaboliteAdductDecharger` tool. `"cliquems"` is the format used by **cliqueMS**.
isPositive	A logical that specifies whether the adduct should be positive. Should only be set when `format="metfrag"` and `x` is a `numeric` identifier.
charge	The final charge. Only needs to be set when `format="openms"`.
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 **RDCOM-Client** 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`

---

check-GUI *Interactive GUI utilities to check workflow data*

Description

These functions provide interactive utilities to explore and review workflow data using a **shiny** graphical user interface (GUI). In addition, unsatisfactory data (*e.g.* noise identified as a feature and unrelated feature groups in a component) can easily be selected for removal.

Usage

```
## S4 method for signature 'components'
checkComponents(
  components,
  fGroups,
  session = "checked-components.yml",
  rtWindow = 30,
  clearSession = FALSE
)

importCheckFeaturesSession(
  sessionIn,
  sessionOut,
  fGroups,
  rtWindow = 6,
  mzWindow = 0.002,
  overWrite = FALSE
)
```

```
## S4 method for signature 'featureGroups'
checkFeatures(
  fGroups,
  session = "checked-features.yml",
  rtWindow = 30,
  clearSession = FALSE
)

getMCTrainData(fGroups, session)

predictCheckFeaturesSession(fGroups, session, model = NULL, overWrite = FALSE)
```

Arguments

components	The components to be checked.
fGroups	A featureGroups object. This should be the 'new' object for importCheckFeaturesSession for which the session needs to be imported.
session	The session file name.
rtWindow	For checkFeatures and checkComponents: the 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. For importCheckFeaturesSession: the retention time window (seconds) used to relate 'old' with 'new' feature groups.
clearSession	If TRUE the session will be completely cleared before starting the GUI. This effectively removes all selections for data removal.
sessionIn, sessionOut	The file names for the input and output sessions.
mzWindow	The *m/z* window (in Da) used to relate 'old' with 'new' feature groups.
overWrite	Set to TRUE to overwrite the output session file if it already exists. If FALSE, the function will stop with an error message.
model	The model that was created with **MetaClean** and that should be used to predict pass/fail data. If NULL, the example model of the **MetaCleanData** package is used.

Details

The data selected for removal is stored in *sessions*. These are 'YAML' files to allow easy external manipulation. The sessions can be used to restore the selections that were made for data removal when the GUI tool is executed again. Furthermore, functionality is provided to import and export sessions. To actually remove the data the filter method should be used with the session file as input.

checkComponents is used to review components and their feature groups contained within. A typical use case is to verify that peaks from features that were annotated as related adducts and/or isotopes are correctly aligned.

`importCheckFeaturesSession` is used to import a session file that was generated from a different `featureGroups` object. This is useful to avoid re-doing manual interpretation of chromatographic peaks when, for instance, feature group data is re-created with different parameters.

`checkFeatures` is used to review chromatographic information for feature groups. Its main purpose is to assist in reviewing the quality of detected feature (groups) and easily select unwanted data such as features with poor peak shapes or noise.

`getMCTrainData` converts a session created by `checkFeatures` to a `data.frame` that can be used by the **MetaClean** to train a new model. The output format is comparable to that from `getPeakQualityMetrics`.

`predictCheckFeaturesSession` Uses ML data from **MetaClean** to predict the quality (Pass/Fail) of feature group data, and converts this to a session which can be reviewed with `checkFeatures` and used to remove unwanted feature groups by `filter`.

Note

`checkComponents`: Some componentization algorithms (*e.g.* `generateComponentsNontarget` and `generateComponentsTPs`) may output components where the same feature group in a component is present multiple times, for instance, when multiple TPs are matched to the same feature group. If such a feature group is selected for removal, then *all* of its result in the component will be marked for removal.

`getMCTrainData` only uses session data for selected feature groups. Selected features for removal are ignored, as this is not supported by **MetaClean**.

References

Chetnik K, Petrick L, Pandey G (2020). "MetaClean: a machine learning-based classifier for reduced false positive peak detection in untargeted LC-MS metabolomics data." *Metabolomics*, **16**(11). doi: 10.1007/s11306020017383.

`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, ...)

## S4 method for signature 'featureGroups'
generateComponentsCAMERA(
  fGroups,
  ionization = NULL,
  onlyIsotopes = FALSE,
  minSize = 2,
  relMinReplicates = 0.5,
  extraOpts = NULL
)

## S4 method for signature 'featureGroupsSet'
generateComponentsCAMERA(fGroups, ionization = NULL, ...)

## S4 method for signature 'featureGroups'
generateComponentsCliqueMS(
  fGroups,
  ionization = NULL,
  maxCharge = 1,
  maxGrade = 2,
  ppm = 10,
  adductInfo = NULL,
  absMzDev = 0.005,
  minSize = 2,
  relMinAdductAbundance = 0.75,
  adductConflictsUsePref = TRUE,
  NMConflicts = c("preferential", "mostAbundant", "mostIntense"),
  prefAdducts = c("[M+H]+", "[M-H]-"),
  extraOptsCli = NULL,
```

```
  extraOptsIso = NULL,
  extraOptsAnn = NULL,
  parallel = TRUE
)

## S4 method for signature 'featureGroupsSet'
generateComponentsCliqueMS(fGroups, ionization = NULL, ...)

## S4 method for signature 'featureGroups'
generateComponentsIntClust(
  fGroups,
  method = "complete",
  metric = "euclidean",
  normFunc = max,
  average = TRUE,
  maxTreeHeight = 1,
  deepSplit = TRUE,
  minModuleSize = 1
)

## S4 method for signature 'featureGroups'
generateComponentsNontarget(
  fGroups,
  ionization = NULL,
  rtRange = c(-120, 120),
  mzRange = c(5, 120),
  elements = c("C", "H", "O"),
  rtDev = 30,
  absMzDev = 0.002,
  absMzDevLink = absMzDev * 2,
  traceHack = all(R.Version()[c("major", "minor")] >= c(3, 4)),
  ...
)

## S4 method for signature 'featureGroupsSet'
generateComponentsNontarget(fGroups, ionization = NULL, ...)

## S4 method for signature 'featureGroups'
generateComponentsOpenMS(
  fGroups,
  ionization = NULL,
  chargeMin = 1,
  chargeMax = 1,
  chargeSpan = 3,
  qTry = "heuristic",
  potentialAdducts = NULL,
  minRTOverlap = 0.66,
  retWindow = 1,
```

```
  absMzDev = 0.005,
  minSize = 2,
  relMinAdductAbundance = 0.75,
  adductConflictsUsePref = TRUE,
  NMConflicts = c("preferential", "mostAbundant", "mostIntense"),
  prefAdducts = c("[M+H]+", "[M-H]-"),
  extraOpts = NULL
)

## S4 method for signature 'featureGroupsSet'
generateComponentsOpenMS(
  fGroups,
  ionization = NULL,
  chargeMin = 1,
  chargeMax = 1,
  chargeSpan = 3,
  qTry = "heuristic",
  potentialAdducts = NULL,
  ...
)

defaultOpenMSAdducts(ionization)

## S4 method for signature 'featureGroups'
generateComponentsRAMClustR(
  fGroups,
  ionization = NULL,
  st = NULL,
  sr = NULL,
  maxt = 12,
  hmax = 0.3,
  normalize = "TIC",
  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
)

## S4 method for signature 'featureGroupsSet'
generateComponentsRAMClustR(fGroups, ionization = NULL, ...)

## S4 method for signature 'featureGroups'
generateComponentsSpecClust(
  fGroups,
```

```
  MSPeakLists,
  method = "complete",
  specSimParams = getDefSpecSimParams(),
  maxTreeHeight = 1,
  deepSplit = TRUE,
  minModuleSize = 1
)

## S4 method for signature 'featureGroups'
generateComponentsTPs(
  fGroups,
  fGroupsTPs = fGroups,
  ignoreParents = FALSE,
  TPs = NULL,
  MSPeakLists = NULL,
  formulas = NULL,
  compounds = NULL,
  minRTDiff = 20,
  specSimParams = getDefSpecSimParams()
)

## S4 method for signature 'featureGroupsSet'
generateComponentsTPs(
  fGroups,
  fGroupsTPs = fGroups,
  ignoreParents = FALSE,
  TPs = NULL,
  MSPeakLists = NULL,
  formulas = NULL,
  compounds = NULL,
  minRTDiff = 20,
  specSimParams = getDefSpecSimParams()
)
```

Arguments

fGroups	featureGroups object for which components should be generated. For generateComponentsTPs: also see the fGroupsTPs argument.
algorithm	A character string describing the algorithm that should be used: "ramclustr", "camera", "nontarget", "intclust", "openms", "cliquems", "specclust", "tp"
...	Any parameters to be passed to the selected component generation algorithm. For sets workflows: further arguments directly passed to the non-sets method.
ionization	Which ionization polarity was used to generate the data: should be "positive" or "negative". If the featureGroups object has adduct annotations, and ionization=NULL, the ionization will be detected automatically. (**sets workflow**) This parameter is not supported for sets workflows, as the ionization will always be detected automatically.

`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 `CAMERA::annotate` (`generateComponentsCAMERA`) or as extra command line parameters to `MetaboliteAdductDecharger` (`generateComponentsCAMERA`). Set to `NULL` to ignore.

`maxCharge, maxGrade, ppm`
Arguments passed to `cliqueMS::getIsotopes` and/or `cliqueMS::getAnnotation`.

`adductInfo` Sets the `adinfo` argument to `cliqueMS::getAnnotation`. If `NULL` then the default adduct information from **cliqueMS** is used (*i.e.* the `positive.adinfo`/`negative.adinfo` package datasets).

`relMinAdductAbundance`
The minimum relative abundance ('`0-1`') that an adduct should be assigned to features within the same feature group. See the Feature components section for more details.

`adductConflictsUsePref`
If set to `TRUE`, and not all adduct assigments to the features within a feature group are equal and at least one of those adducts is a preferential adduct (`prefAdducts` argument), then only the features with (the lowest ranked) preferential adduct are considered. In all other cases or when `adductConflictsUsePref=FALSE` only features with the most frequently assigned adduct is considered. See the Feature components section for more details.

`NMConflicts` The strategies to employ when not all neutral masses within a component are equal. Valid options are: `"preferential"`, `"mostAbundant"` and `"mostIntense"`. Multiple strategies are possible, and will be executed in the given order until one succeeds. See the Feature components section for more details.

`prefAdducts` A `character` vector with one or more *preferential adducts*. See the Feature components section for more details.

`extraOptsCli, extraOptsIso, extraOptsAnn`
Named `list` with further arguments to be passed to `cliqueMS::getCliques`, `cliqueMS::getIsotopes` and `cliqueMS::getAnnotation`, respectively. Set to `NULL` to ignore.

`parallel` If set to `TRUE` then code is executed in parallel through the **futures** package. Please see the parallelization section in the handbook for more details.

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

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

`normFunc, average`
Passed to `as.data.table` to perform normalization and averaging of data.

`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`.

  For `generateComponentsOpenMS`: Sets the `algorithm:MetaboliteFeatureDeconvolution:mass_max` option.

`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).

`chargeMin, chargeMax`
: The minimum/maximum charge to consider. Corresponds to the `algorithm:MetaboliteFeatureDeconv` options.

`chargeSpan`
: The maximum charge span for a single analyte. Corresponds to `algorithm:MetaboliteFeatureDeconvo`

`qTry`
: Sets how charges are determined. Corresponds to `algorithm:MetaboliteFeatureDeconvolution:q_t` Valid options are `"heuristic"` and `"all"` (the `"feature"` option from OpenMS is currently not supported).

`potentialAdducts`
: The adducts to consider. Should be a `numeric` vector with probabilities for each adduct, *e.g.* `potentialAdducts=c("[M+H]+" = 0.8,"[M+Na]+" = 0.2)`. Note that the sum of probabilities should always be '`1`'. Furthermore, note that additions of multiple adducts should be controlled by the `chargeMin`/`chargeMax` arguments (and *not* with `potentialAdducts`), *e.g.* if `chargeMax=2` then both `[M+H]+` and `[2M+H]2+` may be considered. Please see the `algorithm:MetaboliteFeatureDeconvolutio` option of MetaboliteAdductDecharger for more details. If `NULL` then the a default is chosen with `defaultOpenMSAdducts` (which is *not* the same as `OpenMS`).

  (**sets workflow**) Should be a `list` where each entry specifies the potential adducts for a set. Should either be named with the sets names or follow the same order as `sets(fGroups)`. Example: `potentialAdducts=list(positive=c("[M+H]+" = 0.8,"[M+Na]+" = 0.2),negative=c("[M-H]-" = 0.8,"[M-H2O-H]-" = 0.2))`

`minRTOverlap, retWindow`
: Sets feature retention tolerances when grouping features. Sets the "`algorithm:MetaboliteFeatureDeco` and `algorithm:MetaboliteFeatureDeconvolution:min_rt_overlap` options.

`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.

`MSPeakLists`
: The `MSPeakLists` object for the given feature groups that should be used for MS spectral similarity calculations. For `generateComponentsTPs` it can be `NULL`, in which case no calculations are performed.

`specSimParams`
: A named `list` with parameters that influence the calculation of MS spectra similarities. See the spectral similarity parameters documentation for more details.

`fGroupsTPs`
: A `featureGroups` object containing the feature groups that are expected to be transformation products. If a distinction between parents and TPs is not yet known, `fGroupsTPs` should equal the `fGroups` argument. Otherwise, `fGroups` should only contain the parent feature groups, and both `fGroups` and `fGroupsTPs` *must* be a subset of the same `featureGroups` object.

`ignoreParents`
: If `TRUE` then feature groups present in both `fGroups` and `fGroupsTPs` are not considered as TPs.

`TPs`
: A `transformationProducts` object. Set to `NULL` to perform linking without this data.

`formulas, compounds`
: A `formulas`/`compounds` object to calculate annotation similarities between parents and TPs. If `NULL` then this data is not calculated. For more details see the Linking parents and transformation products section below.

`minRTDiff`
: Minimum retention time (in seconds) difference between the parent and a TP to determine whether a TP elutes prior/after the parent (to calculate `retDir` values, see Details in componentsTPs))

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.

generateComponentsCliqueMS uses cliqueMS to generate components using the cliqueMS::getCliques function. The grouping of features in each component ('clique') is based on high similarity of chromatographic elution profiles. All features in each component are then annotated with the cliqueMS::getIsotopes and cliqueMS::getAnnotation functions.

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 fastcluster::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.

generateComponentsOpenMS uses the MetaboliteAdductDecharger utility (see http://www.openms.de) to generate components. Features that show highly similar chromatographic elution profiles are grouped, and subsequently annotated with their adducts.

defaultOpenMSAdducts returns the default adducts and their probabilities when the OpenMS algorithm is used for componentization. See the potentialAdducts argument for more details.

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.

generateComponentsSpecClust generates components based on MS/MS similarity between feature groups. The similarities are converted to a distance matrix and used as input for hierarchical clustering, and the resulting dendrogram is automatically cut with cutreeDynamicTree. The clustering is performed with fastcluster::hclust.

generateComponentsTPs generates components by linking feature groups of transformation products and their parents. Moreover, this method typically employs data from generated transformation products to find parents and their TPs. However, this data is not necessary, and components can also be made based on MS/MS similarity and/or other annotation similarities between the parent and its TPs. For more details see the Linking parents and transformation products section below.

Value

A components (derived) object containing all generated components.

The algorithms based on 'feature components' return a componentsFeatures derived object.

generateComponentsIntClust and generateComponentsSpecClust return objects derived from componentsSpecClust.

generateComponentsTPs returns objects derived from componentsTPs.

Feature components

Components resulting from `generateComponentsCliqueMS` and `generateComponentsOpenMS` are based on so called *feature components*. Unlike other algorithms, components are first made on a feature level (per analysis), instead of for complete feature groups. In the final step the feature components are converted to 'regular' components by employing a consensus approach with the following steps:

1. If an adduct assigned to a feature only occurs as a minority compared to other adduct assignments within the same feature group, it is considered as an outlier and removed accordingly (controlled by the `relMinAdductAbundance` argument).
2. For features within a feature group, only keep their adduct assignment if it occurs as the most frequent or is preferential (controlled by `adductConflictsUsePref` and `prefAdducts` arguments).
3. Components are made by combining the feature groups for which at least one of their features are jointly present in the same feature component.
4. Conflicts of neutral mass assignments within a component (*i.e.* not all are the same) are dealt with. Firstly, all feature groups with an unknown neutral mass are split in another component. Then, if conflicts still occur, the feature groups with similar neutral mass (determined by `absMzDev` argument) are grouped. Depending on the `NMConflicts` argument, the group with one or more preferential adduct(s) or that is the largest or most intense is selected, whereas others are removed from the component. In case multiple groups contain preferential adducts, and '`>1`' preferential adducts are available, the group with the adduct that matches first in `prefAdducts` 'wins'. In case of ties, one of the next strategies in `NMConflicts` is tried.
5. If a feature group occurs in multiple components it will be removed completely.
6. the `minSize` filter is applied.

Sets workflows

In a sets workflow the componentization data is generated differently depending on the used algorithm:

- The componentization algorithms that support annotation of adducts/isotopes, *e.g.* **RAMClustR**, **CAMERA** and `OpenMS`, first perform componentization for each set independently. This is necessary, because *e.g.* adducts are specific to the MS ionization mode that was used. The resulting components are then all combined in a `componentsSet` object. Note that the components themselves are never merged. The components are renamed to include the set name for which they were generated (*e.g.* `"CMP1"` becomes `"CMP1-positive"`).
- For componentization with **nontarget** a similar strategy is applied as above. However, the output is stored in a `componentsNTSet` object, which supports additional methods such as `plotGraph`.
- For the componentization based on hierarchical clustering, *i.e.* the `"intclust"` and `"specclust"` algorithms, the output format is not different than for a non-sets workflow. However, for intensity based clustering, normalization of feature intensities occurs per set. For MS/MS based clustering, spectral similarities from each sets are combined as is described for the `spectrumSimilarity` method for sets workflows.

- For componentization of transformation products, the output class is the same (componentsTPs). However, the component tables are amended with extra information such as overall/specific set spectrum similarities. As sets data is mixed, transformation products are able to be linked with a parent, even if they were not measured in the same set.

Parallelization

generateComponentsOpenMS uses multiprocessing to parallelize computations. Please see the parallelization section in the handbook for more details and patRoon options for configuration options.

Note that for caching purposes, the analyses files must always exist on the local host computer, even if it is not participating in computations.

Linking parents and transformation products

With generateComponentsTPs, each component consists of feature groups that are considered to be transformation products for one parent (the parent that 'belongs' to the component can be retrieved with the componentInfo method). The parent feature groups are taken from the fGroups parameter, while the feature groups for TPs are taken from fGroupsTPs. If a feature group occurs in both variables, it may therefore be considered as both a parent or TP.

If transformation product data is given, *i.e.* the TPs argument is set, then a suspect screening of the TPs must be performed in advance (see screenSuspects and convertToSuspects to create the suspect list). Furthermore, if TPs were generated with generateTPsBioTransformer or generateTPsLibrary then the suspect screening must also include the parents (*e.g.* by setting includeParents=TRUE when calling convertToSuspects or by amending results by setting amend=TRUE to screenSuspects). The suspect screening is necessary for the componentization algorithm to map the feature groups of the parent or TP. If the the suspect screening yields multiple TP hits, all will be reported. Similarly, if the suspect screening contains multiple hits for a parent, a component is made for each of the parent hits.

In case no transformation product data is provided (TPs=NULL), the componentization algorithm simply assumes that each feature group from fGroupsTPs is a potential TP for every parent feature group in fGroups. For this reason, it is highly recommended to specify which feature groups are parents/TPs (see the fGroupsTPs argument description above) and *crucial* that the data is post-processed, for instance by only retaining TPs that have high annotation similarity with their parents (see the filter method for componentsTPs).

An typical way to distinguish which feature groups are parents or TPs from two different (groups of) samples is by calculating Fold Changes (see the as.data.table method for feature groups and plotVolcano). Of course, other statistical techniques from R are also suitable.

During componentization, several characteristics are calculated which may be useful for post-processing:

- specSimilarity: the MS/MS spectral similarity between the feature groups of the TP and its parent ('0-1').
- specSimilarityPrec,specSimilarityBoth: as specSimilarity, but calculated with binned data using the "precursor" and "both" method, respectively (see MS spectral similarity parameters for more details).

- `fragmentMatches` The number of MS/MS fragment formula annotations that overlap between the TP and parent. If both the `formulas` and `compounds` arguments are specified then the annotation data is pooled prior to calculation. Note that only unique matches are counted. Furthermore, note that annotations from *all* candidates are considered, even if the formula/structure of the parent/TP is known. Hence, `fragmentMatches` is mainly useful when little or no chemical information is known on the parents/TPs, *i.e.*, when `TPs=NULL` or originates from `generateTPsLogic`. Since annotations for all candidates are used, it is highly recommended that the annotation objects are first processed with the `filter` method, for instance, to select only the top ranked candidates.
- `neutralLossMatches` As `fragmentMatches`, but counting overlapping neutral loss formulae.
- `retDir` The retention time direction of the TP relative to its parent. See Details in componentsTPs. If TP data was specified, the expected direction is stored in `TP_retDir`.
- `retDiff`,`mzDiff`,`formulaDiff` The retention time, *m/z* and formula difference between the parent and TP (latter only available if data TP formula is available).

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.

The `shift` parameter of `specSimParams` is ignored by `generateComponentsTPs`, since it always calculates similarities with all supported options.

Author(s)

Rick Helmus <<r.helmus@uva.nl>> and Bas van de Velde (major contributions to spectral binning and similarity calculation).

References

Schollee JE, Bourgin M, von Gunten U, McArdell CS, Hollender J (2018). "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.

Daniel Müllner (2013). fastcluster: Fast Hierarchical, Agglomerative Clustering Routines for R and Python. Journal of Statistical Software, 53(9), 1-18. URL https://www.jstatsoft.org/v53/i09/.

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)

Senan O, Aguilar-Mogas A, Navarro M, Capellades J, Noon L, Burks D, Yanes O, Guimera R, Sales-Pardo M (2019). "CliqueMS: a computational tool for annotating in-source metabolite ions from LC-MS untargeted metabolomics data based on a coelution similarity network." *Bioinformatics*, **35**(20), 4089–4097. doi: 10.1093/bioinformatics/btz207.

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.

Bielow C, Ruzek S, Huber CG, Reinert K (2010). "Optimal Decharging and Clustering of Charge Ladders Generated in ESI-MS." *Journal of Proteome Research*, **9**(5), 2688–2695. doi: 10.1021/pr100177k.

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'
delete(obj, i = NULL, j = NULL, ...)

## 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,
  checkComponentsSession = NULL,
  negate = FALSE,
  verbose = TRUE
)

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

## S4 method for signature 'components'
plotSpectrum(obj, index, markFGroup = NULL, xlim = NULL, ylim = NULL, ...)

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

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

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

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

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

## S4 method for signature 'componentsSet'
filter(obj, ..., negate = FALSE, sets = NULL)

## S4 method for signature 'componentsSet'
```

```
delete(obj, i = NULL, j = NULL, ...)

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

## S4 method for signature 'componentsSet'
unset(obj, set)
```

Arguments

obj, object, x	The component object.
i, j	For [/[[: 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 delete: The data to remove from. i are the components as numeric index, logical or character, j the feature groups as numeric index/logical (relative to component) or character. If either is NULL then data for all is removed. j may also be a function: it will be called for each component, with the component (a data.table) as first argument, the component name as second argument, and any other arguments passed as ... to delete. The return value of this function specifies the feature groups to be removed (same format as j).
...	For delete: passed to the function specified as j. For plotChroms: Further (optional) arguments passed to the plotChroms 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 plotSpectrum: Further arguments passed to plot. For consensus: components objects that should be used to generate the consensus. For sets workflow methods: further arguments passed to the base components method.
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.

`checkComponentsSession` If set then components and/or feature groups are removed that were selected for removal (see check-GUI and the `checkComponents` function). The value of `checkComponentsSession` should either by a path to the session file or `TRUE`, in which case the default session file name is used. If `negate=TRUE` then all non-selected data is removed instead.

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

`verbose` If set to `FALSE` then no text output is shown.

`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.

`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 `plotChroms` method for the `featureGroups` class.

`sets` (**sets workflow**) A `character` with name(s) of the sets to keep (or remove if `negate=TRUE`).

`set` (**sets workflow**) The name of the set.

Details

`components` objects are obtained from component generators.

Value

`delete` returns the object for which the specified data was removed.

`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.
- delete: Completely deletes specified (parts of) components.
- 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.
- plotSpectrum: Plot a *pseudo* mass spectrum for a single component.
- plotChroms: 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
    - componentsCamera
    - componentsFeatures
      - componentsCliqueMS
      - componentsOpenMS
    - componentsClust
      - componentsIntClust
      - componentsSpecClust
    - componentsSet
      - componentsNTSet
    - componentsUnset
    - componentsNT
      - componentsNTUnset
    - componentsRC
    - componentsTPs

Sets workflows

The `componentsSet` class is applicable for sets workflows. This class is derived from `components` and therefore largely follows the same user interface.

The following methods are specifically defined for sets workflows:

- All the methods from base class `workflowStepSet`.
- `unset` Converts the object data for a specified set into a 'non-set' object (`componentsUnset`), which allows it to be used in 'regular' workflows. Only the components in the specified set are kept.

The following methods are changed or with new functionality:

- `filter` and the subset operator (`[`) Can be used to select components that are only present for selected sets.

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`

---

componentsClust-class *Base class for components that are based on hierarchical clustered data.*

Description

This base class is derived from `components` and is used to store components resulting from hierarchical clustering information, for instance, generated by `generateComponentsIntClust` and `generateComponentsSpecClust`.

Usage

```
## S4 method for signature 'componentsClust'
delete(obj, ...)

## S4 method for signature 'componentsClust'
clusters(obj)

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

## S4 method for signature 'componentsClust'
```

```
clusterProperties(obj)

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

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

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

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

Arguments

`...`	Further options passed to `plot.dendrogram` (`plot`) or `plot` (`plotSilhouettes`).
`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`.
`x, obj`	A `componentsClust` (derived) 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.
`pch, type`	Passed to `plot`.

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**.
- `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

`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.

`altered` Set to `TRUE` if the object was altered (*e.g.* filtered) after its creation.

S4 class hierarchy

- `components`
  - `componentsClust`
    - `componentsIntClust`
    - `componentsSpecClust`

Note

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

When the object is altered (*e.g.* by filtering or subsetting it), methods that need the original clustered data such as plotting methods do not work anymore and stop with an error.

References

Schollee JE, Bourgin M, von Gunten U, McArdell CS, Hollender J (2018). "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

componentsIntClust-class
Components based on clustered intensity profiles.

Description

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

Usage

```
## 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, ...)
```

Arguments

obj	A componentsIntClust object.
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 (plotInt).
index	Numeric component/cluster index.
pch, type, lty	Passed to lines.

Details

Objects from this class are generated by generateComponentsIntClust

Value

plotHeatMap returns the same as heatmap.2 or heatmaply.

Methods (by generic)

- 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.

Slots

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

S4 class hierarchy

- componentsClust
  - componentsIntClust

Note

When the object is altered (*e.g.* by filtering or subsetting it), methods that need the original clustered data such as plotting methods do not work anymore and stop with an error.

See Also

componentsClust for other relevant methods 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)

## S4 method for signature 'componentsNTSet'
plotGraph(obj, onlyLinked, set)

## S4 method for signature 'componentsNTSet'
unset(obj, set)
```

Arguments

obj	The componentsNT object to plot.
onlyLinked	If TRUE then only components with links are plotted.
set	(**sets workflow**) The name of the set.

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`

Sets workflows

The `componentsNTSet` class is applicable for sets workflows. This class is derived from `componentsNT` and therefore largely follows the same user interface.

The following methods are specifically defined for sets workflows:

- All the methods from base class `workflowStepSet`.
- `unset` Converts the object data for a specified set into a 'non-set' object (`componentsNTUnset`), which allows it to be used in 'regular' workflows. Only the components in the specified set are kept. Furthermore, the component names are restored to non-set specific names (see `generateComponents` for more details).

The following methods are changed or with new functionality:

- `plotGraph` Currently can only create graph networks from one set (specified by the `set` argument).

Note that the `componentsNTSet` class does not have a `homol` slot. Instead, the `setObjects` method can be used to access this data for a specific set.

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. and Contributors, Benoit Thieurmel and Titouan Robert (2021). visNetwork: Network Visualization using 'vis.js' Library. R package version 2.1.0. http://datastorm-open.github.io/visNetwork/

See Also

components and component-generation

---

componentsSpecClust-class *Components based on MS/MS similarity.*

Description

This class is derived from componentsClust and is used to store components from feature groups that were clustered based on their MS/MS similarities.

Details

Objects from this class are generated by generateComponentsSpecClust

S4 class hierarchy

- componentsClust
  - componentsSpecClust

Note

When the object is altered (*e.g.* by filtering or subsetting it), methods that need the original clustered data such as plotting methods do not work anymore and stop with an error.

See Also

componentsClust for other relevant methods and component-generation

---

componentsTPs-class *Components based on parent and transformation product (TP) linkage.*

Description

This class is derived from components and is used to store components that result from linking feature groups that are (predicted to be) parents with feature groups that (are predicted to be) transformation products. For more details, see generateComponentsTPs.

Usage

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

## S4 method for signature 'componentsTPs'
filter(
  obj,
  ...,
  retDirMatch = FALSE,
  minSpecSim = NULL,
  minSpecSimPrec = NULL,
  minSpecSimBoth = NULL,
  minFragMatches = NULL,
  minNLMatches = NULL,
  formulas = NULL,
  verbose = TRUE,
  negate = FALSE
)

## S4 method for signature 'componentsTPs'
plotGraph(obj, onlyLinked)
```

Arguments

`x`, `obj`	A `componentsTPs` object.
`...`, `verbose`	Further arguments passed to the base `filter` method.
`retDirMatch`	If set to `TRUE`, only keep TPs for which the retention time direction (`retDir`, see Details in componentsTPs) matches with the observed direction. TPs will never be removed if the expected/observed direction is '`0`' (*i.e.* unknown or not significantly different than the parent).
`minSpecSim`, `minSpecSimPrec`, `minSpecSimBoth`	The minimum spectral similarity of a TP compared to its parent ('`0-1`'). The `minSpecSimPrec` and `minSpecSimBoth` apply to binned data that is shifted with the `"precursor"` and `"both"` method, respectively (see MS spectral similarity parameters for more details). Set to `NULL` to ignore.
`minFragMatches`, `minNLMatches`	Minimum number of parent/TP fragment and neutral loss matches, respectively. Set to `NULL` to ignore. See the Linking parents and transformation products section in component-generation for more details.
`formulas`	A `formulas` object. The formula annotation data in this object is to verify if elemental additions/subtractions from metabolic logic reactions are possible (hence, it only works with data from `generateTPsLogic`). To verify elemental additions, only TPs with at least one candidate formula that has these elements are kept. Similarly, for elemental subtractions, any of the parent candidate formulae must contain the subtraction elements. Note that TPs are currently not filtered if either the parent or the TP has no formula annotations. Set to `NULL` to ignore.

negate	If TRUE then filters are applied in opposite manner.
onlyLinked	If TRUE then only components with links are plotted.

Value

filter returns a filtered componentsTPs object.

plotGraph returns the result of visNetwork.

Methods (by generic)

- as.data.table: Returns all component data as a data.table.
- filter: Provides various rule based filtering options to clean and prioritize TP data.
- plotGraph: Plots an interactive network graph for linked components. Components are linked with each other if one or more transformation products overlap. The graph is constructed with the **igraph** package and rendered with **visNetwork**.

S4 class hierarchy

- components
  - componentsTPs

Note

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

References

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

Almende B.V. and Contributors, Benoit Thieurmel and Titouan Robert (2021). visNetwork: Network Visualization using 'vis.js' Library. R package version 2.1.0. http://datastorm-open.github.io/visNetwork/

See Also

components for other relevant methods 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, ...)

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

## S4 method for signature 'featureGroups'
generateCompoundsMetFrag(
  fGroups,
  MSPeakLists,
  method = "CL",
  timeout = 300,
  timeoutRetries = 2,
  errorRetries = 2,
  topMost = 100,
  dbRelMzDev = 5,
  fragRelMzDev = 5,
  fragAbsMzDev = 0.002,
  adduct = NULL,
  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
)

## S4 method for signature 'featureGroupsSet'
generateCompoundsMetFrag(
```

```
  fGroups,
  MSPeakLists,
  method = "CL",
  timeout = 300,
  timeoutRetries = 2,
  errorRetries = 2,
  topMost = 100,
  dbRelMzDev = 5,
  fragRelMzDev = 5,
  fragAbsMzDev = 0.002,
  adduct = NULL,
  ...,
  setThreshold = 0,
  setThresholdAnn = 0
)

## S4 method for signature 'featureGroups'
generateCompoundsSIRIUS(
  fGroups,
  MSPeakLists,
  relMzDev = 5,
  adduct = NULL,
  elements = "CHNOP",
  profile = "qtof",
  formulaDatabase = NULL,
  fingerIDDatabase = "pubchem",
  noise = NULL,
  errorRetries = 2,
  cores = NULL,
  topMost = 100,
  topMostFormulas = 5,
  extraOptsGeneral = NULL,
  extraOptsFormula = NULL,
  verbose = TRUE,
  splitBatches = FALSE
)

## S4 method for signature 'featureGroupsSet'
generateCompoundsSIRIUS(
  fGroups,
  MSPeakLists,
  relMzDev = 5,
  adduct = NULL,
  ...,
  setThreshold = 0,
  setThresholdAnn = 0
)
```

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. For sets workflows: further arguments directly passed to the non-sets method.
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.
includeSuspectLists, onlyDefault, includeNoDB	A logical specifying whether scoring terms related to suspect lists, default scoring terms and non-database specific scoring terms should be included in the output, respectively.
method	Which method should be used for MetFrag execution: "CL" for MetFragCL and "R" for MetFragR. The former might be faster.
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 'DatabaseSearchRelativeMassDeviatio option.
fragRelMzDev	Relative mass deviation (in ppm) for fragment matching. Sets the 'FragmentPeakMatchRelativeMassDe option.
fragAbsMzDev	Absolute mass deviation (in Da) for fragment matching. Sets the 'FragmentPeakMatchAbsoluteMassDev option.
adduct	An adduct object (or something that can be converted to it with as.adduct). Examples: "[M-H]-", "[M+Na]+". If the featureGroups object has adduct annotations then these are used if adducts=NULL. (**sets workflow**) The adduct argument is not supported for sets workflows, since the adduct annotations will then always be used.
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.

`setThreshold`
: (**sets workflow**) Minimum abundance for a candidate among all sets ('`0-1`'). For instance, a value of '`1`' means that the candidate needs to be present in all the set data.

`setThresholdAnn`
: (**sets workflow**) As `setThreshold`, but only taking into account the set data that contain annotations for the feature group of the candidate.

`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.

`splitBatches` If `TRUE` then the calculations done by `SIRIUS` will be evenly split over multiple `SIRIUS` calls (which may be run in parallel depending on the set package options). If `splitBatches=FALSE` then all feature calculations are performed from a single `SIRIUS` execution, which is often the fastest if calculations are performed on a single computer.

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.

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

`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.

Value

`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.

`generateCompoundsMetFrag` returns a `compoundsMF` object.

`generateCompoundsSIRIUS` returns a `compounds` object.

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	pubchemlite
pubMedReferences	PubChemNumberPubMedReferences	pubchem
pubMedReferences	ChemSpiderNumberPubMedReferences	chemspider
pubMedReferences	NUMBER_OF_PUBMED_ARTICLES	comptox
pubMedReferences	PubMed_Count	pubchemlite

extReferenceCount	ChemSpiderNumberExternalReferences	chemspider
dataSourceCount	ChemSpiderDataSourceCount	chemspider
referenceCount	ChemSpiderReferenceCount	chemspider
RSCCount	ChemSpiderRSCCount	chemspider
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	pubchemlite
bioPathway	BioPathway	pubchemlite
drugMedicInfo	DrugMedicInfo	pubchemlite
foodRelated	FoodRelated	pubchemlite
pharmacoInfo	PharmacoInfo	pubchemlite
safetyInfo	SafetyInfo	pubchemlite
toxicityInfo	ToxicityInfo	pubchemlite
knownUse	KnownUse	pubchemlite
disorderDisease	DisorderDisease	pubchemlite
identification	Identification	pubchemlite

annoTypeCount	FPSum	pubchemlite
annoTypeCount	AnnoTypeCount	pubchemlite

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)$name`.

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

Sets workflows

With a sets workflow, annotation is first performed for each set. This is important, since the annotation algorithms typically cannot work with data from mixed ionization modes. The annotation results are then combined to generate a *sets consensus*:

- The annotation tables for each feature group from the set specific data are combined. Rows with overlapping candidates (determined by the first-block INCHIKEY) are merged.
- Set specific data (*e.g.* the ionic formula) is retained by renaming their columns with set specific names.
- The MS/MS fragment annotations (`fragInfo` column) from each set are combined.
- The scorings for each set are averaged to calculate overall scores.
- The candidates are re-ranked based on their average ranking among the set data (if a candidate is absent in a set it is assigned the poorest rank in that set).
- The coverage of each candidate among sets is calculated. Depending on the `setThreshold` and `setThresholdAnn` arguments, candidates with low abundance are removed.

Parallelization

`generateCompoundsMetFrag` and `generateCompoundsSIRIUS` uses multiprocessing to parallelize computations. Please see the parallelization section in the handbook for more details and patRoon options for configuration options.

When local database files are used with `generateCompoundsMetFrag` (*e.g.* when `database` is set to `"pubchemlite"`, `"csv"` etc.) and '`patRoon.MP.method="future"`', then the database file must be present on all the nodes. When `pubchemlite` or `comptox` is used, the location for these databases can be configured on the host with the respective package options ('`patRoon.path.MetFragPubChemLite`' and '`patRoon.path.MetFragCompTox`'). Note that these files must *also* be present on the local host computer, even if it is not participating in computations.

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#.XnjMAHLvKUk 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 `splitBatches=FALSE`. In this case, all `SIRIUS` output will be printed to the terminal (unless `verbose=FALSE` or '`patRoon.MP.method="future"`'). Furthermore, please note that only annotations to be performed for the same adduct are grouped in a single batch execution.

References

Ruttkies C, Schymanski EL, Wolf S, Hollender J, Neumann S (2016). "MetFrag relaunched: incorporating strategies beyond in silico fragmentation." *Journal of Cheminformatics*, **8**(1). doi: 10.1186/s1332101601159.

Duhrkop K, Fleischauer M, Ludwig M, Aksenov AA, Melnik AV, Meusel M, Dorrestein PC, Rousu J, Bocker S (2019). "SIRIUS 4: a rapid tool for turning tandem mass spectra into metabolite structure information." *Nature Methods*, **16**(4), 299–302. doi: 10.1038/s4159201903448.

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

Duhrkop K, Shen H, Meusel M, Rousu J, Bocker S (2015). "Searching molecular structure databases with tandem mass spectra using CSI:FingerID." *Proceedings of the National Academy of Sciences*, **112**(41), 12580–12585. doi: 10.1073/pnas.1509788112.

Bocker S, Letzel MC, Liptak Z, Pervukhin A (2008). "SIRIUS: decomposing isotope patterns for metabolite identification." *Bioinformatics*, **25**(2), 218–224. doi: 10.1093/bioinformatics/btn603.

See Also

`compounds-class`

`compounds-class` *Compound annotations class*

Description

Contains data for compound annotations for feature groups.

Usage

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

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

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

## S4 method for signature 'compounds'
filter(
  obj,
  minExplainedPeaks = NULL,
  minScore = NULL,
  minFragScore = NULL,
  minFormulaScore = NULL,
  scoreLimits = NULL,
  ...
)

## 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)

## S4 method for signature 'compounds'
plotScores(
  obj,
  index,
  groupName,
  normalizeScores = "max",
  excludeNormScores = defaultExclNormScores(obj),
  onlyUsed = TRUE
)

## S4 method for signature 'compounds'
annotatedPeakList(
  obj,
```

```
  index,
  groupName,
  MSPeakLists,
  formulas = NULL,
  onlyAnnotated = FALSE
)

## S4 method for signature 'compounds'
plotSpectrum(
  obj,
  index,
  groupName,
  MSPeakLists,
  formulas = NULL,
  plotStruct = FALSE,
  title = NULL,
  specSimParams = getDefSpecSimParams(),
  mincex = 0.9,
  xlim = NULL,
  ylim = NULL,
  maxMolSize = c(0.2, 0.4),
  molRes = c(100, 100),
  ...
)

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

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

## S4 method for signature 'compoundsSet'
delete(obj, i, j, ...)

## S4 method for signature 'compoundsSet,ANY,missing,missing'
x[i, j, ..., sets = NULL, updateConsensus = FALSE, drop = TRUE]

## S4 method for signature 'compoundsSet'
filter(obj, ..., sets = NULL, updateConsensus = FALSE, negate = FALSE)
```

```
## S4 method for signature 'compoundsSet'
plotSpectrum(
  obj,
  index,
  groupName,
  MSPeakLists,
  formulas = NULL,
  plotStruct = FALSE,
  title = NULL,
  specSimParams = getDefSpecSimParams(),
  mincex = 0.9,
  xlim = NULL,
  ylim = NULL,
  maxMolSize = c(0.2, 0.4),
  molRes = c(100, 100),
  perSet = TRUE,
  mirror = TRUE,
  ...
)

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

## S4 method for signature 'compoundsSet'
annotatedPeakList(obj, index, groupName, MSPeakLists, formulas = NULL, ...)

## S4 method for signature 'compoundsSet'
consensus(
  obj,
  ...,
  absMinAbundance = NULL,
  relMinAbundance = NULL,
  uniqueFrom = NULL,
  uniqueOuter = FALSE,
  rankWeights = 1,
  labels = NULL,
  filterSets = FALSE,
  setThreshold = 0,
  setThresholdAnn = 0
)

## S4 method for signature 'compoundsSet'
```

```
unset(obj, set)

## S4 method for signature 'compoundsConsensusSet'
unset(obj, set)
```

Arguments

obj, object, compounds, x	The compound object.
minExplainedPeaks, scoreLimits	Passed to the featureAnnotations method.
minScore, minFragScore, minFormulaScore	Minimum overall score, in-silico fragmentation score and formula score, respectively. Set to NULL to ignore. The scoreLimits argument allows for more advanced score filtering.
...	For plotSpectrum: Further arguments passed to plot. For delete: passed to the function specified as j. for filter: passed to the featureAnnotations method. For consensus: any further (and unique) compounds objects. For sets workflow methods: further arguments passed to the base compounds method.
formulas	The formulas object that should be used for scoring/annotation. For plotSpectrum and annotatedPeakList: 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. For plotStructure and getMCS: multiple indices (*i.e.* vector with length >=2) should be specified to plot/calculate the most common substructure (MCS). Alternatively, '-1' may be specified to select all candidates. For plotSpectrum: two indices can be specified to compare spectra. In this case groupName should specify values for the spectra to compare.
groupName	The name of the feature group (or feature groups when comparing spectra) to which the candidate belongs.
width, height	The dimensions (in pixels) of the raster image that should be plotted.
normalizeScores	A character that specifies how normalization of annotation 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.

`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. Currently not supported when comparing spectra.

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

`specSimParams`
: A named `list` with parameters that influence the calculation of MS spectra similarities. See the spectral similarity parameters documentation for more details.

`mincex`
: The formula annotation labels are automatically scaled. The `mincex` argument forces a minimum `cex` value for readability.

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

`maxMolSize`
: Numeric vector of size two with the maximum width/height of the candidate structure (relative to the plot size).

`molRes`
: Numeric vector of size two with the resolution of the candidate structure (in pixels).

`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 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`.

`rankWeights`
: A numeric vector with weights of to calculate 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.

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

`i, j, drop`
: Passed to the `featureAnnotations` method.

`sets`
: (**sets workflow**) A `character` with name(s) of the sets to keep (or remove if `negate=TRUE`). Note: if `updateConsensus=FALSE` then the `setCoverage` column of the annotation results is not updated.

`updateConsensus` (**sets workflow**) If `TRUE` then the annonation consensus among set results is updated. See the Sets workflows section for more details.

`negate` Passed to the `featureAnnotations` method.

`perSet, mirror` (**sets workflow**) If `perSet=TRUE` then the set specific mass peaks are annotated separately. Furthermore, if `mirror=TRUE` (and there are two sets in the object) then a mirror plot is generated.

`filterSets` (**sets workflow**) Controls how algorithms concensus abundance filters are applied. See the Sets workflows section below.

`setThreshold, setThresholdAnn` (**sets workflow**) Thresholds used to create the annotation set consensus. See `generateCompounds`.

`set` (**sets workflow**) The name of the set.

Details

`compounds` objects are obtained from compound generators. This class is derived from the `featureAnnotations` class, please see its documentation for more methods and other details.

Value

`addFormulaScoring` returns a `compounds` object updated with formula scoring.

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

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

Methods (by generic)

- `defaultExclNormScores`: Returns default scorings that are excluded from normalization.
- `show`: Show summary information for this object.
- `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. Also see the `featureAnnotations` method.
- `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.
- plotSpectrum: Plots an annotated spectrum for a given candidate compound for a feature group. Two spectra can be compared by specifying a two-sized vector for the index and groupName arguments.
- 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

setThreshold,setThresholdAnn (**sets workflow**) A copy of the equally named arguments that were passed when this object was created by generateCompounds.

origFGNames (**sets workflow**) The original (order of) names of the featureGroups object that was used to create this object.

S4 class hierarchy

- featureAnnotations
  - compounds
    - compoundsConsensus
    - compoundsMF
    - compoundsSet
      - compoundsConsensusSet
    - compoundsUnset

Source

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

Sets workflows

The compoundsSet class is applicable for sets workflows. This class is derived from compounds and therefore largely follows the same user interface.

The following methods are specifically defined for sets workflows:

- All the methods from base class workflowStepSet.
- unset Converts the object data for a specified set into a 'non-set' object (compoundsUnset), which allows it to be used in 'regular' workflows. Only the annotation results that are present in the specified set are kept (based on the set consensus, see below for implications).

The following methods are changed or with new functionality:

- `filter` and the subset operator (`[`) Can be used to select data that is only present for selected sets. Depending on the `updateConsenus`, both either operate on set consensus or original data (see below for implications).
- `annotatedPeakList` Returns a combined annotation table with all sets.
- `plotSpectrum` Is able to highlight set specific mass peaks (`perSet` and `mirror` arguments).
- `consensus` Creates the algorithm consensus based on the original annotation data (see below for implications). Then, like the sets workflow method for `generateCompounds`, a consensus is made for all sets, which can be controlled with the `setThreshold` and `setThresholdAnn` arguments. The candidate coverage among the different algorithms is calculated for each set (*e.g.* `coverage-positive` column) and for all sets (`coverage` column), which is based on the presence of a candidate in all the algorithms from all sets data. The `consensus` method for sets workflow data supports the `filterSets` argument. This controls how the algorithm consensus abundance filters (`absMinAbundance`/`relMinAbundance`) are applied: if `filterSets=TRUE` then the minimum of all `coverage` set specific columns is used to obtain the algorithm abundance. Otherwise the overall `coverage` column is used. For instance, consider a consensus object to be generated from two objects generated by different algorithms (*e.g.* `SIRIUS` and `MetFrag`), which both have a positive and negative set. Then, if a candidate occurs with both algorithms for the positive mode set, but only with the first algorithm in the negative mode set, `relMinAbundance=1` will remove the candidate if `filterSets=TRUE` (because the minimum relative algorithm abundance is '`0.5`'), while `filterSets=FALSE` will not remove the candidate (because based on all sets data the candidate occurs in both algorithms).
- `addFormulaScoring` Adds the formula scorings to the original data and re-creates the annotation set consensus (see below for implications).

Two types of annotation data are stored in a `compoundsSet` object:

1. Annotations that are produced from a consensus between set results (see `generateCompounds`).
2. The 'original' annotation data per set, prior to when the set consensus was made. This includes candidates that were filtered out because of the thresholds set by `setThreshold` and `setThresholdAnn`. However, when `filter` or subsetting (`[`) operations are performed, the original data is also updated.

In most cases the first data is used. However, in a few cases the original annotation data is used (as indicated above), for instance, to re-create the set consensus. It is important to realize that the original annotation data may have *additional* candidates, and a newly created set consensus may therefore have 'new' candidates. For instance, when the object consists of the sets `"positive"` and `"negative"` and `setThreshold=1` was used to create it, then `compounds[,sets = "positive",updateConsensus = TRUE]` may now have additional candidates, *i.e.* those that were not present in the `"negative"` set and were previously removed due to the consensus threshold filter.

Note

The values ranges in the `scoreLimits` slot, which are used for normalization of scores, are based on the *original* scorings when the compounds were generated (*prior* to employing the `topMost` filter to `generateCompounds`).

References

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

See Also

The featureAnnotations base class for more relevant methods and compound-generation.

---

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 method for calculating similarities (i.e. not dissimilarity!). 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**.

References

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

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,missing'
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	For [: 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). "MetFrag relaunched: incorporating strategies beyond in silico fragmentation." *Journal of Cheminformatics*, **8**(1). 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
)
```

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.

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.

Parallelization

convertMSFiles (except if algorithm="bruker") uses multiprocessing to parallelize computations. Please see the parallelization section in the handbook for more details and patRoon options for configuration options.

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). "OpenMS: a flexible open-source software platform for mass spectrometry data analysis." *Nature Methods*, **13**(9), 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). "A cross-platform toolkit for mass spectrometry and proteomics." *Nature Biotechnology*, **30**(10), 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,
  featQualityRange = NULL,
  groupQualityRange = NULL,
  rGroups = NULL,
  results = NULL,
  removeBlanks = FALSE,
  checkFeaturesSession = NULL,
  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 chosen 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 minimum 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.

`featQualityRange`
: Used to filter features by their peak qualities/scores (see `calculatePeakQualities`). Should be a named `list` with min/max ranges for each quality/score to be filtered (the `featureQualityNames` function can be used to obtain valid names). Example: `featQualityRange=list(ModalityScore=c(0.3,Inf),SymmetryScore=c(0.5,Inf))`. Set to `NULL` to ignore.

`groupQualityRange`
: Like `featQualityRange`, but filters on group specific or averaged qualities/scores.

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

`results`
: Only keep feature groups that have results in the object specified by `results`. Valid classes are `featureAnnotations` (*e.g.* formula/compound annotations)

and components. Can also be a list with multiple objects: in this case a feature group is kept if it has a result in *any* of the objects. Set to NULL to ignore.

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.

checkFeaturesSession — If set then features and/or feature groups are removed that were selected for removal (see check-GUI). The session files are typically generated with the checkFeatures and predictCheckFeaturesSession functions. The value of checkFeaturesSession should either by a path to the session file or TRUE, in which case the default session file name is used. If negate=TRUE then all non-selected features/feature groups are removed instead.

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. Features/feature groups selected for removal by the session specified by checkFeaturesSession.

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

## S4 method for signature 'features'
makeSet(obj, ..., adducts, labels = NULL)

## S4 method for signature 'featuresSet'
makeSet(obj, ...)

findFeaturesBruker(
  analysisInfo,
  doFMF = "auto",
  startRange = 0,
  endRange = 0,
```

```
  save = TRUE,
  close = save,
  verbose = TRUE
)

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

importFeaturesEnviMass(analysisInfo, enviProjPath)

findfeaturesKPIC2(
  analysisInfo,
  kmeans = TRUE,
  level = 1000,
  ...,
  parallel = TRUE,
  verbose = TRUE
)

importfeaturesKPIC2(picsList, analysisInfo)

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 = 1,
  maxFWHM = 30,
  traceSNRFiltering = FALSE,
  localRTRange = 10,
  localMZRange = 6.5,
  isotopeFilteringModel = "metabolites (5% RMS)",
  MZScoring13C = FALSE,
  useSmoothedInts = TRUE,
  extraOpts = NULL,
  intSearchRTWindow = 3,
  useFFMIntensities = FALSE,
  verbose = TRUE
)

findFeaturesSAFD(
```

```
  analysisInfo,
  profPath = NULL,
  mzRange = c(0, 400),
  maxNumbIter = 1000,
  maxTPeakW = 300,
  resolution = 30000,
  minMSW = 0.02,
  RThreshold = 0.75,
  minInt = 2000,
  sigIncThreshold = 5,
  S2N = 2,
  minPeakWS = 3,
  verbose = TRUE
)

findFeaturesSIRIUS(analysisInfo, 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", "sirius", "kpic2", "safd"
...	Further parameters passed to xcmsSet (findFeaturesXCMS), xcms::findChromPeaks (findFeaturesXCMS3), enviPickwrap (featurefinderEnviPick), getPIC/getPIC.kmeans (findFeaturesKPIC2) or to selected feature finding or importing algorithms (findFeatures and importFeatures). For makeSet: further features objects that should be used for the sets workflow.
verbose	If set to FALSE then no text output is shown.
type	What type of data should be imported: "xcms", "xcms3", "kpic2" or "envimass".
obj	A features object that is used (together with other objects passed to ...) to create a sets object with makeSet.
adducts	The adduct assignments to each set. Should either be a list with adduct objects or a character vector (*e.g.* "[M+H]+"). The order should follow that of the objects given to the obj and ... arguments.

`labels` The labels, or *set names*, for each set to be created. The order should follow that of the objects given to the `obj` and `...` arguments. If `NULL`, then labels are automatically generated from the polarity of the specified `adducts` argument (*e.g.* `"positive"`, `"negative"`).

`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.

`parallel` If set to `TRUE` then code is executed in parallel through the **futures** package. Please see the parallelization section in the handbook for more details.

`enviProjPath` The path of the enviMass project.

`kmeans` If `TRUE` then `getPIC.kmeans` is used to obtain PICs, otherwise it is `getPIC`.

`level` Passed to `getPIC` or `getPIC.kmeans`

`picsList` A `list` with a `pics` objects obtained with `getPIC` or `getPIC.kmeans` for each analysis.

`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_out` 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_filter` `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_ 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:com (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).

useFFMIntensities
: If TRUE then peak intensities are directly loaded from FeatureFinderMetabo output. Otherwise, intensities are loaded afterwards from the input 'mzML' files, which is potentially much slower, especially with many analyses files. However, useFFMIntensities=TRUE is still somewhat experimental, may be less accurate and requires a recent version of OpenMS (>=2.7).

profPath
: A character vector with paths to the profile MS data for each analysis (will be re-cycled if necessary). See the Using SAFD section for more details.

mzRange
: The *m/z* window to be imported (passed to the import_files_MS1 function).

maxNumbIter, maxTPeakW, resolution, minMSW, RThreshold, minInt, sigIncThreshold, S2N, minPeakWS
: Parameters directly passed to the safd_s3D function.

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 importFeaturesKPIC2.

makeSet is used to initiate a sets workflow. See the Sets workflows section for more details.

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**.

findFeaturesKPIC2 uses the KPIC2 R package to extract features.

importfeaturesKPIC2 converts features obtained with with the **KPIC2** package to a new features object.

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

findFeaturesSAFD uses SAFD to obtain features. This functionality is still experimental. Please see the Using SAFD section below for more details.

findFeaturesSIRIUS uses SIRIUS to find features. The features are collected by running the lcms-align command for every analysis.

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.

makeSet returns a featuresSet object.

Parallelization

findFeaturesOpenMS, findFeaturesSIRIUS and findFeaturesSAFD uses multiprocessing to parallelize computations. Please see the parallelization section in the handbook for more details and patRoon options for configuration options.

Note that for caching purposes, the analyses files must always exist on the local host computer, even if it is not participating in computations.

Sets workflows

The makeSet method function is used to initiate a sets workflow. This method simply combines all the features from specified features objects in a new featuresSet object. The analysis information for this object is updated with all analyses, and a set column is added to designate the set of each analysis. Note that currently, all analyses names **must** be unique across different sets. In the last step the features are *neutralized*: the specified adducts to makeSet is used calculate the neutral monoisotopic mass for each feature per set. This neutralization is important to group features afterwards (see the details on sets workflows in feature grouping).

Using SAFD

The support for SAFD is still experimental, and its interface might change in the future.

In order to use SAFD, please make sure that its julia packages are installed and you have verified that everything works, *e.g.* by running the test data.

This algorithm only supports profile and centroided MS data. If the use of profile data is desired, centroided data must still be available for other functionality of patRoon. The centroided data is specified through the 'regular' analysis info mechanism. The location to any profile data is specified through the profPath argument (NULL for no profile data). The base file names (*i.e.* the file name without path and extension) of both centroid and profile data must be the same. Furthermore, the format of the profile data must be mzXML.

Note

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

makeSet Currently does not support making sets from featuresSet objects.

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

Ji H, Zeng F, Xu Y, Lu H, Zhang Z (2017). "KPIC2: An Effective Framework for Mass Spectrometry-Based Metabolomics Using Pure Ion Chromatograms." *Analytical Chemistry*, **89**(14), 7631–7640. doi: 10.1021/acs.analchem.7b01547.

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). "OpenMS: a flexible open-source software platform for mass spectrometry data analysis." *Nature Methods*, **13**(9), 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 https://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 (2018). Extending R with C++: A Brief Introduction to Rcpp. The American Statistician. 72(1). URL https://doi.org/10.1080/00031305.2017.1375990.

Samanipour S, OBrien JW, Reid MJ, Thomas KV (2019). "Self Adjusting Algorithm for the Nontargeted Feature Detection of High Resolution Mass Spectrometry Coupled with Liquid Chromatography Profile Data." *Analytical Chemistry*, **91**(16), 10800–10807. doi: 10.1021/acs.analchem.9b02422.

Duhrkop K, Fleischauer M, Ludwig M, Aksenov AA, Melnik AV, Meusel M, Dorrestein PC, Rousu J, Bocker S (2019). "SIRIUS 4: a rapid tool for turning tandem mass spectra into metabolite structure information." *Nature Methods*, **16**(4), 299–302. doi: 10.1038/s4159201903448.

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(obj, algorithm, ..., verbose = TRUE)

## S4 method for signature 'data.frame'
groupFeatures(obj, algorithm, ..., verbose = TRUE)

importFeatureGroups(path, type, ...)

## S4 method for signature 'featuresSet'
groupFeatures(obj, algorithm, ..., verbose = TRUE)

## S4 method for signature 'featureGroups'
makeSet(
  obj,
  ...,
  groupAlgo,
  groupArgs = NULL,
  verbose = TRUE,
  adducts = NULL,
  labels = NULL
)

## S4 method for signature 'featureGroupsSet'
makeSet(obj, ...)

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

importFeatureGroupsEnviMass(path, feat, positive)

## S4 method for signature 'features'
groupFeaturesKPIC2(
```

```
  feat,
  rtalign = TRUE,
  loadRawData = TRUE,
  groupArgs = list(tolerance = c(0.005, 12)),
  alignArgs = list(),
  verbose = TRUE
)

## S4 method for signature 'featuresSet'
groupFeaturesKPIC2(
  feat,
  groupArgs = list(tolerance = c(0.005, 12)),
  verbose = TRUE
)

importFeatureGroupsKPIC2(picsSetGrouped, analysisInfo)

## S4 method for signature 'featureGroupsKPIC2'
delete(obj, ...)

## S4 method for signature 'features'
groupFeaturesOpenMS(
  feat,
  rtalign = TRUE,
  QT = FALSE,
  maxAlignRT = 30,
  maxAlignMZ = 0.005,
  maxGroupRT = 12,
  maxGroupMZ = 0.005,
  extraOptsRT = NULL,
  extraOptsGroup = NULL,
  verbose = TRUE
)

groupFeaturesSIRIUS(analysisInfo, verbose = TRUE)

## S4 method for signature 'features'
groupFeaturesXCMS(
  feat,
  rtalign = TRUE,
  loadRawData = TRUE,
  groupArgs = list(mzwid = 0.015),
  retcorArgs = list(method = "obiwarp"),
  verbose = TRUE
)

## S4 method for signature 'featuresSet'
groupFeaturesXCMS(feat, groupArgs = list(mzwid = 0.015), verbose = TRUE)
```

```
importFeatureGroupsXCMS(xs, analysisInfo)

## S4 method for signature 'featureGroupsXCMS'
delete(obj, ...)

## S4 method for signature 'features'
groupFeaturesXCMS3(
  feat,
  rtalign = TRUE,
  loadRawData = TRUE,
  groupParam = xcms::PeakDensityParam(sampleGroups = analysisInfo(feat)$group),
  preGroupParam = groupParam,
  retAlignParam = xcms::ObiwarpParam(),
  verbose = TRUE
)

## S4 method for signature 'featuresSet'
groupFeaturesXCMS3(
  feat,
  groupParam = xcms::PeakDensityParam(sampleGroups = analysisInfo(feat)$group),
  verbose = TRUE
)

importFeatureGroupsXCMS3(xdata, analysisInfo)

## S4 method for signature 'featureGroupsXCMS3'
delete(obj, ...)
```

Arguments

obj	Either a features object to be grouped, or a data.frame with analysis info to be passed to groupFeaturesSIRIUS or a featureGroups objects passed to makeSet.
algorithm	A character that specifies the algorithm to be used: either "openms", "xcms", "xcms3" or "kpic2" (features method), or "sirius" (data.frame method).
...	Any parameters to be passed to the selected grouping/importing algorithm. For makeSet: further featureGroups objects that should be used for the sets workflow.
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 importFeatureGroupsBrukerTAS 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).

groupAlgo groupAlgo The name of the feature grouping algorithm. See the algorithm argument description.

groupArgs A list with arguments directly passed to groupFeatures (can be named). Example: groupArgs=list(maxAlignMZ=0.002).

adducts The adduct assignments to each set. Should either be a list with adduct objects or a character vector (*e.g.* "[M+H]+"). The order should follow that of the objects given to the obj and ... arguments. if NULL then adduct annotations are used.

labels The labels, or *set names*, for each set to be created. The order should follow that of the objects given to the obj and ... arguments. If NULL, then labels are automatically generated from the polarity of the specified adducts argument (*e.g.* "positive", "negative").

feat The features to be grouped. importFeatureGroupsBrukerPA and importFeatureGroupsEnviMass only support features generated by findFeaturesBruker and importFeaturesEnviMass, respectively.

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.

loadRawData Set to TRUE if analyses are available as mzXML or mzML files. Otherwise MS data is not loaded, and some dummy data (*e.g.* file paths) is used in the returned object.

alignArgs named character vector that may contain extra parameters to be used by KPIC::PICset.align.

picsSetGrouped A grouped PIC set object (*e.g.* as returned by KPIC::PICset.group).

analysisInfo A data.frame with analysis info.

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_diff 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("-algorithm:di (corresponds to setting maxGroupRT=12). Set to NULL to ignore.

`retcorArgs` named `character vector` that may contain extra parameters to be used by `xcms::retcor`.

`xs` An `xcmsSet` object.

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

`preGroupParam` grouping parameters applied when features are grouped *prior* to alignment (only with peak groups alignment).

`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.

The `data.frame` method for `groupFeatures` is a special case that currently only supports the `"sirius"` algorithm.

`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`.

`makeSet` is used to initiate a sets workflow. See the Sets workflows section for more details.

`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).

groupFeaturesKPIC2 uses the the KPIC2 R package for grouping of features. Grouping of features and alignment of their retention times are performed with the KPIC::PICset.group and KPIC::PICset.align functions, respectively.

importFeatureGroupsKPIC2 imports grouped features from an **KPIC** object.

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.

groupFeaturesSIRIUS uses SIRIUS to find *and* group features. This is done by running the lcms-align command on every analyses at once. Note that grouping feature data from other algorithms than SIRIUS are therefore not supported.

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.

makeSet and the featuresSet method of groupFeatures return a featureGroupsSet object.

Sets workflows

With a sets workflow the features are first neutralized, *i.e.* their *m/z* values is replaced by their neutral monoisotopic mass. The latter is calculated from adduct annotations, specified either directly, or already present in the input objects. The features are then grouped by any of the groupFeatures algorithms. The neutralization step ensures that features measured with *e.g.* different ionization polarities can be grouped since their neutral mass will be the same. The sets workflow is initiated in either of the following two ways:

1. The groupFeatures method can be called with a featuresSet object (generated with the features method of makeSet).
2. By using the makeSet method for featureGroups. This uses the features method internally to first create a featuresSet object from all the feature data inside the input featureGroups objects. The (neutralized) features are then (re-)grouped with groupFeatures.

The advantage of the second method is that it preserves adduct annotations present in the input featureGroups objects (*e.g.* as set by selectIons or adducts<-). Furthermore, the second approach allows more advanced workflows where the input featureGroups are first pre-treated with *e.g.* filter before the sets object is made. On the other hand, the first approach is easier as it doesn't

require intermediate feature grouping steps, and is often sufficient as adduct annotations can be made afterwards with `selectIons/adducts<-`, and most `filter` operations do not need to be done per individual set.

With both approaches the resulting `featureGroupsSet` object will contain adduct annotations, either based on those already present from input `featureGroups` objects or as specified by the `adducts` argument to `makeSet`.

NOTE the `loadRawData` and arguments related to retention time alignment (*e.g.* `rtalign`, `retcorArgs`) are currently not supported for sets workflows when using algorithms from **KPIC2** or **XCMS**.

Note

`makeSet` Currently does not support making sets from `featureGroupsSet` objects.

References

Ji H, Zeng F, Xu Y, Lu H, Zhang Z (2017). "KPIC2: An Effective Framework for Mass Spectrometry-Based Metabolomics Using Pure Ion Chromatograms." *Analytical Chemistry*, **89**(14), 7631–7640. doi: 10.1021/acs.analchem.7b01547.

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). "OpenMS: a flexible open-source software platform for mass spectrometry data analysis." *Nature Methods*, **13**(9), 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 https://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 (2018). Extending R with C++: A Brief Introduction to Rcpp. The American Statistician. 72(1). URL https://doi.org/10.1080/00031305.2017.1375990.

Duhrkop K, Fleischauer M, Ludwig M, Aksenov AA, Melnik AV, Meusel M, Dorrestein PC, Rousu J, Bocker S (2019). "SIRIUS 4: a rapid tool for turning tandem mass spectra into metabolite structure information." *Nature Methods*, **16**(4), 299–302. doi: 10.1038/s4159201903448.

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,
  parallel = TRUE
)

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,
  parallel = TRUE
)

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 getDefFeaturesOptParamRange 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.
parallel	If set to TRUE then code is executed in parallel through the **futures** package. Please see the parallelization section in the handbook for more details.
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).
- Parallelization is performed via the **future** package instead of **BiocParallel**. If this is enabled (`parallel=TRUE`) then any parallelization supported by the feature finding or grouping algorithm is disabled.

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 passed to the 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.

Similary, for KPIC2, the following parameters should be split:

- the `width` parameter (feature optimization) is optimized by specifying the `min_width` and `max_width` parameters.
- the `tolerance` and `weight` parameters (feature grouping optimization) are optimized by setting `mz_tolerance`/`rt_tolerance` and `mz_weight`/`rt_weight` parameters, 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). "IPO: a tool for automated optimization of XCMS parameters." *BMC Bioinformatics*, **16**(1). 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)
```

`featureAnnotations-class`

Base feature annotations class

Description

Holds information for all feature group annotations.

Usage

```
## S4 method for signature 'featureAnnotations'
annotations(obj)

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

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

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

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

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

## S4 method for signature 'featureAnnotations'
as.data.table(
  x,
  fGroups = NULL,
  fragments = FALSE,
  countElements = NULL,
  countFragElements = NULL,
  OM = FALSE,
  normalizeScores = "none",
  excludeNormScores = defaultExclNormScores(x)
)

## S4 method for signature 'featureAnnotations'
delete(obj, i = NULL, j = NULL, ...)

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

```
  negate = FALSE
)

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

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

Arguments

obj, x	featureAnnotations object to be accessed
i, j	For [/[[: 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 delete: The data to remove from. i are the feature groups as numeric index, logical or character, j the candidates as numeric indices (rows). If either is NULL then data for all is removed. j may also be a function: it will be called for each feature group, with the annotation table (a data.table) as first argument, the feature group name as second argument, and any other arguments passed as ... to delete. The return value of this function specifies the candidate indices (rows) to be removed (specified as an integer or logical vector).
...	For the "[" operator: ignored. For delete: passed to the function specified as j. Others: Any further (and unique) featureAnnotations 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).
fragments	If TRUE then information on annotated fragments will be included. Automatically set to TRUE if countFragElements is set.
countElements, countFragElements	A character vector with elements that should be counted for each candidate's formula. 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 (ion_formula column in fragInfo data) 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.

normalizeScores — A character that specifies how normalization of annotation 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 explained peaks. Set to NULL to ignore.

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 name column of the table returned by formulaScorings$name and compoundScorings()$name. For instance, scoreLimits=list(numberPatents=c(10,Inf)) specifies that numberPatents should be at least '10'. Note that a result without a specified scoring is never removed. If a score term exists multiple times, *i.e.* due to a consensus, then a candidate is kept if at least one of the terms falls within the range. 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 sulfur 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.

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.

Details

This class stores annotation data for feature groups, such as molecular formulae, SMILES identifiers, compound names etc. The class of objects that are generated by formula and compound annotation (generateFormulas and generateCompounds) are based on this class.

Value

as.data.table returns a data.table.

delete returns the object for which the specified data was removed.

filter returns a filtered featureAnnotations 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).

Methods (by generic)

- annotations: Accessor for the groupAnnotations slot.
- 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 candidates.
- [: Subset on feature groups.
- [[: Extracts annotation data for a feature group.
- $: Extracts annotation data for a feature group.
- as.data.table: Generates a table with all annotation data for each feature group and other information such as element counts.

- `delete`: Completely deletes specified annotations.
- `filter`: Provides rule based filtering for feature group annotations. Useful to eliminate unlikely candidates and speed up further processing.
- `plotVenn`: plots a Venn diagram (using **VennDiagram**) outlining unique and shared candidates of up to five different `featureAnnotations` objects.
- `plotUpSet`: plots an UpSet diagram (using the `upset` function) outlining unique and shared formula candidates between different `featureAnnotations` objects.

Slots

`groupAnnotations` A `list` with for each annotated feature group a `data.table` with annotation data. Use the `annotations` method for access.

`scoreTypes` A `character` with all the score types present in this object.

`scoreRanges` The minimum and maximum score values of all candidates for each feature group. 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).

S4 class hierarchy

- `workflowStep`
  - `featureAnnotations`
    - `formulas`
      - `formulasConsensus`
      - `formulasSet`
      - `formulasUnset`
    - `compounds`
      - `compoundsConsensus`
      - `compoundsMF`
      - `compoundsSet`
      - `compoundsUnset`

References

Koch BP, Dittmar T (2015). "From mass to structure: an aromaticity index for high-resolution mass data of natural organic matter." *Rapid Communications in Mass Spectrometry*, **30**(1), 250–250. doi: 10.1002/rcm.7433.

Abdulla HA, Sleighter RL, Hatcher PG (2013). "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**(8), 3895–3902. doi: 10.1021/ac303221j.

Koch BP, Dittmar T (2006). "From mass to structure: an aromaticity index for high-resolution mass data of natural organic matter." *Rapid Communications in Mass Spectrometry*, **20**(5), 926–932. doi: 10.1002/rcm.2386.

Kujawinski EB, Behn MD (2006). "Automated Analysis of Electrospray Ionization Fourier Transform Ion Cyclotron Resonance Mass Spectra of Natural Organic Matter." *Analytical Chemistry*, **78**(13), 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**(18), 2938-2940. doi: 10.1093/bioinformatics/btx364, http://dx.doi.org/10.1093/bioinformatics/btx364.

Lex A, Gehlenborg N, Strobelt H, Vuillemot R, Pfister H (2014). "UpSet: Visualization of Intersecting Sets." *IEEE Transactions on Visualization and Computer Graphics*, **20**(12), 1983–1992. doi: 10.1109/tvcg.2014.2346248.

See Also

formulas-class and compounds-class

The derived formulas and compounds classes.

---

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'
groupTable(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'
groupQualities(fGroups)

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

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

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

## S4 replacement method for signature 'featureGroups'
adducts(obj) <- value

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

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

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

## S4 method for signature 'featureGroups'
delete(obj, i = NULL, j = NULL, ...)

## S4 method for signature 'featureGroups'
```

```
export(obj, type, out)

getFCParams(rGroups, ...)

## S4 method for signature 'featureGroups'
as.data.table(
  x,
  average = FALSE,
  areas = FALSE,
  features = FALSE,
  qualities = FALSE,
  regression = FALSE,
  averageFunc = mean,
  normFunc = NULL,
  FCParams = NULL
)

## S4 method for signature 'featureGroups,missing'
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,
  normFunc = NULL,
  xnames = TRUE,
  showLegend = 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'
plotChroms(
  obj,
  rtWindow = 30,
  mzExpWindow = 0.001,
  retMin = FALSE,
  topMost = NULL,
  topMostByRGroup = FALSE,
  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'
plotVolcano(
  obj,
  FCParams,
  showLegend = TRUE,
  averageFunc = mean,
  col = NULL,
  pch = 19,
  ...
)

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

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

## S4 method for signature 'featureGroups'
calculatePeakQualities(
  obj,
  weights,
  flatnessFactor,
  avgFunc = mean,
  parallel = TRUE
)

## S4 method for signature 'featureGroups'
selectIons(
  fGroups,
  components,
  prefAdduct,
  onlyMonoIso = TRUE,
  chargeMismatch = "adduct"
)

## S4 method for signature 'featureGroupsSet'
sets(obj)

## S4 method for signature 'featureGroupsSet'
adducts(obj, set, ...)

## S4 replacement method for signature 'featureGroupsSet'
adducts(obj, set, reGroup = TRUE) <- value

## S4 method for signature 'featureGroupsSet'
delete(obj, i = NULL, j = NULL, ...)

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

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

## S4 method for signature 'featureGroupsSet,ANY,ANY,missing'
x[i, j, ..., rGroups, sets = NULL, drop = TRUE]

## S4 method for signature 'featureGroupsSet'
export(obj, type, out, set)

## S4 method for signature 'featureGroupsSet'
as.data.table(
  x,
```

```
  average = FALSE,
  areas = FALSE,
  features = FALSE,
  qualities = FALSE,
  regression = FALSE,
  averageFunc = mean,
  normFunc = NULL,
  FCParams = NULL
)

## S4 method for signature 'featureGroupsSet'
filter(
  obj,
  ...,
  negate = FALSE,
  sets = NULL,
  absMinSets = NULL,
  relMinSets = NULL
)

## S4 method for signature 'featureGroupsSet'
plotInt(
  obj,
  average = FALSE,
  normFunc = NULL,
  xnames = !sets,
  showLegend = sets,
  pch = 20,
  type = "b",
  lty = 3,
  col = NULL,
  ...,
  sets = FALSE
)

## S4 method for signature 'featureGroupsSet'
plotVenn(obj, which = NULL, ..., sets = FALSE)

## S4 method for signature 'featureGroupsSet'
unique(x, which, ..., sets = FALSE)

## S4 method for signature 'featureGroupsSet'
overlap(fGroups, which, exclusive, sets = FALSE)

## S4 method for signature 'featureGroupsSet'
selectIons(fGroups, components, prefAdduct, ...)

## S4 method for signature 'featureGroupsSet'
```

```
unset(obj, set)
```

Arguments

areas	If set to TRUE then areas are considered instead of peak intensities. For as.data.table: ignored if features=TRUE, as areas of features are always reported.
fGroups, obj, x, object	featureGroups object to be accessed.
value	For adducts<-: A character with adduct annotations assigned to each feature group. The length should equal the number of feature groups. Can be named with feature group names to customize the assignment order.
i, j	For [/[[: 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. For delete: The data to remove from. i are the analyses as numeric index, logical or character, j the feature groups as numeric index, logical or character. If either is NULL then data for all is removed. j may also be a function: it will be called for each feature group, with a vector of the group intensities as first argument, the group name as second argument, and any other arguments passed as ... to delete. The return value of this function specifies the analyses of the features in the group to be removed (same format as i).
...	For the "[" operator: ignored. For delete: passed to the function specified as j. For getFCParams: optional named arguments that override defaults. For sets workflow methods: further arguments passed to the base featureGroups method. Otherwise passed to plot (plot and plotChroms), lines (plotInt), **VennDiagram** plotting functions (plotVenn), chordDiagram (plotChord) or upset (plotUpSet).
rGroups	For [: An optional character vector: if specified only keep results for the given replicate groups (equivalent to the rGroups argument to filter). For getFCParams: a character vector with the names of the two replicate groups to be compared.
results	Optional argument. If specified only feature groups with results in the specified object are kept. The class of results should be featureAnnotations or components. Multiple objects can be specified in a list: in this case a feature group is kept if it has a result in *any* of the objects (equivalent to the results argument to filter).
drop	ignored.

`name`	The feature group name (partially matched).
`out`	The destination file for the exported data.
`average`	If `TRUE` then data within replicate groups are averaged. For `as.data.table`: if `features=TRUE` other feature properties are also averaged.
`features`	If `TRUE` then feature specific data will be added. If `average=TRUE` this data will be averaged for each feature group.
`qualities`	Adds feature (group) qualities (`qualities="quality"`), scores (`qualities="score"`) or both (`qualities="both"`), if this data is available (*i.e.* from `calculatePeakQualities`). If `qualities=FALSE` then nothing is reported.
`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).
`averageFunc`	Function used for averaging. for `as.data.table`: only used when `average=TRUE` or `FCParams != NULL`.
`normFunc`	Function that should be used for normalization of data. The function is called for all intensities/areas of a feature group and these quantities are divided by the result of the function call. For example, when `max` is used normalized intensities will be between zero and one. If all quantities are zero then the function will not be called. Set to `NULL` to perform no normalization.
`FCParams`	A parameter list to calculate Fold change data (see `getFCParams` and the Fold change calculation section). For `as.data.table`: set to `NULL` to not perform FC calculations.
`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`	Plot a legend if `TRUE`.
`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.
`xnames`	Plot analysis (or replicate group if `average=TRUE`) names on the x axis.
`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.

mzExpWindow
: In case the *m/z* window to plot an EIC for a particular analysis is not known (*i.e.* no feature was detected of the feature group to be plot and onlyPresent=FALSE) then the EIC *m/z* range is estimated from the range for the complete feature group and expanded by the offset defined by mzExpWindow.

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

topMostByRGroup
: If set to TRUE and topMost is set: only plot EICs for the top most features in each replicate group. For instance, when topMost=1 and topMostByRGroup=TRUE, then EICs will be plotted for the most intense feature of each replicate group.

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.

  For plotVenn: alternatively a named list containing elements of character vectors with replicate groups to compare. For instance, which=list(infl = c("influent-A","influent-B"),effl = c("effluent-A","effluent-B")), will compare the features in replicate groups '"influent-A/B"' against those in '"effluent-A/B"'. The names of the list are used for labelling in the plot, and will be made automatically if not specified.

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.

`weights` A named `numeric` vector that defines the weight for each score to calculate the totalScore. The names of the vector follow the score names. Unspecified weights are defaulted to '1'. Example: `weights=c(ApexBoundaryRatioScore=0.5,GaussianSimilari`

`flatnessFactor` Passed to **MetaClean** as the `flatness.factor` argument to `calculateJaggedness` and `calculateModality`.

`avgFunc` The function used to average the peak qualities and scores for each feature group.

`parallel` If set to `TRUE` then code is executed in parallel through the **futures** package. Please see the parallelization section in the handbook for more details.

`components` The `components` object that was generated for the given `featureGroups` object. Obviously, the components must be created with algorithms that support adduct/isotope annotations, such as those from **RAMClustR** and **cliqueMS**.

`prefAdduct` The 'preferred adduct' (see method description). This is often `"[M+H]+"` or `"[M-H]-"`.

`onlyMonoIso` Set to `TRUE` to only keep feature groups that were annotated as monoisotopic. Feature groups are never removed by this setting if no isotope annotations are available.

`chargeMismatch` Specifies how to deal with a mismatch in charge between adduct and isotope annotations. Valid values are: `"adduct"` (ignore isotope annotation), `"isotope"` (ignore adduct annotation), `"none"` (ignore both annotations) and `"ignore"` (don't check for charge mismatches). *Important*: when `OpenMS` is used to find features, it already removes any detected non-monoisotopic features by default. Hence, in such case setting `chargeMismatch="adduct"` is more appropriate.

`set` (**sets workflow**) The name of the set.

`reGroup` Set to `TRUE` to re-group the features after the adduct annotations are changed. See the Sets workflow section for more details.

`sets` (**sets workflow**) For `[` and `filter`: a `character` with name(s) of the sets to keep (or remove if `negate=TRUE`).

For `plotInt`: if `TRUE` then feature intensities are plot per set (order follows the analysis information).

For `plotVenn`, `overlap` and `unique`: If `TRUE` then the `which` argument changes its meaning and is used to specify the names of the sets to be compared.

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

`absMinSets, relMinSets`

(**sets workflow**) Feature groups are only kept when they contain data for at least this (absolute or relative) amount of sets. Set to `NULL` to ignore.

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.

`getFCParams` creates a parameter `list` to calculate Fold changes (see the Fold change calculation section).

Value

`delete` returns the object for which the specified data was removed.

`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`).

`calculatePeakQualities` returns a modified object amended with peak qualities and scores.

`selectIons` returns a `featureGroups` object with only the selected feature groups and amended with adduct annotations.

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.
- `groupTable`: 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.
- `groupQualities`: Accessor for `groupQualities` slot.
- `groupScores`: Accessor for `groupScores` slot.
- `annotations`: Accessor for `annotations` slot.

- `adducts`: Returns a named `character` with adduct annotations assigned to each feature group (if available).
- `adducts<-`: Sets adduct annotations for feature groups.
- `[`: Subset on analyses/feature groups.
- `[[`: Extract intensity values.
- `$`: Extract intensity values for a feature group.
- `delete`: Completely deletes specified feature groups.
- `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'.
- `plotChroms`: 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.
- `plotVolcano`: Plots Fold change data in a 'Volcano plot'.
- `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).
- `calculatePeakQualities`: Calculates peak and group qualities for all features and feature groups. The peak qualities (and scores) are calculated with the features method of this function, and subsequently averaged per feature group. Then, **MetaClean** is used to calculate the Elution Shift and Retention Time Consistency group quality metrics (see the **MetaClean** publication cited below for more details). Similarly to the `features` method, these metrics are scored by normalizing qualities among all groups and scaling them from '`0`' (worst) to '`1`' (best). The totalScore for each group is then calculated as the weighted sum from all feature (group) scores. The `getMCTrainData` and `predictCheckFeaturesSession` functions can be used to train and apply Pass/Fail ML models from **MetaClean**.
- `selectIons`: uses componentization results to select feature groups with preferred adduct ion and/or isotope annotation. Typically, this means that only feature groups are kept if they are

(de-)protonated adducts and are monoisotopic. The adduct annotation assignments for the selected feature groups are copied from the components to the annotations slot. If the adduct for a feature group is unknown, its annotation is defaulted to the 'preferred' adduct, and hence, the feature group will never be removed. Furthermore, if a component does not contain an annotation with the preferred adduct, the most intense feature group is selected instead. Similarly, if no isotope annotation is available, the feature group is assumed to be monoisotopic and thus not removed. An important advantage of selectIons is that it may considerably simplify your dataset. Furthermore, the adduct assignments allow formula/compound annotation steps later in the workflow to improve their annotation accuracy. On the other hand, it is important the componentization results are reliable. Hence, it is highly recommended that, prior to calling selectIons, the settings to generateComponents are optimized and its results are reviewed with checkComponents. Finally, the adducts<- method can be used to manually correct adduct assignments afterwards if necessary.

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).

groupQualities,groupScores A data.table with qualities/scores for each feature group (see the calculatePeakQualities method).

annotations A data.table with adduct annotations for each group (see the selectIons method).

groupAlgo,groupArgs,groupVerbose (**sets workflow**) Grouping parameters that were used when this object was created. Used by adducts<- and selectIons when these methods perform a re-grouping of features.

annotations (**sets workflow**) As the featureGroups slot, but contains the annotation data per set.

Fold change calculation

the as.data.table and plotVolcano methods can be used to calculate/plot (log2) Fold changes (FC) between two replicate groups to easily identify significant changes. The calculation process is configured through a pramater list, which can be constructed with the getFCParams function. The parameter list has the following entries:

- rGroups the name of the two replicate groups to compare (taken from the rGroups argument to getFCParams).
- thresholdFC: the threshold log FC for a feature group to be classified as increasing/decreasing.
- thresholdPV: the threshold log P for a feature group to be significantly different.

- `zeroMethod`,`zeroValue`: how to handle zero values when calculating the FC: `add` adds an offset to zero values, `"fixed"` sets zero values to a fixed number and `"omit"` removes zero data. The number that is added/set by the former two options is defined by `zeroValue`.
- `PVTestFunc`: a function that is used to calculate P values (usually using `t.test`).
- `PVAdjFunc`: a function that is used to adjust P values (usually using `p.adjust`)

S4 class hierarchy

- `workflowStep`
  - `featureGroups`
    - `featureGroupsSet`
      - `featureGroupsScreeningSet`
    - `featureGroupsUnset`
    - `featureGroupsScreening`
      - `featureGroupsSetScreeningUnset`
    - `featureGroupsBruker`
    - `featureGroupsConsensus`
    - `featureGroupsEnviMass`
    - `featureGroupsKPIC2`
    - `featureGroupsOpenMS`
    - `featureGroupsSIRIUS`
    - `featureGroupsBrukerTASQ`
    - `featureGroupsXCMS`
    - `featureGroupsXCMS3`

Sets workflows

The `featureGroupsSet` class is applicable for sets workflows. This class is derived from `featureGroups` and therefore largely follows the same user interface.

The following methods are specifically defined for sets workflows:

- `sets` Returns the set names for this object.
- `unset` Converts the object data for a specified set into a 'non-set' object (`featureGroupsUnset`), which allows it to be used in 'regular' workflows. The adduct annotations for the selected set are used to convert all feature (group) masses to ionic *m/z* values. The annotations persist in the converted object.

The following methods are changed or with new functionality:

- `adducts`, `adducts<-` require the `set` argument. The order of the data that is returned/changed follows that of the `annotations` slot. Furthermore, `adducts<-` will perform a re-grouping of features when its `reGroup` parameter is set to `TRUE`. The implications for this are discussed below.
- `filter` and the subset operator (`[`) have specific arguments to choose/filter by (feature presence in) sets. See the argument descriptions.
- `as.data.table`: normalization of intensities is performed per set.

- `export` Only allows to export data from one set. The `unset` method is used prior to exporting the data.
- `overlap`, `unique`, `plotVenn`, `plotInt` allow to handle data per set. See the `sets` argument description.
- `selectIons` Will perform a re-grouping of features. The implications of this are discussed below.

A re-grouping of features occurs if `selectIons` is called or `adducts<-` is used with `reGroup=TRUE`. Afterwards, it is very likely that feature group names are changed. Since data generated later in the workflow (*e.g.* annotation steps) rely on feature group names, these objects are **not valid** anymore, and **must** be re-generated.

Author(s)

The code to calculate and plot Fold change data was created by Bas van de Velde.

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**(18), 2938-2940. doi: 10.1093/bioinformatics/btx364, http://dx.doi.org/10.1093/bioinformatics/btx364.

Lex A, Gehlenborg N, Strobelt H, Vuillemot R, Pfister H (2014). "UpSet: Visualization of Intersecting Sets." *IEEE Transactions on Visualization and Computer Graphics*, **20**(12), 1983–1992. doi: 10.1109/tvcg.2014.2346248.

Chetnik K, Petrick L, Pandey G (2020). "MetaClean: a machine learning-based classifier for reduced false positive peak detection in untargeted LC-MS metabolomics data." *Metabolomics*, **16**(11). doi: 10.1007/s11306020017383.

---

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,missing'
plot(x, retMin = FALSE, ...)

## 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
)

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

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

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"`, `xcms3`, `kpic2` or `"openms"`.
`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. Not yet supported for sets workflows.

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`	For `[`/`[[`: 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.

`featureGroupsScreening-class` *Class for suspect screened feature groups.*

Description

This class derives from `featureGroups` and adds suspect screening information.

Usage

```
## S4 method for signature 'featureGroupsScreening'
screenInfo(obj)

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

## S4 method for signature 'featureGroupsScreening,ANY,ANY,missing'
x[i, j, ..., rGroups, suspects = NULL, drop = TRUE]

## S4 method for signature 'featureGroupsScreening'
delete(obj, i = NULL, j = NULL, ...)

## S4 method for signature 'featureGroupsScreening'
as.data.table(x, ..., collapseSuspects = ",", onlyHits = FALSE)

## S4 method for signature 'featureGroupsScreening'
annotateSuspects(
  fGroups,
  MSPeakLists,
  formulas,
  compounds,
  absMzDev = 0.005,
  relMinMSMSIntensity = 0.05,
```

```
  simMSMSMethod = "cosine",
  checkFragments = c("mz", "formula", "compound"),
  formulasNormalizeScores = "max",
  compoundsNormalizeScores = "max",
  IDFile = system.file("misc", "IDLevelRules.yml", package = "patRoon"),
  logPath = file.path("log", "ident")
)

## S4 method for signature 'featureGroupsScreening'
filter(
  obj,
  ...,
  onlyHits = NULL,
  selectHitsBy = NULL,
  selectBestFGroups = FALSE,
  maxLevel = NULL,
  maxFormRank = NULL,
  maxCompRank = NULL,
  minAnnSimForm = NULL,
  minAnnSimComp = NULL,
  minAnnSimBoth = NULL,
  absMinFragMatches = NULL,
  relMinFragMatches = NULL,
  negate = FALSE
)

## S4 method for signature 'featureGroupsScreeningSet'
screenInfo(obj)

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

## S4 method for signature 'featureGroupsScreeningSet,ANY,ANY,missing'
x[i, j, ..., rGroups, suspects = NULL, sets = NULL, drop = TRUE]

## S4 method for signature 'featureGroupsScreeningSet'
delete(obj, i = NULL, j = NULL, ...)

## S4 method for signature 'featureGroupsScreeningSet'
as.data.table(x, ..., collapseSuspects = ",", onlyHits = FALSE)

## S4 method for signature 'featureGroupsScreeningSet'
annotateSuspects(
  fGroups,
  MSPeakLists = NULL,
  formulas = NULL,
  compounds = NULL,
  ...
```

```
)

## S4 method for signature 'featureGroupsScreeningSet'
filter(
  obj,
  ...,
  onlyHits = NULL,
  selectHitsBy = NULL,
  selectBestFGroups = FALSE,
  maxLevel = NULL,
  maxFormRank = NULL,
  maxCompRank = NULL,
  minAnnSimForm = NULL,
  minAnnSimComp = NULL,
  minAnnSimBoth = NULL,
  absMinFragMatches = NULL,
  relMinFragMatches = NULL,
  negate = FALSE
)

## S4 method for signature 'featureGroupsScreeningSet'
unset(obj, set)
```

Arguments

`obj`, `object`, `x`, `fGroups`
: The `featureGroupsScreening` object.

`i`, `j`, `rGroups`
: Used for subsetting data analyses, feature groups and replicate groups, see `featureGroups`.

`...`
: Further arguments passed to the base method.

`suspects`
: An optional `character` vector with suspect names. If specified, only `featureGroups` will be kept that are assigned to these suspects.

`drop`
: Ignored.

`collapseSuspects`
: If a `character` then any suspects that were matched to the same feature group are collapsed to a single row and suspect names are separated by the value of `collapseSuspects`. If `NULL` then no collapsing occurs, and each suspect match is reported on a single row. Note that some columns will not be reported when collapsing is enabled.

`onlyHits`
: For `as.data.table`: if `TRUE` then only feature groups with suspect hits are reported.

  For `filter`

  - if `negate=FALSE` and `onlyHits=TRUE` then all feature groups without suspect hits will be removed. Otherwise nothing will be done.
  - if `negate=TRUE` then `onlyHits=TRUE` will select feature groups without suspect hits, `onlyHits=FALSE` will only retain feature groups with suspect matches and this filter is ignored if `onlyHits=NULL`.

`MSPeakLists, formulas, compounds`
: Annotation data (`MSPeakLists`, `formulas` and `compounds`) obtained for this `featureGroupsScreening` object. All arguments can be `NULL` to exclude it from the annotation.

`absMzDev`
: Maximum absolute *m/z* deviation.

`relMinMSMSIntensity`
: Minimum relative intensity ('`0-1`') threshold applied when calculating annotation similarities.

`simMSMSMethod`
: Either `"cosine"` or `"jaccard"`: used to compare MS/MS peak lists for annotation similarity calculation.

`checkFragments`
: Which type(s) of MS/MS fragments from workflow data should be checked to evaluate the number of suspect fragment matches (*i.e.* from the `fragments_mz/fragments_formula` columns in the suspect list). Valid values are: `"mz"`, `"formula"`, `"compounds"`. The former uses *m/z* values in the specified `MSPeakLists` object, whereas the others use the formulae that were annotated to MS/MS peaks in the given `formulas` or `compounds` objects. Multiple values are possible: in this case the maximum number of fragment matches will be reported.

`compoundsNormalizeScores, formulasNormalizeScores`
: A `character` that specifies how normalization of annotation scorings occurs. Either
`"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`).

`IDFile`
: A file path to a YAML file with rules used for estimation of identification levels. See the Suspect annotation section for more details. If not specified then a default rules file will be used.

`logPath`
: A directory path to store logging information. If `NULL` then logging is disabled.

`selectHitsBy`
: Should be `"intensity"` or `"level"`. For cases where the same suspect is matched to multiple feature groups, only the suspect to the feature group with highest mean intensity (`selectHitsBy="intensity"`) or best identification level (`selectHitsBy="level"`) is kept. In case of ties only the first hit is kept. Set to `NULL` to ignore this filter. If `negate=TRUE` then only those hits with lowest mean intensity/poorest identification level are kept.

`selectBestFGroups`
: If `TRUE` then for any cases where a single feature group is matched to several suspects only the suspect assigned to the feature group with best identification score is kept. In case of ties only the first is kept.

`maxLevel, maxFormRank, maxCompRank, minAnnSimForm, minAnnSimComp, minAnnSimBoth`
: Filter suspects by maximum identification level (*e.g.* `"3a"`), formula/compound rank or with minimum formula/compound/combined annotation similarity. Set to `NULL` to ignore.

`absMinFragMatches, relMinFragMatches`
: Only retain suspects with this minimum number MS/MS matches with the fragments specified in the suspect list (*i.e.* `fragments_mz/fragments_formula`).

	relMinFragMatches sets the minimum that is relative ('0-1') to the maximum number of MS/MS fragments specified in the fragments_* columns of the suspect list. Set to NULL to ignore.
negate	If set to TRUE then filtering operations are performed in opposite manner.
sets	(**sets workflow**) A character with name(s) of the sets to keep (or remove if negate=TRUE).
set	(**sets workflow**) The name of the set.

Value

annotateSuspects returns a featureGroupsScreening object, which is a featureGroups object amended with annotation data.

filter returns a filtered featureGroupsScreening object.

Methods (by generic)

- screenInfo: Returns a table with screening information (see screenInfo slot).
- show: Shows summary information for this object.
- [: Subset on analyses, feature groups and/or suspects.
- as.data.table: Obtain a summary table (a data.table) with retention, *m/z*, intensity and optionally other feature data. Furthermore, the output table will be merged with information from screenInfo, such as suspect names and other properties and annotation data.
- annotateSuspects: Incorporates annotation data obtained during the workflow to annotate suspects with matched known MS/MS fragments, formula/candidate ranks and automatic estimation of identification levels. See the Suspect annotation section for more details. The estimation of identification levels for each suspect is logged in the log/ident directory.
- filter: Performs rule based filtering. This method builds on the comprehensive filter functionality from the base filter,featureGroups-method. It adds several filters to select *e.g.* the best ranked suspects or those with a minimum estimated identification level. **NOTE**: most filters *only* affect suspect hits, not feature groups. Set onlyHits=TRUE to subsequently remove any feature groups that lost any suspect matches due to other filter steps.

Slots

screenInfo A (data.table) with results from suspect screening. This table will be amended with annotation data when annotateSuspects is run.

Suspect annotation

The annotateSuspects method is used to annotate suspects after screenSuspects was used to collect suspect screening results and other workflow steps such as formula and compound annotation steps have been completed. The annotation results, which can be acquired with the as.data.table and screenInfo methods, amends the current screening data with the following columns:

- formRank,compRank The rank of the suspect within the formula/compound annotation results.

- `annSimForm`,`annSimComp`,`annSimBoth` A similarity measure between measured and annotated MS/MS peaks from annotation of formulae, compounds or both. The similarity is calculated as the spectral similarity between a peaklist with (a) all MS/MS peaks and (b) only annotated peaks. Thus, a value of one means that all MS/MS peaks were annotated. If both formula and compound annotations are available then `annSimBoth` is calculated after combining all the annotated peaks, otherwise `annSimBoth` equals the available value for `annSimForm` or `annSimComp`. The similarity calculation can be configured with the `relMinMSMSIntensity` and `simMSMSMethod` arguments to `annotateSuspects`.
- `maxFrags` The maximum number of MS/MS fragments that can be matched for this suspect (based on the `fragments_*` columns from the suspect list).
- `maxFragMatches`,`maxFragMatchesRel` The absolute and relative amount of experimental MS/MS peaks that were matched from the fragments specified in the suspect list. The value for `maxFragMatchesRel` is relative to the value for `maxFrags`. The calculation of this column is influenced by the `checkFragments` argument to `annotateSuspects`.
- `estIDLevel` Provides an *estimation* of the identification level, roughly following that of (Schymanski et al. 2014). However, please note that this value is only an estimation, and manual interpretation is still necessary to assign final identification levels. The estimation is done through a set of rules, see the Identification level rules section below.

Note that only columns are present is sufficient data is available for their calculation.

Identification level rules

The estimation of identification levels is configured through a YAML file which specifies the rules for each level. The default file is shown below.

```
1:
    suspectFragments: 3
    retention: 12
2a:
    individualMoNAScore:
        min: 0.9
        higherThanNext: .inf
    rank:
        max: 1
        type: compound
3a:
    individualMoNAScore: 0.4
3b:
    suspectFragments: 3
3c:
    annMSMSSim:
        type: compound
        min: 0.7
4a:
    annMSMSSim:
        type: formula
        min: 0.7
```

```
        higherThanNext: 0.2
    isoScore:
        min: 0.5
        higherThanNext: 0.2
    rank:
        max: 1
        type: formula
4b:
    isoScore:
        min: 0.9
        higherThanNext: 0.2
    rank:
        max: 1
        type: formula
5:
    all: yes
```

Most of the file should be self-explanatory. Some notes:

- Each rule is either a field of `suspectFragments` (minimum number of MS/MS fragments matched from suspect list), `retention` (maximum retention deviation from suspect list), `rank` (the maximum annotation rank from formula or compound annotations), `all` (this level is always matched) or any of the scorings available from the formula or compound annotations.
- In case any of the rules could be applied to either formula or compound annotations, the annotation type must be specified with the `type` field (`formula` or `compound`).
- Identification levels should start with a number and may optionally be followed by a alphabetic character. The lowest levels are checked first.
- If `relative=yes` then the relative scoring will be used for testing.
- For `suspectFragments`: if the number of fragments from the suspect list (`maxFrags` column) is less then the minimum rule value, the minimum is adjusted to the number of available fragments.

A template rules file can be generated with the `genIDLevelRulesFile` function, and this file can subsequently passed to `annotateSuspects`. The file format is highly flexible and (sub)levels can be added or removed if desired. Note that the default file is currently only suitable when annotation is performed with GenForm and MetFrag, for other algorithms it is crucial to modify the rules.

S4 class hierarchy

- `featureGroups`
  - `featureGroupsScreening`
    - `featureGroupsSetScreeningUnset`

Source

Cosine spectral similarity calculation was based on the code from `SpectrumSimilarity()` function of **OrgMassSpecR**.

Sets workflows

The `featureGroupsScreeningSet` class is applicable for sets workflows. This class is derived from `featureGroupsScreening` and therefore largely follows the same user interface.

The following methods are specifically defined for sets workflows:

- All the methods from base class `workflowStepSet`.
- `unset` Converts the object data for a specified set into a 'non-set' object (`featureGroupsScreeningUnset`), which allows it to be used in 'regular' workflows. Only the screening results present in the specified set are kept.

The following methods are changed or with new functionality:

- `annotateSuspects` Suspect annotation is performed per set. Thus, formula/compound ranks, estimated identification levels etc are calculated for each set. Subsequently, these results are merged in the final `screenInfo`. In addition, an overall `formRank` and `compRank` column is created based on the rankings of the suspect candidate in the set consensus data. Furthermore, an overall `estIDLevel` is generated that is based on the 'best' estimated identification level among the sets data (*i.e.* the lowest). In case there is a tie between sub-levels (*e.g.* '`3a`' and '`3b`'), then the sub-level is stripped (*e.g.* '`3`').
- `filter` All filters releated to estimated identification levels and formula/compound rankings are applied to the overall set data (see above). All others are applied to set specific data: in this case candidates are only removed if none of the set data confirms to the filter.

This class derives also from `featureGroupsSet`. Please see its documentation for more relevant details with sets workflows.

Note that the `formRank` and `compRank` columns are *not* updated when the data is subset.

Note

The `relMinMSMSIntensity` filter argument to `annotateSuspects` is applied *after* removing the precursor ion from the peak lists (if present). Thus, intensity scales may be different when this filter is applied when the most abundant peak resulted from the precursor ion.

`filter` removes suspect hits with `NA` values when any of the filters related to minimum or maximum values are applied (unless `negate=TRUE`).

Author(s)

Rick Helmus <<r.helmus@uva.nl>>, Emma Schymanski <<emma.schymanski@uni.lu>> (contributions to identification level rules), Bas van de Velde (contributions to spectral similarity calculation).

References

Schymanski EL, Jeon J, Gulde R, Fenner K, Ruff M, Singer HP, Hollender J (2014). "Identifying Small Molecules via High Resolution Mass Spectrometry: Communicating Confidence." *Environmental Science and Technology*, **48**(4), 2097–2098. doi: 10.1021/es5002105.

Stein SE, Scott DR (1994). "Optimization and testing of mass spectral library search algorithms for compound identification." *Journal of the American Society for Mass Spectrometry*, **5**(9), 859–866. doi: 10.1016/10440305(94)870098, https://doi.org/10.1016/1044-0305(94)87009-8.

See Also

featureGroups

`featureQualityNames` *Returns chromatographic peak quality and score names for features and/or feature groups.*

Description

Returns chromatographic peak quality and score names for features and/or feature groups.

Usage

```
featureQualityNames(feat = TRUE, group = TRUE, scores = FALSE, totScore = TRUE)
```

Arguments

`feat`	If `TRUE` then names specific to features are returned.
`group`	If `TRUE` then names specific to groups are returned.
`scores`	If `TRUE` the score names are returned, otherwise the quality names.
`totScore`	If `TRUE` (and `scores=TRUE`) then the name of the total score is included.

`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,
  qualityRange = 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 'features'
delete(obj, i = NULL, j = NULL, ...)

## S4 method for signature 'features'
calculatePeakQualities(obj, weights, flatnessFactor, parallel = TRUE)

## S4 method for signature 'featuresSet'
sets(obj)

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

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

## S4 method for signature 'featuresSet,ANY,missing,missing'
x[i, ..., sets = NULL, drop = TRUE]
```

```
## S4 method for signature 'featuresSet'
filter(obj, ..., negate = FALSE, sets = NULL)

## S4 method for signature 'featuresSet'
unset(obj, set)

## S4 method for signature 'featuresKPIC2'
delete(obj, i = NULL, j = NULL, ...)

## S4 method for signature 'featuresXCMS'
delete(obj, i = NULL, j = NULL, ...)

## S4 method for signature 'featuresXCMS3'
delete(obj, i = NULL, j = NULL, ...)
```

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.

`qualityRange` Used to filter features by their peak qualities/scores (see `calculatePeakQualities`). Should be a named `list` with min/max ranges for each quality/score to be filtered (the `featureQualityNames` function can be used to obtain valid names). Example: `qualityRange=list(ModalityScore=c(0.3,Inf),SymmetryScore=c(0.5,Inf))`. Set to `NULL` to ignore.

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

`i`, `j` For `[`/`[[`: 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.

For `delete`: The data to remove from. `i` are the analyses as numeric index, logical or character, `j` the features as numeric index (row) of the feature. If either is `NULL` then data for all is removed. `j` may also be a function: it will be called for each analysis, with the feature table (a `data.table`) as first argument, the analysis name as second argument, and any other arguments passed as `...`

	to delete. The return value of this function specifies the feature indices (rows) to be removed (specified as an integer or logical vector).
...	For delete: passed to the function specified as j. For sets workflow methods: further arguments passed to the base features method.
drop	ignored.
name	The analysis name (partially matched).
weights	A named numeric vector that defines the weight for each score to calculate the totalScore. The names of the vector follow the score names. Unspecified weights are defaulted to '1'. Example: weights=c(ApexBoundaryRatioScore=0.5,GaussianSimilari
flatnessFactor	Passed to **MetaClean** as the flatness.factor argument to calculateJaggedness and calculateModality.
parallel	If set to TRUE then code is executed in parallel through the **futures** package. Please see the parallelization section in the handbook for more details.
sets	(**sets workflow**) For [and filter: a character with name(s) of the sets to keep (or remove if negate=TRUE).
set	(**sets workflow**) The name of the set.

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).

delete returns the object for which the specified data was removed.

calculatePeakQualities returns a modified object amended with peak qualities and scores.

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.
- `delete`: Completely deletes specified features.
- `calculatePeakQualities`: Calculates peak qualities for each feature. This uses MetaClean R package to calculate the following metrics: Apex-Boundary Ratio, FWHM2Base, Jaggedness, Modality, Symmetry, Gaussian
  Similarity, Sharpness, Triangle Peak Area Similarity Ratio and Zig-Zag index. Please see the **MetaClean** publication (referenced below) for more details. For each metric, an additional score is calculated by normalizing all feature values (unless the quality metric definition has a fixed range) and scale from '`0`' (worst) to '`1`' (best). Then, a totalScore for each feature is calculated by the (weighted) sum of all score values.

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`
    - `featuresSet`
    - `featuresUnset`
    - `featuresFromFeatGroups`
    - `featuresConsensus`
    - `featuresBruker`
    - `featuresEnviPick`
    - `featuresKPIC2`
    - `featuresOpenMS`
    - `featuresSAFD`
    - `featuresSIRIUS`
    - `featuresBrukerTASQ`
    - `featuresXCMS`
    - `featuresXCMS3`

Sets workflows

The `featuresSet` class is applicable for sets workflows. This class is derived from `features` and therefore largely follows the same user interface.

The following methods are specifically defined for sets workflows:

- `sets` Returns the set names for this object.
- `unset` Converts the object data for a specified set into a 'non-set' object (`featuresUnset`), which allows it to be used in 'regular' workflows. The adduct annotations for the selected set (*e.g.* as passed to `makeSet`) are used to convert all feature masses to ionic *m/z* values.

The following methods are changed or with new functionality:

- `filter` and the subset operator (`[`) have specific arguments to choose/filter by (feature presence in) sets. See the `sets` argument description.

Note

For `calculatePeakQualities`: sometimes **MetaClean** may return `NA` for the Gaussian Similarity metric, in which case it will be set to '`0`'.

References

Chetnik K, Petrick L, Pandey G (2020). "MetaClean: a machine learning-based classifier for reduced false positive peak detection in untargeted LC-MS metabolomics data." *Metabolomics*, **16**(11). doi: 10.1007/s11306020017383.

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, MSPeakLists, algorithm, ...)

## S4 method for signature 'featureGroups'
generateFormulasDA(
  fGroups,
  MSPeakLists,
  precursorMzSearchWindow = 0.002,
  MSMode = "both",
  adduct = NULL,
  featThreshold = 0,
  featThresholdAnn = 0.75,
  absAlignMzDev = 0.002,
  save = TRUE,
```

```
  close = save
)

## S4 method for signature 'featureGroupsSet'
generateFormulasDA(
  fGroups,
  MSPeakLists,
  precursorMzSearchWindow = 0.002,
  MSMode = "both",
  adduct = NULL,
  ...,
  setThreshold = 0,
  setThresholdAnn = 0
)

## S4 method for signature 'featureGroups'
generateFormulasGenForm(
  fGroups,
  MSPeakLists,
  relMzDev = 5,
  adduct = NULL,
  elements = "CHNOP",
  hetero = TRUE,
  oc = FALSE,
  extraOpts = NULL,
  calculateFeatures = TRUE,
  featThreshold = 0,
  featThresholdAnn = 0.75,
  absAlignMzDev = 0.002,
  MSMode = "both",
  isolatePrec = TRUE,
  timeout = 120,
  topMost = 50,
  batchSize = 8
)

## S4 method for signature 'featureGroupsSet'
generateFormulasGenForm(
  fGroups,
  MSPeakLists,
  relMzDev = 5,
  adduct = NULL,
  ...,
  setThreshold = 0,
  setThresholdAnn = 0
)

## S4 method for signature 'featureGroups'
```

```
generateFormulasSIRIUS(
  fGroups,
  MSPeakLists,
  relMzDev = 5,
  adduct = NULL,
  elements = "CHNOP",
  profile = "qtof",
  database = NULL,
  noise = NULL,
  cores = NULL,
  topMost = 100,
  extraOptsGeneral = NULL,
  extraOptsFormula = NULL,
  calculateFeatures = TRUE,
  featThreshold = 0,
  featThresholdAnn = 0.75,
  absAlignMzDev = 0.002,
  verbose = TRUE,
  splitBatches = FALSE
)

## S4 method for signature 'featureGroupsSet'
generateFormulasSIRIUS(
  fGroups,
  MSPeakLists,
  relMzDev = 5,
  adduct = NULL,
  ...,
  setThreshold = 0,
  setThresholdAnn = 0
)

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.
MSPeakLists	An MSPeakLists object that was generated for the supplied fGroups.
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. For sets workflows: further arguments directly passed to the non-sets method.
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]+". If the featureGroups object has adduct annotations then these are used if adducts=NULL.

(**sets workflow**) The adduct argument is not supported for sets workflows, since the adduct annotations will then always be used.

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=0.75 dictates that a formula should be present in at least 75% of the features inside a feature group.

featThresholdAnn — As featThreshold, but only considers features with annotations. For instance, featThresholdAnn=0.75 dictates that a formula should be present in at least 75% of the features with annotations inside a feature group.

absAlignMzDev — When the group formula annotation consensus is made from feature annotations, the *m/z* values of annotated MS/MS fragments may slightly deviate from those of the corresponding group MS/MS peak list. The absAlignMzDev argument specifies the maximum *m/z* window used to re-align the mass peaks.

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.

setThreshold — (**sets workflow**) Minimum abundance for a candidate among all sets ('0-1'). For instance, a value of '1' means that the candidate needs to be present in all the set data.

setThresholdAnn — (**sets workflow**) As setThreshold, but only taking into account the set data that contain annotations for the feature group of the candidate.

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 or FALSE for no isolation (*e.g.* when you already performed isolation with the filter method). The z parameter (charge) is automatically deduced from the adduct used for annotation (unless isolatePrec=FALSE), hence any custom z setting is ignored.

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.

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. Note that this is ignored if 'patRoon.MP.method="future"'.

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.

splitBatches — If TRUE then the calculations done by SIRIUS will be evenly split over multiple SIRIUS calls (which may be run in parallel depending on the set package options). If splitBatches=FALSE then all feature calculations are performed from a single SIRIUS execution, which is often the fastest if calculations are performed on a single computer.

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 and featThresholdAnn arguments). 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. It is recommended, but not mandatory, that the MSPeakLists are also generated by DataAnalysis.

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
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

Sets workflows

With a sets workflow, annotation is first performed for each set. This is important, since the annotation algorithms typically cannot work with data from mixed ionization modes. The annotation results are then combined to generate a *sets consensus*:

- The annotation tables for each feature group from the set specific data are combined. Rows with overlapping candidates (determined by the neutral formula) are merged.
- Set specific data (*e.g.* the ionic formula) is retained by renaming their columns with set specific names.
- The MS/MS fragment annotations (fragInfo column) from each set are combined.
- The scorings for each set are averaged to calculate overall scores.
- The candidates are re-ranked based on their average ranking among the set data (if a candidate is absent in a set it is assigned the poorest rank in that set).
- The coverage of each candidate among sets is calculated. Depending on the setThreshold and setThresholdAnn arguments, candidates with low abundance are removed.

Parallelization

generateFormulasGenForm and generateFormulasSIRIUS uses multiprocessing to parallelize computations. Please see the parallelization section in the handbook for more details and patRoon options for configuration options.

When futures are used for parallel processing (patRoon.MP.method="future"), calculations with GenForm are done with batch mode is disabled (see batchSize argument), which generally limit overall performance.

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 splitBatches=FALSE. In this case, all SIRIUS output will be printed to the terminal (unless verbose=FALSE or 'patRoon.MP.method="future"'). Furthermore, please note that only annotations to be performed for the same adduct are grouped in a single batch execution.

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**(2), 259–290.

Duhrkop K, Fleischauer M, Ludwig M, Aksenov AA, Melnik AV, Meusel M, Dorrestein PC, Rousu J, Bocker S (2019). "SIRIUS 4: a rapid tool for turning tandem mass spectra into metabolite structure information." *Nature Methods*, **16**(4), 299–302. doi: 10.1038/s4159201903448.

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

Duhrkop K, Shen H, Meusel M, Rousu J, Bocker S (2015). "Searching molecular structure databases with tandem mass spectra using CSI:FingerID." *Proceedings of the National Academy of Sciences*, **112**(41), 12580–12585. doi: 10.1073/pnas.1509788112.

Bocker S, Letzel MC, Liptak Z, Pervukhin A (2008). "SIRIUS: decomposing isotope patterns for metabolite identification." *Bioinformatics*, **25**(2), 218–224. doi: 10.1093/bioinformatics/btn603.

See Also

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

---

formulas-class *Formula annotations class*

Description

Contains data of generated chemical formulae for given feature groups.

Usage

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

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

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

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

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

## S4 method for signature 'formulas'
delete(obj, i = NULL, j = NULL, ...)

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

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

## S4 method for signature 'formulas'
plotSpectrum(
  obj,
  index,
  groupName,
  analysis = NULL,
  MSPeakLists,
  title = NULL,
  specSimParams = getDefSpecSimParams(),
  mincex = 0.9,
  xlim = NULL,
  ylim = NULL,
  ...
)

## S4 method for signature 'formulas'
plotScores(
  obj,
  index,
  groupName,
  analysis = NULL,
  normalizeScores = "max",
  excludeNormScores = defaultExclNormScores(obj)
)

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

## S4 method for signature 'formulasSet'
```

```
show(object)

## S4 method for signature 'formulasSet'
delete(obj, i, j, ...)

## S4 method for signature 'formulasSet,ANY,missing,missing'
x[i, j, ..., sets = NULL, updateConsensus = FALSE, drop = TRUE]

## S4 method for signature 'formulasSet'
filter(obj, ..., sets = NULL, updateConsensus = FALSE, negate = FALSE)

## S4 method for signature 'formulasSet'
plotSpectrum(
  obj,
  index,
  groupName,
  analysis = NULL,
  MSPeakLists,
  title = NULL,
  specSimParams = getDefSpecSimParams(),
  mincex = 0.9,
  xlim = NULL,
  ylim = NULL,
  perSet = TRUE,
  mirror = TRUE,
  ...
)

## S4 method for signature 'formulasSet'
annotatedPeakList(obj, index, groupName, analysis = NULL, MSPeakLists, ...)

## S4 method for signature 'formulasSet'
consensus(
  obj,
  ...,
  absMinAbundance = NULL,
  relMinAbundance = NULL,
  uniqueFrom = NULL,
  uniqueOuter = FALSE,
  rankWeights = 1,
  labels = NULL,
  filterSets = FALSE,
  setThreshold = 0,
  setThresholdAnn = 0
)

## S4 method for signature 'formulasSet'
unset(obj, set)
```

```
## S4 method for signature 'formulasConsensusSet'
unset(obj, set)
```

Arguments

obj, x, object	The formulas object.
features	If TRUE returns formula data for features, otherwise for feature groups.
i, j	For [[: If both i and j are specified then i specifies the analysis and j the feature group of the feature for which annotations should be returned. Otherwise i specifies the feature group for which group annotations should be returned. i/j can be specified as integer index or as a character name. Otherwise passed to the featureAnnotations method.
...	For plotSpectrum: Further arguments passed to plot. For delete: passed to the function specified as j. For consensus: Any further (and unique) formulas objects. For sets workflow methods: further arguments passed to the base formulas method.
fGroups, fragments, countElements, countFragElements, OM	Passed to the featureAnnotations method.
normalizeScores	A character that specifies how normalization of annotation 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.
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. Setting this to TRUE disables reporting of most other data.
index	The candidate index (row). For plotSpectrum two indices can be specified to compare spectra. In this case groupName and analysis (if not NULL) should specify values for the spectra to compare.
groupName	The name of the feature group (or feature groups when comparing spectra) to which the candidate belongs.

analysis	A character specifying the analysis (or analyses when comparing spectra) 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.
title	The title of the plot. Set to NULL for an automatically generated title.
specSimParams	A named list with parameters that influence the calculation of MS spectra similarities. See the spectral similarity parameters documentation for more details.
mincex	The formula annotation labels are automatically scaled. The mincex argument forces a minimum cex value for readability.
xlim, ylim	Sets the plot size limits used by plot. Set to NULL for automatic plot sizing.
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 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.
rankWeights	A numeric vector with weights of to calculate 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.
labels	A character with names to use for labelling. If NULL labels are automatically generated.
sets	(**sets workflow**) A character with name(s) of the sets to keep (or remove if negate=TRUE). Note: if updateConsensus=FALSE then the setCoverage column of the annotation results is not updated.
updateConsensus	(**sets workflow**) If TRUE then the annonation consensus among set results is updated. See the Sets workflows section for more details.
drop	Passed to the featureAnnotations method.
negate	Passed to the featureAnnotations method.
perSet, mirror	(**sets workflow**) If perSet=TRUE then the set specific mass peaks are annotated separately. Furthermore, if mirror=TRUE (and there are two sets in the object) then a mirror plot is generated.
filterSets	(**sets workflow**) Controls how algorithms concensus abundance filters are applied. See the Sets workflows section below.

`setThreshold, setThresholdAnn`
: (**sets workflow**) Thresholds used to create the annotation set consensus. See `generateFormulas`.

`set`
: (**sets workflow**) The name of the set.

Details

`formulas` objects are obtained from formula generators. This class is derived from the `featureAnnotations` class, please see its documentation for more methods and other details.

Value

`annotations` 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.

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

Methods (by generic)

- `annotations`: Accessor method to obtain generated formulae.
- `analyses`: returns a `character` vector with the names of the analyses for which data is present in this object.
- `defaultExclNormScores`: Returns default scorings that are excluded from normalization.
- `show`: Show summary information for this object.
- `[[`: Extracts a formula table, either for a feature group or for features in an analysis.
- `as.data.table`: Generates a table with all candidate formulae for each feature group and other information such as element counts.
- `annotatedPeakList`: Returns an MS/MS peak list annotated with data from a given candidate formula.
- `plotSpectrum`: Plots an annotated spectrum for a given candidate formula of a feature or feature group. Two spectra can be compared by specifying a two-sized vector for the `index`, `groupName` and (if desired) `analysis` arguments.
- `plotScores`: Plots a barplot with scoring of a candidate formula.
- `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

`featureFormulas` A `list` with all generated formulae for each analysis/feature group. Use the `annotations` method for access.

`setThreshold,setThresholdAnn` (**sets workflow**) A copy of the equally named arguments that were passed when this object was created by `generateFormulas`.

`origFGNames` (**sets workflow**) The original (order of) names of the `featureGroups` object that was used to create this object.

S4 class hierarchy

- `featureAnnotations`
  - `formulas`
    - `formulasConsensus`
    - `formulasSet`
      - `formulasConsensusSet`
    - `formulasUnset`

Source

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

Sets workflows

The `formulasSet` class is applicable for sets workflows. This class is derived from `formulas` and therefore largely follows the same user interface.

The following methods are specifically defined for sets workflows:

- All the methods from base class `workflowStepSet`.
- `unset` Converts the object data for a specified set into a 'non-set' object (`formulasUnset`), which allows it to be used in 'regular' workflows. Only the annotation results that are present in the specified set are kept (based on the set consensus, see below for implications).

The following methods are changed or with new functionality:

- `filter` and the subset operator (`[`) Can be used to select data that is only present for selected sets. Depending on the `updateConsenus`, both either operate on set consensus or original data (see below for implications).
- `annotatedPeakList` Returns a combined annotation table with all sets.
- `plotSpectrum` Is able to highlight set specific mass peaks (`perSet` and `mirror` arguments).
- `consensus` Creates the algorithm consensus based on the original annotation data (see below for implications). Then, like the sets workflow method for `generateFormulas`, a consensus is made for all sets, which can be controlled with the `setThreshold` and `setThresholdAnn` arguments. The candidate coverage among the different algorithms is calculated for each set (*e.g.* `coverage-positive` column) and for all sets (`coverage` column), which is based on the presence of a candidate in all the algorithms from all sets data. The `consensus` method for sets workflow data supports the `filterSets` argument. This controls how the algorithm consensus abundance filters (`absMinAbundance`/`relMinAbundance`) are applied: if `filterSets=TRUE` then the minimum of all `coverage` set specific columns is used to obtain the algorithm abundance. Otherwise the overall `coverage` column is used. For instance, consider a consensus

object to be generated from two objects generated by different algorithms (*e.g.* SIRIUS and GenForm), which both have a positive and negative set. Then, if a candidate occurs with both algorithms for the positive mode set, but only with the first algorithm in the negative mode set, relMinAbundance=1 will remove the candidate if filterSets=TRUE (because the minimum relative algorithm abundance is '0.5'), while filterSets=FALSE will not remove the candidate (because based on all sets data the candidate occurs in both algorithms).

Two types of annotation data are stored in a formulasSet object:

1. Annotations that are produced from a consensus between set results (see generateFormulas).
2. The 'original' annotation data per set, prior to when the set consensus was made. This includes candidates that were filtered out because of the thresholds set by setThreshold and setThresholdAnn. However, when filter or subsetting ([) operations are performed, the original data is also updated.

In most cases the first data is used. However, in a few cases the original annotation data is used (as indicated above), for instance, to re-create the set consensus. It is important to realize that the original annotation data may have *additional* candidates, and a newly created set consensus may therefore have 'new' candidates. For instance, when the object consists of the sets "positive" and "negative" and setThreshold=1 was used to create it, then formulas[,sets = "positive",updateConsensus = TRUE] may now have additional candidates, *i.e.* those that were not present in the "negative" set and were previously removed due to the consensus threshold filter.

See Also

The featureAnnotations base class for more relevant methods and formula-generation.

---

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

```
adducts(obj, ...)

adducts(obj, ...) <- value

algorithm(obj)

analysisInfo(obj)

analyses(obj)
```

```
annotatedPeakList(obj, ...)

annotations(obj, ...)

calculatePeakQualities(obj, weights = NULL, flatnessFactor = 0.05, ...)

clusterProperties(obj)

clusters(obj)

consensus(obj, ...)

convertToMFDB(TPs, out, ...)

convertToSuspects(TPs, ...)

cutClusters(obj)

defaultExclNormScores(obj)

export(obj, type, out, ...)

featureTable(obj, ...)

filter(obj, ...)

getFeatures(obj)

getMCS(obj, ...)

groupNames(obj)

makeSet(obj, ...)

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

plotChroms(obj, ...)

plotGraph(obj, onlyLinked = TRUE, ...)

plotInt(obj, ...)

plotScores(obj, ...)

plotSilhouettes(obj, kSeq, ...)

plotSpectrum(obj, ...)
```

```
plotStructure(obj, ...)

plotVenn(obj, ...)

plotUpSet(obj, ...)

delete(obj, ...)

plotVolcano(obj, ...)

replicateGroups(obj)

setObjects(obj)

sets(obj)

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

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

unset(obj, set)
```

Arguments

obj	The object the generic should be applied to.
...	Any further method specific arguments. See method documentation for details.
value	The replacement value.
weights, flatnessFactor	See method documentation.
TPs	The transformationProducts derived object.
out	Output file.
type	The export type.
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.
onlyLinked	Only plots linked objects if TRUE.
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.

set
: The name of the set.

Details

adducts returns assigned adducts of the object.

- Methods are defined for: featureGroups; featureGroupsSet.

adducts<- sets adducts of the object.

- Methods are defined for: featureGroups; featureGroupsSet.

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

- Methods are defined for: optimizationResult; workflowStep.

analysisInfo returns the analysis information from an object.

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

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; compoundsSet; formulas; formulasSet.

annotations returns annotations.

- Methods are defined for: featureAnnotations; featureGroups; formulas.

calculatePeakQualities calculates chromatographic peak qualities and scores.

- Methods are defined for: featureGroups; features.

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

- Methods are defined for: componentsClust; compoundsCluster.

clusters Obtain clustering object(s).

- Methods are defined for: componentsClust; compoundsCluster.

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

- Methods are defined for: components; componentsSet; compounds; compoundsSet; featureGroupsComparison; featureGroupsComparisonSet; formulas; formulasSet.

convertToMFDB Exports the object to a local database that can be used with MetFrag.

- Methods are defined for: .

convertToSuspects Converts an object to a suspect list.

- Methods are defined for: .

cutClusters Returns assigned cluster indices of a cut cluster.

- Methods are defined for: componentsClust; compoundsCluster.

defaultExclNormScores Returns default scorings that are excluded from normalization.

- Methods are defined for: compounds; formulas.

export exports workflow data to a given format.

- Methods are defined for: featureGroups; featureGroupsSet.

featureTable returns feature information.

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

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

- Methods are defined for: components; componentsSet; componentsTPs; compounds; compoundsSet; featureAnnotations; featureGroups; featureGroupsScreening; featureGroupsScreeningSet; featureGroupsSet; features; featuresSet; formulasSet; MSPeakLists; MSPeakListsSet; transformationProductsBT.

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; compoundsCluster; featureAnnotations; featureGroups; MSPeakLists.

makeSet creates a set from given workflow objects.

- Methods are defined for: featureGroups; featureGroupsSet; features; featuresSet.

plotChord plots a Chord diagram to assess overlapping data.

- Methods are defined for: featureGroups; featureGroupsComparison.

plotChroms plots extracted ion chromatogram(s).

- Methods are defined for: components; featureGroups.

plotGraph Plots an interactive network graph.

- Methods are defined for: componentsNT; componentsNTSet; componentsTPs.

plotInt plots the intensity of all contained features.

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

plotScores plots candidate scorings.

- Methods are defined for: compounds; formulas.

plotSilhouettes plots silhouette widths to evaluate the desired cluster size.

- Methods are defined for: componentsClust; compoundsCluster.

plotSpectrum plots a (annotated) spectrum.

- Methods are defined for: components; compounds; compoundsSet; formulas; formulasSet; MSPeakLists; MSPeakListsSet.

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: featureAnnotations; featureGroups; featureGroupsComparison; featureGroupsSet.

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

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

delete Deletes results.

- Methods are defined for: components; componentsClust; componentsSet; compoundsSet; featureAnnotations; featureGroups; featureGroupsKPIC2; featureGroupsScreening; featureGroupsScreeningSet; featureGroupsSet; featureGroupsXCMS; featureGroupsXCMS3; features; featuresKPIC2; featuresXCMS; featuresXCMS3; formulas; formulasSet.

plotVolcano plots a volcano plot.

- Methods are defined for: featureGroups.

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

- Methods are defined for: featureGroups; features.

setObjects returns the *set objects* of this object. See the documentation of workflowStepSet.

- Methods are defined for: workflowStepSet.

sets returns the names of the sets inside this object. See the documentation for sets workflows.

- Methods are defined for: featureGroupsSet; featuresSet; workflowStepSet.

treeCut Manually cut a cluster.

- Methods are defined for: componentsClust; compoundsCluster.

treeCutDynamic Automatically cut a cluster.

- Methods are defined for: componentsClust; compoundsCluster.

unset Converts this object to a regular non-set object. See the documentation for sets workflows.

- Methods are defined for: componentsNTSet; componentsSet; compoundsConsensusSet; compoundsSet; featureGroupsScreeningSet; featureGroupsSet; featuresSet; formulasConsensusSet; formulasSet; MSPeakListsSet.

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; componentsSet,ANY,ANY,missing; compoundsCluster,ANY,missing,missing; compoundsSet,ANY,missing,missing; featureAnnotations,ANY,mi featureGroups,ANY,ANY,missing; featureGroupsComparison,ANY,missing,missing; featureGroupsScreenin featureGroupsScreeningSet,ANY,ANY,missing; featureGroupsSet,ANY,ANY,missing; features,ANY,missing,missing; featuresSet,ANY,missing,missing; formulasSet,ANY,missing,missing; MSPeakLists,ANY,ANY,missing; MSPeakListsSet,ANY,ANY,missing; transformationProducts,ANY,missing,m

[[Extract data from an object.

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

$ Extract data from an object.

- Methods are defined for: components; featureAnnotations; featureGroups; featureGroupsComparison; features; MSPeakLists; transformationProducts.

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

- Methods are defined for: components; componentsTPs; featureAnnotations; featureGroups; featureGroupsScreening; featureGroupsScreeningSet; featureGroupsSet; features; featuresSet; formulas; MSPeakLists; MSPeakListsSet; transformationProducts; 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; compoundsCluster; featureAnnotations; featureGroups; featureGroupsComparison; features; MSPeakLists; optimizationResult; transformationProducts.

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; transformationProducts.

plot Generates a plot for an object.

- Methods are defined for: componentsClust,missing; compoundsCluster,missing; featureGroups,missing; featureGroupsComparison,missing; optimizationResult,missing.

show Prints information about this object.

- Methods are defined for: adduct; components; componentsFeatures; componentsSet; compounds; compoundsCluster; compoundsSet; featureGroups; featureGroupsScreening; featureGroupsScreeningSet; featureGroupsSet; features; featuresSet; formulas; formulasSet; MSPeakLists; MSPeakListsSet; optimizationResult; transformationProducts; workflowStep; workflowStepSet.

---

getPICSet,features-method

Conversion to KPIC2 objects

Description

Converts a features object to an **KPIC** object.

Usage

```
## S4 method for signature 'features'
getPICSet(obj, loadRawData = TRUE)

## S4 method for signature 'featuresKPIC2'
getPICSet(obj, ...)
```

Arguments

obj	The features object that should be converted.
loadRawData	Set to TRUE if analyses are available as mzXML or mzML files. Otherwise MS data is not loaded, and some dummy data (*e.g.* file paths) is used in the returned object.
...	Ignored

getXCMSSet *Conversion to XCMS objects*

Description

Converts a features or featureGroups object to an xcmsSet or XCMSnExp object.

Usage

```
getXCMSSet(obj, verbose = TRUE, ...)

getXCMSnExp(obj, verbose = TRUE, ...)

## S4 method for signature 'features'
getXCMSSet(obj, verbose, loadRawData)

## S4 method for signature 'featuresXCMS'
getXCMSSet(obj, verbose = TRUE, ...)

## S4 method for signature 'featureGroups'
getXCMSSet(obj, verbose, loadRawData)

## S4 method for signature 'featureGroupsXCMS'
getXCMSSet(obj, verbose, loadRawData)

## S4 method for signature 'featuresSet'
getXCMSSet(obj, ..., set)

## S4 method for signature 'featureGroupsSet'
getXCMSSet(obj, ..., set)

## S4 method for signature 'features'
getXCMSnExp(obj, verbose, loadRawData)

## S4 method for signature 'featuresXCMS3'
getXCMSnExp(obj, verbose = TRUE, ...)

## S4 method for signature 'featureGroups'
getXCMSnExp(obj, verbose, loadRawData)

## S4 method for signature 'featureGroupsXCMS3'
getXCMSnExp(obj, verbose, loadRawData)

## S4 method for signature 'featuresSet'
getXCMSnExp(obj, ..., set)

## S4 method for signature 'featureGroupsSet'
```

```
getXCMSnExp(obj, ..., set)
```

Arguments

obj	The object that should be converted.
verbose	If FALSE then no text output is shown.
...	(**sets workflow**) Further arguments passed to non-sets method. Otherwise ignored.
loadRawData	Set to TRUE if analyses are available as mzXML or mzML files. Otherwise MS data is not loaded, and some dummy data (*e.g.* file paths) is used in the returned object.
set	(**sets workflow**) The name of the set to be exported.

Sets workflows

In a sets workflow, unset is used to convert the feature (group) data before the object is exported.

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 = FALSE, 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,
  minMSMSPeaks = NULL,
  isolatePrec = NULL,
  deIsotopeMS = FALSE,
  deIsotopeMSMS = FALSE,
  withMSMS = FALSE,
  annotatedBy = NULL,
  retainPrecursorMSMS = TRUE,
  reAverage = FALSE,
  negate = FALSE
)

## S4 method for signature 'MSPeakLists'
plotSpectrum(
  obj,
  groupName,
  analysis = NULL,
  MSLevel = 1,
  title = NULL,
  specSimParams = getDefSpecSimParams(),
  xlim = NULL,
  ylim = NULL,
  ...
)

## S4 method for signature 'MSPeakLists'
spectrumSimilarity(
  obj,
  groupName1,
  groupName2 = NULL,
```

```
  analysis1 = NULL,
  analysis2 = NULL,
  MSLevel = 1,
  specSimParams = getDefSpecSimParams(),
  NAToZero = FALSE,
  drop = TRUE
)

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

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

## S4 method for signature 'MSPeakListsSet,ANY,ANY,missing'
x[i, j, ..., reAverage = FALSE, sets = NULL, drop = TRUE]

## S4 method for signature 'MSPeakListsSet'
as.data.table(x, fGroups = NULL, averaged = TRUE)

## S4 method for signature 'MSPeakListsSet'
filter(
  obj,
  ...,
  annotatedBy = NULL,
  retainPrecursorMSMS = TRUE,
  reAverage = FALSE,
  negate = FALSE,
  sets = NULL
)

## S4 method for signature 'MSPeakListsSet'
plotSpectrum(
  obj,
  groupName,
  analysis = NULL,
  MSLevel = 1,
  title = NULL,
  specSimParams = getDefSpecSimParams(),
  xlim = NULL,
  ylim = NULL,
  perSet = TRUE,
  mirror = TRUE,
  ...
)

## S4 method for signature 'MSPeakListsSet'
spectrumSimilarity(
```

```
  obj,
  groupName1,
  groupName2 = NULL,
  analysis1 = NULL,
  analysis2 = NULL,
  MSLevel = 1,
  specSimParams = getDefSpecSimParams(),
  NAToZero = FALSE,
  drop = TRUE
)

## S4 method for signature 'MSPeakListsSet'
unset(obj, set)

getDefIsolatePrecParams(...)
```

Arguments

`obj, x, object`	The `MSPeakLists` object to access.
`i, j`	For `[`/`[[`: 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`. For sets workflow methods: further arguments passed to the base `MSPeakLists` method.
`reAverage`	Set to `TRUE` to regenerate group averaged MS peak lists. **NOTE** it is very important that any annotation data relying on MS peak lists (formulae/compounds) are regenerated afterwards! Otherwise it is likely that *e.g.* plotting methods will use wrong MS/MS data.
`drop`	If set to `TRUE` and if the comparison is made between two spectra then `drop` is used to reduce the `matrix` return value to a `numeric` vector.
`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.

`minMSMSPeaks` If the number of peaks in an MS/MS peak list (**excluding** the precursor peak) is lower than this it will be completely removed. Set to `NULL` to ignore.

`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.

`annotatedBy` Either a `formulas` or `compounds` object, or a `list` with both. Any MS/MS peaks that are *not* annotated by any of the candidates in the specified objects are removed.

`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. To compare spectra, two group names can be specified.

`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. When comparing spectra, either `NULL` or the analyses for both spectra should be specified.

`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.

`specSimParams` A named `list` with parameters that influence the calculation of MS spectra similarities. See the spectral similarity parameters documentation for more details.

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

`groupName1, groupName2`
The names of the feature groups for which the comparison should be made. If both arguments are specified then a comparison is made with the spectra specified by `groupName1` *vs* those specified by `groupName2`. The length of either can be '>1' to generate a comparison matrix. Alternatively, if `groupName2` is `NULL` then all the spectra specified in `groupName1` will be compared with eachother, *i.e.* resulting in a square similarity matrix.

`analysis1, analysis2`
The name of the analysis (analyses) for the comparison. If `NULL` then data from the feature group averaged peak list is used. Otherwise, should be the same length as `groupName1`/`groupName2`.

`NAToZero` Set to `TRUE` to convert `NA` similarities (*i.e.* when no similarity could be calculated) to zero values.

`sets` (**sets workflow**) A `character` with name(s) of the sets to keep (or remove if `negate=TRUE`).

`perSet, mirror` (**sets workflow**) If `perSet=TRUE` then the set specific mass peaks are annotated separately. Furthermore, if `mirror=TRUE` (and there are two sets in the object) then a mirror plot is generated.

`set` (**sets workflow**) The name of the set.

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`.

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. The filters are applied to all peak lists for each analysis. These peak lists are subsequently averaged to update group averaged peak lists. However, since version '`1.1`', the resulting feature group lists are *not* filtered afterwards.
- `plotSpectrum`: Plots a spectrum using MS or MS/MS peak lists for a given feature group. Two spectra can be compared when two feature groups are specified.
- `spectrumSimilarity`: Calculates the spectral similarity between two or more spectra.

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.

`analysisInfo` (**sets workflow**) Analysis information. Use the `analysisInfo` method for access.

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
    - MSPeakListsSet
    - MSPeakListsUnset

Source

spectrumSimilarity: The principles of spectral binning and cosine similarity calculations were loosely was based on the code from SpectrumSimilarity() function of **OrgMassSpecR**.

Sets workflows

The MSPeakListsSet class is applicable for sets workflows. This class is derived from MSPeakLists and therefore largely follows the same user interface.

The following methods are specifically defined for sets workflows:

- All the methods from base class workflowStepSet.
- unset Converts the object data for a specified set into a 'non-set' object (MSPeakListsUnset), which allows it to be used in 'regular' workflows. Only the MS peaks that are present in the specified set are kept.
- analysisInfo Returns the analysis info for this object.

The following methods are changed or with new functionality:

- filter and the subset operator ([) Can be used to select data that is only present for selected sets. The filter method is applied for each set individually, and afterwards the results are combined again (see generateMSPeakLists). Note that this has important implications for *e.g.* relative intensity filters (relMSIntThr/relMSMSIntThr), topMSPeaks/topMSMSPeaks and minMSMSPeaks. Similarly, when the annotatedBy filter is applied, each set specific MS peak list is filtered by the annotation results from only that set.
- plotSpectrum Is able to highlight set specific mass peaks (perSet and mirror arguments).
- spectrumSimilarity First calculates similarities for each spectral pair per set (*e.g.* all positive mode spectra are compared and then all negative mode spectra are compared). This data is then combined into an overall similarity value. How this combination is performed depends on the setCombineMethod field of the specSimParams argument.

Author(s)

For spectrumSimilarity: major contributions by Bas van de Velde for spectral binning and similarity calculation.

MSPeakLists-generation

Generation of MS Peak Lists

Description

Functionality to generate MS peak lists.

Usage

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

## S4 method for signature 'featureGroups'
generateMSPeakListsDA(
  fGroups,
  bgsubtr = TRUE,
  maxMSRtWindow = 5,
  minMSIntensity = 500,
  minMSMSIntensity = 500,
  clear = TRUE,
  close = TRUE,
  save = close,
  MSMSType = "MSMS",
  avgFGroupParams = getDefAvgPListParams()
)

## S4 method for signature 'featureGroupsSet'
generateMSPeakListsDA(fGroups, ...)

## S4 method for signature 'featureGroups'
generateMSPeakListsDAFMF(
  fGroups,
  minMSIntensity = 500,
  minMSMSIntensity = 500,
  close = TRUE,
  save = close,
  avgFGroupParams = getDefAvgPListParams()
)

## S4 method for signature 'featureGroupsSet'
generateMSPeakListsDAFMF(fGroups, ...)

## S4 method for signature 'featureGroups'
generateMSPeakListsMzR(
  fGroups,
  maxMSRtWindow = 5,
```

```
  precursorMzWindow = 4,
  topMost = NULL,
  avgFeatParams = getDefAvgPListParams(),
  avgFGroupParams = getDefAvgPListParams()
)

## S4 method for signature 'featureGroupsSet'
generateMSPeakListsMzR(fGroups, ...)

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. For sets workflows: further arguments passed to the non-sets workflow method.
`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 chromatographic 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`

Sets workflows

With a sets workflow, the feature group averaged peak lists are made per set. This is important, because for averaging peak lists cannot be mixed, for instance, when different ionization modes were used to generate the sets. The group averaged peaklists are then simply combined and labelled in the final peak lists. However, please note that annotation and other functionality typically uses only the set specific peak lists, as this functionality cannot work with mixed peak lists.

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 https://www.jstatsoft.org/v53/i09/.

See Also

MSPeakLists-class

---

newProject *Easily create new* **patRoon** *projects*

Description

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.

Usage

```
newProject(destPath = NULL)
```

Arguments

destPath	Set destination path value to this value (useful for debugging). Set to NULL for a default value.

---

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,missing'
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)
```

printPackageOpts *Prints all the package options of* patRoon *and their currently set values.*

Description

Prints all the package options of patRoon and their currently set values.

Usage

```
printPackageOpts()
```

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",
  reportFeatures = FALSE,
  formulas = NULL,
  formulasNormalizeScores = "max",
  formulasExclNormScores = NULL,
  compounds = NULL,
  compoundsNormalizeScores = "max",
  compoundsExclNormScores = c("score", "individualMoNAScore", "annoTypeCount"),
  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", "annoTypeCount"),
  compoundsOnlyUsedScorings = TRUE,
  compoundsTopMost = 5,
  compsCluster = NULL,
  components = NULL,
  MSPeakLists = NULL,
  retMin = TRUE,
  EICGrid = c(2, 1),
  EICRtWindow = 20,
  EICMzExpWindow = 0.001,
  EICTopMost = 1,
  EICTopMostByRGroup = TRUE,
  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", "annoTypeCount"),
  compoundsOnlyUsedScorings = TRUE,
  compoundsTopMost = 5,
  compsCluster = NULL,
  includeMFWebLinks = "compounds",
  components = NULL,
  interactiveHeat = FALSE,
  MSPeakLists = NULL,
  specSimParams = getDefSpecSimParams(),
  retMin = TRUE,
  EICRtWindow = 20,
  EICMzExpWindow = 0.001,
  EICTopMost = 1,
  EICTopMostByRGroup = TRUE,
  EICOnlyPresent = TRUE,
```

```
  selfContained = TRUE,
  optimizePng = FALSE,
  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.
`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 annotation 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,formulasExclNormScores` 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 (otherwise 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, EICMzExpWindow, EICTopMost, EICTopMostByRGroup, EICOnlyPresent — Plotting parameters passed to plotChroms (*i.e.* rtWindow, mzExpWindow, topMost, topMostByRGroup 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.

specSimParams — A named list with parameters that influence the calculation of MS spectra similarities. See the spectral similarity parameters documentation for more details.

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 significant reduction in disk space usage may be seen, however, at the cost additional processing time. Multiple pngquant processes will be executed in parallel, which can be configured with 'patRoon.MP.maxProcs' (parallelization will always happen with the "classic" method, see patRoon options).

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.

Parallelization

`reportHTML` uses multiprocessing to parallelize computations. Please see the parallelization section in the handbook for more details and patRoon options for configuration options.

Currently, `reportHTML` only uses `"classic"` multiprocessing, regardless of the '`patRoon.MP.method`' option.

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

`sets-workflow` *Sets workflows*

Description

With sets workflows in **patRoon** a complete non-target (or suspect) screening workflow is performed with sample analyses that were measured with different MS methods (typically positive and negative ionization).

Details

The analyses files that were measured with a different method are grouped in *sets*. In the most typical case, there is a `"positive"` and `"negative"` set, for the positively/negatively ionized data, respectively. However, other distinctions than polarity are also possible (although currently the chromatographic method should be the same between sets). A sets workflow is typically initiated with the `makeSet` method. The handbook contains much more details about sets workflows.

See Also

`workflowStepSet`, the Sets workflows sections in other documentation pages and the **patRoon** handbook.

---

specSimParams *MS spectral similarity calculation parameters*

Description

Parameters relevant for calculation of similarities between mass spectra.

Usage

```
getDefSpecSimParams(...)
```

Arguments

`...`	optional named arguments that override defaults.

Details

For the calculation of spectral similarities the following parameters exist:

- `method` The similarity method: either `"cosine"` or `"jaccard"`.
- `removePrecursor` If `TRUE` then precursor peaks (*i.e.* the mass peak corresponding to the feature) are removed prior to similarity calculation.
- `mzWeight`,`intWeight` Mass and intensity weights used for cosine calculation.
- `absMzDev` Maximum absolute *m/z* deviation between mass peaks, used for binning spectra.
- `relMinIntensity` The minimum relative intensity for mass peaks (`'0-1'`). Peaks with lower intensities are not considered for similarity calculation. The relative intensities are called after the precursor peak is removed when `removePrecursor=TRUE`.
- `minPeaks` Only consider spectra that have at least this amount of peaks (*after* the spectrum is filtered).
- `shift` If and how shifting is applied prior to similarity calculation. Valid options are: `"none"` (no shifting), `"precursor"` (all mass peaks of the second spectrum are shifted by the mass difference between the precursors of both spectra) or `"both"` (the spectra are first binned without shifting, and peaks still unaligned are then shifted as is done when `shift="precursor"`).

- `setCombinedMethod` (**sets workflow**) Determines how spectral similarities from different sets are combined. Possible values are `"mean"`, `"min"` or `"max"`, which calculates the combined value as the mean, minimum or maximum value, respectively. `NA` values (*e.g.* if a set does not have peak list data to combine) are removed in advance.

These parameters are typically passed as a named `list` as the `specSimParams` argument to functions that do spectral similarity calculations. The `getDefSpecSimParams` function can be used to generate such parameter list with defaults.

`suspect-screening` *Target and suspect screening*

Description

Utilities to screen for analytes with known or suspected identity.

Usage

```
## S4 method for signature 'featureGroups'
screenSuspects(
  fGroups,
  suspects,
  rtWindow,
  mzWindow,
  adduct,
  skipInvalid,
  onlyHits
)

## S4 method for signature 'featureGroupsScreening'
screenSuspects(
  fGroups,
  suspects,
  rtWindow,
  mzWindow,
  adduct,
  skipInvalid,
  onlyHits,
  amend = FALSE
)

numericIDLevel(level)

genIDLevelRulesFile(out, inLevels = NULL, exLevels = NULL)

## S4 method for signature 'featureGroupsSet'
screenSuspects(
```

```
  fGroups,
  suspects,
  rtWindow,
  mzWindow,
  adduct,
  skipInvalid,
  onlyHits
)

## S4 method for signature 'featureGroupsScreeningSet'
screenSuspects(
  fGroups,
  suspects,
  rtWindow,
  mzWindow,
  adduct,
  skipInvalid,
  onlyHits,
  amend = FALSE
)

importFeatureGroupsBrukerTASQ(path, analysisInfo, clusterRTWindow = 12)
```

Arguments

`fGroups`	The `featureGroups` object that should be screened.
`suspects`	A `data.frame` with suspect information. See the Suspect list format section below. (**sets workflow**) Can also be a `list` with suspect lists to be used for each set (otherwise the same suspect lists is used for all sets). The `list` can be named with the sets names to mark which suspect list is to be used with which set (*e.g.* `suspects=list(positive=suspsPos,negative=suspsNeg)`).
`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]+"`. May be `NULL`, see Suspect list format and Matching of suspect masses sections below.
`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.
`onlyHits`	If `TRUE` then all feature groups not matched by any of the suspects will be removed.
`amend`	If `TRUE` then screening results will be *amended* to the original object.
`level`	The identification level to be converted.

`out` The file path to the target file.

`inLevels, exLevels`
A regular expression for the identification levels to include or exclude, respectively. For instance, `exLevels="4|5"` would exclude level 4 and 5 from the output file. Set to `NULL` to ignore.

`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 identified with little to no prior knowledge, a common strategy is to screen for compounds with known or suspected identity. This may be a generally favorable approach if possible, as it can significantly reduce the load on data interpretation.

`screenSuspects` is used to perform suspect screening. The input `featureGroups` object will be screened for suspects by *m/z* values and optionally retention times. Afterwards, any feature groups not matched may be kept or removed, depending whether a full non-target analysis is desired.

`numericIDLevel` Extracts the numeric part of a given identification level (*e.g.* `"3a"` becomes '`3`').

`genIDLevelRulesFile` Generates a template YAML file that is used to configure the rules for automatic estimation of identification levels. This file can then be used as input for `annotateSuspects`.

`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.

Value

`screenSuspects` returns a `featureGroupsScreening` object, which is a copy of the input `fGroups` object amended with additional screening information.

`importFeatureGroupsBrukerTASQ` returns a new `featureGroups` object containing converted screening results from Bruker TASQ.

Sets workflows

In a sets workflow, `screenSuspects` performs suspect screening for each set separately, and the screening results are combined afterwards. The `sets` column in the `screenInfo` data marks in which sets the suspect hit was found.

`importFeatureGroupsBrukerTASQ` Is currently not supported for sets workflows.

Suspect list format

the `suspects` argument for `screenSuspects` should be a `data.frame` with the following mandatory and optional columns:

- `name` The suspect name. Must be file-compatible. (**mandatory**)
- `rt` The retention time (in seconds) for the suspect. If specified the suspect will only be matched if its retention matches the experimental value (tolerance defined by the `rtWindow` argument). (**optional**)
- `neutralMass`,`formula`,`SMILES`,`InChI` The neutral monoisotopic mass, chemical formula, SMILES or InChI for the suspect. (data from one of these columns are **mandatory** in case no value from the `mz` column is available for a suspect)
- `mz` The ionized *m/z* of the suspect. (**mandatory** unless it can be calculated from one of the aforementioned columns)
- `adduct` A `character` that can be converted with `as.adduct`. Can be used to automatically calculate values for the `mz` column. (**mandatory** unless data from the `mz` column is available, the `adduct` argument is set or `fGroups` has adduct annotations)
- `fragments_mz`,`fragments_formula` One or more MS/MS fragments (specified as *m/z* or formulae, respectively). Multiple values can be specified by separating them with a semicolon (;). This data is used by `annotateSuspects` to report detected MS/MS fragments and calculate identification levels. (**optional**)

Matching of suspect masses

How the mass of a suspect is matched with the mass of a feature depends on the available data:

- If the suspect has data from the `mz` column of the suspect list, then this data is matched with the detected feature *m/z*.
- Otherwise, if the suspect has data in the `adduct` column of the suspect list, this data is used to calculate its `mz` value, which is then used like above.
- In the last case, the neutral mass of the suspect is matched with the neutral mass of the feature. Hence, either the `adduct` argument needs to be specified, or the `featureGroups` input object must have adduct annotations.

Note

Both `screenSuspects` and `importFeatureGroupsBrukerTASQ` may use the suspect names to base file names used for reporting, logging etc. Therefore, it is important that these are file-compatible names. For this purpose, `screenSuspects` will automatically try to convert long, non-unique and/or otherwise incompatible suspect names.

For `screenSuspects` in some cases you may need to install OpenBabel (*e.g.* when only InChI data is available for mass calculation).

`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 `plotChroms`) the integrated chromatographic peak range shown is incorrect.

References

OBoyle NM, Banck M, James CA, Morley C, Vandermeersch T, Hutchison GR (2011). "Open Babel: An open chemical toolbox." *Journal of Cheminformatics*, **3**(1). doi: 10.1186/17582946333.

Daniel Müllner (2013). fastcluster: Fast Hierarchical, Agglomerative Clustering Routines for R and Python. Journal of Statistical Software, 53(9), 1-18. URL https://www.jstatsoft.org/v53/i09/.

See Also

`featureGroupsScreening`

`TP-generation` *Generation of transformation products (TPs)*

Description

Functionality to automatically obtain transformation products for a given list of parent compounds.

Usage

```
generateTPs(algorithm, ...)

generateTPsBioTransformer(
  parents,
  type = "env",
  steps = 2,
  extraOpts = NULL,
  skipInvalid = TRUE,
  fpType = "extended",
  fpSimMethod = "tanimoto",
  MP = FALSE
)

generateTPsLibrary(
  parents = NULL,
  TPLibrary = NULL,
  skipInvalid = TRUE,
  matchParentsBy = "InChIKey"
)

## S4 method for signature 'featureGroups'
generateTPsLogic(fGroups, minMass = 40, adduct = NULL, transformations = NULL)

## S4 method for signature 'featureGroupsSet'
generateTPsLogic(fGroups, minMass = 40, transformations = NULL)
```

Arguments

algorithm	A character string describing the algorithm that should be used: "biotransformer", "logic", "library"
...	Any parameters to be passed to the selected TP generation algorithm.
parents	The parents for which transformation products should be obtained. This can be (1) a suspect list (see suspect screening for more information), (2) the resulting output screenSuspects or (3) a compounds annotation object. In the former two cases, the suspect (hits) are used as parents, whereas in the latter case all candidates are used as parents. For generateTPsLibrary: can be NULL, in this case TPs for all parents in the library are obtained.
type	The type of prediction. Valid values are: "env", "ecbased", "cyp450", "phaseII", "hgut", "superbio", "allHuman". Sets the -b command line option.
steps	The number of steps for the predictions. Sets the -s command line option.
extraOpts	A character with extra command line options passed to the biotransformer.jar tool.
skipInvalid	If set to TRUE then the parents will be skipped (with a warning) for which insufficient information (*e.g.* SMILES) is available.
fpType	The type of structural fingerprint that should be calculated. See the type argument of the get.fingerprint function of **rcdk**.
fpSimMethod	The method for calculating similarities (i.e. not dissimilarity!). See the method argument of the fp.sim.matrix function of the **fingerprint** package.
MP	If TRUE then multiprocessing is enabled. Since BioTransformer supports native parallelization, additional multiprocessing generally doesn't lead to significant reduction in computational times. Furthermore, enabling multiprocessing can lead to very high CPU/RAM usage.
TPLibrary	If NULL, a default PubChem based library is used. Otherwise, TPLibrary should be a data.frame. See section below.
matchParentsBy	A character that specifies how the input parents are matched with the data from the TP library. Valid options are: "InChIKey", "InChIKey1", "InChI", "SMILES".
fGroups	A featureGroups object for which TPs should be calculated.
minMass	A numeric that specifies the minimum mass of calculated TPs. If below this mass it will be removed.
adduct	An adduct object (or something that can be converted to it with as.adduct). Examples: "[M-H]-", "[M+Na]+". If the featureGroups object has adduct annotations then these are used if adducts=NULL. (**sets workflow**) The adduct argument is not supported for sets workflows, since the adduct annotations will then always be used.
transformations	A data.frame with transformation reactions to be used for calculating the TPs (see section below). If NULL, a default table from Schollee *et al.* is used (see references).

Details

Several algorithms are available that obtain TPs for a set of parent compounds. Depending on the algorithm, the parents can be a suspect list, all the candidates in a compounds object or the screening results or all feature groups from a featureGroups object. The resulting objects can (again depending on the used algorithm) then be used for conversion to suspect lists (convertToSuspects), a MetFrag database (convertToMFDB) or further processed to generate parent to TP links from features (generateComponentsTPs).

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

generateTPsBioTransformer uses BioTransformer to predict TPs. An important advantage of this algorithm is that it provides full structural information for TPs. However, to do so, structural information also needs to be present for the parents. Afterwards, structural similarities between the parent and its TPs are calculated, which can be used to filter the results. In order to use this function the '.jar' command line utility should be installed and specified in the patRoon.path.BioTransformer option. The '.jar' file can be obtained via https://bitbucket.org/djoumbou/biotransformer/src/master.

generateTPsLibrary obtains transformation products from a library. Similar to generateTPsBioTransformer, this algorithm relies and provides structural information for parents/TPs (SMILES). By default, a library is used that is based on data from PubChem. However, it also possible to use your own library.

generateTPsLogic applies *metabolic logic* to predict transformation products. With this algorithm, TPs are predicted from common (environmental) chemical reactions, such as hydroxylation, demethylation etc. The generated TPs result from calculating the mass differences between a parent feature after it underwent the reaction. While this only results in little information on chemical properties of the TP, an advantage of this method is that it does not rely on structural information of the parent, which may be unknown in a full non-target analysis.

Custom TP libraries and transformations

The TPLibrary argument is used to specify a custom TP library for generateTPsLibrary. This should be a data.frame where each row specifies a TP for a parent, with the following columns:

- parent_name and TP_name: The name of the parent/TP.
- parent_SMILES and TP_SMILES The SMILES of the parent structure.
- parent_LogP and TP_LogP The log P values for the parent/TP. (**optional**)
- LogPDiff The difference between parent and TP Log P values. Ignored if *both* parent_LogP and TP_LogP are specified. (**optional**)

Other columns are allowed, and will be included in the final object. Multiple TPs for a single parent are specified by repeating the value within parent_ columns.

Similarly, the transformations argument to generateTPsLogic is used to specify custom rules to calculate transformation products. This should be a data.frame with the following columns:

- transformation The name of the chemical transformation

- add The elements that are added by this reaction (*e.g.* "O").
- sub The elements that are removed by this reaction (*e.g.* "H2O").
- retDir The expected retention time direction relative to the parent (assuming a reversed phase like LC separation). Valid values are: '-1' (elutes before the parent), '1' (elutes after the parent) or '0' (no significant change or unknown).

Parallelization

generateTPsBioTransformer uses multiprocessing to parallelize computations. Please see the parallelization section in the handbook for more details and patRoon options for configuration options.

Source

The algorithm of generateTPsLogic is directly based on the work done by Schollee *et al.* (see references).

Note

When the parents argument is a compounds object, the candidate library identifier is used in case the candidate has no defined compoundName.

References

Djoumbou-Feunang Y, Fiamoncini J, Gil-de-la-Fuente A, Greiner R, Manach C, Wishart DS (2019). "BioTransformer: a comprehensive computational tool for small molecule metabolism prediction and metabolite identification." *Journal of Cheminformatics*, **11**(1). doi: 10.1186/s1332101803245.

Wicker J, Lorsbach T, Gutlein M, Schmid E, Latino D, Kramer S, Fenner K (2015). "enviPath - The environmental contaminant biotransformation pathway resource." *Nucleic Acids Research*, **44**(D1), D502–D508. doi: 10.1093/nar/gkv1229.

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

Schollee JE, Schymanski EL, Avak SE, Loos M, Hollender J (2015). "Prioritizing Unknown Transformation Products from Biologically-Treated Wastewater Using High-Resolution Mass Spectrometry, Multivariate Statistics, and Metabolic Logic." *Analytical Chemistry*, **87**(24), 12121–12129. doi: 10.1021/acs.analchem.5b02905.

See Also

The transformationProducts class and derived classes transformationProductsBT and transformationProductsLibr for methods to post-process TP data.

transformationProducts-class
Base transformation products (TP) class

Description

Holds information for all TPs for a set of parents.

Usage

```
## S4 method for signature 'transformationProducts'
parents(TPs)

## S4 method for signature 'transformationProducts'
products(TPs)

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

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

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

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

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

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

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

## S4 method for signature 'transformationProducts'
convertToSuspects(TPs, includeParents = FALSE)
```

Arguments

TPs, x, object	transformationProducts object to be accessed
i	For [/[[: A numeric or character value which is used to select parents 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 parents are selected. For [[: should be a scalar value.
...	Unused.
drop, j	ignored.
name	The parent name (partially matched).
includeParents	If TRUE then parents are also included in the returned suspect list.

Details

This class holds all generated data for transformation products for a set of parents. The class is virtual and derived objects are created by TP generators.

The TP data in objects from this class include a retDir column. These are numeric values that hint what the the chromatographic retention order of a TP might be compared to its parent: a value of '-1' means it will elute earlier, '1' it will elute later and '0' that there is no significant difference or the direction is unknown. These values are based on a typical reversed phase separation. When generateTPsBioTransformer or generateTPsLibrary was used to generate the data, the retDir values are based on calculated log P values of the parent and its TPs.

Methods (by generic)

- parents: Accessor method for the parents slot of a transformationProducts class.
- products: Accessor method for the products slot.
- length: Obtain total number of transformation products.
- names: Obtain the names of all parents in this object.
- show: Show summary information for this object.
- [: Subset on parents.
- [[: Extracts a table with TPs for a parent.
- $: Extracts a table with TPs for a parent.
- as.data.table: Returns all TP data in a table.
- convertToSuspects: Converts this object to a suspect list that can be used as input for screenSuspects.

Slots

parents A data.table with metadata for all parents that have TPs in this object. Use the parents method for access.

products A list with data.table entries with TP information for each parent. Use the products method for access.

S4 class hierarchy

- workflowStep
  - transformationProducts
    - transformationProductsBT
    - transformationProductsLibrary
    - transformationProductsLogic

See Also

Derived classes transformationProductsBT and transformationProductsLibrary for specific algorithm methods and TP-generation

---

transformationProductsBT-class

Class to store transformation products (TPs) predicted by BioTransformer

---

Description

This class is used to store prediction results that are generated with BioTransformer.

Usage

```
## S4 method for signature 'transformationProductsBT'
convertToMFDB(TPs, out, includeParents = FALSE)

## S4 method for signature 'transformationProductsBT'
filter(
  obj,
  removeParentIsomers = FALSE,
  removeTPIsomers = FALSE,
  removeDuplicates = FALSE,
  minSimilarity = NULL,
  negate = FALSE
)
```

Arguments

out	The file name of the the output MetFrag database.
includeParents	Set to TRUE to include the parents in the database.
obj, TPs	transformationProductsBTs object to be accessed
removeParentIsomers	If TRUE then TPs with an equal formula as their parent (isomers) are removed.

removeTPIsomers
: If TRUE then all TPs with equal formula as any sibling TPs (isomers) are removed. Unlike removeDuplicates, *all* TP candidates are removed (including the first match). This filter automatically sets removeDuplicates=TRUE to avoid complete removal of TPs with equal structure.

removeDuplicates
: If TRUE then the TPs of a parent with duplicate structures (SMILES) are removed. Such duplicates may occur when different transformation pathways yield the same TPs. The first TP candidate with duplicate structure will be kept.

minSimilarity
: Minimum structure similarity ('0-1') that a TP should have relative to its parent. For details on how these similarities are calculated, see the generateTPsBioTransformer function. May be useful under the assumption that parents and TPs who have a high structural similarity, also likely have a high MS/MS spectral similarity (which can be evaluated after componentization with generateComponentsTPs.

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

Details

Objects from this class are generate with generateTPsBioTransformer. This class is derived from the transformationProducts base class, please see its documentation for more details.

Value

filter returns a filtered transformationProductsBT object.

Methods (by generic)

- convertToMFDB: Exports this object as a '.csv' file that can be used as a MetFrag local database.
- filter: Performs rule-based filtering of the BioTransformer predictions. Useful to simplify and clean-up the data.

S4 class hierarchy

- transformationProducts
  - transformationProductsBT

References

Djoumbou-Feunang Y, Fiamoncini J, Gil-de-la-Fuente A, Greiner R, Manach C, Wishart DS (2019). "BioTransformer: a comprehensive computational tool for small molecule metabolism prediction and metabolite identification." *Journal of Cheminformatics*, **11**(1). doi: 10.1186/s1332101803245.

Wicker J, Lorsbach T, Gutlein M, Schmid E, Latino D, Kramer S, Fenner K (2015). "enviPath - The environmental contaminant biotransformation pathway resource." *Nucleic Acids Research*, **44**(D1), D502–D508. doi: 10.1093/nar/gkv1229.

See Also

The base class transformationProducts for more relevant methods and TP-generation

---

transformationProductsLibrary-class

Class to store transformation products (TPs) obtained from a library

---

Description

This class is used to store prediction results that are available in a TP library.

Usage

```
## S4 method for signature 'transformationProductsLibrary'
convertToMFDB(TPs, out, includeParents = FALSE)
```

Arguments

TPs	transformationProductsLibrary object to be accessed
out	The file name of the the output MetFrag database.
includeParents	Set to TRUE to include the parents in the database.

Details

Objects from this class are generate with generateTPsLibrary. This class is derived from the transformationProducts base class, please see its documentation for more details.

Methods (by generic)

- convertToMFDB: Exports this object as a '.csv' file that can be used as a MetFrag local database.

S4 class hierarchy

- transformationProducts
  - transformationProductsLibrary

See Also

The base class transformationProducts for more relevant methods and TP-generation

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()
```

withOpt *Temporarily changes package options*

Description

This function is inspired by `withr::with_options`: it can be used to execute some code where package options are temporarily changed. This function uses a shortened syntax, especially when changing options for `patRoon`.

Usage

```
withOpt(code, ..., prefix = "patRoon.")
```

Arguments

`code`	The code to be executed.
`...`	Named arguments with options to change.
`prefix`	A `character` that will be used to prefix given option names.

Examples

```
## Not run:
# Set max parallel processes to five while performing formula calculations
withOpt(MP.maxProcs = 5, {
    formulas <- generateFormulas(fGroups, "genform", ...)
})

## End(Not run)
```

`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
  - transformationProducts
    - transformationProductsBT
    - transformationProductsLibrary
    - transformationProductsLogic
  - features
    - featuresSet
    - featuresUnset
    - featuresFromFeatGroups
    - featuresConsensus
    - featuresBruker
    - featuresEnviPick
    - featuresKPIC2
    - featuresOpenMS
    - featuresSAFD
    - featuresSIRIUS
    - featuresBrukerTASQ
    - featuresXCMS
    - featuresXCMS3
  - featureGroups
    - featureGroupsSet
      - featureGroupsScreeningSet
    - featureGroupsUnset
    - featureGroupsScreening
      - featureGroupsSetScreeningUnset
    - featureGroupsBruker
    - featureGroupsConsensus
    - featureGroupsEnviMass
    - featureGroupsKPIC2
    - featureGroupsOpenMS
    - featureGroupsSIRIUS
    - featureGroupsBrukerTASQ
    - featureGroupsXCMS
    - featureGroupsXCMS3
  - components
    - componentsCamera
    - componentsFeatures
      - componentsCliqueMS
      - componentsOpenMS
    - componentsClust
      - componentsIntClust

- componentsSpecClust
  - componentsSet
    - componentsNTSet
  - componentsUnset
  - componentsNT
    - componentsNTUnset
  - componentsRC
  - componentsTPs
- featureAnnotations
  - formulas
    - formulasConsensus
    - formulasSet
    - formulasUnset
  - compounds
    - compoundsConsensus
    - compoundsMF
    - compoundsSet
    - compoundsUnset
- MSPeakLists
  - MSPeakListsSet
  - MSPeakListsUnset

`workflowStepSet-class` *(Virtual) base class for sets related workflow objects*

Description

This class is the base for many sets workflows related classes. This class is virtual, and therefore never created directly.

Usage

```
## S4 method for signature 'workflowStepSet'
setObjects(obj)

## S4 method for signature 'workflowStepSet'
sets(obj)

## S4 method for signature 'workflowStepSet'
show(object)
```

Arguments

`obj, object`	An object that is derived from `workflowStepSet`.

Details

The most important purpose of this class is to hold data that is specific for a set. These *set objects* are typically objects with classes from a regular non-sets workflow (*e.g.* `components`, `compounds`), and are used by the sets workflow object to *e.g.* form a consensus. Since the set objects may contain additional data, such as algorithm specific slots, it may in some cases be of interest to access them directly with the `setObjects` method (described below).

Methods (by generic)

- `setObjects`: Accessor for the `setObjects` slot.
- `sets`: Returns the names for each set in this object.
- `show`: Shows summary information for this object.

Slots

`setObjects` A `list` with the *set objects* (see the Details section). The `list` is named with the set names.

S4 class hierarchy

- `workflowStepSet`
  - `componentsSet`
    - `componentsNTSet`
  - `featureGroupsScreeningSet`
  - `compoundsSet`
    - `compoundsConsensusSet`
  - `formulasSet`
    - `formulasConsensusSet`
  - `MSPeakListsSet`

Index