Package 'RFLOMICS'

May 9, 2026

Title Interactive web application for Omics-data analysis

Version 1.5.0

Description R-package with shiny interface, provides a framework for the analysis of transcriptomics, proteomics and/or metabolomics data. The interface offers a guided experience for the user, from the definition of the experimental design to the integration of several omics table together. A report can be generated with all settings and analysis results.

URL https://github.com/RFLOMICS/RFLOMICS

BugReports https://github.com/RFLOMICS/RFLOMICS/issues

Depends R (>= 4.4.0), SummarizedExperiment, MultiAssayExperiment, shinyBS, dplyr, ggplot2, htmltools, knitr, coseq

biocViews ShinyApps, DifferentialExpression, Metabolomics, Proteomics, Transcriptomics

License Artistic-2.0

Encoding UTF-8

RoxygenNote 7.3.3

Suggests testthat, shinytest2, BiocStyle, org.Hs.eg.db

VignetteBuilder knitr

Imports vroom, org.At.tair.db, AnnotationDbi, clusterProfiler, ComplexHeatmap, data.table, DT, edgeR, FactoMineR, ggpubr, ggnetwork, ggrepel, grDevices, grid, httr, limma, magrittr, methods, mixOmics, MOFA2, plotly, purrr, RColorBrewer, reshape2, reticulate, rmarkdown, S4Vectors, shiny, shinydashboard, shinyWidgets, stats, stringr, tidyr, tibble, tidyselect, UpSetR,

SystemRequirements Python (>=3), numpy, pandas, h5py, scipy, argparse, sklearn, mofapy2 (>=0.7.1)

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

git_branch devel

git_last_commit f2a9e77

git_last_commit_date 2026-04-28

Repository Bioconductor 3.24

Date/Publication 2026-05-08

Author Nadia Bessoltane [aut, cre] (ORCID:
<https://orcid.org/0000-0001-6931-2529>),
Delphine Charif [aut] (ORCID: <https://orcid.org/0000-0002-1949-5969>),
Audrey Hulot [aut] (ORCID: <https://orcid.org/0000-0002-9647-6470>),
Christine Paysant-Leroux [aut] (ORCID:
<https://orcid.org/0000-0003-2046-6492>),
Gwendal Cueff [aut]

Maintainer Nadia Bessoltane <nadia.bessoltane@inrae.fr>

Contents

`createRflomicsMAE` *RflomicsMAE class constructor*

Description

This function is a constructor for the class RflomicsMAE-class. It initializes an object of class RflomicsMAE-class from a list of omics datasets, a vector of dataset names, a vector of omics types, and an experimental design.

Usage

```
createRflomicsMAE(
  projectName = NULL,
  omicsData = NULL,
  omicsNames = NULL,
  omicsTypes = NULL,
  ExpDesign = NULL,
  factorInfo = NULL
)
```

Arguments

projectName	Project name
omicsData	list of omics dataset: named list of data.frame, matrix or SummarizedExperiment objects, or MultiAssayExperiment object.
omicsNames	Vector of dataset names that we want to analyze.
omicsTypes	vector of dataset types: "RNAseq", "metabolomics", "proteomics"
ExpDesign	a data.frame which describes the experimental design.
factorInfo	a data.frame describing the experimental factors. • factorName: factor names • factorRef: factor references • factorType: factor type : "Bio", "batch", "Meta" • factorLevels: levels of each factor with "," separation.

Value

An object of class RflomicsMAE-class

See Also

RflomicsMAE-class

Examples

```
# load ecoseed data
data(ecoseed.mae)

factorInfo <- data.frame(
  "factorName"   = c("Repeat", "temperature", "imbibition"),
  "factorType"   = c("batch", "Bio", "Bio")
)

# create rflomicsMAE object with ecoseed data
MAE <- RFLOMICS::createRflomicsMAE(
  projectName = "Tests",
  omicsData   = ecoseed.mae,
  omicsTypes  = c("RNAseq","proteomics","metabolomics"),
  factorInfo  = factorInfo)
names(MAE) <- c("RNAtest", "protetest", "metatest")

# generate upset plot
# MultiAssayExperiment::upsetSamples(MAE)

# generate data overview plot
# plotDataOverview(MAE)

# generate plot of coverage of condition by data
# plotConditionsOverview(MAE)
```

ecoseed.df *Ecoseed project data*

Description

This dataset is provided by the EcoSeed project (FP7-KBBE; Impacts of Environmental Conditions on Seed Quality). that investigates the effect of seed production temperature on the germination potential of Arabidopsis thaliana.

This dataset is a multi-omics dataset composed of three data matrices: transcriptomics (raw RNAseq read count data matrix), metabolomics and proteomics (relative abundance matrix as XIC).

These data are provided in 2 object: ecoseed.df and ecoseed.mae

Usage

```
data("ecoseed.df")
```

Format

A list of data.frame containing

- design: a data.frame with experiment design,
- RNAtest: a data.frame with RNAseq data,
- protetest: a data.frame with proteomics data,
- metatest: a data.frame with metabolomics data

References

FP7-KBBE; Impacts of Environmental Conditions on Seed Quality

Examples

```
data("ecoseed.df")

# list of data.frames
names(ecoseed.df)
head(ecoseed.df$design)
```

ecoseed.mae *Ecoseed project data*

Description

This dataset is provided by the EcoSeed project (FP7-KBBE; Impacts of Environmental Conditions on Seed Quality). that investigates the effect of seed production temperature on the germination potential of Arabidopsis thaliana.

This dataset is a multi-omics dataset composed of three data matrices: transcriptomics (raw RNAseq read count data matrix), metabolomics and proteomics (relative abundance matrix as XIC).

These data are provided in 2 object: ecoseed.df and ecoseed.mae

Usage

```
data("ecoseed.mae")
```

Format

ecoseed.mae: a MultiAssayExperiment object, of RNAtest, protetest and metatest data in SummarizedExperiment

- ExperimentList class object of length 3:
  - RNAtest: a SummarizedExperiment object with RNAseq data,
  - protetest: a SummarizedExperiment object with proteomics data,
  - metatest: a SummarizedExperiment object with metabolomics data
- DataFrame with experiment design
- ...

References

FP7-KBBE; Impacts of Environmental Conditions on Seed Quality

Examples

```
data("ecoseed.mae")
```

---

generateExpressionContrast

Contrast expressions

Description

Generate expression of contrasts based on chosed model formula.

- generateExpressionContrast: This function allows, from a model formulae, to give the expression contrast data frames. Three types of contrasts are expressed:
  - pairwise comparison
  - averaged expression
  - interaction expression
- setSelectedContrasts: Set the selected contrasts stored in `metadata` slot
- getSelectedContrasts: List the selected contrasts

Usage

```
## S4 method for signature 'RflomicsSE'
generateExpressionContrast(object, contrastType = NULL)

## S4 method for signature 'RflomicsMAE'
generateExpressionContrast(object, contrastType = NULL)

## S4 method for signature 'RflomicsMAE'
```

```
setSelectedContrasts(object, contrastList = NULL)

## S4 method for signature 'RflomicsSE'
setSelectedContrasts(object, contrastList = NULL)

## S4 method for signature 'RflomicsMAE'
getSelectedContrasts(object)

## S4 method for signature 'RflomicsSE'
getSelectedContrasts(object)
```

Arguments

object	an object of class RflomicsSE or class RflomicsMAE-class
contrastType	type of contrasts from which the possible contrasts are extracted ("average", "simple", "interaction"). Default is all contrasts types.
contrastList	a data.frame of contrasts generated by generateExpressionContrast

Value

list of 1 or 3 data.frames of contrast expression

an object of RflomicsSE class or RflomicsMAE-class class

Author(s)

Christine Paysant-Le Roux, adapted by Nadia Bessoltane

Examples

```
library(RFLOMICS)

# load ecoseed data
data(ecoseed.mae)

factorInfo <- data.frame(
  "factorName"   = c("Repeat", "temperature", "imbibition"),
  "factorType"   = c("batch", "Bio", "Bio")
)

# create rflomicsMAE object with ecoseed data
MAE <- RFLOMICS::createRflomicsMAE(
  projectName = "Tests",
  omicsData   = ecoseed.mae,
  omicsTypes  = c("RNAseq","proteomics","metabolomics"),
  factorInfo  = factorInfo)

# generate all statistical model formulae
formulae <- generateModelFormulae(MAE)

# chose and set model formula to rflomicsMAE object
MAE <- setModelFormula(MAE, formulae[[1]])

# Generate expression of contrasts from chosen model
contrastList <- generateExpressionContrast(MAE, "averaged")
```

```
# Set the contrasts List and choose the first 3 contrasts of type averaged
MAE <- setSelectedContrasts(MAE, contrastList = contrastList[c(1,2,3),])
```

`generateModelFormulae` *Statistical model formulae*

Description

These methods allow the user to select and set the formula of the statistical method.

- generateModelFormulae(): From a vector of character giving the name of the factors of an omics experiment, and their type of effect: biological or batch, it returns all models formulae that can be formulated in association with this factors. Batch effect factors do not appear in interaction terms with biological factor. Model formulae stop in second order interaction.
- setModelFormula(): Set the model formula stored in `metadata` slot
- getModelFormula: Access to the model formula of the statistical analysis

Usage

```
## S4 method for signature 'RflomicsMAE'
generateModelFormulae(object)

## S4 method for signature 'RflomicsMAE'
setModelFormula(object, modelFormula = NULL)

## S4 method for signature 'RflomicsSE'
setModelFormula(object, modelFormula = NULL)

## S4 method for signature 'RflomicsMAE'
getModelFormula(object)

## S4 method for signature 'RflomicsSE'
getModelFormula(object)
```

Arguments

`object`	an object of class RflomicsSE or class RflomicsMAE-class
`modelFormula`	a string of model formula generated by generateModelFormulae

Value

a named list of object of class formula

an object of RflomicsSE class or RflomicsMAE-class class

Examples

```
library(RFLOMICS)

# load ecoseed data
data(ecoseed.mae)

factorInfo <- data.frame(
  "factorName"   = c("Repeat", "temperature", "imbibition"),
  "factorType"   = c("batch", "Bio", "Bio")
)

# create rflomicsMAE object with ecoseed data
MAE <- RFLOMICS::createRflomicsMAE(
  projectName = "Tests",
  omicsData   = ecoseed.mae,
  omicsTypes  = c("RNAseq","proteomics","metabolomics"),
  factorInfo  = factorInfo)

# generate all statistical model formulae
formulae <- generateModelFormulae(MAE)

# chose and set model formula to rflomicsMAE object
MAE <- setModelFormula(MAE, formulae[[1]])

# Generate expression of contrasts from chosen model
contrastList <- generateExpressionContrast(MAE, "averaged")

# Set the contrasts List and choose the first 3 contrasts of type averaged
MAE <- setSelectedContrasts(MAE, contrastList = contrastList[c(1,2,3),])
```

`generateReport` *Generate RFLOMICS html report or archive*

Description

This function is used to generate a html report from a RflomicsMAE-class object or archive with results.

Usage

```
## S4 method for signature 'RflomicsMAE'
generateReport(
  object,
  reportName = NULL,
  archiveName = NULL,
  tmpDir = NULL,
  ...
)
```

Arguments

object	a object of RflomicsSE class or RflomicsMAE-class class.
reportName	Name of the html report.
archiveName	Name of archive with all analysis results
tmpDir	temporary directory (default: working directory)
...	other arguments to pass into the render function.

Value

An html report or archive (tar.gz)

Examples

```
library(RFLOMICS)
# load ecoseed data
data(ecoseed.mae)

factorInfo <- data.frame(
  "factorName"   = c("Repeat", "temperature", "imbibition"),
  "factorType"   = c("batch", "Bio", "Bio")
)

# create rflomicsMAE object with ecoseed data
MAE <- createRflomicsMAE(
  projectName = "Tests",
  omicsData   = ecoseed.mae,
  omicsTypes  = c("RNAseq","proteomics","metabolomics"),
  factorInfo  = factorInfo)

formulae <- generateModelFormulae(MAE)
MAE <- setModelFormula(MAE, modelFormula = formulae[[1]])

selectedContrasts <-
  generateExpressionContrast(MAE, contrastType="simple")

MAE <- setSelectedContrasts(MAE, contrastList = selectedContrasts)

## data processing
MAE <- runDataProcessing(
  object = MAE,
  SE.name = "protetest",
  samples=NULL,
  normalize = list(normMethod="none"),
  transform = list(transformMethod="none"))

## diff analysis
MAE <- runDiffAnalysis(
  object = MAE,
  SE.name = "protetest",
  contrastList =
    selectedContrasts,
  p.adj.method="BH",
  method = "limmalmFit",
  p.adj.cutoff = 0.05,
  logFC.cutoff = 0)
```

```
## Enrichment
# MAE <- runAnnotationEnrichment(
#   object = MAE,
#   SE.name = "protetest",
#   database = "GO",
#   domain = c("MF"),
#   list_args = list(OrgDb = "org.At.tair.db",
#                    keyType = "TAIR",
#                    pvalueCutoff = 0.05))

# get name of performed analysis
getAnalyzedDatasetNames(MAE)

# generate report
#generateReport(object = MAE, reportName = "ecoseed_report.html")
```

`getAnalysis` *Get results from RFLOMICS object*

Description

Get a specific analysis results from a RflomicsMAE or a SE.

- getAnalyzedDatasetNames: return a list of performed analysis names.

Usage

```
## S4 method for signature 'RflomicsMAE'
getAnalysis(object, name = NULL, subName = NULL)

## S4 method for signature 'RflomicsSE'
getAnalysis(object, name = NULL, subName = NULL)

## S4 method for signature 'RflomicsMAE'
getAnalyzedDatasetNames(object, analyses = NULL)
```

Arguments

`object`	The RflomicsMAE or RflomicsSE object from which to extract the analysis.
`name`	the name of element to add to metadata slot.
`subName`	the name of sub element to add to metadata slot.
`analyses`	vector of list of analysis name

Value

The analysis metadata slot (a list of results)

- getAnalysis: return list of results from a specific analysis.

Examples

```
# See generateReport for an example that includes getAnalyzedDatasetNames
```

`prepareForIntegration,RflomicsMAE-method`
Preparation step for integration

Description

This function transforms a RflomicsMAE produced by rflomics into an untrained MOFA object or a list to use for mixOmics. It checks for batch effects to correct them before integration. It also transforms RNASeq counts data into continuous data using `voom`. This is the second step into the integration.

Usage

```
## S4 method for signature 'RflomicsMAE'
prepareForIntegration(
  object,
  omicsNames = NULL,
  rnaSeq_transfo = "limma (voom)",
  variableLists = NULL,
  group = NULL,
  method = "MOFA",
  transformData = TRUE,
  cmd = FALSE
)
```

Arguments

`object`	An object of class RflomicsMAE-class. It is expected the MAE object is produced by RFLOMICS previous analyses, as it relies on their results.
`omicsNames`	vector of characters strings, referring to the names of the filtered table in 'object@ExperimentList'.
`rnaSeq_transfo`	character string, only supports 'limma (voom)' for now. Transformation of the rnaSeq data from counts to continuous data.
`variableLists`	list of variables to keep per dataset. Default is keeping all features.
`group`	Not implemented yet in the interface. Useful for MOFA2 run.
`method`	one of MOFA or mixOmics. Method for which the object is prepared.
`transformData`	boolean. Transform the data with the transform and normalization method? Default is TRUE.
`cmd`	used in the interface. Print cmd lines.

Value

An untrained MOFA object or a list of dataset

Examples

```
library(RFLOMICS)

# load ecoseed data
data("ecoseed.mae")

factorInfo <- data.frame(
  "factorName"   = c("Repeat", "temperature", "imbibition"),
  "factorType"   = c("batch", "Bio", "Bio")
)

# create rflomicsMAE object with ecoseed data
MAE <- RFLOMICS::createRflomicsMAE(
  projectName = "Tests",
  omicsData   = ecoseed.mae,
  omicsTypes  = c("RNAseq","proteomics","metabolomics"),
  factorInfo  = factorInfo)
names(MAE) <- c("RNAtest", "protetest", "metatest")

formulae <- generateModelFormulae( MAE)
MAE <- setModelFormula(MAE, formulae[[1]])
contrastList <- Reduce(rbind, generateExpressionContrast(MAE))

MAE <- MAE |>
    setSelectedContrasts(contrastList[c(3,6,25)]) |>
    runDataProcessing(SE.name = "metatest",
                      transform = list(transformMethod = "log2"),
                      normalize = list(normMethod = "median")) |>
    runDataProcessing(SE.name = "protetest",
                      transform = list(transformMethod = "none"),
                      normalize = list(normMethod = "median")) |>
    runDiffAnalysis(SE.name = "metatest", method = "limmalmFit") |>
    runDiffAnalysis(SE.name = "protetest", method = "limmalmFit")

# Integration using MOFA
# Prepare mofa object:
mofaObj <- prepareForIntegration(MAE,
                                 omicsNames = c("protetest", "metatest"),
                                 variableLists = rownames(MAE),
                                 method = "MOFA")
class(mofaObj)

# Integration using MixOmics
mixObj <- prepareForIntegration(MAE,
                                omicsNames = c("protetest", "metatest"),
                                variableLists = rownames(MAE),
                                method = "mixOmics")
class(mixObj)
```

RflomicsMAE-class *RflomicsMAE Class*

Description

RflomicsMAE is a class that extends the MultiAssayExperiment class by imposing a structure to the metadata slot. This class is used by the Rflomics analysis workflow to store the experimental design, the settings and results of a multi-omics integration analysis.

Usage

```
## S4 method for signature 'RflomicsMAE'
getProjectName(object)

## S4 method for signature 'RflomicsMAE'
getDesignMat(object)

## S4 method for signature 'RflomicsMAE'
getDatasetNames(object)

## S4 method for signature 'RflomicsMAE'
getOmicsTypes(object)

## S4 method for signature 'RflomicsMAE'
getFactorNames(object)

## S4 method for signature 'RflomicsMAE'
getFactorTypes(object)

## S4 method for signature 'RflomicsMAE'
getBioFactors(object)

## S4 method for signature 'RflomicsMAE'
getBatchFactors(object)

## S4 method for signature 'RflomicsMAE'
getMetaFactors(object)

## S4 method for signature 'RflomicsMAE'
getRflomicsSE(object, datasetName = NULL)

## S4 method for signature 'RflomicsMAE'
getFactorModalities(object, factorName)

## S4 method for signature 'RflomicsMAE'
subRflomicsMAE(object, omicNames = NULL)

## S4 method for signature 'RflomicsMAE'
plotDataOverview(
  object,
  omicNames = NULL,
  realSize = FALSE,
  raw = FALSE,
  completeCases = FALSE
)
```

```
## S4 method for signature 'RflomicsMAE'
plotConditionsOverview(object, omicNames = NULL)
```

Arguments

object	An object of class RflomicsMAE-class
datasetName	the name of the RflomicsSE to retrieve
factorName	factor name
omicNames	a vector with dataset names
realSize	booleen value, influence the display size
raw	boolean. If TRUE, displays the raw data without any selection. If FALSE, displays the data with removed samples.
completeCases	boolean. If true, only shows the complete cases of the object.

Value

A RflomicsMAE object.

Slots

- ExperimentList:
  - A ExperimentList class object of RflomicsSE object for each assay dataset
- colData: see MultiAssayExperiment
- sampleMap: see MultiAssayExperiment
- metadata:
  - projectName: string. Project name.
  - omicList: list. Contains the list of omics datasets, with the type and name.
  - design: The experimental design.
  - IntegrationAnalysis: A list containing the multi-omics integration analysis settings and results.
  - design: The experimental design
  - sessionInfo:
  - IntegrationAnalysis: A list containing the multi-omics integration analysis settings and results.

Consructor

createRflomicsMAE

Accessors

- getProjectName: return a string with the name of the project
- getDesignMat: return a data.frame with experimental design.
- getDatasetNames: return a vector with dataset names.
- getOmicsTypes: return a named vector with omics type of each dataset ("RNAseq", "proteomics", "metabolomics")

- getFactorNames: return a vector with the experimental factor names.
- getFactorTypes: return a named vector with experimental factor types ("bio", "batch" or "meta").
- getBioFactors: return a vector with the biological factor names.
- getBatchFactors: return a vector with the batch factor names.
- getMetaFactors: return a vector with the metadata factor names.
- getRflomicsSE: return a RflomicsSE object with selected dataset
- getFactorModalities: return a vector with the modality names of selected factor.
- subRflomicsMAE: return a RflomicsMAE-class object with selected datasets.

Plots

- plotDataOverview: This function plot an overview of the loaded datasets displaying per sample (n=number of entities (genes/metabolites/proteins); k=number of samples)
- plotConditionsOverview: A complete design and at least one biological and one batch factors are required for using RFLOMICS workflow.

Methods

`generateModelFormulae` `generateExpressionContrast` `runDataProcessing` `runDataProcessing` `runDiffAnalysis` `runCoExpression` `runAnnotationEnrichment`

See Also

`MultiAssayExperiment`

Examples

```
# load ecoseed data
data(ecoseed.mae)

factorInfo <- data.frame(
  "factorName"   = c("Repeat", "temperature", "imbibition"),
  "factorType"   = c("batch", "Bio", "Bio")
)

# create rflomicsMAE object with ecoseed data
MAE <- RFLOMICS::createRflomicsMAE(
  projectName = "Tests",
  omicsData   = ecoseed.mae,
  omicsTypes  = c("RNAseq","proteomics","metabolomics"),
  factorInfo  = factorInfo)
names(MAE) <- c("RNAtest", "protetest", "metatest")

# generate upset plot
# MultiAssayExperiment::upsetSamples(MAE)

# generate data overview plot
# plotDataOverview(MAE)
```

```
# generate plot of coverage of condition by data
# plotConditionsOverview(MAE)
# See createRflomicsMAE for an example that includes plotDataOverview
# See createRflomicsMAE for an example that includes plotConditionsOverview
```

`rflomicsMAE2MAE` *convert a RflomicsMAE to a MultiAssayExperiment*

Description

Convert RflomicsMAE object to MultiAssayExperiment object.

Usage

```
## S4 method for signature 'RflomicsMAE'
rflomicsMAE2MAE(object, raw = FALSE)
```

Arguments

`object`	The RflomicsMAE object to convert.
`raw`	Boolean. If TRUE raw omics data values.

Value

object of MultiAssayExperiment class.

`RflomicsSE-class` *RflomicsSE Class*

Description

RflomicsSE is a class that extends the SummarizedExperiment by imposing a structure on the metadata slot. This class is used by the Rflomics analysis workflow to store the experimental design, the settings and results of a single omic analysis. The slot metadata is structured as follows:

Usage

```
## S4 method for signature 'RflomicsSE'
getDesignMat(object)

## S4 method for signature 'RflomicsSE'
getDatasetNames(object)

## S4 method for signature 'RflomicsSE'
getOmicsTypes(object)

## S4 method for signature 'RflomicsSE'
getFactorNames(object)
```

```
## S4 method for signature 'RflomicsSE'
getFactorTypes(object)

## S4 method for signature 'RflomicsSE'
getBioFactors(object)

## S4 method for signature 'RflomicsSE'
getBatchFactors(object)

## S4 method for signature 'RflomicsSE'
getMetaFactors(object)

## S4 method for signature 'RflomicsSE'
getFactorModalities(object, factorName)
```

Arguments

`object`	An object of class RflomicsSE
`factorName`	factor name

Value

A `RflomicsSE` object.

Slots

See SummarizedExperiment

The slot metadata is structured as follows:

- omicType: the type of omics dataset
- design: experimental design
- DataProcessing: a list containing the data processing settings and results
- PCAlist: a list containing the PCA settings and results
- DiffExpAnal: a list containing the Differential Analysis settings and results
- CoExpAnal: a list containing the Coexpression Analysis settings and results
- DiffExpEnrichAnal: a list containing the enrichment analysis of the list of DE features settings and results
- CoExpEnrichAnal: a list containing the enrichment analysis of the list of co-expressed features settings and results

Accessors

- getDesignMat: return a data.frame with experimental design.
- getDatasetNames: return a string with dataset name.
- getOmicsTypes: return a named vector with omics type of dataset ("RNAseq", "proteomics", "metabolomics")
- getFactorNames: return a vector with the experimental factor names.

- getFactorTypes: return a named vector with experimental factor types ("bio", "batch" or "meta").
- getBioFactors: return a vector with the biological factor names.
- getBatchFactors: return a vector with the batch factor names.
- getMetaFactors: return a vector with the metadata factor names.
- getFactorModalities: return a vector with the modality names of selected factor.

See Also

SummarizedExperiment

`runAnnotationEnrichment` *Run Gene Enrichment Analysis and process results*

Description

This function performs over representation analysis (ORA) using clusterprofiler functions. It can be used with custom annotation file (via enricher), GO (enrichGO) or KEGG (enrichKEGG) annotations.

Usage

```
## S4 method for signature 'RflomicsSE'
runAnnotationEnrichment(
  object,
  featureList = NULL,
  from = "DiffExp",
  universe = NULL,
  database = "custom",
  domain = "no-domain",
  annotation = NULL,
  OrgDb = NULL,
  organism = NULL,
  keyType = NULL,
  pvalueCutoff = 0.05,
  qvalueCutoff = 1,
  minGSSize = 10,
  maxGSSize = 500,
  ...
)

## S4 method for signature 'RflomicsMAE'
runAnnotationEnrichment(
  object,
  SE.name,
  featureList = NULL,
  from = "DiffExp",
```

```
  universe = NULL,
  database = "custom",
  domain = "no-domain",
  annotation = NULL,
  OrgDb = NULL,
  organism = NULL,
  keyType = NULL,
  pvalueCutoff = 0.05,
  qvalueCutoff = 1,
  minGSSize = 10,
  maxGSSize = 500,
  ...
)

## S4 method for signature 'RflomicsSE'
plotClusterProfiler(
  object,
  featureListName = NULL,
  database = NULL,
  domain = "no-domain",
  plotType = "dotplot",
  showCategory = 15,
  searchExpr = "",
  nodeLabel = "all",
  p.adj.cutoff = NULL,
  ...
)

## S4 method for signature 'RflomicsSE'
plotEnrichComp(
  object,
  from = "DiffExp",
  database = NULL,
  domain = "no-domain",
  matrixType = "presence",
  clustering = TRUE,
  ...
)

## S4 method for signature 'RflomicsSE'
getEnrichRes(
  object,
  featureListName = NULL,
  from = "DiffExp",
  database = "GO",
  domain = NULL
)

## S4 method for signature 'RflomicsMAE'
getEnrichRes(
  object,
  experiment,
```

```
  featureListName = NULL,
  from = "DiffExp",
  database = "GO",
  domain = NULL
)

## S4 method for signature 'RflomicsSE'
sumORA(object, from = "DiffExp", database = NULL, featureListName = NULL)

## S4 method for signature 'RflomicsSE'
getEnrichSettings(object, from = "DiffExp", database = "GO")

## S4 method for signature 'RflomicsMAE'
getAnnotAnalysesSummary(
  object,
  from = "DiffExp",
  listNames = NULL,
  omicNames = NULL,
  databaseList = NULL,
  ...
)
```

Arguments

object	An object of class RflomicsSE or class RflomicsMAE-class
featureList	name of contrasts (tags or names) from which to extract DE genes if from is DiffExpAnal.
from	indicates if the enrichment results are taken from differential analysis results (DiffExp) or from the co-expression analysis results (CoExp)
universe	description
database	which database (GO, KEGG, custom...)
domain	the subonology or subdomain for the database (eg CC, MF or BP for GO.)
annotation	for custom annotation, a data frame of the annotation. The data frame must contains at least two columns: gene and term, with the omics name and the associated term id respectively. A column name can be added with the full name of the term (if term is not the full name already). The column domain can be used to indicate either different databases (grouped analyses of kegg and go for example) or different domains for a single database (CC, MF and BP) for GO.
OrgDb	OrgDb (with enrichGO)
organism	supported organism listed in 'https://www.genome.jp/kegg/catalog/org_list.html' (with enrichKEGG)
keyType	keytype of input gene with enrichGO (one of "kegg", 'ncbi-geneid', 'ncbi-proteinid' and 'uniprot' with enrichKEGG)
pvalueCutoff	adjusted pvalue cutoff on enrichment tests to report
qvalueCutoff	qvalue cutoff on enrichment tests to report as significant. Tests must pass i) pvalueCutoff on unadjusted pvalues, ii) pvalueCutoff on adjusted pvalues and iii) qvalueCutoff on qvalues to be reported.
minGSSize	minimal size of genes annotated by Ontology term for testing.
maxGSSize	maximal size of genes annotated for testing

...	more arguments
SE.name	SE.name the name of the dataset if the input object is a RflomicsMAE-class
featureListName	the contrastName or clusterName to retrieve the results from. If NULL, all results are returned.
plotType	type of plot. Define the function used inside. One of dotplot, heatplot or cnetplot.
showCategory	max number of terms to show.
searchExpr	expression to search in the showCategory terms.
nodeLabel	same as in enrichplot::cnetplot function, defines
p.adj.cutoff	pvalueCutoff to define the enrichment threshold.
matrixType	Heatmap matrix to plot, one of GeneRatio, p.adjust or presence.
clustering	if TRUE, a hierarchical clustering is performed on rows and columns and they will be ordered accordingly. It will not be displayed as a dendrogram.
experiment	if the object is a RflomicsMAE, then experiment is the name of the RflomicsSE to look for.
listNames	List of names of differential expression feature lists (from = "DiffExp") or list of names of annotated co-expression clusters (from = "CoExp")
omicNames	List of dataset names
databaseList	List of used databases

Value

A RflomicsMAE or a RflomicsSE, depending on the class of object parameter. The enrichment results are added to the metadata slot, either in DiffExpEnrichAnal or CoExpEnrichAnal.

a list of tables or a table

a list with all settings

Accessors

A set of getters and setters generic functions to access and modify objects of the slot metadata of a RflomicsMAE-class object or a RflomicsMAE-class object.

- getEnrichRes: get a particular enrichment result. return enrichment results given in the form of lists of clusterprofiler results.
- sumORA: Get summary tables from ORA analyses - once an enrichment has been conducted.
- getEnrichSettings: get the settings of an enrichment analysis.
- getAnnotAnalysesSummary: return A list of heatmaps, one for each ontology/domain.

Plots

A collection of functions for plotting results from omics analysis steps.

- plotClusterProfiler: Plot a dotplot, a cnetplot or an heatplot, using enrichplot package. It is a wrapper method destined for the RflomicsSE class..
- plotEnrichComp: plot an heatmap of all the enriched term found for a given database and a given source (differential analysis or coexpression clusters). Allow for the comparison of several enrichment results.

Examples

```
# load ecoseed data
library(RFLOMICS)
data(ecoseed.mae)

factorInfo <- data.frame(
  "factorName"   = c("Repeat", "temperature", "imbibition"),
  "factorType"   = c("batch", "Bio", "Bio")
)

# create rflomicsMAE object with ecoseed data
MAE <- createRflomicsMAE(
  projectName = "Tests",
  omicsData   = ecoseed.mae,
  omicsTypes  = c("RNAseq","proteomics","metabolomics"),
  factorInfo  = factorInfo)
#names(MAE) <- c("RNAtest", "protetest", "metatest")

# Set the statistical model and contrasts to test
formulae <- generateModelFormulae(MAE)
MAE <- setModelFormula(MAE, formulae[[1]])

# Get the contrasts List and choose the first 3 contrasts of type averaged
contrastList <- generateExpressionContrast(MAE, "averaged")

MAE <- setSelectedContrasts(MAE, contrastList = contrastList[c(1, 2, 3),])

# Run the data preprocessing and perform the differential analysis
MAE <- runDataProcessing(MAE, SE.name = "protetest",
                         transform = list(transformMethod = "log2"),
                         normalize = list(normMethod = "median"))

MAE <- runDiffAnalysis(MAE,
                       SE.name = "protetest",
                       method = "limmalmFit")

# Run GO annotation (enrichGO)
# Not run: need org.At.tair.db package
# MAE <- runAnnotationEnrichment(MAE, SE.name = "protetest",
#                                OrgDb = "org.At.tair.db",
#                                keyType = "TAIR",
#                                pvalueCutoff = 0.05),
#                                from = "DiffExp", database = "GO",
#                                domain = "CC")

# Run KEGG annotation (enrichKEGG)
# need internet connection
# MAE <- runAnnotationEnrichment(MAE, SE.name = "protetest",
#                                organism = "ath",
#                                keyType = "kegg",
#                                pvalueCutoff = 1,
#                                from = "DiffExp", database = "KEGG")


# Search for the pvalue cutoff:
# sumORA(MAE[["protetest"]], from = "DiffExp", database = "KEGG")
```

```
# From differential analysis proteins lists:
# plotClusterProfiler(MAE[["protetest"]],
#                  featureListName = "(temperatureElevated - temperatureMedium) in mean",
#                     database = "KEGG", from = "DiffExp",
#                     plotType = "heatplot", p.adj.cutoff = 0.05,
#                     domain = "no-domain")
#
# plotEnrichComp(MAE[["protetest"]], from = "DiffExp",
#                database = "KEGG", matrixType = "FC")

# Get all results from KEGG on differential expression lists:
# results <- getEnrichRes(MAE[["protetest"]],
#                         from = "diffexp", database = "KEGG")

# Search for the pvalue cutoff:
# usedPvalue <-
#     getEnrichPvalue(MAE[["protetest"]], from = "diffexp", database = "KEGG")
# settings <-
#     getEnrichSettings(MAE[["protetest"]], from = "diffexp", database = "KEGG")
```

`runCoExpression` *Run CoExpression Analysis and process results*

Description

This method performs a co-expression/co-abundance analysis of omic-data.

Usage

```
## S4 method for signature 'RflomicsSE'
runCoExpression(
  object,
  K = 2:20,
  replicates = 5,
  contrastNames = NULL,
  merge = "union",
  model = "Normal",
  GaussianModel = NULL,
  transformation = NULL,
  normFactors = NULL,
  meanFilterCutoff = NULL,
  scale = NULL,
  min.data.size = 100,
  ...
)

## S4 method for signature 'RflomicsMAE'
runCoExpression(
  object,
  SE.name,
  K = 2:20,
```

```
  replicates = 5,
  contrastNames = NULL,
  merge = "union",
  model = "Normal",
  GaussianModel = NULL,
  transformation = NULL,
  normFactors = NULL,
  meanFilterCutoff = NULL,
  scale = NULL,
  min.data.size = 100,
  ...
)

## S4 method for signature 'RflomicsMAE'
getCoExpAnalysesSummary(object, omicNames = NULL)

## S4 method for signature 'RflomicsSE'
plotCoExpression(object)

## S4 method for signature 'RflomicsMAE'
plotCoExpression(object, SE.name)

## S4 method for signature 'RflomicsSE'
plotCoExpressionProfile(
  object,
  cluster = 1,
  condition = "groups",
  features = NULL
)

## S4 method for signature 'RflomicsMAE'
plotCoExpressionProfile(
  object,
  SE.name,
  cluster = 1,
  condition = "groups",
  features = NULL
)

## S4 method for signature 'RflomicsSE'
plotCoseqContrasts(object)

## S4 method for signature 'RflomicsMAE'
plotCoseqContrasts(object, SE.name)

## S4 method for signature 'RflomicsSE'
getCoexpSettings(object)

## S4 method for signature 'RflomicsMAE'
getCoexpSettings(object, SE.name)

## S4 method for signature 'RflomicsSE'
```

```
getCoexpClusters(object, clusterName = NULL)

## S4 method for signature 'RflomicsMAE'
getCoexpClusters(object, SE.name, clusterName = NULL)
```

Arguments

object	An object of class RflomicsSE or class RflomicsMAE-class
K	Number of clusters (a single value or a vector of values)
replicates	The number of iteration for each K.
contrastNames	names of the contrasts from which the DE entities have to be taken. Can be NULL, in that case every contrasts from the differential analysis are taken into consideration.
merge	"union" or "intersection"
model	Type of mixture model to use "Poisson" or "normal". By default, it is the normal.
GaussianModel	Type of GaussianModel to be used for the Normal mixture model only. This parameters is set to "Gaussian_pk_Lk_Ck" by default and doesn't have to be changed except if an error message proposed to try another model like "Gaussian_pk_Lk_Bk".
transformation	The transformation type to be used. By default, it is the "arcsin" one.
normFactors	The type of estimator to be used to normalize for differences in library size. By default, it is the "TMM" one.
meanFilterCutoff	a cutoff to filter a gene with a mean expression lower than this value. (only for RNAseq data, set to NULL for others).
scale	Boolean. If TRUE scale all variables to unit variance.
min.data.size	The minimum allowed number of variables (default: 100)
...	Additional arguments.
SE.name	SE.name the name of the dataset if the input object is a RflomicsMAE-class
omicNames	the name of the experiment to summarize.
cluster	cluster number
condition	Default is group.
features	Default is NULL.
clusterName	name of the cluster

Details

For now, only the coseq function of the coseq package is used. For RNAseq data, parameters used are those recommended in DiCoExpress workflow (see the reference). This parameters are: model="normal", transformation="arcsin", GaussianModel="Gaussian_pk_Lk_Ck", normFactors="TMM", meanFilterCutoff = 50 For proteomics or metabolomics, data are scaled by protein or metabolite to group them by expression profiles rather than by expression intensity. After data scaling, recommended parameters (from coseq developers) for co-expression analysis are: model="normal", transformation="none", GaussianModel="Gaussian_pk_Lk_Ck", normFactors="none", meanFilterCutoff = NULL.

Value

An S4 object of class RflomicsSE All the results are stored as a named list `CoExpAnal` in the metadata slot of a given `RflomicsSE` object. Objects are: The runCoExpression method return several results, for coseq method, objects are:

- `settings`: co-expression analysis settings. See `getCoexpSetting`
- `results`: boolean indicating if the co-expression analysis succeed
  - `coseqResults`: the raw results of `coseq`
  - `clusters`: a List of clusters
  - `cluster.nb`: The number of cluster
  - `plots`: The plots of `coseq` results
  - `stats`: A tibble summarizing failed jobs: reason, proportion, if any
- `errors`: error list.

Accessors

A set of getters and setters generic functions to access and modify objects of the slot metadata of a RflomicsMAE-class object or a RflomicsMAE-class object.

- getCoexpSettings: Access to the co-expression analysis settings of a given omics dataset
- getCoexpClusters: get members of a cluster. return The list of entities inside this cluster.

Plots

A collection of functions for plotting results from omics analysis steps.

- getCoExpAnalysesSummary: ...
- plotCoExpression: list plot of ICL, logLikelihood and coseq object with min ICL
- plotCoExpressionProfile: ...
- plotCoseqContrasts: This function describes the composition of clusters according to the contrast to which the gene belongs

References

Lambert, I., Paysant-Le Roux, C., Colella, S. et al. DiCoExpress: a tool to process multifactorial RNAseq experiments from quality controls to co-expression analysis through differential analysis based on contrasts inside GLM models. Plant Methods 16, 68 (2020).

See Also

`coseq`

`createRflomicsMAE`

`generateModelFormulae`

`generateExpressionContrast`

`runDataProcessing`

`runDiffAnalysis`

Examples

```
# load ecoseed data
data(ecoseed.mae)

factorInfo <- data.frame(
  "factorName"   = c("Repeat", "temperature", "imbibition"),
  "factorType"   = c("batch", "Bio", "Bio")
)

# create rflomicsMAE object with ecoseed data
MAE <- RFLOMICS::createRflomicsMAE(
  projectName = "Tests",
  omicsData   = ecoseed.mae,
  omicsTypes  = c("RNAseq","proteomics","metabolomics"),
  factorInfo  = factorInfo)

# Set the statistical model and contrasts to test
formulae <- generateModelFormulae(MAE)
MAE <- setModelFormula(MAE, formulae[[1]])

# Get the contrasts List and choose the first 3 contrasts of type averaged
contrastList <- generateExpressionContrast(MAE, "averaged")

MAE <- setSelectedContrasts(MAE, contrastList = contrastList[c(1, 2, 3),])

# Run the data preprocessing and perform the differential analysis
MAE <- runDataProcessing(MAE, SE.name = "protetest",
                         transform = list(transformMethod = "log2"),
                         normalize = list(normMethod = "median"))
MAE <- runDiffAnalysis(MAE, SE.name = "protetest")

# Run co-expression analysis
MAE <- runCoExpression(MAE, SE.name = "protetest",
                       K = 2:5, replicates = 5,
                       merge = "union")

# get parametres used to run co-expression analysis
coExp.set.list <- getCoexpSettings(MAE[["protetest"]])
coExp.set.list$method

# get results
clusters <- getCoexpClusters(MAE[["protetest"]], clusterName = "cluster_1")

# plots
#plotCoExpression(MAE[["protetest"]])

#plotCoExpressionProfile(MAE[["protetest"]], cluster = 2)

#plotCoseqContrasts(MAE[["protetest"]])
```

---

runDataProcessing *Omic Data Exploratory and Preprocessing*

Description

These functions applied a data processing (filtering, normalization and/or transformation, PCA) on RNAseq, proteomics, or metabolomics data.

runDataProcessing() calls the following functions:

- runSampleFiltering: This function applied sample filtering on an dataset.
- **runTransformData**: Performs feature filtering depending on the omics data type. For **RNA-seq data**, lowly expressed transcripts are removed based on count statistics. For **proteomics and metabolomics data**, missing value filtering is applied.
- runTransformData: This function applied a transformation to the dataset. The transformation method is chosen according to the dataset omicstype (RNAseq: none, metabolomics/proteomics: log2 or log10)
- runNormalization: This function applied a normalization on a dataset. The normalization method is chosen according to the dataset omics type (RNAseq: TMM, metabolomics/proteomics: median)
- runMVImputation: Missing value imputation approach, applied to proteomics and metabolomics data, replaces missing values (0 or NA) with the minimum value among all non-zero values. Additionally, variables with at least one condition group without any missing values are retained without further filtering.
- runOmicsPCA: This function performs a principal component analysis on omic data stored in an object of class RflomicsSE-class Results are stored in the metadata slot of the same object. If a "Normalization" slot is present in the metadata slot, then data are normalized before running the PCA according to the indicated transform method.

This function performs a principal component analysis on omic data stored in an object of class RflomicsSE-class Results are stored in the metadata slot of the same object. If a "Normalization" slot is present in the metadata slot, then data are normalized before running the PCA according to the indicated transform method.

- checkExpDesignCompleteness: return a string with message. This method checks some experimental design characteristics. A complete design (all combinations of factor modalities with at least 2 replicates for each have to be present) with at least one biological and one batch factors are required to use the RFLOMICS workflow.
- plotDataDistribution: return boxplot or density plot of expression or abundance distribution.
- plotMissingValues: return barplot of

Usage

```
## S4 method for signature 'RflomicsSE'
runDataProcessing(
  object,
  samples = NULL,
  lowCountFilter = list(filterMethod = "filterByExpr", filterStrategy = "groups",
    cpmCutoff = NULL),
  missingValueFilter = list(MVencoding = "NA", method = "none", globalProp = NULL,
    nbCondition = NULL, propPerCondition = NULL),
  transform = list(transformMethod = "none", userTransMethod = "unknown"),
```

```
  normalize = list(normMethod = "none", userNormMethod = "unknown"),
  impute = list(imputMethod = "none", factor = NULL),
  ...
)

## S4 method for signature 'RflomicsMAE'
runDataProcessing(
  object,
  SE.name,
  samples = NULL,
  lowCountFilter = list(filterMethod = "filterByExpr", filterStrategy = "groups",
    cpmCutoff = NULL),
  missingValueFilter = list(MVencoding = "NA", method = "none", globalProp = NULL,
    nbCondition = NULL, propPerCondition = NULL),
  transform = list(transformMethod = "none", userTransMethod = "unknown"),
  normalize = list(normMethod = "none", userNormMethod = "unknown"),
  impute = list(imputMethod = "none", factor = NULL)
)

## S4 method for signature 'RflomicsSE'
runSampleFiltering(object, samples = NULL)

## S4 method for signature 'RflomicsMAE'
runSampleFiltering(object, SE.name, samples = NULL)

## S4 method for signature 'RflomicsSE'
runFeatureFiltering(
  object,
  lowCountFilter = list(filterMethod = "filterByExpr", filterStrategy = "groups",
    cpmCutoff = NULL),
  missingValueFilter = list(MVencoding = "NA", method = "GlobalFiltering", globalProp =
    0.5, nbCondition = NULL, propPerCondition = NULL)
)

## S4 method for signature 'RflomicsMAE'
runFeatureFiltering(
  object,
  SE.name,
  lowCountFilter = list(filterMethod = "filterByExpr", filterStrategy = "groups",
    cpmCutoff = NULL),
  missingValueFilter = list(MVencoding = "NA", globalProp = 0.5, nbCondition = NULL,
    propPerCondition = NULL)
)

## S4 method for signature 'RflomicsSE'
runTransformData(object, transformMethod = NULL, userTransMethod = "unknown")

## S4 method for signature 'RflomicsMAE'
runTransformData(
  object,
  SE.name,
  transformMethod = NULL,
```

```
  userTransMethod = "unknown"
)

## S4 method for signature 'RflomicsSE'
runNormalization(object, normMethod = NULL, userNormMethod = "unknown")

## S4 method for signature 'RflomicsMAE'
runNormalization(
  object,
  SE.name,
  normMethod = NULL,
  userNormMethod = "unknown"
)

## S4 method for signature 'RflomicsSE'
runMVImputation(object, imputMethod = "minFeatureValue", factor = 1.8)

## S4 method for signature 'RflomicsMAE'
runMVImputation(object, SE.name, imputMethod = "minFeatureValue", factor = 0.1)

## S4 method for signature 'RflomicsSE'
runOmicsPCA(object, ncomp = 5, raw = FALSE)

## S4 method for signature 'RflomicsMAE'
runOmicsPCA(object, SE.name, ncomp = 5, raw = FALSE)

## S4 method for signature 'RflomicsSE'
checkExpDesignCompleteness(object, sampleList = NULL)

## S4 method for signature 'RflomicsMAE'
checkExpDesignCompleteness(object, omicName, sampleList = NULL)

## S4 method for signature 'RflomicsSE'
getProcessedData(
  object,
  filter = FALSE,
  trans = FALSE,
  norm = FALSE,
  imput = FALSE,
  log = FALSE
)

## S4 method for signature 'RflomicsMAE'
getProcessedData(
  object,
  SE.name,
  filter = FALSE,
  trans = FALSE,
  norm = FALSE,
  imput = FALSE,
  log = FALSE
)
```

```
## S4 method for signature 'RflomicsSE'
getTransSettings(object)

## S4 method for signature 'RflomicsMAE'
getTransSettings(object, SE.name)

## S4 method for signature 'RflomicsSE'
getFilterSettings(object)

## S4 method for signature 'RflomicsMAE'
getFilterSettings(object, SE.name)

## S4 method for signature 'RflomicsSE'
getImputSettings(object)

## S4 method for signature 'RflomicsMAE'
getImputSettings(object, SE.name)

## S4 method for signature 'RflomicsSE'
getFilteredFeatures(object)

## S4 method for signature 'RflomicsMAE'
getFilteredFeatures(object, SE.name)

## S4 method for signature 'RflomicsSE'
getSelectedSamples(object)

## S4 method for signature 'RflomicsMAE'
getSelectedSamples(object, SE.name)

## S4 method for signature 'RflomicsSE'
getCoeffNorm(object)

## S4 method for signature 'RflomicsMAE'
getCoeffNorm(object, SE.name)

## S4 method for signature 'RflomicsSE'
getNormSettings(object)

## S4 method for signature 'RflomicsMAE'
getNormSettings(object, SE.name)

## S4 method for signature 'RflomicsSE'
plotLibrarySize(object, raw = FALSE)

## S4 method for signature 'RflomicsMAE'
plotLibrarySize(object, SE.name, raw = FALSE)

## S4 method for signature 'RflomicsSE'
plotDataDistribution(object, plot = "boxplot", raw = FALSE)
```

```
## S4 method for signature 'RflomicsMAE'
plotDataDistribution(object, SE.name, plot = "boxplot", raw = FALSE)

## S4 method for signature 'RflomicsSE'
plotOmicsPCA(object, raw = TRUE, axes = c(1, 2), groupColor = "groups")

## S4 method for signature 'RflomicsMAE'
plotOmicsPCA(
  object,
  SE.name,
  raw = FALSE,
  axes = c(1, 2),
  groupColor = "groups"
)

## S4 method for signature 'RflomicsSE'
plotExpDesignCompleteness(object, sampleList = NULL)

## S4 method for signature 'RflomicsMAE'
plotExpDesignCompleteness(object, omicName, sampleList = NULL)

## S4 method for signature 'RflomicsSE'
plotMissingValues(object, raw = FALSE)

## S4 method for signature 'RflomicsMAE'
plotMissingValues(object, SE.name, raw = FALSE)

## S4 method for signature 'RflomicsSE'
isProcessedData(
  object,
  filter = FALSE,
  trans = FALSE,
  norm = FALSE,
  log = FALSE
)

## S4 method for signature 'RflomicsMAE'
isProcessedData(
  object,
  SE.name,
  filter = TRUE,
  trans = TRUE,
  norm = TRUE,
  log = FALSE
)
```

Arguments

`object`	An object of class RflomicsSE-class.
`samples`	samples to keep.
`lowCountFilter`	A list specifying parameters for **RNA-seq** low-count filtering. `filterMethod` defines the filtering approach and supports two options:

"filterByExpr" **(default)** Uses edgeR::filterByExpr(). In this case, **filterStrategy** must be "groups" (default and only supported value), and **cpmCutoff** is not used (should be NULL).

"CPM" Applies a CPM-based filtering approach. The **filterStrategy** parameter controls how features are retained: "NbReplicates" (default) keeps features expressed in at least a minimum number of replicates, while "NbConditions" keeps features expressed in a minimum number of conditions. The **cpmCutoff** defines the CPM threshold used to determine expressed features (default: 1).

"none" ...

missingValueFilter

A list specifying parameters for missing value handling in **proteomics and metabolomics data**.

MVencoding defines how missing values are encoded: "NA" or "0". If "0" is used, zero values are converted to NA in the processed data.

method defines the filtering approach and supports two options:

"GlobalFiltering" Filters features based on the overall proportion of missing values. The **globalProp** parameter defines the maximum allowed proportion of missing values across all samples (default: 0.5).

"ConditionFiltering" Filters features based on missing values within conditions. The **nbCondition** parameter defines the minimum number of conditions required (default: 1), and **propPerCondition** defines the maximum allowed proportion of missing values per condition (default: 0.7).

"none" ...

transform — transformation list of arguments.

normalize — normalization list of arguments.

impute — imputation list of arguments.

... — additional arguments

SE.name — the name of the data the normalization have to be applied to.

transformMethod

The transformation method to store in the metadata

userTransMethod

to rm

normMethod — Normalization method. Accepted values: TMM for RNAseq, and median, totalSum, or none for proteomics and metabolomics data. Default values: TMM for RNAseq data and median for proteomics and metabolomics data

userNormMethod — to rm

imputMethod — The imputation method ("minFeatureValue") for proteomics and metabolomics data.

factor — factor

ncomp — Number of components to compute. Default is 5.

raw — a boolean

sampleList — list of samples to check.

omicName — a character string with the name of the dataset

filter — boolean. If TRUE, check if data is filtered (low counts/RNAseq)

trans — boolean. If TRUE, check if data is transformed

norm	boolean. If TRUE, check if data is normalized
imput	boolean. If TRUE, returned imputed data
log	boolean. If TRUE, check if the data has been log-transformed (RNAseq).
plot	plot type ("boxplot" or "density")
axes	A vector giving the two axis that have to be drawn for the factorial map
groupColor	All combination of level's factor

Value

An object of class RflomicsSE or class RflomicsSE

An object of class RflomicsSE The applied normalization method and computed scaling factors (by samples) are stored as a named list ("normalization") of two elements (respectively "method" and "coefNorm") in the metadata slot of a given data set, stored itself in the ExperimentList slot of a RflomicsSE object.

An object of class RflomicsSE

Accessors

- getProcessedData: return RflomicsSE object with a processed data (filtering, normalization and/or transformation)
- getTransSettings: return a list of transformation settings of a given omics dataset
- getFilterSettings: return a list the filtering settings of a given omics dataset
- getImputSettings: return a list the imputation settings of a given omics dataset
- getFilteredFeatures: return a vector of filtered features of a given omics dataset
- getSelectedSamples: return a vector of selected samples of a given omics dataset
- getCoeffNorm: return a named vector with normalization coefficients of a given omics dataset
- getNormSettings: return a list of normalization settings of a given omics dataset
- isProcessedData: return

Plots

- plotLibrarySize: return barplot of library size by sample.
- plotOmicsPCA: This function plot the factorial map from a PCA object stored in a RflomicsSE-class object. By default, samples are colored by groups (all combinations of level's factor)
- plotExpDesignCompleteness: This method checks that experimental design constraints are satisfied and plot a summary of the design. A complete design (all combinations of factor modalities with at least 2 replicates for each have to be present) with at least one biological and one batch factors are required to use the RFLOMICS workflow.

References

Lambert, I., Paysant-Le Roux, C., Colella, S. et al. DiCoExpress: a tool to process multifactorial RNAseq experiments from quality controls to co-expression analysis through differential analysis based on contrasts inside GLM models. Plant Methods 16, 68 (2020).

See Also

RflomicsMAE-class RflomicsSE-class getProcessedData getTransSettings getFilterSettings getFilteredFeatures getCoeffNorm getNormSettings plotLibrarySize plotDataDistribution plotOmicsPCA

Examples

```
# load ecoseed data
data(ecoseed.mae)

factorInfo <- data.frame(
  "factorName"   = c("Repeat", "temperature", "imbibition"),
  "factorType"   = c("batch", "Bio", "Bio")
)

# create rflomicsMAE object with ecoseed data
MAE <- RFLOMICS::createRflomicsMAE(
  projectName = "Tests",
  omicsData   = ecoseed.mae,
  omicsTypes  = c("RNAseq","proteomics","metabolomics"),
  factorInfo  = factorInfo)


# Set the statistical model
formulae <- generateModelFormulae(MAE)
MAE <- setModelFormula(MAE, formulae[[1]])

# set the contrast list
contrastList <- generateExpressionContrast(MAE, "averaged")
MAE <- setSelectedContrasts(MAE, contrastList = contrastList[c(1, 2, 3),])

# Data processing of RNAseq dataset : RNAtest
## using data processing functions for RNAseq data
### filter low RNAseq count
# MAE <- filterLowAbundance(MAE, SE.name = "RNAtest",
#                           filterStrategy = "NbReplicates",
#                           cpmCutoff = 1)
# ### filter outlier samples
# MAE <- runSampleFiltering(MAE, SE.name = "RNAtest",
#                           samples = colnames(MAE[["RNAtest"]])[-1])
# ### data normalisation outlier samples
# MAE <- runNormalization(MAE, SE.name = "RNAtest",
#                          normMethod = "TMM")

## use runDataProcessing function that combines the previous three functions
MAE <- runDataProcessing(MAE, SE.name = "RNAtest",
                         samples = colnames(MAE[["RNAtest"]])[-1],
                         lowCountFilter =
                           list(filterMethod = "CPM",
                                filterStrategy = "NbReplicates",
                                cpmCutoff = 1),
                         normalize = list(normMethod = "TMM"))

## check completness of RNAtest data
checkExpDesignCompleteness(MAE, omicName = "RNAtest")$messages

# Data processing of proteimics dataset : protetest
```

```
# ## transform data
# MAE <- runTransformData(MAE, SE.name = "protetest",  transformMethod = "log2")
# ## normalise data
# MAE <- runNormalization(MAE, SE.name = "protetest", normMethod = "median")

## use runDataProcessing function
MAE <- runDataProcessing(MAE, SE.name = "protetest",
                         normalize = list(normMethod = "median"),
                         transform = list(transformMethod = "log2"))

# plotExpDesignCompleteness(MAE[["RNAtest"]])

# plot Library Size
# plotDataDistribution(MAE[["RNAtest"]], raw=TRUE)
# plotDataDistribution(MAE[["RNAtest"]], raw=FALSE)

# plot gene expression distribution
# plotDataDistribution(MAE[["RNAtest"]], raw=TRUE, plot = "boxplot")
# plotDataDistribution(MAE[["RNAtest"]], raw=FALSE, plot = "boxplot")

# plot PCA
# plotOmicsPCA(MAE[["RNAtest"]], raw="raw", groupColor = "imbibition")
# plotOmicsPCA(MAE[["RNAtest"]], raw="norm", groupColor = "imbibition")

# See runDataProcessing for an example that includes getTransSettings
# See runDataProcessing for an example that includes getFilterSettings
# See runDataProcessing for an example that includes getImputSettings
# See runDataProcessing for an example that includes getFilteredFeatures
# See runDataProcessing for an example that includes getSelectedSamples
# See runDataProcessing for an example that includes getCoeffNorm
# See runDataProcessing for an example that includes getNormSettings
# See runDataProcessing for an example that includes plotLibrarySize
# See runDataProcessing for an example that includes plotDataDistribution
# See runDataProcessing for an example that includes plotOmicsPCA
# See runDataProcessing for an example that includes plotExpDesignCompleteness
# See runDataProcessing for an example that includes plotMissingValues
```

`runDiffAnalysis` *Run Differential Expression Analysis and process results*

Description

This is an interface method which run a differential analysis on omics datasets stored in an object of class RflomicsSE or RflomicsMAE-class. According to the type of omics and to a list of contrasts, a differential analysis is performed for each contrasts. Two methods are available according to the type of object:

- For RNAseq data: the `glmFit` function/model of the `edgeR` package is applied.
- For proteomics and metabolomics data: the `lmFit` function/model of the `limma` package is applied.
- filterDiffAnalysis: The filterDiffAnalysis method allows filtering the results of the differential analysis based on a new cutoff for p-value and fold change.
- setValidContrasts: Set the valid contrasts stored in `metadata` slot.

Usage

```
## S4 method for signature 'RflomicsSE'
runDiffAnalysis(
  object,
  contrastList = NULL,
  method = NULL,
  p.adj.method = "BH",
  p.adj.cutoff = 0.05,
  logFC.cutoff = 0,
  cmd = FALSE,
  ...
)

## S4 method for signature 'RflomicsMAE'
runDiffAnalysis(
  object,
  SE.name,
  contrastList = NULL,
  method = NULL,
  p.adj.method = "BH",
  p.adj.cutoff = 0.05,
  logFC.cutoff = 0,
  cmd = FALSE,
  ...
)

## S4 method for signature 'RflomicsSE'
filterDiffAnalysis(object, p.adj.cutoff = 0.05, logFC.cutoff = 0)

## S4 method for signature 'RflomicsMAE'
filterDiffAnalysis(object, SE.name, p.adj.cutoff = 0.05, logFC.cutoff = 0)

## S4 method for signature 'RflomicsSE'
setValidContrasts(object, contrastList = NULL)

## S4 method for signature 'RflomicsMAE'
setValidContrasts(object, omicName = NULL, contrastList = NULL)

## S4 method for signature 'RflomicsSE'
plotDiffAnalysis(
  object,
  contrastName,
  typeofplots = c("MA.plot", "volcano", "histogram")
)

## S4 method for signature 'RflomicsMAE'
plotDiffAnalysis(
  object,
  SE.name,
  contrastName,
  typeofplots = c("MA.plot", "volcano", "histogram")
)
```

```
## S4 method for signature 'RflomicsSE'
plotHeatmapDesign(
  object,
  contrastName,
  splitFactor = "none",
  title = "",
  annotNames = NULL,
  modalities = NULL,
  drawArgs = list(),
  heatmapArgs = list()
)

## S4 method for signature 'RflomicsMAE'
plotHeatmapDesign(
  object,
  SE.name,
  contrastName,
  splitFactor = "none",
  title = "",
  annotNames = NULL,
  modalities = NULL,
  drawArgs = list(),
  heatmapArgs = list()
)

## S4 method for signature 'RflomicsSE'
plotBoxplotDE(object, featureName = NULL, groupColor = "groups", raw = FALSE)

## S4 method for signature 'RflomicsMAE'
plotBoxplotDE(
  object,
  SE.name,
  featureName = NULL,
  groupColor = "groups",
  raw = FALSE
)

## S4 method for signature 'RflomicsSE'
getDEMatrix(object)

## S4 method for signature 'RflomicsMAE'
getDEMatrix(object, SE.name)

## S4 method for signature 'RflomicsSE'
getDEList(object, contrasts = NULL, operation = "union")

## S4 method for signature 'RflomicsMAE'
getDEList(object, SE.name, contrasts = NULL, operation = "union")

## S4 method for signature 'RflomicsSE'
getDiffSettings(object)
```

```
## S4 method for signature 'RflomicsMAE'
getDiffSettings(object, SE.name)

## S4 method for signature 'RflomicsSE'
getValidContrasts(object)

## S4 method for signature 'RflomicsMAE'
getValidContrasts(object, omicName)

## S4 method for signature 'RflomicsSE'
getDiffStat(object)

## S4 method for signature 'RflomicsMAE'
getDiffStat(object, SE.name = NULL)

## S4 method for signature 'RflomicsMAE'
getDiffAnalysesSummary(
  object,
  plot = FALSE,
  ylabelLength = 30,
  nbMaxLabel = 20,
  interface = FALSE
)
```

Arguments

object	An object of class RflomicsSE or class RflomicsMAE-class
contrastList	A data.frame of contrast
method	A character vector giving the name of the differential analysis method to run. Either "edgeRglmfit" or "limmalmFit".
p.adj.method	The method chosen to adjust pvalue. Takes the same values as the ones of adj.p.adjust method.
p.adj.cutoff	adjusted pvalue cutoff. Default is the parameter from the differential analysis.
logFC.cutoff	cutoff for absolute value of log2FC. Default is the parameter from the differential analysis.
cmd	Boolean. Used in the interface. If TRUE, print cmd for the user.
...	Additional arguments.
SE.name	SE.name the name of the dataset if the input object is a RflomicsMAE-class
omicName	a dataset name
contrastName	The contrastName for which the MAplot has to be drawn
typeofplots	The plots you want to return. Default is all possible plots: MA plot, Volcano plot and non adjusted pvalues histogram.
splitFactor	characters. Default to none. Name of a feature in the design matrix, splits the samples on the heatmap according to its modalities.
title	characters. Title of the heatmap.
annotNames	vector. Names of the annotations to keep in the Heatmap. Default takes all available information.

modalities	named list of vectors of modalities to subset and print on the heatmap.
drawArgs, heatmapArgs	named lists. Any additional parameter passed to ComplexHeatmap::Heatmap or ComplexHeatmap::draw
featureName	variable name (gene/protein/metabolite name)
groupColor	default to groups, indicate a variable in the design to color the boxplots accordingly.
raw	Boolean. Plot the raw data or the transformed ones (TRUE)
contrasts	Vector of characters, expect to be contrast names. Default is null, the operation (union) is performed on every contrasts found.
operation	character. Either union or intersection. Defines the operation to perform on the DE lists from the contrasts.
plot	FALSE or TRUE
ylabelLength	max length of the labels (characters)
nbMaxLabel	number of labels to print
interface	Boolean. Is this plot for the interface or commandline?

Details

Functions and parameters used for RNAseq are those recommended in DiCoExpress workflow (see the paper in reference). Functions and parameters used for proteomics and metabolomics data are those recommended in the (Efstathiou *et al.*, 2017)

Value

A RflomicsSE or a RflomicsMAE-class object. All the results are stored as a named list `DiffExpAnal` in the metadata slot of a given RflomicsSE object. Objects are:

- stats: data.frame giving a summary of the differential statistical analysis results by contrast: number of DE features, number of up and down regulated features
- setting: Parameters used for the differential analysis
- method: The method used for the differential analysis
- p.adj.method: The applied p-value correction method
- p.adj.cutoff: The cut-off applied for the adjusted p-value
- logFC.cutoff: The absolute log FC cut-off
- RawDEFres: a list giving for each contrast the raw results of the differential analysis method
- DEF: a list giving for each contrast a data.frame of non filtered differential expressed features with their statistics
- TopDEF: a list giving for each contrast a data.frame of differential expressed features ordered and filtered by p.adj.cutoff with their statistics
- mergeDEF: a data frame of 0 and 1 indicating for each features in row, if it is DE in a given contrasts in column
- contrasts: a data.table of the contrasts used for the differential analysis

a data.frame with differential analyses summary

Accessors

- getDEMatrix: return a matrix of experimental design.
- getDEList: return a vector of union or intersection of differential expressed features from list of contrasts.
- getDiffSettings: return a list of differential expression analysis settings of a given omics dataset
- getValidContrasts: return a data.frame of validated contrasts
- getDiffStat: Get summary table from diffExpAnalysis analysis
- getDiffAnalysesSummary: ...

Plots

- plotDiffAnalysis method draws a MAplot, a volcano plot and the p-values distribution from the results of a differential analysis.
- plotHeatmapDesign method draws a heatmap from the results of a differential analysis.
- plotBoxplotDE method draws a boxplot showing the expression of given differentially expressed feature.

References

Lambert, I., Paysant-Le Roux, C., Colella, S. et al. DiCoExpress: a tool to process multifactorial RNAseq experiments from quality controls to co-expression analysis through differential analysis based on contrasts inside GLM models. Plant Methods 16, 68 (2020).

Efstathiou G, Antonakis AN, Pavlopoulos GA, et al. ProteoSign: an end-user online differential proteomics statistical analysis platform. Nucleic Acids Res. 2017;45(W1):W300-W306.

See Also

getDiffSettings, getDEList, getDEMatrix

plotDiffAnalysis, plotHeatmapDesign, plotBoxplotDE

Examples

```
# load ecoseed data
library(RFLOMICS)
data(ecoseed.mae)

factorInfo <- data.frame(
  "factorName"   = c("Repeat", "temperature", "imbibition"),
  "factorType"   = c("batch", "Bio", "Bio")
)

# create rflomicsMAE object with ecoseed data
MAE <- RFLOMICS::createRflomicsMAE(
  projectName = "Tests",
  omicsData   = ecoseed.mae,
  omicsTypes  = c("RNAseq","proteomics","metabolomics"),
  factorInfo  = factorInfo)
```

```
# Set the statistical model and contrasts to test
formulae <- generateModelFormulae(MAE)
MAE <- setModelFormula(MAE, formulae[[1]])

# Get the contrasts List and choose the first 3 contrasts of type averaged
contrastList <- generateExpressionContrast(MAE, "averaged")

MAE <- setSelectedContrasts(MAE, contrastList = contrastList[c(1, 2, 3),])

# Run the data preprocessing and perform the differential analysis
MAE <- runDataProcessing(MAE, SE.name = "protetest",
                         transform = list(transformMethod = "log2"),
                         normalize = list(normMethod = "median"))

MAE <- runDiffAnalysis(MAE, SE.name = "protetest",
                       method = "limmalmFit")
# or
# MAE[["protetest"]] <- runDiffAnalysis(MAE[["protetest"]],
#                                       method = "limmalmFit",
#                                       contrastList = contrastList)

# Filter the results of the differential analysis with new cut-off values
# for p-value and fold change.
MAE <- filterDiffAnalysis(MAE, SE.name = "protetest",
                          p.adj.cutoff = 0.01,
                          logFC.cutoff = 0)
# or
# MAE[["protetest"]] <- filterDiffAnalysis(MAE[["protetest"]],
#                                          p.adj.cutoff = 0.01,
#                                          logFC.cutoff = 0)

# Access to the diff analysis settings
## Get DE matrix from DiffExpAnalysis
head(getDEMatrix(MAE[["protetest"]]))

## Get union or intersection from list of contrasts
getDEList(MAE[["protetest"]], contrasts = "(temperatureMedium - temperatureLow) in mean")

## Get diff setting
getDiffSettings(MAE[["protetest"]])

# generate plot results of a differential analysis
thiscontrast <- "(temperatureMedium - temperatureLow) in mean"

## generate MAplot from diff analysis
# plotDiffAnalysis(MAE[["protetest"]],
#                  contrastName = thiscontrast,
#                  typeofplots = "MA.plot")

## plot the heatmap
# plotHeatmapDesign(MAE[["protetest"]],
#                   contrastName = thiscontrast)

## plot boxplot with feature expression
# plotBoxplotDE(MAE[["protetest"]],
#               features = "AT1G47128",
```

```
#               groupColor = "temperature")
# plotBoxplotDE(MAE[["protetest"]],
#               features = "AT1G79550",
#               groupColor = "imbibition")

# See runDiffAnalysis for an example that includes plotDiffAnalysis
# See runDiffAnalysis for an example that includes plotHeatmapDesign
# See runDiffAnalysis for an example that includes plotBoxplotDE
# See runDiffAnalysis for an example that includes getDEMatrix
# See runDiffAnalysis for an example that includes getDEList
# See runDiffAnalysis for an example that includes getDiffSettings
```

`runOmicsIntegration,RflomicsMAE-method`
runOmicsIntegration

Description

Runs the integration according to the selected method (MOFA or mixOmics) and the settings given by the user. Requires to have the correct entry format in preparedObject before running.

These methods are used to directly access the results of multi-omics analyses or their settings, usually stored in the metadata of the RflomicsMAE-class object. Setters are also available.

Usage

```
## S4 method for signature 'RflomicsMAE'
runOmicsIntegration(
  object,
  preparedObject = NULL,
  method = "MOFA",
  scale_views = FALSE,
  maxiter = 1000,
  num_factors = 10,
  selectedResponse = NULL,
  ncomp = 2,
  link_datasets = 1,
  link_response = 1,
  sparsity = FALSE,
  cases_to_try = 5,
  cmd = FALSE,
  ...
)

## S4 method for signature 'RflomicsMAE'
getMixOmics(object, response = NULL, onlyResults = TRUE)

## S4 method for signature 'RflomicsMAE'
getMOFA(object, onlyResults = TRUE)

## S4 method for signature 'RflomicsMAE'
getMOFASettings(object)
```

```
## S4 method for signature 'RflomicsMAE'
getMixOmicsSettings(object)

## S4 method for signature 'RflomicsMAE'
setMOFA(object, results = NULL)

## S4 method for signature 'RflomicsMAE'
setMixOmics(object, results = NULL)

## S4 method for signature 'RflomicsMAE'
sumMixOmics(object, selectedResponse = NULL)
```

Arguments

object	a RflomicsMAE object.
preparedObject	An untrained MOFA object or a list of dataset. Usually a result of prepareForIntegration.
method	one of MOFA or mixOmics. Method for which the object is prepared.
scale_views	boolean. If TRUE, scale each dataset to unit variance.
maxiter	MOFA2 parameter. Number of max iteration (otherwise stop when converged.)
num_factors	MOFA2 parameter. The number of factor to compute.
selectedResponse	a character. Useful if MixOmics was run on several response variable. If NULL, all variables are taken into account.
ncomp	mixOmics parameter. Number of components to compute.
link_datasets	mixOmics parameter. Link between datasets in the computation.
link_response	mixOmics parameter. Link between dataset and response.
sparsity	boolean. Used to determine which mixOmics function to apply (either block.plsda if FALSE or block.splsda if TRUE).
cases_to_try	integer. If sparsity is set to TRUE, then cases_to_try is used to determine the number of sets of variables to test for tuning.
cmd	boolean. Used in the interface. If TRUE, print cmd in the console.
...	not in use at the moment
response	a character giving the response variable to access specifically.
onlyResults	default return only the MixOmics or MOFA2 results. If you want to access all information of the integration, set onlyResults to FALSE. In MixOmics case, works only when response is specified.
results	The MOFA or mixOmics results to set in the object. If null, set to NULL.

Value

a RflomicsMAE object with the correct metadata slot filled with the results and the settings.

For getters: in getMixOmics, if response is NULL, then all the mixOmics results are returned. Otherwise, it gives the particular mixOmics result. For MOFA, returns the untrained object and the trained object as a list.

For setters: always returns a RflomicsMAE-class object.

sumMixOmics: A data frame or a list of dataframe (if selectedResponse is NULL) presenting the summary of mixOmics analyses.

Examples

```
# load ecoseed data
library(RFLOMICS)
data(ecoseed.mae)

factorInfo <- data.frame(
  "factorName"   = c("Repeat", "temperature", "imbibition"),
  "factorType"   = c("batch", "Bio", "Bio")
)

# create rflomicsMAE object with ecoseed data
MAE <- RFLOMICS::createRflomicsMAE(
  projectName = "Tests",
  omicsData   = ecoseed.mae,
  omicsTypes  = c("RNAseq","proteomics","metabolomics"),
  factorInfo  = factorInfo)

formulae <- generateModelFormulae( MAE)
MAE <- setModelFormula(MAE, formulae[[1]])
contrastList <- Reduce(rbind, generateExpressionContrast(MAE))

MAE <- MAE |>
  setSelectedContrasts(contrastList[c(3,6,25)]) |>
  runDataProcessing(SE.name = "metatest",
                    transform = list(transformMethod = "log2"),
                    normalize = list(normMethod = "median")) |>
  runDataProcessing(SE.name = "protetest",
                    transform = list(transformMethod = "none"),
                    normalize = list(normMethod = "median"))   |>
  runDiffAnalysis(SE.name = "metatest", method = "limmalmFit")     |>
  runDiffAnalysis(SE.name = "protetest", method = "limmalmFit")

# Integration using MOFA
# Prepare mofa object:
mofaObj <- prepareForIntegration(MAE,
                                 omicsNames = c("protetest", "metatest"),
                                 variableLists = rownames(MAE),
                                 method = "MOFA")

# Perform integration:
# Not run: MAEtest <- runOmicsIntegration(MAE,
#                                 preparedObject = mofaObj,
#                                 method = "MOFA", num_factors = 5)

# Integration using MixOmics
mixObj <- prepareForIntegration(MAE,
                                omicsNames = c("protetest", "metatest"),
                                variableLists = rownames(MAE),
                                method = "mixOmics")
MAEtest <- runOmicsIntegration(MAE, preparedObject = mixObj,
                               method = "mixOmics")


# Access mixOmics results:
#getMixOmics(MAEtest, response = "temperature")
getMixOmicsSettings(MAEtest)
```

```
# mixOmics::plotIndiv(getMixOmics(MAEtest, response = "imbibition"))

# Access MOFA2 results:
# getMOFA(MAEtest)
# getMOFASettings(MAEtest)
# MOFA2::plot_variance_explained(getMOFA(MAEtest))
```

`runRFLOMICS` *Run RFLOMICS interface*

Description

running this function will open the shiny application. Run the shiny application

Usage

```
runRFLOMICS(...)
```

Arguments

`...`	More arguments to pass to shinyApp.

Value

shinyApp

Examples

```
library(RFLOMICS)
## Not run: runRFLOMICS()
```

Index