Package 'mia'

June 5, 2023

Type Package

Version 1.8.0

Title Microbiome analysis

Description mia implements tools for microbiome analysis based on the SummarizedExperiment, SingleCellExperiment and TreeSummarizedExperiment infrastructure. Data wrangling and analysis in the context of taxonomic data is the main scope. Additional functions for common task are implemented such as community indices calculation and summarization.

biocViews Microbiome, Software, DataImport

License Artistic-2.0 | file LICENSE

Encoding UTF-8

LazyData false

Depends R (>= 4.0), SummarizedExperiment, SingleCellExperiment, TreeSummarizedExperiment (>= 1.99.3), MultiAssayExperiment

Imports methods, stats, utils, MASS, ape, decontam, vegan, BiocGenerics, S4Vectors, IRanges, Biostrings, DECIPHER, BiocParallel, DelayedArray, DelayedMatrixStats, scuttle, scater, DirichletMultinomial, rlang, dplyr, tibble, tidyr

Suggests testthat, knitr, patchwork, BiocStyle, yaml, phyloseq, dada2, stringr, biomformat, reldist, ade4, microbiomeDataSets, rmarkdown

URL https://github.com/microbiome/mia

BugReports https://github.com/microbiome/mia/issues

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

VignetteBuilder knitr

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

git_branch RELEASE_3_17

git_last_commit e5f4407

git_last_commit_date 2023-04-25

Date/Publication 2023-06-05

Author Felix G.M. Ernst [aut] (<https://orcid.org/0000-0001-5064-0928>),
Sudarshan A. Shetty [aut] (<https://orcid.org/0000-0001-7280-9915>),
Tuomas Borman [aut, cre] (<https://orcid.org/0000-0002-8563-8884>),
Leo Lahti [aut] (<https://orcid.org/0000-0001-5537-637X>),
Yang Cao [ctb],
Nathan D. Olson [ctb],
Levi Waldron [ctb],
Marcel Ramos [ctb],
Héctor Corrada Bravo [ctb],
Jayaram Kancherla [ctb],
Domenick Braccia [ctb]

Maintainer Tuomas Borman <tuomas.v.borman@utu.fi>

R topics documented:

`mia-package` `mia` *Package.*

Description

`mia` implements tools for microbiome analysis based on the `SummarizedExperiment`, `SingleCellExperiment` and `TreeSummarizedExperiment` infrastructure. Data wrangling and analysis in the context of taxonomic data is the main scope. Additional functions for common task are implemented such as community indices calculation and summarization.

See Also

TreeSummarizedExperiment class

`agglomerate-methods` *Agglomerate data using taxonomic information*

Description

`agglomerateByRank` can be used to sum up data based on the association to certain taxonomic ranks given as `rowData`. Only available `taxonomyRanks` can be used.

Usage

```
## S4 method for signature 'SummarizedExperiment'
agglomerateByRank(
  x,
  rank = taxonomyRanks(x)[1],
  onRankOnly = FALSE,
  na.rm = FALSE,
  empty.fields = c(NA, "", " ", "\t", "-", "_"),
  ...
)

## S4 method for signature 'SingleCellExperiment'
agglomerateByRank(x, ..., altexp = NULL, strip_altexp = TRUE)

## S4 method for signature 'TreeSummarizedExperiment'
agglomerateByRank(x, ..., agglomerateTree = FALSE)
```

Arguments

x	a SummarizedExperiment object
rank	a single character defining a taxonomic rank. Must be a value of taxonomyRanks() function.
onRankOnly	TRUE or FALSE: Should information only from the specified rank be used or from ranks equal and above? See details. (default: onRankOnly = FALSE)
na.rm	TRUE or FALSE: Should taxa with an empty rank be removed? Use it with caution, since empty entries on the selected rank will be dropped. This setting can be tweaked by defining empty.fields to your needs. (default: na.rm = TRUE)
empty.fields	a character value defining, which values should be regarded as empty. (Default: c(NA, "", " ", "\t")). They will be removed if na.rm = TRUE before agglomeration.
...	arguments passed to agglomerateByRank function for SummarizedExperiment objects, mergeRows and sumCountsAcrossFeatures. • remove_empty_ranksA single boolean value for selecting whether to remove those columns of rowData that include only NAs after agglomeration. (By default: remove_empty_ranks = FALSE) • make_uniqueA single boolean value for selecting whether to make rownames unique. (By default: make_unique = TRUE)
altexp	String or integer scalar specifying an alternative experiment containing the input data.
strip_altexp	TRUE or FALSE: Should alternative experiments be removed prior to agglomeration? This prevents to many nested alternative experiments by default (default: strip_altexp = TRUE)
agglomerateTree	TRUE or FALSE: should rowTree() also be agglomerated? (Default: agglomerateTree = FALSE)

Details

Based on the available taxonomic data and its structure setting onRankOnly = TRUE has certain implications on the interpretability of your results. If no loops exist (loops meaning two higher ranks containing the same lower rank), the results should be comparable. you can check for loops using detectLoop.

Agglomeration sum up values of assays at specified taxonomic level. Certain assays, e.g. those that include binary or negative values, can lead to meaningless values, when values are summed. In those cases, consider doing agglomeration first and then transformation.

Value

A taxonomically-agglomerated, optionally-pruned object of the same class as x.

See Also

mergeRows, sumCountsAcrossFeatures

Examples

```
data(GlobalPatterns)
# print the available taxonomic ranks
colnames(rowData(GlobalPatterns))
taxonomyRanks(GlobalPatterns)

# agglomerate at the Family taxonomic rank
x1 <- agglomerateByRank(GlobalPatterns, rank="Family")
## How many taxa before/after agglomeration?
nrow(GlobalPatterns)
nrow(x1)

# with agglomeration of the tree
x2 <- agglomerateByRank(GlobalPatterns, rank="Family",
                        agglomerateTree = TRUE)
nrow(x2) # same number of rows, but
rowTree(x1) # ... different
rowTree(x2) # ... tree

 # If assay contains binary or negative values, summing might lead to meaningless
 # values, and you will get a warning. In these cases, you might want to do
 # agglomeration again at chosen taxonomic level.
 tse <- transformCounts(GlobalPatterns, method = "pa")
 tse <- agglomerateByRank(tse, rank = "Genus")
 tse <- transformCounts(tse, method = "pa")

# removing empty labels by setting na.rm = TRUE
sum(is.na(rowData(GlobalPatterns)$Family))
x3 <- agglomerateByRank(GlobalPatterns, rank="Family", na.rm = TRUE)
nrow(x3) # different from x2

# Because all the rownames are from the same rank, rownames do not include
# prefixes, in this case "Family:".
print(rownames(x3[1:3,]))

# To add them, use getTaxonomyLabels function.
rownames(x3) <- getTaxonomyLabels(x3, with_rank = TRUE)
print(rownames(x3[1:3,]))

# use 'remove_empty_ranks' to remove columns that include only NAs
x4 <- agglomerateByRank(GlobalPatterns, rank="Phylum", remove_empty_ranks = TRUE)
head(rowData(x4))

# If assay contains NAs, you might want to consider replacing them since summing-up
# NAs lead to NA
x5 <- GlobalPatterns
# Replace first value with NA
assay(x5)[1,1] <- NA
x6 <- agglomerateByRank(x5, "Kingdom")
head( assay(x6) )
# Replace NAs with 0. It is justified when we are summing-up counts
assay(x5)[ is.na(assay(x5)) ] <- 0
```

```
x6 <- agglomerateByRank(x5, "Kingdom")
head( assay(x6) )

## Look at enterotype dataset...
data(enterotype)
## print the available taxonomic ranks. Shows only 1 rank available
## not useful for agglomerateByRank
taxonomyRanks(enterotype)
```

`calculateDMN` *Dirichlet-Multinomial Mixture Model: Machine Learning for Microbiome Data*

Description

These functions are accessors for functions implemented in the `DirichletMultinomial` package

Usage

```
calculateDMN(x, ...)

## S4 method for signature 'ANY'
calculateDMN(
  x,
  k = 1,
  BPPARAM = SerialParam(),
  seed = runif(1, 0, .Machine$integer.max),
  ...
)

## S4 method for signature 'SummarizedExperiment'
calculateDMN(
  x,
  assay.type = assay_name,
  assay_name = exprs_values,
  exprs_values = "counts",
  transposed = FALSE,
  ...
)

runDMN(x, name = "DMN", ...)

getDMN(x, name = "DMN", ...)

## S4 method for signature 'SummarizedExperiment'
getDMN(x, name = "DMN")

bestDMNFit(x, name = "DMN", type = c("laplace", "AIC", "BIC"), ...)
```

```
## S4 method for signature 'SummarizedExperiment'
bestDMNFit(x, name = "DMN", type = c("laplace", "AIC", "BIC"))

getBestDMNFit(x, name = "DMN", type = c("laplace", "AIC", "BIC"), ...)

## S4 method for signature 'SummarizedExperiment'
getBestDMNFit(x, name = "DMN", type = c("laplace", "AIC", "BIC"))

calculateDMNgroup(x, ...)

## S4 method for signature 'ANY'
calculateDMNgroup(
  x,
  variable,
  k = 1,
  seed = runif(1, 0, .Machine$integer.max),
  ...
)

## S4 method for signature 'SummarizedExperiment'
calculateDMNgroup(
  x,
  variable,
  assay.type = assay_name,
  assay_name = exprs_values,
  exprs_values = "counts",
  transposed = FALSE,
  ...
)

performDMNgroupCV(x, ...)

## S4 method for signature 'ANY'
performDMNgroupCV(
  x,
  variable,
  k = 1,
  seed = runif(1, 0, .Machine$integer.max),
  ...
)

## S4 method for signature 'SummarizedExperiment'
performDMNgroupCV(
  x,
  variable,
  assay.type = assay_name,
  assay_name = exprs_values,
```

```
  exprs_values = "counts",
  transposed = FALSE,
  ...
)
```

Arguments

x	a numeric matrix with samples as rows or a SummarizedExperiment object.
...	optional arguments not used.
k	the number of Dirichlet components to fit. See dmn
BPPARAM	A BiocParallelParam object specifying whether the UniFrac calculation should be parallelized.
seed	random number seed. See dmn
assay.type	a single character value for specifying which assay to use for calculation.
assay_name	a single character value for specifying which assay to use for calculation. (Please use assay.type instead. At some point assay_name will be disabled.)
exprs_values	a single character value for specifying which assay to use for calculation. (Please use assay.type instead.)
transposed	Logical scalar, is x transposed with samples in rows?
name	the name to store the result in metadata
type	the type of measure used for the goodness of fit. One of 'laplace', 'AIC' or 'BIC'.
variable	a variable from colData to use as a grouping variable. Must be a character of factor.

Value

calculateDMN and getDMN return a list of DMN objects, one element for each value of k provided.

bestDMNFit returns the index for the best fit and getBestDMNFit returns a single DMN object.

calculateDMNgroup returns a DMNGroup object

performDMNgroupCV returns a data.frame

See Also

DMN-class, DMNGroup-class, dmn, dmngroup, cvdmngroup , accessors for DMN objects

Examples

```
fl <- system.file(package="DirichletMultinomial", "extdata", "Twins.csv")
counts <- as.matrix(read.csv(fl, row.names=1))
fl <- system.file(package="DirichletMultinomial", "extdata", "TwinStudy.t")
pheno0 <- scan(fl)
lvls <- c("Lean", "Obese", "Overwt")
pheno <- factor(lvls[pheno0 + 1], levels=lvls)
colData <- DataFrame(pheno = pheno)
```

```
tse <- TreeSummarizedExperiment(assays = list(counts = counts),
                                colData = colData)


#
dmn <- calculateDMN(tse)
dmn[[1L]]

# since this take a bit of resources to calculate for k > 1, the data is
# loaded
## Not run:
tse <- runDMN(tse, name = "DMN", k = 1:7)

## End(Not run)
data(dmn_se)
# Convert SE to TreeSE to enable all features of mia and TreeSE object
# (Optional step at this point)
dmn_tse <- as(dmn_se, "TreeSummarizedExperiment")
names(metadata(dmn_tse))

# return a list of DMN objects
getDMN(dmn_tse)
# return, which objects fits best
bestDMNFit(dmn_tse, type = "laplace")
# return the model, which fits best
getBestDMNFit(dmn_tse, type = "laplace")
```

`calculateJSD` *Calculate the Jensen-Shannon Divergence*

Description

This function calculates the Jensen-Shannon Divergence (JSD) in a `SummarizedExperiment` object.

Usage

```
## S4 method for signature 'ANY'
calculateJSD(x, ...)

## S4 method for signature 'SummarizedExperiment'
calculateJSD(
  x,
  assay.type = assay_name,
  assay_name = exprs_values,
  exprs_values = "counts",
  transposed = FALSE,
  ...
)

runJSD(x, BPPARAM = SerialParam(), chunkSize = nrow(x))
```

Arguments

x	a numeric matrix or a SummarizedExperiment.
...	optional arguments not used.
assay.type	a single character value for specifying which assay to use for calculation.
assay_name	a single character value for specifying which assay to use for calculation. (Please use assay.type instead. At some point assay_name will be disabled.)
exprs_values	a single character value for specifying which assay to use for calculation. (Please use assay.type instead.)
transposed	Logical scalar, is x transposed with cells in rows?
BPPARAM	A BiocParallelParam object specifying whether the JSD calculation should be parallelized.
chunkSize	an integer scalar, defining the size of data send to the individual worker. Only has an effect, if BPPARAM defines more than one worker. (default: chunkSize = nrow(x))

Value

a sample-by-sample distance matrix, suitable for NMDS, etc.

Author(s)

Susan Holmes <susan@stat.stanford.edu>. Adapted for phyloseq by Paul J. McMurdie. Adapted for mia by Felix G.M. Ernst

References

Jensen-Shannon Divergence and Hilbert space embedding. Bent Fuglede and Flemming Topsoe University of Copenhagen, Department of Mathematics http://www.math.ku.dk/~topsoe/ISIT2004JSD.pdf

See Also

http://en.wikipedia.org/wiki/Jensen-Shannon_divergence

Examples

```
data(enterotype)
library(scater)


jsd <- calculateJSD(enterotype)
class(jsd)
head(jsd)

enterotype <- runMDS(enterotype, FUN = calculateJSD, name = "JSD",
                     exprs_values = "counts")
head(reducedDim(enterotype))
head(attr(reducedDim(enterotype),"eig"))
attr(reducedDim(enterotype),"GOF")
```

`calculateOverlap` *Estimate overlap*

Description

This function calculates overlap for all sample-pairs in a `SummarizedExperiment` object.

Usage

```
calculateOverlap(
  x,
  assay.type = assay_name,
  assay_name = "counts",
  detection = 0,
  ...
)

## S4 method for signature 'SummarizedExperiment'
calculateOverlap(
  x,
  assay.type = assay_name,
  assay_name = "counts",
  detection = 0,
  ...
)

runOverlap(x, ...)

## S4 method for signature 'SummarizedExperiment'
runOverlap(x, name = "overlap", ...)
```

Arguments

`x`	a `SummarizedExperiment` object containing a tree.
`assay.type`	A single character value for selecting the `assay` to calculate the overlap.
`assay_name`	a single `character` value for specifying which assay to use for calculation. (Please use `assay.type` instead. At some point `assay_name` will be disabled.)
`detection`	A single numeric value for selecting detection threshold for absence/presence of features. Feature that has abundance under threshold in either of samples, will be discarded when evaluating overlap between samples.
`...`	Optional arguments not used.
`name`	A single character value specifying the name of overlap matrix that is stored in reducedDim(x).

Details

These function calculates overlap between all the sample-pairs. Overlap reflects similarity between sample-pairs.

When overlap is calculated using relative abundances, the higher the value the higher the similarity is, When using relative abundances, overlap value 1 means that all the abundances of features are equal between two samples, and 0 means that samples have completely different relative abundances.

Value

calculateOverlap returns sample-by-sample distance matrix. runOverlap returns x that includes overlap matrix in its reducedDim.

Author(s)

Leo Lahti and Tuomas Borman. Contact: microbiome.github.io

See Also

calculateJSD calculateUnifrac

Examples

```
data(esophagus)
tse <- esophagus
tse <- transformCounts(tse, method = "relabundance")
overlap <- calculateOverlap(tse, assay_name = "relabundance")
overlap

# Store result to reducedDim
tse <- runOverlap(tse, assay.type = "relabundance", name = "overlap_between_samples")
head(reducedDims(tse)$overlap_between_samples)
```

---

calculateUnifrac *Calculate weighted or unweighted (Fast) Unifrac distance*

Description

This function calculates the (Fast) Unifrac distance for all sample-pairs in a TreeSummarizedExperiment object.

Usage

```
calculateUnifrac(x, tree, ...)

## S4 method for signature 'ANY,phylo'
calculateUnifrac(
  x,
  tree,
  weighted = FALSE,
  normalized = TRUE,
  BPPARAM = SerialParam(),
  ...
)

## S4 method for signature 'TreeSummarizedExperiment,missing'
calculateUnifrac(
  x,
  assay.type = assay_name,
  assay_name = exprs_values,
  exprs_values = "counts",
  tree_name = "phylo",
  transposed = FALSE,
  ...
)

runUnifrac(
  x,
  tree,
  weighted = FALSE,
  normalized = TRUE,
  nodeLab = NULL,
  BPPARAM = SerialParam(),
  ...
)
```

Arguments

x	a numeric matrix or a TreeSummarizedExperiment object containing a tree. Please note that runUnifrac expects a matrix with samples per row and not per column. This is implemented to be compatible with other distance calculations such as dist as much as possible.
tree	if x is a matrix, a phylo object matching the matrix. This means that the phylo object and the columns should relate to the same type of features (aka. microorganisms).
...	optional arguments not used.
weighted	TRUE or FALSE: Should use weighted-Unifrac calculation? Weighted-Unifrac takes into account the relative abundance of species/taxa shared between samples, whereas unweighted-Unifrac only considers presence/absence. Default is

FALSE, meaning the unweighted-Unifrac distance is calculated for all pairs of samples.

normalized — TRUE or FALSE: Should the output be normalized such that values range from 0 to 1 independent of branch length values? Default is TRUE. Note that (unweighted) Unifrac is always normalized by total branch-length, and so this value is ignored when weighted == FALSE.

BPPARAM — A BiocParallelParam object specifying whether the Unifrac calculation should be parallelized.

assay.type — a single character value for specifying which assay to use for calculation.

assay_name — a single character value for specifying which assay to use for calculation. (Please use assay.type instead. At some point assay_name will be disabled.)

exprs_values — a single character value for specifying which assay to use for calculation. (Please use assay.type instead.)

tree_name — a single character value for specifying which tree will be used in calculation. (By default: tree_name = "phylo")

transposed — Logical scalar, is x transposed with cells in rows, i.e., is Unifrac distance calculated based on rows (FALSE) or columns (TRUE). (By default: transposed = FALSE)

nodeLab — if x is a matrix, a character vector specifying links between rows/columns and tips of tree. The length must equal the number of rows/columns of x. Furthermore, all the node labs must be present in tree.

Details

Please note that if calculateUnifrac is used as a FUN for runMDS, the argument ntop has to be set to nrow(x).

Value

a sample-by-sample distance matrix, suitable for NMDS, etc.

Author(s)

Paul J. McMurdie. Adapted for mia by Felix G.M. Ernst

References

http://bmf.colorado.edu/unifrac/

The main implementation (Fast Unifrac) is adapted from the algorithm's description in:

Hamady, Lozupone, and Knight, "Fast UniFrac: facilitating high-throughput phylogenetic analyses of microbial communities including analysis of pyrosequencing and PhyloChip data." The ISME Journal (2010) 4, 17–27.

See also additional descriptions of Unifrac in the following articles:

Lozupone, Hamady and Knight, "Unifrac - An Online Tool for Comparing Microbial Community Diversity in a Phylogenetic Context.", BMC Bioinformatics 2006, 7:371

Lozupone, Hamady, Kelley and Knight, "Quantitative and qualitative (beta) diversity measures lead to different insights into factors that structure microbial communities." Appl Environ Microbiol. 2007

Lozupone C, Knight R. "Unifrac: a new phylogenetic method for comparing microbial communities." Appl Environ Microbiol. 2005 71 (12):8228-35.

Examples

```
data(esophagus)
library(scater)
calculateUnifrac(esophagus, weighted = FALSE)
calculateUnifrac(esophagus, weighted = TRUE)
calculateUnifrac(esophagus, weighted = TRUE, normalized = FALSE)
# for using calculateUnifrac in conjunction with runMDS the tree argument
# has to be given separately. In addition, subsetting using ntop must
# be disabled
esophagus <- runMDS(esophagus, FUN = calculateUnifrac, name = "Unifrac",
                    tree = rowTree(esophagus),
                    exprs_values = "counts",
                    ntop = nrow(esophagus))
reducedDim(esophagus)
```

---

estimateDivergence *Estimate divergence*

Description

Estimate divergence against a given reference sample.

Usage

```
estimateDivergence(
  x,
  assay.type = assay_name,
  assay_name = "counts",
  name = "divergence",
  reference = "median",
  FUN = vegan::vegdist,
  method = "bray",
  ...
)

## S4 method for signature 'SummarizedExperiment'
estimateDivergence(
  x,
  assay.type = assay_name,
  assay_name = "counts",
  name = "divergence",
```

```
  reference = "median",
  FUN = vegan::vegdist,
  method = "bray",
  ...
)
```

Arguments

x	a SummarizedExperiment object.
assay.type	the name of the assay used for calculation of the sample-wise estimates.
assay_name	a single character value for specifying which assay to use for calculation. (Please use assay.type instead. At some point assay_name will be disabled.)
name	a name for the column of the colData the results should be stored in. By defaut, name is "divergence".
reference	a numeric vector that has length equal to number of features, or a non-empty character value; either 'median' or 'mean'. reference specifies the reference that is used to calculate divergence. by default, reference is "median".
FUN	a function for distance calculation. The function must expect the input matrix as its first argument. With rows as samples and columns as features. By default, FUN is vegan::vegdist.
method	a method that is used to calculate the distance. Method is passed to the function that is specified by FUN. By default, method is "bray".
...	optional arguments

Details

Microbiota divergence (heterogeneity / spread) within a given sample set can be quantified by the average sample dissimilarity or beta diversity with respect to a given reference sample.

This measure is sensitive to sample size. Subsampling or bootstrapping can be applied to equalize sample sizes between comparisons.

Value

x with additional colData named *name*

Author(s)

Leo Lahti and Tuomas Borman. Contact: microbiome.github.io

See Also

plotColData

- estimateRichness
- estimateEvenness
- estimateDominance

Examples

```
data(GlobalPatterns)
tse <- GlobalPatterns

# By default, reference is median of all samples. The name of column where results
# is "divergence" by default, but it can be specified.
tse <- estimateDivergence(tse)

# The method that are used to calculate distance in divergence and
# reference can be specified. Here, euclidean distance and dist function from
# stats package are used. Reference is the first sample.
tse <- estimateDivergence(tse, name = "divergence_first_sample",
                          reference = assays(tse)$counts[,1],
                          FUN = stats::dist, method = "euclidean")

# Reference can also be median or mean of all samples.
# By default, divergence is calculated by using median. Here, mean is used.
tse <- estimateDivergence(tse, name = "divergence_average", reference = "mean")

# All three divergence results are stored in colData.
colData(tse)
```

---

estimateDiversity *Estimate (alpha) diversity measures*

Description

Several functions for calculating (alpha) diversity indices, including the vegan package options and some others.

Usage

```
estimateDiversity(
  x,
  assay.type = "counts",
  assay_name = NULL,
  index = c("coverage", "fisher", "gini_simpson", "inverse_simpson",
    "log_modulo_skewness", "shannon"),
  name = index,
  ...
)

## S4 method for signature 'SummarizedExperiment'
estimateDiversity(
  x,
  assay.type = "counts",
  assay_name = NULL,
```

```
  index = c("coverage", "fisher", "gini_simpson", "inverse_simpson",
    "log_modulo_skewness", "shannon"),
  name = index,
  ...,
  BPPARAM = SerialParam()
)

## S4 method for signature 'TreeSummarizedExperiment'
estimateDiversity(
  x,
  assay.type = "counts",
  assay_name = NULL,
  index = c("coverage", "faith", "fisher", "gini_simpson", "inverse_simpson",
    "log_modulo_skewness", "shannon"),
  name = index,
  tree_name = "phylo",
  ...,
  BPPARAM = SerialParam()
)

estimateFaith(
  x,
  tree = "missing",
  assay.type = "counts",
  assay_name = NULL,
  name = "faith",
  ...
)

## S4 method for signature 'SummarizedExperiment,phylo'
estimateFaith(
  x,
  tree,
  assay.type = "counts",
  assay_name = NULL,
  name = "faith",
  node_lab = NULL,
  ...
)

## S4 method for signature 'TreeSummarizedExperiment,missing'
estimateFaith(
  x,
  assay.type = "counts",
  assay_name = NULL,
  name = "faith",
  tree_name = "phylo",
  ...
```

)

Arguments

x	a SummarizedExperiment object or TreeSummarizedExperiment. The latter is recommended for microbiome data sets and tree-based alpha diversity indices.
assay.type	the name of the assay used for calculation of the sample-wise estimates.
assay_name	a single character value for specifying which assay to use for calculation. (Please use assay.type instead. At some point assay_name will be disabled.)
index	a character vector, specifying the diversity measures to be calculated.
name	a name for the column(s) of the colData the results should be stored in. By default this will use the original names of the calculated indices.
...	optional arguments: • threshold A numeric value in the unit interval, determining the threshold for coverage index. By default, threshold is 0.9. • quantile Arithmetic abundance classes are evenly cut up to to this quantile of the data. The assumption is that abundances higher than this are not common, and they are classified in their own group. By default, quantile is 0.5. • num_of_classes The number of arithmetic abundance classes from zero to the quantile cutoff indicated by quantile. By default, num_of_classes is 50.
BPPARAM	A BiocParallelParam object specifying whether calculation of estimates should be parallelized.
tree_name	a single character value for specifying which rowTree will be used to calculate faith index. (By default: tree_name = "phylo")
tree	A phylogenetic tree that is used to calculate 'faith' index. If x is a TreeSummarizedExperiment, rowTree(x) is used by default.
node_lab	NULL or a character vector specifying the links between rows and node labels of tree. If a certain row is not linked with the tree, missing instance should be noted as NA. When NULL, all the rownames should be found from the tree. (By default: node_lab = NULL)

Details

The available indices include the 'Coverage', 'Faith's phylogenetic diversity', 'Fisher alpha', 'Gini-Simpson', 'Inverse Simpson', 'log-modulo skewness', and 'Shannon' indices. See details for more information and references.

Alpha diversity is a joint quantity that combines elements or community richness and evenness. Diversity increases, in general, when species richness or evenness increase.

By default, this function returns all indices.

- 'coverage' Number of species needed to cover a given fraction of the ecosystem (50 percent by default). Tune this with the threshold argument.

- 'faith' Faith's phylogenetic alpha diversity index measures how long the taxonomic distance is between taxa that are present in the sample. Larger values represent higher diversity. Using this index requires rowTree. (Faith 1992)
- 'fisher' Fisher's alpha; as implemented in `vegan::fisher.alpha`. (Fisher et al. 1943)
- 'gini_simpson' Gini-Simpson diversity i.e. $1 - lambda$, where $lambda$ is the Simpson index, calculated as the sum of squared relative abundances. This corresponds to the diversity index 'simpson' in `vegan::diversity`. This is also called Gibbs–Martin, or Blau index in sociology, psychology and management studies. The Gini-Simpson index (1-lambda) should not be confused with Simpson's dominance (lambda), Gini index, or inverse Simpson index (1/lambda).
- 'inverse_simpson' Inverse Simpson diversity: $1/lambda$ where $lambda = sum(p^2)$ and p refers to relative abundances. This corresponds to the diversity index 'invsimpson' in vegan::diversity. Don't confuse this with the closely related Gini-Simpson index
- 'log_modulo_skewness' The rarity index characterizes the concentration of species at low abundance. Here, we use the skewness of the frequency distribution of arithmetic abundance classes (see Magurran & McGill 2011). These are typically right-skewed; to avoid taking log of occasional negative skews, we follow Locey & Lennon (2016) and use the log-modulo transformation that adds a value of one to each measure of skewness to allow logarithmization.
- 'shannon' Shannon diversity (entropy).

Value

x with additional `colData` named `*name*`

Author(s)

Leo Lahti and Tuomas Borman. Contact: `microbiome.github.io`

References

Beisel J-N. et al. (2003) A Comparative Analysis of Diversity Index Sensitivity. *Internal Rev. Hydrobiol.* 88(1):3-15. https://portais.ufg.br/up/202/o/2003-comparative_evennes_index.pdf

Bulla L. (1994) An index of diversity and its associated diversity measure. *Oikos* 70:167–171

Faith D.P. (1992) Conservation evaluation and phylogenetic diversity. *Biological Conservation* 61(1):1-10.

Fisher R.A., Corbet, A.S. & Williams, C.B. (1943) The relation between the number of species and the number of individuals in a random sample of animal population. *Journal of Animal Ecology 12*, 42-58.

Locey K.J. & Lennon J.T. (2016) Scaling laws predict global microbial diversity. *PNAS* 113(21):5970-5975.

Magurran A.E., McGill BJ, eds (2011) Biological Diversity: Frontiers in Measurement and Assessment. (Oxford Univ Press, Oxford), Vol 12.

Smith B. & Wilson JB. (1996) A Consumer's Guide to Diversity Indices. *Oikos* 76(1):70-82.

See Also

plotColData

- estimateRichness
- estimateEvenness
- estimateDominance
- diversity
- estimateR

Examples

```
data(GlobalPatterns)
tse <- GlobalPatterns

# All index names as known by the function
index <- c("shannon","gini_simpson","inverse_simpson", "coverage", "fisher",
"faith",  "log_modulo_skewness")

# Corresponding polished names
name <- c("Shannon","GiniSimpson","InverseSimpson", "Coverage", "Fisher",
"Faith",  "LogModSkewness")

# Calculate diversities
tse <- estimateDiversity(tse, index = index)

# The colData contains the indices with their code names by default
colData(tse)[, index]

# Removing indices
colData(tse)[, index] <- NULL

# 'threshold' can be used to determine threshold for 'coverage' index
tse <- estimateDiversity(tse, index = "coverage", threshold = 0.75)
# 'quantile' and 'num_of_classes' can be used when
# 'log_modulo_skewness' is calculated
tse <- estimateDiversity(tse, index = "log_modulo_skewness",
       quantile = 0.75, num_of_classes = 100)

# It is recommended to specify also the final names used in the output.
tse <- estimateDiversity(tse,
  index = c("shannon", "gini_simpson", "inverse_simpson", "coverage",
              "fisher", "faith", "log_modulo_skewness"),
   name = c("Shannon", "GiniSimpson",  "InverseSimpson",  "Coverage",
              "Fisher", "Faith", "LogModSkewness"))

# The colData contains the indices by their new names provided by the user
colData(tse)[, name]

# Compare the indices visually
pairs(colData(tse)[, name])
```

```
# Plotting the diversities - use the selected names
library(scater)
plotColData(tse, "Shannon")
# ... by sample type
plotColData(tse, "Shannon", "SampleType")
## Not run:
# combining different plots
library(patchwork)
plot_index <- c("Shannon","GiniSimpson")
plots <- lapply(plot_index,
               plotColData,
               object = tse,
               x = "SampleType",
               colour_by = "SampleType")
plots <- lapply(plots,"+",
   theme(axis.text.x = element_text(angle=45,hjust=1)))
names(plots) <- plot_index
plots$Shannon + plots$GiniSimpson + plot_layout(guides = "collect")

## End(Not run)
```

`estimateDominance` *Estimate dominance measures*

Description

This function calculates community dominance indices. This includes the 'Absolute', 'Berger-Parker', 'Core abundance', 'Gini', 'McNaughton's', 'Relative', and 'Simpson's' indices.

Usage

```
estimateDominance(
  x,
  assay.type = assay_name,
  assay_name = "counts",
  index = c("absolute", "dbp", "core_abundance", "gini", "dmn", "relative",
    "simpson_lambda"),
  ntaxa = 1,
  aggregate = TRUE,
  name = index,
  ...,
  BPPARAM = SerialParam()
)

## S4 method for signature 'SummarizedExperiment'
estimateDominance(
  x,
  assay.type = assay_name,
  assay_name = "counts",
```

```
  index = c("absolute", "dbp", "core_abundance", "gini", "dmn", "relative",
    "simpson_lambda"),
  ntaxa = 1,
  aggregate = TRUE,
  name = index,
  ...,
  BPPARAM = SerialParam()
)
```

Arguments

x	a SummarizedExperiment object
assay.type	A single character value for selecting the assay to calculate the sample-wise estimates.
assay_name	a single character value for specifying which assay to use for calculation. (Please use assay.type instead. At some point assay_name will be disabled.)
index	a character vector, specifying the indices to be calculated.
ntaxa	Optional and only used for the Absolute and Relative dominance indices: The n-th position of the dominant taxa to consider (default: ntaxa = 1). Disregarded for the indices "dbp", "core_abundance", "Gini", "dmn", and "Simpson".
aggregate	Optional and only used for the Absolute, dbp, Relative, and dmn dominance indices: Aggregate the values for top members selected by ntaxa or not. If TRUE, then the sum of relative abundances is returned. Otherwise the relative abundance is returned for the single taxa with the indicated rank (default: aggregate = TRUE). Disregarded for the indices "core_abundance", "gini", "dmn", and "simpson".
name	A name for the column(s) of the colData where the calculated Dominance indices should be stored in.
...	additional arguments currently not used.
BPPARAM	A BiocParallelParam object specifying whether calculation of estimates should be parallelized. (Currently not used)

Details

A dominance index quantifies the dominance of one or few species in a community. Greater values indicate higher dominance.

Dominance indices are in general negatively correlated with alpha diversity indices (species richness, evenness, diversity, rarity). More dominant communities are less diverse.

estimateDominance calculates the following community dominance indices:

- 'absolute' Absolute index equals to the absolute abundance of the most dominant n species of the sample (specify the number with the argument ntaxa). Index gives positive integer values.
- 'dbp' Berger-Parker index (See Berger & Parker 1970) calculation is a special case of the 'relative' index. dbp is the relative abundance of the most abundant species of the sample. Index gives values in interval 0 to 1, where bigger value represent greater dominance.

$$dbp = \frac{N_1}{N_{tot}}$$

where N_1 is the absolute abundance of the most dominant species and N_{tot} is the sum of absolute abundances of all species.

- 'core_abundance' Core abundance index is related to core species. Core species are species that are most abundant in all samples, i.e., in whole data set. Core species are defined as those species that have prevalence over 50\ species must be prevalent in 50\ calculate the core abundance index. Core abundance index is sum of relative abundances of core species in the sample. Index gives values in interval 0 to 1, where bigger value represent greater dominance.

$$core_abundance = \frac{N_{core}}{N_{tot}}$$

where N_{core} is the sum of absolute abundance of the core species and N_{tot} is the sum of absolute abundances of all species.

- 'gini' Gini index is probably best-known from socio-economic contexts (Gini 1921). In economics, it is used to measure, for example, how unevenly income is distributed among population. Here, Gini index is used similarly, but income is replaced with abundance.

  If there is small group of species that represent large portion of total abundance of microbes, the inequality is large and Gini index closer to 1. If all species has equally large abundances, the equality is perfect and Gini index equals 0. This index should not be confused with Gini-Simpson index, which quantifies diversity.

- 'dmn' McNaughton's index is the sum of relative abundances of the two most abundant species of the sample (McNaughton & Wolf, 1970). Index gives values in the unit interval:

$$dmn = (N_1 + N_2)/N_tot$$

where N_1 and N_2 are the absolute abundances of the two most dominant species and N_{tot} is the sum of absolute abundances of all species.

- 'relative' Relative index equals to the relative abundance of the most dominant n species of the sample (specify the number with the argument `ntaxa`). This index gives values in interval 0 to 1.

$$relative = N_1/N_tot$$

where N_1 is the absolute abundance of the most dominant species and N_{tot} is the sum of absolute abundances of all species.

- 'simpson_lambda' Simpson's (dominance) index or Simpson's lambda is the sum of squared relative abundances. This index gives values in the unit interval. This value equals the probability that two randomly chosen individuals belongs to the same species. The higher the probability, the greater the dominance (See e.g. Simpson 1949).

$$lambda = \sum(p^2)$$

where p refers to relative abundances.

There is also a more advanced Simpson dominance index (Simpson 1949). However, this is not provided and the simpler squared sum of relative abundances is used instead as the alternative index is not in the unit interval and it is highly correlated with the simpler variant implemented here.

Value

x with additional colData named *name*

Author(s)

Leo Lahti and Tuomas Borman. Contact: microbiome.github.io

References

Berger WH & Parker FL (1970) Diversity of Planktonic Foraminifera in Deep-Sea Sediments. *Science* 168(3937):1345-1347. doi: 10.1126/science.168.3937.1345

Gini C (1921) Measurement of Inequality of Incomes. *The Economic Journal* 31(121): 124-126. doi: 10.2307/2223319

McNaughton, SJ and Wolf LL. (1970). Dominance and the niche in ecological systems. *Science* 167:13, 1–139

Simpson EH (1949) Measurement of Diversity. *Nature* 163(688). doi: 10.1038/163688a0

See Also

- estimateRichness
- estimateEvenness
- estimateDiversity

Examples

```
data(esophagus)

# Calculates Simpson's lambda (can be used as a dominance index)
esophagus <- estimateDominance(esophagus, index="simpson_lambda")

# Shows all indices
colData(esophagus)

# Indices must be written correctly (e.g. dbp, not dbp), otherwise an error
# gets thrown
## Not run: esophagus <- estimateDominance(esophagus, index="dbp")
# Calculates dbp and Core Abundance indices
esophagus <- estimateDominance(esophagus, index=c("dbp", "core_abundance"))
# Shows all indices
colData(esophagus)
# Shows dbp index
colData(esophagus)$dbp
# Deletes dbp index
colData(esophagus)$dbp <- NULL
# Shows all indices, dbp is deleted
colData(esophagus)
# Deletes all indices
colData(esophagus) <- NULL
```

```
# Calculates all indices
esophagus <- estimateDominance(esophagus)
# Shows all indices
colData(esophagus)
# Deletes all indices
colData(esophagus) <- NULL

# Calculates all indices with explicitly specified names
esophagus <- estimateDominance(esophagus,
    index = c("dbp", "dmn", "absolute", "relative",
              "simpson_lambda", "core_abundance", "gini"),
    name  = c("BergerParker", "McNaughton", "Absolute", "Relative",
              "SimpsonLambda", "CoreAbundance", "Gini")
)
# Shows all indices
colData(esophagus)
```

`estimateEvenness` *Estimate Evenness measures*

Description

This function calculates community evenness indices. These include the 'Camargo', 'Pielou', 'Simpson', 'Evar' and 'Bulla' evenness measures. See details for more information and references.

Usage

```
estimateEvenness(
  x,
  assay.type = assay_name,
  assay_name = "counts",
  index = c("pielou", "camargo", "simpson_evenness", "evar", "bulla"),
  name = index,
  ...
)

## S4 method for signature 'SummarizedExperiment'
estimateEvenness(
  x,
  assay.type = assay_name,
  assay_name = "counts",
  index = c("camargo", "pielou", "simpson_evenness", "evar", "bulla"),
  name = index,
  ...,
  BPPARAM = SerialParam()
)
```

Arguments

x	a SummarizedExperiment object
assay.type	A single character value for selecting the assay used for calculation of the sample-wise estimates.
assay_name	a single character value for specifying which assay to use for calculation. (Please use assay.type instead. At some point assay_name will be disabled.)
index	a character vector, specifying the eveness measures to be calculated.
name	a name for the column(s) of the colData the results should be stored in.
...	optional arguments: • threshold a numeric threshold. assay values below or equal to this threshold will be set to zero.
BPPARAM	A BiocParallelParam object specifying whether calculation of estimates should be parallelized.

Details

Evenness is a standard index in community ecology, and it quantifies how evenly the abundances of different species are distributed. The following evenness indices are provided:

By default, this function returns all indices.

The available evenness indices include the following (all in lowercase):

- 'camargo' Camargo's evenness (Camargo 1992)
- 'simpson_evenness' Simpson's evenness is calculated as inverse Simpson diversity (1/lambda) divided by observed species richness S: (1/lambda)/S.
- 'pielou' Pielou's evenness (Pielou, 1966), also known as Shannon or Shannon-Weaver/Wiener/Weiner evenness; H/ln(S). The Shannon-Weaver is the preferred term; see Spellerberg and Fedor (2003).
- 'evar' Smith and Wilson's Evar index (Smith & Wilson 1996).
- 'bulla' Bulla's index (O) (Bulla 1994).

Desirable statistical evenness metrics avoid strong bias towards very large or very small abundances; are independent of richness; and range within the unit interval with increasing evenness (Smith & Wilson 1996). Evenness metrics that fulfill these criteria include at least camargo, simpson, smith-wilson, and bulla. Also see Magurran & McGill (2011) and Beisel et al. (2003) for further details.

Value

x with additional colData named *name*

References

Beisel J-N. et al. (2003) A Comparative Analysis of Evenness Index Sensitivity. *Internal Rev. Hydrobiol.* 88(1):3-15. URL: https://portais.ufg.br/up/202/o/2003-comparative_evennes_index.pdf

Bulla L. (1994) An index of evenness and its associated diversity measure. *Oikos* 70:167–171.

Camargo, JA. (1992) New diversity index for assessing structural alterations in aquatic communities. *Bull. Environ. Contam. Toxicol.* 48:428–434.

Locey KJ and Lennon JT. (2016) Scaling laws predict global microbial diversity. *PNAS* 113(21):5970-5975; doi:10.1073/pnas.1521291113.

Magurran AE, McGill BJ, eds (2011) Biological Diversity: Frontiers in Measurement and Assessment (Oxford Univ Press, Oxford), Vol 12.

Pielou, EC. (1966) The measurement of diversity in different types of biological collections. *J Theoretical Biology* 13:131–144.

Smith B and Wilson JB. (1996) A Consumer's Guide to Evenness Indices. *Oikos* 76(1):70-82.

Spellerberg and Fedor (2003). A tribute to Claude Shannon (1916 –2001) and a plea for more rigorous use of species richness, species diversity and the 'Shannon–Wiener' Index. *Alpha Ecology & Biogeography* 12, 177–197.

See Also

plotColData

- estimateRichness
- estimateDominance
- estimateDiversity

Examples

```
data(esophagus)
tse <- esophagus

# Specify index and their output names
index <- c("pielou", "camargo", "simpson_evenness", "evar", "bulla")
name  <- c("Pielou", "Camargo", "SimpsonEvenness",  "Evar", "Bulla")

# Estimate evenness and give polished names to be used in the output
tse <- estimateEvenness(tse, index = index, name = name)

# Check the output
head(colData(tse))
```

---

estimateRichness *Estimate richness measures*

Description

Several functions for calculation of community richness indices available via wrapper functions. They are implemented via the vegan package.

Usage

```
estimateRichness(
  x,
  assay.type = assay_name,
  assay_name = "counts",
  index = c("ace", "chao1", "hill", "observed"),
  name = index,
  detection = 0,
  ...,
  BPPARAM = SerialParam()
)

## S4 method for signature 'SummarizedExperiment'
estimateRichness(
  x,
  assay.type = assay_name,
  assay_name = "counts",
  index = c("ace", "chao1", "hill", "observed"),
  name = index,
  detection = 0,
  ...,
  BPPARAM = SerialParam()
)
```

Arguments

x	a SummarizedExperiment object.
assay.type	the name of the assay used for calculation of the sample-wise estimates.
assay_name	a single character value for specifying which assay to use for calculation. (Please use assay.type instead. At some point assay_name will be disabled.)
index	a character vector, specifying the richness measures to be calculated.
name	a name for the column(s) of the colData the results should be stored in.
detection	a numeric value for selecting detection threshold for the abundances. The default detection threshold is 0.
...	additional parameters passed to estimateRichness
BPPARAM	A BiocParallelParam object specifying whether calculation of estimates should be parallelized.

Details

These include the 'ace', 'Chao1', 'Hill', and 'Observed' richness measures. See details for more information and references.

The richness is calculated per sample. This is a standard index in community ecology, and it provides an estimate of the number of unique species in the community. This is often not directly observed for the whole community but only for a limited sample from the community. This has led to alternative richness indices that provide different ways to estimate the species richness.

Richness index differs from the concept of species diversity or evenness in that it ignores species abundance, and focuses on the binary presence/absence values that indicate simply whether the species was detected.

The function takes all index names in full lowercase. The user can provide the desired spelling through the argument `name` (see examples).

The following richness indices are provided.

- 'ace' Abundance-based coverage estimator (ACE) is another nonparametric richness index that uses sample coverage, defined based on the sum of the probabilities of the observed species. This method divides the species into abundant (more than 10 reads or observations) and rare groups in a sample and tends to underestimate the real number of species. The ACE index ignores the abundance information for the abundant species, based on the assumption that the abundant species are observed regardless of their exact abundance. We use here the bias-corrected version (O'Hara 2005, Chiu et al. 2014) implemented in `estimateR`. For an exact formulation, see `estimateR`. Note that this index comes with an additional column with standard error information.
- 'chao1' This is a nonparametric estimator of species richness. It assumes that rare species carry information about the (unknown) number of unobserved species. We use here the bias-corrected version (O'Hara 2005, Chiu et al. 2014) implemented in `estimateR`. This index implicitly assumes that every taxa has equal probability of being observed. Note that it gives a lower bound to species richness. The bias-corrected for an exact formulation, see `estimateR`. This estimator uses only the singleton and doubleton counts, and hence it gives more weight to the low abundance species. Note that this index comes with an additional column with standard error information.
- 'hill' Effective species richness aka Hill index (see e.g. Chao et al. 2016). Currently only the case 1D is implemented. This corresponds to the exponent of Shannon diversity. Intuitively, the effective richness indicates the number of species whose even distribution would lead to the same diversity than the observed community, where the species abundances are unevenly distributed.
- 'observed' The *observed richness* gives the number of species that is detected above a given `detection` threshold in the observed sample (default 0). This is conceptually the simplest richness index. The corresponding index in the **vegan** package is "richness".

Value

`x` with additional `colData` named `*name*`

Author(s)

Leo Lahti. Contact: `microbiome.github.io`

References

Chao A. (1984) Non-parametric estimation of the number of classes in a population. *Scand J Stat.* 11:265–270.

Chao A, Chun-Huo C, Jost L (2016). Phylogenetic Diversity Measures and Their Decomposition: A Framework Based on Hill Numbers. Biodiversity Conservation and Phylogenetic Systematics, Springer International Publishing, pp. 141–172, doi:10.1007/978-3-319-22461-9_8.

Chiu, C.H., Wang, Y.T., Walther, B.A. & Chao, A. (2014). Improved nonparametric lower bound of species richness via a modified Good-Turing frequency formula. *Biometrics* 70, 671-682.

O'Hara, R.B. (2005). Species richness estimators: how many species can dance on the head of a pin? *J. Anim. Ecol.* 74, 375-386.

See Also

plotColData

- estimateR

Examples

```
data(esophagus)

# Calculates all richness indices by default
esophagus <- estimateRichness(esophagus)

# Shows all indices
colData(esophagus)

# Shows Hill index
colData(esophagus)$hill

# Deletes hill index
colData(esophagus)$hill <- NULL

# Shows all indices, hill is deleted
colData(esophagus)

# Delete the remaining indices
colData(esophagus)[, c("observed", "chao1", "ace")] <- NULL

# Calculates observed richness index and saves them with specific names
esophagus <- estimateRichness(esophagus,
    index = c("observed", "chao1", "ace", "hill"),
     name = c("Observed", "Chao1", "ACE", "Hill"))

# Show the new indices
colData(esophagus)

# Deletes all colData (including the indices)
colData(esophagus) <- NULL

# Calculate observed richness excluding singletons (detection limit 1)
esophagus <- estimateRichness(esophagus, index="observed", detection = 1)

# Deletes all colData (including the indices)
colData(esophagus) <- NULL

# Indices must be written correctly (all lowercase), otherwise an error
# gets thrown
```

```
## Not run: esophagus <- estimateRichness(esophagus, index="ace")

# Calculates Chao1 and ACE indices only
esophagus <- estimateRichness(esophagus, index=c("chao1", "ace"),
                                          name=c("Chao1", "ACE"))

# Deletes all colData (including the indices)
colData(esophagus) <- NULL

# Names of columns can be chosen arbitrarily, but the length of arguments
# must match.
esophagus <- estimateRichness(esophagus,
                              index = c("ace", "chao1"),
                              name = c("index1", "index2"))
# Shows all indices
colData(esophagus)
```

---

getAbundance *Get abundance values by "SampleID" or "FeatureID"*

Description

These are functions for extracting abundances present in `assay(x)`. These functions are convenience wrapper around subsetting columns or rows from `assay(x,name)`.

Usage

```
getAbundanceSample(
  x,
  sample_id,
  assay.type = assay_name,
  assay_name = "counts"
)

## S4 method for signature 'SummarizedExperiment'
getAbundanceSample(
  x,
  sample_id = NULL,
  assay.type = assay_name,
  assay_name = "counts"
)

getAbundanceFeature(
  x,
  feature_id,
  assay.type = assay_name,
  assay_name = "counts"
```

```
)

## S4 method for signature 'SummarizedExperiment'
getAbundanceFeature(
  x,
  feature_id = NULL,
  assay.type = assay_name,
  assay_name = "counts"
)
```

Arguments

x	A SummarizedExperiment object.
sample_id	A "SampleID" from which user wants to extract the abundances of "FeatureID". This is essentially a column name in assay(x).
assay.type	a character value to select an assayNames
assay_name	a single character value for specifying which assay to use for calculation. (Please use assay.type instead. At some point assay_name will be disabled.)
feature_id	A "FeatureID" for which user wants to extract the abundances from all of "SampleID" in assayNames. This is essentially a rowname in assay(x).

Details

getAbundanceSample returns abundance values for all "FeatureIDs" in a user specified "SampleID".

getAbundanceFeature returns abundance values in all "SampleIDs" for user specified "FeatureID".

Value

getAbundanceSample and getAbundanceFeature return a numeric matrix of the abundance values for all "SampleIDs"/"FeatureIDs"

Author(s)

Sudarshan A. Shetty

Examples

```
# getAbundanceSample
data(GlobalPatterns)
getAbundanceSample(GlobalPatterns,
                   sample_id = 'CC1',
                   assay.type = 'counts')
# getAbundanceFeature
getAbundanceFeature(GlobalPatterns,
                    feature_id = '522457',
                    assay.type = 'counts')
```

getExperimentCrossAssociation
Calculate correlations between features of two experiments.

Description

Calculate correlations between features of two experiments.

Usage

```
getExperimentCrossAssociation(x, ...)

## S4 method for signature 'MultiAssayExperiment'
getExperimentCrossAssociation(
  x,
  experiment1 = 1,
  experiment2 = 2,
  assay.type1 = assay_name1,
  assay_name1 = "counts",
  assay.type2 = assay_name2,
  assay_name2 = "counts",
  altexp1 = NULL,
  altexp2 = NULL,
  colData_variable1 = NULL,
  colData_variable2 = NULL,
  MARGIN = 1,
  method = c("kendall", "spearman", "categorical", "pearson"),
  mode = "table",
 p_adj_method = c("fdr", "BH", "bonferroni", "BY", "hochberg", "holm", "hommel", "none"),
  p_adj_threshold = NULL,
  cor_threshold = NULL,
  sort = FALSE,
  filter_self_correlations = FALSE,
  verbose = TRUE,
  test_significance = FALSE,
  show_warnings = TRUE,
  paired = FALSE,
  ...
)

## S4 method for signature 'SummarizedExperiment'
getExperimentCrossAssociation(x, experiment2 = x, ...)

testExperimentCrossAssociation(x, ...)

## S4 method for signature 'ANY'
testExperimentCrossAssociation(x, ...)
```

```
testExperimentCrossCorrelation(x, ...)

## S4 method for signature 'ANY'
testExperimentCrossCorrelation(x, ...)

getExperimentCrossCorrelation(x, ...)

## S4 method for signature 'ANY'
getExperimentCrossCorrelation(x, ...)
```

Arguments

x	A MultiAssayExperiment or SummarizedExperiment object.
...	Additional arguments: • symmetric A single boolean value for specifying if measure is symmetric or not. When symmetric = TRUE, associations are calculated only for unique variable-pairs, and they are assigned to corresponding variable-pair. This decreases the number of calculations in 2-fold meaning faster execution. (By default: symmetric = FALSE) • association_FUN A function that is used to calculate (dis-)similarity between features. Function must take matrix as an input and give numeric values as an output. Adjust method and other parameters correspondingly. Supported functions are, for example, stats::dist and vegan::vegdist.
experiment1	A single character or numeric value for selecting the experiment 1 from experiments(x) of MultiassayExperiment object. (By default: experiment1 = 1)
experiment2	A single character or numeric value for selecting the experiment 2 fromexperiments(x) of MultiAssayExperiment object or altExp(x) of TreeSummarizedExperiment object. Alternatively, experiment2 can also be TreeSE object when x is TreeSE object. (By default: experiment2 = 2 when x is MAE and experiment2 = x when x is TreeSE)
assay.type1	A single character value for selecting the assay of experiment 1 to be transformed. (By default: assay.type1 = "counts")
assay_name1	a single character value for specifying which assay of experiment 1 to use for calculation. (Please use assay.type1 instead. At some point assay_name1 will be disabled.)
assay.type2	A single character value for selecting the assay of experiment 2 to be transformed. (By default: assay.type2 = "counts")
assay_name2	a single character value for specifying which assay of experiment 2 to use for calculation. (Please use assay.type2 instead. At some point assay_name2 will be disabled.)
altexp1	A single numeric or character value specifying alternative experiment from the altExp of experiment 1. If NULL, then the experiment is itself and altExp option is disabled. (By default: altexp1 = NULL)
altexp2	A single numeric or character value specifying alternative experiment from the altExp of experiment 2. If NULL, then the experiment is itself and altExp option is disabled. (By default: altexp2 = NULL)

colData_variable1
: A character value specifying column(s) from colData of experiment 1. If colData_variable1 is used, assay.type1 is disabled. (By default: `colData_variable1 = NULL`)

colData_variable2
: A character value specifying column(s) from colData of experiment 2. If colData_variable2 is used, assay.type2 is disabled. (By default: `colData_variable2 = NULL`)

MARGIN
: A single numeric value for selecting if association are calculated row-wise / for features (1) or column-wise / for samples (2). Must be 1 or 2. (By default: `MARGIN = 1`)

method
: A single character value for selecting association method ('kendall', pearson', or 'spearman' for continuous/numeric; 'categorical' for discrete) (By default: `method = "kendall"`)

mode
: A single character value for selecting output format Available formats are 'table' and 'matrix'. (By default: `mode = "table"`)

p_adj_method
: A single character value for selecting adjustment method of p-values. Passed to `p.adjust` function. (By default: `p_adj_method = "fdr"`)

p_adj_threshold
: A single numeric value (from 0 to 1) for selecting adjusted p-value threshold for filtering. (By default: `p_adj_threshold = NULL`)

cor_threshold
: A single numeric absolute value (from 0 to 1) for selecting correlation threshold for filtering. (By default: `cor_threshold = NULL`)

sort
: A single boolean value for selecting whether to sort features or not in result matrices. Used method is hierarchical clustering. (By default: `sort = FALSE`)

filter_self_correlations
: A single boolean value for selecting whether to filter out correlations between identical items. Applies only when correlation between experiment itself is tested, i.e., when assays are identical. (By default: `filter_self_correlations = FALSE`)

verbose
: A single boolean value for selecting whether to get messages about progress of calculation.

test_significance
: A single boolean value for selecting whether to test statistical significance of associations.

show_warnings
: A single boolean value for selecting whether to show warnings that might occur when correlations and p-values are calculated.

paired
: A single boolean value for specifying if samples are paired or not. `colnames` must match between twp experiments. `paired` is disabled when `MARGIN = 1`. (By default: `paired = FALSE`)

Details

These functions calculates associations between features of two experiments. `getExperimentCrossAssociation` calculates only associations by default. `testExperimentCrossAssociation` calculates also significance of associations.

We recommend the non-parametric Kendall's tau as the default method for association analysis. Kendall's tau has desirable statistical properties and robustness at lower sample sizes. Spearman rank correlation can provide faster solutions when running times are critical.

Value

These functions return associations in table or matrix format. In table format, returned value is a data frame that includes features and associations (and p-values) in columns. In matrix format, returned value is a one matrix when only associations are calculated. If also significances are tested, then returned value is a list of matrices.

Author(s)

Leo Lahti and Tuomas Borman. Contact: microbiome.github.io

Examples

```
data("HintikkaXOData")
mae <- HintikkaXOData

# Subset so that less observations / quicker to run, just for example
mae[[1]] <- mae[[1]][1:20, 1:10]
mae[[2]] <- mae[[2]][1:20, 1:10]
# Transform data
mae[[1]] <- transformCounts(mae[[1]], method = "rclr")

# Calculate cross-correlations
result <- getExperimentCrossAssociation(mae, method = "pearson", assay.type2 = "nmr")
# Show first 5 entries
head(result, 5)

# Use altExp option to specify alternative experiment from the experiment
altExp(mae[[1]], "Phylum") <- agglomerateByRank(mae[[1]], rank = "Phylum")
# Transform data
altExp(mae[[1]], "Phylum") <- transformCounts(altExp(mae[[1]], "Phylum"), method = "relabundance")
# When mode = matrix, matrix is returned
result <- getExperimentCrossAssociation(mae, experiment2 = 2,
                                        assay.type1 = "relabundance", assay.type2 = "nmr",
                                          altexp1 = "Phylum",
                                          method = "pearson", mode = "matrix")
# Show first 5 entries
head(result, 5)

# testExperimentCorrelation returns also significances
# filter_self_correlations = TRUE filters self correlations
# With p_adj_threshold it is possible to filter those features that do no have
# any correlations that have p-value under threshold
result <- testExperimentCrossAssociation(mae[[1]], experiment2 = mae[[1]], method = "pearson",
                                           filter_self_correlations = TRUE,
                                           p_adj_threshold = 0.05)
# Show first 5 entries
head(result, 5)
```

```
# Also getExperimentCrossAssociation returns significances when
# test_signicance = TRUE
# Warnings can be suppressed by using show_warnings = FALSE
result <- getExperimentCrossAssociation(mae[[1]], experiment2 = mae[[2]], method = "pearson",
                                        assay.type2 = "nmr",
                                        mode = "matrix", test_significance = TRUE,
                                        show_warnings = FALSE)

# Returned value is a list of matrices
names(result)

# Calculate Bray-Curtis dissimilarity between samples. If dataset includes
# paired samples, you can use paired = TRUE.
result <- getExperimentCrossAssociation(mae[[1]], mae[[1]], MARGIN = 2, paired = FALSE,
                                      association_FUN = vegan::vegdist, method = "bray")


# If experiments are equal and measure is symmetric (e.g., taxa1 vs taxa2 == taxa2 vs taxa1),
# it is possible to speed-up calculations by calculating association only for unique
# variable-pairs. Use "symmetric" to choose whether to measure association for only
# other half of of variable-pairs.
result <- getExperimentCrossAssociation(mae, experiment1 = "microbiota", experiment2 = "microbiota",
                                        assay.type1 = "counts", assay.type2 = "counts",
                                        symmetric = TRUE)

# For big data sets, calculation might take long. To make calculations quicker, you can take
# a random sample from data. In a complex biological problems, random sample
# can describe the data enough. Here our random sample is 30 % of whole data.
sample_size <- 0.3
tse <- mae[[1]]
tse_sub <- tse[ sample( seq_len( nrow(tse) ), sample_size * nrow(tse) ), ]
result <- testExperimentCrossAssociation(tse_sub)

# It is also possible to choose variables from colData and calculate association
# between assay and sample metadata or between variables of sample metadata
mae[[1]] <- estimateDiversity(mae[[1]])
# colData_variable works similarly to assay.type. Instead of fetching an assay
# named assay.type from assay slot, it fetches a column named colData_variable
# from colData.
result <- getExperimentCrossAssociation(mae[[1]], assay.type1 = "counts",
                                        colData_variable2 = c("shannon", "coverage"))
```

`getPrevalence` *Calculation prevalence information for features across samples*

Description

These functions calculate the population prevalence for taxonomic ranks in a SummarizedExperiment-class object.

Usage

```
getPrevalence(x, ...)

## S4 method for signature 'ANY'
getPrevalence(x, detection = 0, include_lowest = FALSE, sort = FALSE, ...)

## S4 method for signature 'SummarizedExperiment'
getPrevalence(
  x,
  assay.type = assay_name,
  assay_name = "counts",
  as_relative = TRUE,
  rank = NULL,
  ...
)

getPrevalentTaxa(x, ...)

## S4 method for signature 'ANY'
getPrevalentTaxa(x, prevalence = 50/100, include_lowest = FALSE, ...)

## S4 method for signature 'SummarizedExperiment'
getPrevalentTaxa(
  x,
  rank = NULL,
  prevalence = 50/100,
  include_lowest = FALSE,
  ...
)

getPrevalentFeatures(x, ...)

## S4 method for signature 'ANY'
getPrevalentFeatures(x, ...)

getRareTaxa(x, ...)

## S4 method for signature 'ANY'
getRareTaxa(x, prevalence = 50/100, include_lowest = FALSE, ...)

## S4 method for signature 'SummarizedExperiment'
getRareTaxa(x, rank = NULL, prevalence = 50/100, include_lowest = FALSE, ...)

getRareFeatures(x, ...)

## S4 method for signature 'ANY'
getRareFeatures(x, ...)
```

```
subsetByPrevalentTaxa(x, ...)

## S4 method for signature 'SummarizedExperiment'
subsetByPrevalentTaxa(x, rank = NULL, ...)

subsetByPrevalentFeatures(x, ...)

## S4 method for signature 'ANY'
subsetByPrevalentFeatures(x, ...)

subsetByRareTaxa(x, ...)

## S4 method for signature 'SummarizedExperiment'
subsetByRareTaxa(x, rank = NULL, ...)

subsetByRareFeatures(x, ...)

## S4 method for signature 'ANY'
subsetByRareFeatures(x, ...)

getPrevalentAbundance(
  x,
  assay.type = assay_name,
  assay_name = "relabundance",
  ...
)

## S4 method for signature 'ANY'
getPrevalentAbundance(
  x,
  assay.type = assay_name,
  assay_name = "relabundance",
  ...
)

## S4 method for signature 'SummarizedExperiment'
getPrevalentAbundance(x, assay.type = assay_name, assay_name = "counts", ...)

agglomerateByPrevalence(x, ...)

## S4 method for signature 'SummarizedExperiment'
agglomerateByPrevalence(
  x,
  rank = taxonomyRanks(x)[1L],
  other_label = "Other",
  ...
)
```

Arguments

x	a SummarizedExperiment object
detection	Detection threshold for absence/presence. Either an absolute value compared directly to the values of x or a relative value between 0 and 1, if as_relative = TRUE.
include_lowest	logical scalar: Should the lower boundary of the detection and prevalence cutoffs be included? (default: FALSE)
sort	logical scalar: Should the result be sorted by prevalence? (default: FALSE)
assay.type	A single character value for selecting the assay to use for prevalence calculation.
assay_name	a single character value for specifying which assay to use for calculation. (Please use assay.type instead. At some point assay_name will be disabled.)
as_relative	logical scalar: Should the detection threshold be applied on compositional (relative) abundances? (default: TRUE)
rank, ...	additional arguments • If !is.null(rank) arguments are passed on to agglomerateByRank. See ?agglomerateByRank for more details. • for getPrevalentTaxa, getRareTaxa, subsetByPrevalentTaxa and subsetByRareTaxa additional parameters passed to getPrevalence • for getPrevalentAbundance additional parameters passed to getPrevalentTaxa
prevalence	Prevalence threshold (in 0 to 1). The required prevalence is strictly greater by default. To include the limit, set include_lowest to TRUE.
other_label	A single character valued used as the label for the summary of non-prevalent taxa. (default: other_label = "Other")

Details

getPrevalence calculates the relative frequency of samples that exceed the detection threshold. For SummarizedExperiment objects, the prevalence is calculated for the selected taxonomic rank, otherwise for the rows. The absolute population prevalence can be obtained by multiplying the prevalence by the number of samples (ncol(x)). If as_relative = TRUE the relative frequency (between 0 and 1) is used to check against the detection threshold.

The core abundance index from getPrevalentAbundance gives the relative proportion of the core species (in between 0 and 1). The core taxa are defined as those that exceed the given population prevalence threshold at the given detection level as set for getPrevalentTaxa.

subsetPrevalentTaxa and subsetRareTaxa return a subset of x. The subset includes the most prevalent or rare taxa that are calculated with getPrevalentTaxa or getRareTaxa respectively.

getPrevalentTaxa returns taxa that are more prevalent with the given detection threshold for the selected taxonomic rank.

getRareTaxa returns complement of getPrevalentTaxa.

Value

subsetPrevalentTaxa and subsetRareTaxa return subset of x.

All other functions return a named vectors:

- getPrevalence returns a numeric vector with the names being set to either the row names of x or the names after agglomeration.
- getPrevalentAbundance returns a numeric vector with the names corresponding to the column name of x and include the joint abundance of prevalent taxa.
- getPrevalentTaxa and getRareTaxa return a character vector with only the names exceeding the threshold set by prevalence, if the rownames of x is set. Otherwise an integer vector is returned matching the rows in x.

Author(s)

Leo Lahti For getPrevalentAbundance: Leo Lahti and Tuomas Borman. Contact: microbiome.github.io

References

A Salonen et al. The adult intestinal core microbiota is determined by analysis depth and health status. Clinical Microbiology and Infection 18(S4):16 20, 2012. To cite the R package, see citation('mia')

See Also

agglomerateByRank, getTopTaxa

Examples

```
data(GlobalPatterns)
tse <- GlobalPatterns
# Get prevalence estimates for individual ASV/OTU
prevalence.frequency <- getPrevalence(tse,
                                      detection = 0,
                                      sort = TRUE,
                                      as_relative = TRUE)
head(prevalence.frequency)

# Get prevalence estimates for phylums
# - the getPrevalence function itself always returns population frequencies
prevalence.frequency <- getPrevalence(tse,
                                      rank = "Phylum",
                                      detection = 0,
                                      sort = TRUE,
                                      as_relative = TRUE)
head(prevalence.frequency)

# - to obtain population counts, multiply frequencies with the sample size,
# which answers the question "In how many samples is this phylum detectable"
prevalence.count <- prevalence.frequency * ncol(tse)
```

```
head(prevalence.count)

# Detection threshold 1 (strictly greater by default);
# Note that the data (GlobalPatterns) is here in absolute counts
# (and not compositional, relative abundances)
# Prevalence threshold 50 percent (strictly greater by default)
prevalent <- getPrevalentTaxa(tse,
                              rank = "Phylum",
                              detection = 10,
                              prevalence = 50/100,
                              as_relative = FALSE)
head(prevalent)

# Gets a subset of object that includes prevalent taxa
altExp(tse, "prevalent") <- subsetByPrevalentTaxa(tse,
                                      rank = "Family",
                                      detection = 0.001,
                                      prevalence = 0.55,
                                      as_relative = TRUE)
altExp(tse, "prevalent")

# getRareTaxa returns the inverse
rare <- getRareTaxa(tse,
                    rank = "Phylum",
                    detection = 1/100,
                    prevalence = 50/100,
                    as_relative = TRUE)
head(rare)

# Gets a subset of object that includes rare taxa
altExp(tse, "rare") <- subsetByRareTaxa(tse,
                              rank = "Class",
                              detection = 0.001,
                              prevalence = 0.001,
                              as_relative = TRUE)
altExp(tse, "rare")

# Names of both experiments, prevalent and rare, can be found from slot altExpNames
tse

data(esophagus)
getPrevalentAbundance(esophagus, assay.type = "counts")

# data can be aggregated based on prevalent taxonomic results
agglomerateByPrevalence(tse,
                        rank = "Phylum",
                        detection = 1/100,
                        prevalence = 50/100,
                        as_relative = TRUE)
```

---

isContaminant *decontam functions*

Description

The decontam functions isContaminant and isNotContaminant are made available for SummarizedExperiment objects.

Usage

```
## S4 method for signature 'SummarizedExperiment'
isContaminant(
  seqtab,
  assay.type = assay_name,
  assay_name = "counts",
  name = "isContaminant",
  concentration = NULL,
  control = NULL,
  batch = NULL,
  threshold = 0.1,
  normalize = TRUE,
  detailed = TRUE,
  ...
)

## S4 method for signature 'SummarizedExperiment'
isNotContaminant(
  seqtab,
  assay.type = assay_name,
  assay_name = "counts",
  name = "isNotContaminant",
  control = NULL,
  threshold = 0.5,
  normalize = TRUE,
  detailed = FALSE,
  ...
)

addContaminantQC(x, name = "isContaminant", ...)

## S4 method for signature 'SummarizedExperiment'
addContaminantQC(x, name = "isContaminant", ...)

addNotContaminantQC(x, name = "isNotContaminant", ...)

## S4 method for signature 'SummarizedExperiment'
addNotContaminantQC(x, name = "isNotContaminant", ...)
```

Arguments

seqtab, x	a SummarizedExperiment
assay.type	A single character value for selecting the assay to use.

assay_name	a single character value for specifying which assay to use for calculation. (Please use assay.type instead. At some point assay_name will be disabled.)
name	A name for the column of the colData in which the contaminant information should be stored.
concentration	NULL or a single character value. Defining a column with numeric values from the colData to use as concentration information. (default: concentration = NULL)
control	NULL or a single character value. Defining a column with logical values from the colData to define control and non-control samples. (default: control = NULL)
batch	NULL or a single character value. Defining a column with values interpretable as a factor from the colData to use as batch information. (default: batch = NULL)
threshold	numeric scalar. See decontam:isContaminant or decontam:isNotContaminant
normalize, detailed	logical scalar. See decontam:isContaminant or decontam:isNotContaminant
...	• for isContaminant/ isNotContaminant: arguments passed on to decontam:isContaminant or decontam:isNotContaminant • for addContaminantQC/addNotContaminantQC: arguments passed on to isContaminant/ isNotContaminant

Value

for isContaminant/ isNotContaminant a DataFrame or for addContaminantQC/addNotContaminantQC a modified object of class(x)

See Also

decontam:isContaminant, decontam:isNotContaminant

Examples

```
data(esophagus)
# setup of some mock data
colData(esophagus)$concentration <- c(1,2,3)
colData(esophagus)$control <- c(FALSE,FALSE,TRUE)

isContaminant(esophagus,
              method = "frequency",
              concentration = "concentration")
esophagus <- addContaminantQC(esophagus,
                              method = "frequency",
                              concentration = "concentration")
colData(esophagus)

isNotContaminant(esophagus, control = "control")
esophagus <- addNotContaminantQC(esophagus, control = "control")
colData(esophagus)
```

loadFromMetaphlan *Import Metaphlan results to* TreeSummarizedExperiment

Description

Import Metaphlan results to TreeSummarizedExperiment

Arguments

file	a single character value defining the file path of the Metaphlan file. The file must be in merged Metaphlan format.
sample_meta	a single character value defining the file path of the sample metadata file. The file must be in tsv format (default: sample_meta = NULL).
phy_tree	a single character value defining the file path of the phylogenetic tree. (default: phy_tree = NULL).
...	additional arguments: • assay.type: A single character value for naming assay (default: assay.type = "counts") • assay_name: A single character value for specifying which assay to use for calculation. (Please use assay.type instead. At some point assay_name will be disabled.) • removeTaxaPrefixes: TRUE or FALSE: Should taxonomic prefixes be removed? (default: removeTaxaPrefixes = FALSE)

Details

Import Metaphlan results. Input must be in merged Metaphlan format. Data is imported so that data at the lowest rank is imported as a TreeSummarizedExperiment object. Data at higher rank is imported as a SummarizedExperiment objects which are stored to altExp of TreeSummarizedExperiment object.

Value

A TreeSummarizedExperiment object

Author(s)

Leo Lahti and Tuomas Borman. Contact: microbiome.github.io

References

Beghini F, McIver LJ, Blanco-Míguez A, Dubois L, Asnicar F, Maharjan S, Mailyan A, Manghi P, Scholz M, Thomas AM, Valles-Colomer M, Weingart G, Zhang Y, Zolfo M, Huttenhower C, Franzosa EA, & Segata N (2021) Integrating taxonomic, functional, and strain-level profiling of diverse microbial communities with bioBakery 3. Elife 10:e65088. doi: 10.7554/eLife.65088

See Also

makeTreeSEFromPhyloseq makeTreeSEFromBiom makeTreeSEFromDADA2 loadFromQIIME2 loadFromMothur

Examples

```
# (Data is from tutorial
# https://github.com/biobakery/biobakery/wiki/metaphlan3#merge-outputs)

# File path
file_path <- system.file("extdata", "merged_abundance_table.txt", package = "mia")
# Import data
tse <- loadFromMetaphlan(file_path)
# Data at the lowest rank
tse
# Data at higher rank is stored in altExp
altExps(tse)
# Higher rank data is in SE format, for example, Phylum rank
altExp(tse, "Phylum")
```

---

loadFromMothur *Import Mothur results as a* TreeSummarizedExperiment

---

Description

This method creates a TreeSummarizedExperiment object from Mothur files provided as input.

Usage

```
loadFromMothur(sharedFile, taxonomyFile = NULL, designFile = NULL)
```

Arguments

sharedFile	a single character value defining the file path of the feature table to be imported. The File has to be in shared file format as defined in Mothur documentation.
taxonomyFile	a single character value defining the file path of the taxonomy table to be imported. The File has to be in taxonomy file or constaxonomy file format as defined in Mothur documentation. (default: taxonomyFile = NULL).
designFile	a single character value defining the file path of the sample metadata to be imported. The File has to be in desing file format as defined in Mothur documentation. (default: designFile = NULL).

Details

Results exported from Mothur can be imported as a SummarizedExperiment using loadFromMothur. Except for the sharedFile, the other data types, taxonomyFile, and designFile, are optional, but are highly encouraged to be provided.

Value

A TreeSummarizedExperiment object

Author(s)

Leo Lahti and Tuomas Borman. Contact: microbiome.github.io

References

https://mothur.org/ https://mothur.org/wiki/shared_file/ https://mothur.org/wiki/taxonomy_file/ https://mothur.org/wiki/constaxonomy_file/ https://mothur.org/wiki/design_file/

See Also

makeTreeSEFromPhyloseq makeTreeSEFromBiom makeTreeSEFromDADA2 loadFromQIIME2

Examples

```
# Abundance table
counts <- system.file("extdata", "mothur_example.shared", package = "mia")
# Taxa table (in "cons.taxonomy" or "taxonomy" format)
taxa <- system.file("extdata", "mothur_example.cons.taxonomy", package = "mia")
#taxa <- system.file("extdata", "mothur_example.taxonomy", package = "mia")
# Sample meta data
meta <- system.file("extdata", "mothur_example.design", package = "mia")

# Creates se object from files
se <- loadFromMothur(counts, taxa, meta)
# Convert SE to TreeSE
tse <- as(se, "TreeSummarizedExperiment")
tse
```

---

loadFromQIIME2 *Import QIIME2 results to* TreeSummarizedExperiment

---

Description

Results exported from QIMME2 can be imported as a TreeSummarizedExperiment using loadFromQIIME2. Except for the featureTableFile, the other data types, taxonomyTableFile, refSeqFile and phyTreeFile, are optional, but are highly encouraged to be provided.

Import the QIIME2 artifacts to R.

Usage

```
loadFromQIIME2(
  featureTableFile,
  taxonomyTableFile = NULL,
  sampleMetaFile = NULL,
  featureNamesAsRefSeq = TRUE,
  refSeqFile = NULL,
  phyTreeFile = NULL,
  ...
)

readQZA(file, temp = tempdir(), ...)
```

Arguments

`featureTableFile`	a single `character` value defining the file path of the feature table to be imported.
`taxonomyTableFile`	a single `character` value defining the file path of the taxonomy table to be imported. (default: `taxonomyTableFile = NULL`).
`sampleMetaFile`	a single `character` value defining the file path of the sample metadata to be imported. The file has to be in tsv format. (default: `sampleMetaFile = NULL`).
`featureNamesAsRefSeq`	`TRUE` or `FALSE`: Should the feature names of the feature table be regarded as reference sequences? This setting will be disregarded, if `refSeqFile` is not `NULL`. If the feature names do not contain valid DNA characters only, the reference sequences will not be set.
`refSeqFile`	a single `character` value defining the file path of the reference sequences for each feature. (default: `refSeqFile = NULL`).
`phyTreeFile`	a single `character` value defining the file path of the phylogenetic tree. (default: `phyTreeFile = NULL`).
`...`	additional arguments: • `temp`: the temporary directory used for decompressing the data. (default: `tempdir()`) • `removeTaxaPrefixes`: `TRUE` or `FALSE`: Should taxonomic prefixes be removed? (default: `removeTaxaPrefixes = FALSE`)
`file`	character, path of the input qza file. Only files in format of `BIOMV210DirFmt` (feature table), `TSVTaxonomyDirectoryFormat` (taxonomic table), `NewickDirectoryFormat` (phylogenetic tree) and `DNASequencesDirectoryFormat` (representative sequences) are supported right now.
`temp`	character, a temporary directory in which the qza file will be decompressed to, default `tempdir()`.

Details

Both arguments `featureNamesAsRefSeq` and `refSeqFile` can be used to define reference sequences of features. `featureNamesAsRefSeq` is only taken into account, if `refSeqFile` is `NULL`. No reference sequences are tried to be created, if `featureNameAsRefSeq` is `FALSE` and `refSeqFile` is `NULL`.

Value

A `TreeSummarizedExperiment` object

`matrix` object for feature table, `DataFrame` for taxonomic table, `ape::phylo` object for phylogenetic tree, `Biostrings::DNAStringSet` for representative sequences of taxa.

Author(s)

Yang Cao

References

Bolyen E et al. 2019: Reproducible, interactive, scalable and extensible microbiome data science using QIIME 2. Nature Biotechnology 37: 852–857. https://doi.org/10.1038/s41587-019-0209-9

https://qiime2.org

See Also

`makeTreeSEFromPhyloseq` `makeTreeSEFromBiom` `makeTreeSEFromDADA2` `loadFromMothur`

Examples

```
featureTableFile <- system.file("extdata", "table.qza", package = "mia")
taxonomyTableFile <- system.file("extdata", "taxonomy.qza", package = "mia")
sampleMetaFile <- system.file("extdata", "sample-metadata.tsv", package = "mia")
phyTreeFile <- system.file("extdata", "tree.qza", package = "mia")
refSeqFile <- system.file("extdata", "refseq.qza", package = "mia")
tse <- loadFromQIIME2(
  featureTableFile = featureTableFile,
  taxonomyTableFile = taxonomyTableFile,
  sampleMetaFile = sampleMetaFile,
  refSeqFile = refSeqFile,
  phyTreeFile = phyTreeFile
)

tse
# Read individual files
featureTableFile <- system.file("extdata", "table.qza", package = "mia")
taxonomyTableFile <- system.file("extdata", "taxonomy.qza", package = "mia")
sampleMetaFile <- system.file("extdata", "sample-metadata.tsv", package = "mia")

assay <- readQZA(featureTableFile)
rowdata <- readQZA(taxonomyTableFile, removeTaxaPrefixes = TRUE)
coldata <- read.table(sampleMetaFile, header = TRUE, sep = "\t", comment.char = "")
```

```
# Assign rownames
rownames(coldata) <- coldata[, 1]
coldata[, 1] <- NULL

# Order coldata based on assay
coldata <- coldata[match(colnames(assay), rownames(coldata)), ]

# Create SE from individual files
se <- SummarizedExperiment(assays = list(assay), rowData = rowdata, colData = coldata)
se
```

`makePhyloseqFromTreeSE` *Create a phyloseq object from a TreeSummarizedExperiment object*

Description

This function creates a phyloseq object from a TreeSummarizedExperiment object. By using `assay.type`, it is possible to specify which table from `assay` is added to the phyloseq object.

Usage

```
makePhyloseqFromTreeSE(x, ...)

## S4 method for signature 'SummarizedExperiment'
makePhyloseqFromTreeSE(x, assay.type = "counts", assay_name = NULL, ...)

## S4 method for signature 'TreeSummarizedExperiment'
makePhyloseqFromTreeSE(x, tree_name = "phylo", ...)

makePhyloseqFromTreeSummarizedExperiment(x, ...)

## S4 method for signature 'ANY'
makePhyloseqFromTreeSummarizedExperiment(x, ...)
```

Arguments

`x`	a `TreeSummarizedExperiment` object
`...`	additional arguments
`assay.type`	A single character value for selecting the `assay` to be included in the phyloseq object that is created. (By default: `assay.type = "counts"`)
`assay_name`	a single `character` value for specifying which assay to use for calculation. (Please use `assay.type` instead. At some point `assay_name` will be disabled.)
`tree_name`	a single `character` value for specifying which tree will be included in the phyloseq object that is created, (By default: `tree_name = "phylo"`)

Details

makePhyloseqFromTreeSE is used for creating a phyloseq object from TreeSummarizedExperiment object.

Value

An object of class Phyloseq object.

Author(s)

Leo Lahti and Tuomas Borman. Contact: microbiome.github.io

Examples

```
# Get tse object
data(GlobalPatterns)
tse <- GlobalPatterns

# Create a phyloseq object from it
phy <- makePhyloseqFromTreeSE(tse)
phy

# By default the chosen table is counts, but if there are other tables,
# they can be chosen with assay.type.

# Counts relative abundances table
tse <- transformCounts(tse, method = "relabundance")
phy2 <- makePhyloseqFromTreeSE(tse, assay.type = "relabundance")
phy2
```

---

makeTreeSEFromBiom *Loading a biom file*

Description

For convenience a few functions are available to convert data from a 'biom' file or object into a TreeSummarizedExperiment

Usage

```
loadFromBiom(file, ...)

makeTreeSEFromBiom(
  obj,
  removeTaxaPrefixes = FALSE,
  rankFromPrefix = FALSE,
  ...
)

makeTreeSummarizedExperimentFromBiom(obj, ...)
```

Arguments

file	biom file location
...	optional arguments (not used).
obj	object of type biom
removeTaxaPrefixes	TRUE or FALSE: Should taxonomic prefixes be removed? (default removeTaxaPrefixes = FALSE)
rankFromPrefix	TRUE or FALSE: If file does not have taxonomic ranks on feature table, should they be scraped from prefixes? (default rankFromPrefix = FALSE)

Value

An object of class TreeSummarizedExperiment

See Also

makeTreeSEFromPhyloseq makeTreeSEFromDADA2 loadFromQIIME2 loadFromMothur

Examples

```
if(requireNamespace("biomformat")) {
  library(biomformat)
  # load from file
  rich_dense_file  = system.file("extdata", "rich_dense_otu_table.biom",
                                 package = "biomformat")
  se <- loadFromBiom(rich_dense_file, removeTaxaPrefixes = TRUE, rankFromPrefix = TRUE)

  # load from object
  x1 <- biomformat::read_biom(rich_dense_file)
  se <- makeTreeSEFromBiom(x1)
  # Convert SE to TreeSE
  tse <- as(se, "TreeSummarizedExperiment")
  tse
}
```

---

makeTreeSEFromDADA2 *Coerce 'DADA2' results to* TreeSummarizedExperiment

---

Description

makeTreeSEFromDADA2 is a wrapper for the mergePairs function from the dada2 package.

Usage

```
makeTreeSEFromDADA2(...)

makeTreeSummarizedExperimentFromDADA2(...)
```

Arguments

...	See mergePairs function for more details.

Details

A count matrix is constructed via makeSequenceTable(mergePairs(...)) and rownames are dynamically created as ASV(N) with N from 1 to nrow of the count tables. The colnames and rownames from the output of makeSequenceTable are stored as colnames and in the referenceSeq slot of the TreeSummarizedExperiment, respectively.

Value

An object of class TreeSummarizedExperiment

See Also

makeTreeSEFromPhyloseq makeTreeSEFromBiom loadFromQIIME2 loadFromMothur

Examples

```
if(requireNamespace("dada2")) {
  fnF <- system.file("extdata", "sam1F.fastq.gz", package="dada2")
  fnR = system.file("extdata", "sam1R.fastq.gz", package="dada2")
  dadaF <- dada2::dada(fnF, selfConsist=TRUE)
  dadaR <- dada2::dada(fnR, selfConsist=TRUE)

  tse <- makeTreeSEFromDADA2(dadaF, fnF, dadaR, fnR)
  tse
}
```

---

makeTreeSEFromPhyloseq

Coerce a phyloseq *object to a* TreeSummarizedExperiment

Description

makeTreeSEFromPhyloseq converts phyloseq objects into TreeSummarizedExperiment objects.

Usage

```
makeTreeSEFromPhyloseq(obj)

makeTreeSummarizedExperimentFromPhyloseq(obj)

## S4 method for signature 'ANY'
makeTreeSummarizedExperimentFromPhyloseq(obj)
```

Arguments

obj	a phyloseq object

Details

All data stored in a phyloseq object is transfered.

Value

An object of class TreeSummarizedExperiment

See Also

makeTreeSEFromBiom makeTreeSEFromDADA2 loadFromQIIME2 loadFromMothur

Examples

```
if (requireNamespace("phyloseq")) {
    data(GlobalPatterns, package="phyloseq")
    makeTreeSEFromPhyloseq(GlobalPatterns)
    data(enterotype, package="phyloseq")
    makeTreeSEFromPhyloseq(enterotype)
    data(esophagus, package="phyloseq")
    makeTreeSEFromPhyloseq(esophagus)
}
```

---

meltAssay *Converting a SummarizedExperiment object into a long data.frame*

Description

meltAssay Converts a SummarizedExperiment object into a long data.frame which can be used for tidyverse-tools.

Usage

```
meltAssay(
  x,
  assay.type = assay_name,
  assay_name = "counts",
  add_row_data = NULL,
  add_col_data = NULL,
  feature_name = "FeatureID",
  sample_name = "SampleID",
  ...
)

## S4 method for signature 'SummarizedExperiment'
```

```
meltAssay(
  x,
  assay.type = assay_name,
  assay_name = "counts",
  add_row_data = NULL,
  add_col_data = NULL,
  feature_name = "FeatureID",
  sample_name = "SampleID",
  ...
)
```

Arguments

x	A numeric matrix or a SummarizedExperiment
assay.type	a character value to select an assayNames
assay_name	a single character value for specifying which assay to use for calculation. (Please use assay.type instead. At some point assay_name will be disabled.)
add_row_data	NULL, TRUE or a character vector to select information from the rowData to add to the molten assay data. If add_row_data = NULL no data will be added, if add_row_data = TRUE all data will be added and if add_row_data is a character vector, it will be used to subset to given column names in rowData. (default: add_row_data = NULL)
add_col_data	NULL, TRUE or a character vector to select information from the colData to add to the molten assay data. If add_col_data = NULL no data will be added, if add_col_data = TRUE all data will be added and if add_col_data is a character vector, it will be used to subset to given column names in colData. (default: add_col_data = NULL)
feature_name	a character scalar to use as the output's name for the feature identifier. (default: feature_name = "FeatureID")
sample_name	a character scalar to use as the output's name for the sample identifier. (default: sample_name = "SampleID")
...	optional arguments: • check_names A boolean value passed to data.frame function's check.name argument. Determines if sample names are checked that they are syntactically valid variable names and are not duplicated. If they are not, sample names are modified. (default: check_names = TRUE)

Details

If the colData contains a column "SampleID" or the rowData contains a column "FeatureID", they will be renamed to "SampleID_col" and "FeatureID_row", if row names or column names are set.

Value

A tibble with the molten data. The assay values are given in a column named like the selected assay assay.type. In addition, a column "FeatureID" will contain the rownames, if set, and analogously a column "SampleID" with the colnames, if set

Author(s)

Sudarshan A. Shetty

Examples

```
data(GlobalPatterns)
molten_tse <- meltAssay(GlobalPatterns,
                        assay.type = "counts",
                        add_row_data = TRUE,
                        add_col_data = TRUE
                        )
molten_tse
```

---

merge-methods *Merge a subset of the rows or columns of a* SummarizedExperiment

---

Description

mergeRows/mergeCols merge data on rows or columns of a SummarizedExperiment as defined by a factor alongside the chosen dimension. Metadata from the rowData or colData are retained as defined by archetype.

Usage

```
mergeRows(x, f, archetype = 1L, ...)

mergeCols(x, f, archetype = 1L, ...)

## S4 method for signature 'SummarizedExperiment'
mergeRows(x, f, archetype = 1L, ...)

## S4 method for signature 'SummarizedExperiment'
mergeCols(x, f, archetype = 1L, ...)

## S4 method for signature 'TreeSummarizedExperiment'
mergeRows(x, f, archetype = 1L, mergeTree = FALSE, mergeRefSeq = FALSE, ...)

## S4 method for signature 'TreeSummarizedExperiment'
mergeCols(x, f, archetype = 1L, mergeTree = FALSE, ...)
```

Arguments

x	a SummarizedExperiment or a TreeSummarizedExperiment
f	A factor for merging. Must be the same length as nrow(x)/ncol(x). Rows/Cols corresponding to the same level will be merged. If length(levels(f)) == nrow(x)/ncol(x), x will be returned unchanged.

archetype — Of each level of f, which element should be regarded as the archetype and metadata in the columns or rows kept, while merging? This can be single integer value or an integer vector of the same length as levels(f). (Default: archetype = 1L, which means the first element encountered per factor level will be kept)

... — optional arguments:

- passed onto sumCountsAcrossFeatures, except subset_row, subset_col

mergeTree — TRUE or FALSE: should to rowTree() also be merged? (Default: mergeTree = FALSE)

mergeRefSeq — TRUE or FALSE: should a consensus sequence calculate? If set to FALSE, the result from archetype is returned; If set to TRUE the result from DECIPHER::ConsensusSequence is returned. (Default: mergeRefSeq = FALSE)

Details

assay are agglomerated, i.e.. summed up. Other than counts / absolute values might lead to meaningless values.

These functions are similar to sumCountsAcrossFeatures. However, additional support for TreeSummarizedExperiment was added and science field agnostic names were used. In addition the archetype argument lets the user select how to preserve row or column data.

For merge data of assays the function from scuttle are used.

Value

an object with the same class x with the specified entries merged into one entry in all relevant components.

See Also

sumCountsAcrossFeatures

Examples

```
data(esophagus)
esophagus
plot(rowTree(esophagus))
# get a factor for merging
f <- factor(regmatches(rownames(esophagus),
                       regexpr("^[0-9]*_[0-9]*",rownames(esophagus))))
merged <- mergeRows(esophagus,f, mergeTree = TRUE)
plot(rowTree(merged))
#
data(GlobalPatterns)
GlobalPatterns
merged <- mergeCols(GlobalPatterns,colData(GlobalPatterns)$SampleType)
merged
```

mergeSEs *Merge SE objects into single SE object.*

Description

Merge SE objects into single SE object.

Usage

```
mergeSEs(x, ...)

## S4 method for signature 'SimpleList'
mergeSEs(
  x,
  assay.type = "counts",
  assay_name = NULL,
  join = "full",
  missing_values = NA,
  collapse_samples = FALSE,
  collapse_features = TRUE,
  verbose = TRUE,
  ...
)

## S4 method for signature 'SummarizedExperiment'
mergeSEs(x, y = NULL, ...)

## S4 method for signature 'list'
mergeSEs(x, ...)

full_join(x, ...)

## S4 method for signature 'ANY'
full_join(x, ...)

inner_join(x, ...)

## S4 method for signature 'ANY'
inner_join(x, ...)

left_join(x, ...)

## S4 method for signature 'ANY'
left_join(x, ...)

right_join(x, ...)
```

```
## S4 method for signature 'ANY'
right_join(x, ...)
```

Arguments

x	a SummarizedExperiment object or a list of SummarizedExperiment objects.
...	optional arguments (not used).
assay.type	A character value for selecting the assay to be merged. (By default: assay.type = "counts")
assay_name	(Deprecated) alias for assay.type.
join	A single character value for selecting the joining method. Must be 'full', 'inner', 'left', or 'right'. 'left' and 'right' are disabled when more than two objects are being merged. (By default: join = "full")
missing_values	NA, 0, or a single character values specifying the notation of missing values. (By default: missing_values = NA)
collapse_samples	A boolean value for selecting whether to collapse identically named samples to one. (By default: collapse_samples = FALSE)
collapse_features	A boolean value for selecting whether to collapse identically named features to one. Since all taxonomy information is taken into account, this concerns rownames-level (usually strain level) comparison. Often OTU or ASV level is just an arbitrary number series from sequencing machine meaning that the OTU information is not comparable between studies. With this option, it is possible to specify whether these strains are combined if their taxonomy information along with OTU number matches. (By default: collapse_features = TRUE)
verbose	A single boolean value to choose whether to show messages. (By default: verbose = TRUE)
y	a SummarizedExperiment object when x is a SummarizedExperiment object. Disabled when x is a list.

Details

This function merges multiple SummarizedExperiment objects. It combines rowData, assays, and colData so that the output includes each unique row and column ones. The merging is done based on rownames and colnames. rowTree and colTree are preserved if linkage between rows/cols and the tree is found.

Equally named rows are interpreted as equal. Further matching based on rowData is not done. For samples, collapsing is disabled by default meaning that equally named samples that are stored in different objects are interpreted as unique. Collapsing can be enabled with collapse_samples = TRUE when equally named samples describe the same sample.

If, for example, all rows are not shared with individual objects, there are missing values in assays. The notation of missing can be specified with the missing_values argument. If input consists of TreeSummarizedExperiment objects, also rowTree, colTree, and referenceSeq are preserved if possible. The data is preserved if all the rows or columns can be found from it.

Compared to cbind and rbind mergeSEs allows more freely merging since cbind and rbind expect that rows and columns are matching, respectively.

You can choose joining methods from 'full', 'inner', 'left', and 'right'. In all the methods, all the samples are included in the result object. However, with different methods, it is possible to choose which rows are included.

- full – all unique features
- inner – all shared features
- left – all the features of the first object
- right – all the features of the second object

You can also doe e.g., a full join by using a function full_join which is an alias for mergeSEs. Also other joining methods have dplyr-like aliases.

The output depends on the input. If the input contains SummarizedExperiment object, then the output will be SummarizedExperiment. When all the input objects belong to TreeSummarizedExperiment, the output will be TreeSummarizedExperiment.

Value

A single SummarizedExperiment object.

Author(s)

Leo Lahti and Tuomas Borman. Contact: microbiome.github.io

See Also

- TreeSummarizedExperiment::cbind
- TreeSummarizedExperiment::rbind
- full_join
- inner_join
- left_join
- right_join

Examples

```
data(GlobalPatterns)
data(esophagus)
data(enterotype)

# Take only subsets so that it wont take so long
tse1 <- GlobalPatterns[1:100, ]
tse2 <- esophagus
tse3 <- enterotype[1:100, ]

# Merge two TreeSEs
tse <- mergeSEs(tse1, tse2)
```

```
# Merge a list of TreeSEs
list <- SimpleList(tse1, tse2, tse3)
tse <- mergeSEs(list, assay.type = "counts", missing_values = 0)
tse

# With 'join', it is possible to specify the merging method. Subsets are used
# here just to show the functionality
tse_temp <- mergeSEs(tse[1:10, 1:10], tse[5:100, 11:20], join = "left")
tse_temp

# You can also do a left_join by using alias "left_join"
tse_temp <- left_join(tse[1:10, 1:10], tse[5:100, 11:20])

# If your objects contain samples that describe one and same sample,
# you can collapse equally named samples to one by specifying 'collapse_samples'
tse_temp <- inner_join(list(tse[1:10, 1], tse[1:20, 1], tse[1:5, 1]),
                       collapse_samples = TRUE)
tse_temp

# Merge all available assays
tse <- transformCounts(tse, method="relabundance")
ts1 <- transformCounts(tse1, method="relabundance")
tse_temp <- mergeSEs(tse, tse1, assay.type = assayNames(tse))
```

`mia-datasets` *mia datasets*

Description

These datasets are conversions of the **phyloseq** datasets `GlobalPatterns`, `enterotype`, and `esophagus` into the TreeSummarizedExperiment data container.

`dmn_se` contains an example `SummarizedExperiment` derived from data in the **DirichletMultinomial** package. See `?calculateDMN` for more details.

Global patterns of 16S rRNA diversity at a depth of millions of sequences per sample (2011)

This work compared the microbial communities from 25 environmental samples and three known "mock communities" at a an averag depth of 3.1 million reads per sample. Authors reproduced diversity patterns seen in many other published studies, while invesitigating technical issues/bias by applying the same techniques to simulated microbial communities of known composition. Many thanks to J. Gregory Caporaso for providing the OTU-clustered data files for inclusion in the **phyloseq** package, from which this data `TreeSummarizedExperiment` version was then converted.

The enterotype data of the human gut microbiome (Arumugam et al. 2011) includes taxonomic profiling for 280 fecal samples from 22 subjects based on shotgun DNA sequencing. The authors claimed that the data naturally clumps into three community-level clusters, or "enterotypes", that are not immediately explained by sequencing technology or demographic features of the subjects. A later addendum (2014) the authors stated that enterotypes "should not be seen as discrete clusters, but as a way of stratifying samples to reduce complexity."

The esophagus data set from Pei et al. (2004) includes 3 samples from 3 human adults based on biopsies analysed with 16S rDNA PCR. The 16S rRNA sequence processing has been provided in the mothur wiki at the link below.

This data set from Turnbaugh et al. (2009) was used to introduce Dirichlet Multinomial Mixtures (DMM) for microbiota stratification by Holmes et al. (2012).

PeerJ data by Potbhare et al. (2022) includes skin microbial profiles of 58 volunteers with multiple factors. 16S r-RNA sequencing of V3-V4 regions was done to generate millions of read using illumina platform. A standard bioinformatic and statistical analysis done to explore skin bacterial diversity and its association with age, diet, geographical locations. The authors investigated significant association of skin microbiota with individual's geographical location.

The HintikkaXO dataset contains high-throughput profiling data from 40 rat samples, including 39 biomarkers, 38 metabolites (NMR), and 12706 OTUs from 318 species, measured from Cecum. This is diet comparison study with High/Low fat diet and xylo-oligosaccaride supplementation. Column metadata is common for all experiments (microbiota, metabolites, biomarkers) and includes the following fields:

Usage

```
data(GlobalPatterns)

data(enterotype)

data(esophagus)

data(dmn_se)

data(peerj13075)

data(HintikkaXOData)
```

Format

An object of class `TreeSummarizedExperiment` with 19216 rows and 26 columns.

An object of class `TreeSummarizedExperiment` with 553 rows and 280 columns.

An object of class `TreeSummarizedExperiment` with 58 rows and 3 columns.

An object of class `SummarizedExperiment` with 130 rows and 278 columns.

An object of class `TreeSummarizedExperiment` with 674 rows and 58 columns.

An object of class `MultiAssayExperiment` of length 3.

Details

- Sample: Sample ID (character)
- Rat: Rat ID (factor)
- Site: Site of measurement ("Cecum"); single value
- Diet: Diet group (factor; combination of the Fat and XOS fields)

- Fat: Fat in Diet (factor; Low/High)
- XOS: XOS Diet Supplement (numeric; 0/1)

Row metadata of the microbiota data contains taxonomic information on the Phylum, Class, Order, Family, Genus, Species, and OTU levels.

Biomarker data contains 39 biomarkers.

Metabolite data contains 38 NMR metabolites.

Author(s)

Caporaso, J. G., et al.

Arumugam, M., Raes, J., et al.

Pei et al. <zhiheng.pei@med.nyu.edu>.

Turnbaugh, PJ et al.

Potbhare, R., et al.

Leo Lahti et al.

References

Caporaso, J. G., et al. (2011). Global patterns of 16S rRNA diversity at a depth of millions of sequences per sample. PNAS, 108, 4516-4522. http://www.pnas.org/content/108/suppl.1/4516.short

Arumugam, M., et al. (2011). Enterotypes of the human gut microbiome. Nature, 473(7346), 174-180. http://www.nature.com/doifinder/10.1038/nature09944 Supplemental information includes subject data. OTU-clustered data was initially downloaded from the publicly-accessible: http://www.bork.embl.de/Docu/Arumugam_et_al_2011/downloads.html

Arumugam, M., et al. (2014). Addendum: Enterotypes of the human gut microbiome. Nature 506, 516 (2014). https://doi.org/10.1038/nature13075

Pei, Z., Bini, E. J., Yang, L., Zhou, M., Francois, F., & Blaser, M. J. (2004). Bacterial biota in the human distal esophagus. Proceedings of the National Academy of Sciences of the United States of America, 101(12), 4250-4255. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC384727

McMurdie, J. & Holmes, S. (2013) *phyloseq*: An R Package for reproducible interactive analysis and graphics of microbiome census data. PLoS ONE. 8(4):e61217. https://doi.org/10.1371/journal.pone.0061217

Mothur-processed files and the sequence data can be downloaded at: http://www.mothur.org/wiki/Esophageal_community_analysis

Holmes I, Harris K, Quince C (2012). Dirichlet Multinomial Mixtures: Generative Models for Microbial Metagenomics. PLoS ONE 7(2): e30126. https://doi.org/10.1371/journal.pone.0030126

Turnbaugh PJ, Hamady M, Yatsunenko T, Cantarel BL, Duncan A, et al. (2009). A core gut microbiome in obese and lean twins. Nature 457: 480–484. https://doi.org/10.1038/nature07540

Potbhare, R., RaviKumar, A., Munukka, E., Lahti, L., & Ashma, R. (2022). Skin microbiota diversity among genetically unrelated individuals of Indian origin. PeerJ, 10, e13075. https://peerj.com/articles/13075/ Supplemental information includes OTU table and taxonomy table

and publicly-accessible from: https://www.doi.org/10.7717/peerj.13075/supp-1 https://www.doi.org/10.7717/peerj.13075/supp-2

Hintikka L et al. (2021): Xylo-oligosaccharides in prevention of hepatic steatosis and adipose tissue inflammation: associating taxonomic and metabolomic patterns in fecal microbiotas with bi-clustering. International Journal of Environmental Research and Public Health 18(8):4049 https://doi.org/10.3390/ijerph18084049

`perSampleDominantTaxa` *Get dominant taxa*

Description

These functions return information about the most dominant taxa in a SummarizedExperiment object.

Usage

```
perSampleDominantTaxa(
  x,
  assay.type = assay_name,
  assay_name = "counts",
  rank = NULL,
  ...
)

## S4 method for signature 'SummarizedExperiment'
perSampleDominantTaxa(
  x,
  assay.type = assay_name,
  assay_name = "counts",
  rank = NULL,
  ...
)

perSampleDominantFeatures(x, ...)

## S4 method for signature 'SummarizedExperiment'
perSampleDominantFeatures(x, ...)

addPerSampleDominantTaxa(x, name = "dominant_taxa", ...)

## S4 method for signature 'SummarizedExperiment'
addPerSampleDominantTaxa(x, name = "dominant_taxa", ...)

addPerSampleDominantFeatures(x, ...)

## S4 method for signature 'SummarizedExperiment'
addPerSampleDominantFeatures(x, ...)
```

Arguments

x	A SummarizedExperiment object.
assay.type	A single character value for selecting the assay to use for identifying dominant taxa.
assay_name	a single character value for specifying which assay to use for calculation. (Please use assay.type instead. At some point assay_name will be disabled.)
rank	A single character defining a taxonomic rank. Must be a value of the output of taxonomyRanks().
...	Additional arguments passed on to agglomerateByRank() when rank is specified.
name	A name for the column of the colData where the dominant taxa will be stored in when using addPerSampleDominantTaxa.

Details

addPerSampleDominantTaxa extracts the most abundant taxa in a SummarizedExperiment object, and stores the information in the colData. It is a wrapper for perSampleDominantTaxa.

With rank parameter, it is possible to agglomerate taxa based on taxonomic ranks. E.g. if 'Genus' rank is used, all abundances of same Genus are added together, and those families are returned. See agglomerateByRank() for additional arguments to deal with missing values or special characters.

Value

perSampleDominantTaxa returns a named character vector x while addPerSampleDominantTaxa returns SummarizedExperiment with additional column in colData named *name*.

Author(s)

Leo Lahti, Tuomas Borman and Sudarshan A. Shetty.

Examples

```
data(GlobalPatterns)
x <- GlobalPatterns

# Finds the dominant taxa.
sim.dom <- perSampleDominantTaxa(x, rank="Genus")

# Add information to colData
x <- addPerSampleDominantTaxa(x, rank = "Genus", name="dominant_genera")
colData(x)
```

relabundance *Getter / setter for relative abundance data*

Description

This function will be deprecated. Please use `assay(x, "relabundance")` instead. `relabundance` is a getter/setter for relative abundance stored in the assay slot 'relabundance' of a `TreeSummarizedExperiment` object. This is a shortcut function for `assay(x,"relabundance")`.

Usage

```
relabundance(x, ...)

relabundance(x) <- value

## S4 method for signature 'SummarizedExperiment'
relabundance(x)

## S4 replacement method for signature 'SummarizedExperiment'
relabundance(x) <- value
```

Arguments

x	a `TreeSummarizedExperiment` object
...	optional arguments not used currently.
value	a `matrix` to store as the the 'relabundance' assay

Value

For `relabundance` the matrix stored with the name "relabundance".

Examples

```
data(GlobalPatterns)
# Calculates relative abundances
GlobalPatterns <- transformCounts(GlobalPatterns, method="relabundance")
# Fetches calculated relative abundances
# head(relabundance(GlobalPatterns))
```

runCCA *Canonical Correspondence Analysis*

Description

These functions perform Canonical Correspondence Analysis on data stored in a SummarizedExperiment.

Usage

```
calculateCCA(x, ...)

runCCA(x, ...)

calculateRDA(x, ...)

runRDA(x, ...)

## S4 method for signature 'ANY'
calculateCCA(x, formula, variables, scale = TRUE, ...)

## S4 method for signature 'SummarizedExperiment'
calculateCCA(
  x,
  formula,
  variables,
  ...,
  assay.type = assay_name,
  assay_name = exprs_values,
  exprs_values = "counts"
)

## S4 method for signature 'SingleCellExperiment'
runCCA(x, ..., altexp = NULL, name = "CCA")

## S4 method for signature 'ANY'
calculateRDA(x, formula, variables, ...)

## S4 method for signature 'SummarizedExperiment'
calculateRDA(
  x,
  formula,
  variables,
  ...,
  assay.type = assay_name,
  assay_name = exprs_values,
  exprs_values = "counts"
)
```

```
## S4 method for signature 'SingleCellExperiment'
runRDA(x, ..., altexp = NULL, name = "RDA")
```

Arguments

x	For calculate* a SummarizedExperiment or a numeric matrix with columns as samples For run* a SingleCellExperiment or a derived object.
...	additional arguments passed to vegan::cca or vegan::dbrda
formula	If x is a SummarizedExperiment a formula can be supplied. Based on the right-hand side of the given formula colData is subset to variables. variables and formula can be missing, which turns the CCA analysis into a CA analysis and dbRDA into PCoA/MDS.
variables	When x is a SummarizedExperiment, variables can be used to specify variables from colData. When x is a matrix, variables is a data.frame or an object coercible to one containing the variables to use. All variables are used. Please subset, if you want to consider only some of them. variables and formula can be missing, which turns the CCA analysis into a CA analysis and dbRDA into PCoA/MDS.
scale	a logical scalar, should the expression values be standardized? scale is disabled when using *RDA functions. Please scale before performing RDA (Check examples.)
assay.type	a single character value for specifying which assay to use for calculation.
assay_name	a single character value for specifying which assay to use for calculation. (Please use assay.type instead. At some point assay_name will be disabled.)
exprs_values	a single character value for specifying which assay to use for calculation. (Please use assay.type instead.)
altexp	String or integer scalar specifying an alternative experiment containing the input data.
name	String specifying the name to be used to store the result in the reducedDims of the output.

Details

*CCA functions utilize vegan:cca and *RDA functions vegan:dbRDA. By default dbRDA is done with euclidean distances which equals to RDA.

Value

For calculateCCA a matrix with samples as rows and CCA dimensions as columns

For runCCA a modified x with the results stored in reducedDim as the given name

See Also

For more details on the actual implementation see cca and dbrda

Examples

```
library(scater)
data(GlobalPatterns)
GlobalPatterns <- runCCA(GlobalPatterns, data ~ SampleType)
plotReducedDim(GlobalPatterns,"CCA", colour_by = "SampleType")

GlobalPatterns <- runRDA(GlobalPatterns, data ~ SampleType)
plotReducedDim(GlobalPatterns,"CCA", colour_by = "SampleType")

# To scale values when using *RDA functions, use transformCounts(MARGIN = "features",
tse <- GlobalPatterns
tse <- transformCounts(tse, MARGIN = "features", method = "z")
# Data might include taxa that do not vary. Remove those because after z-transform
# their value is NA
tse <- tse[ rowSums( is.na( assay(tse, "z") ) ) == 0, ]
# Calculate RDA
tse <- runRDA(tse, formula = data ~ SampleType,
              assay.type = "z", name = "rda_scaled", na.action = na.omit)
# Plot
plotReducedDim(tse,"rda_scaled", colour_by = "SampleType")
```

---

runDPCoA *Calculation of Double Principal Correspondance analysis*

Description

Double Principal Correspondance analysis is made available via the ade4 package in typical fashion. Results are stored in the reducedDims and are available for all the expected functions.

Usage

```
calculateDPCoA(x, y, ...)

## S4 method for signature 'ANY,ANY'
calculateDPCoA(
  x,
  y,
  ncomponents = 2,
  ntop = NULL,
  subset_row = NULL,
  scale = FALSE,
  transposed = FALSE,
  ...
)

## S4 method for signature 'TreeSummarizedExperiment,missing'
calculateDPCoA(
  x,
```

```
  ...,
  assay.type = assay_name,
  assay_name = exprs_values,
  exprs_values = "counts",
  tree_name = "phylo"
)

runDPCoA(x, ..., altexp = NULL, name = "DPCoA")
```

Arguments

x	For calculateDPCoA, a numeric matrix of expression values where rows are features and columns are cells. Alternatively, a TreeSummarizedExperiment containing such a matrix. For runDPCoA a TreeSummarizedExperiment containing the expression values as well as a rowTree to calculate y using cophenetic.phylo.
y	a dist or a symmetric matrix compatible with ade4:dpcoa
...	Currently not used.
ncomponents	Numeric scalar indicating the number of DPCoA dimensions to obtain.
ntop	Numeric scalar specifying the number of features with the highest variances to use for dimensionality reduction. Alternatively NULL, if all features should be used. (default: ntop = NULL)
subset_row	Vector specifying the subset of features to use for dimensionality reduction. This can be a character vector of row names, an integer vector of row indices or a logical vector.
scale	Logical scalar, should the expression values be standardized?
transposed	Logical scalar, is x transposed with cells in rows?
assay.type	a single character value for specifying which assay to use for calculation.
assay_name	a single character value for specifying which assay to use for calculation. (Please use assay.type instead. At some point assay_name will be disabled.)
exprs_values	a single character value for specifying which assay to use for calculation. (Please use assay.type instead.)
tree_name	a single character value for specifying which rowTree will be used in calculation. (By default: tree_name = "phylo")
altexp	String or integer scalar specifying an alternative experiment containing the input data.
name	String specifying the name to be used to store the result in the reducedDims of the output.

Details

In addition to the reduced dimension on the features, the reduced dimension for samples are returned as well as sample_red attribute. eig, feature_weights and sample_weights are returned as attributes as well.

Value

For `calculateDPCoA` a matrix with samples as rows and CCA dimensions as columns

For `runDPCoA` a modified `x` with the results stored in `reducedDim` as the given `name`

See Also

`plotReducedDim` `reducedDims`

Examples

```
data(esophagus)
dpcoa <- calculateDPCoA(esophagus)
head(dpcoa)

esophagus <- runDPCoA(esophagus)
reducedDims(esophagus)

library(scater)
plotReducedDim(esophagus, "DPCoA")
```

---

runNMDS *Perform non-metric MDS on sample-level data*

Description

Perform non-metric multi-dimensional scaling (nMDS) on samples, based on the data in a `SingleCellExperiment` object.

Usage

```
calculateNMDS(x, ...)

## S4 method for signature 'ANY'
calculateNMDS(
  x,
  FUN = vegdist,
  nmdsFUN = c("isoMDS", "monoMDS"),
  ncomponents = 2,
  ntop = 500,
  subset_row = NULL,
  scale = FALSE,
  transposed = FALSE,
  keep_dist = FALSE,
  ...
)

## S4 method for signature 'SummarizedExperiment'
```

```
calculateNMDS(
  x,
  ...,
  assay.type = assay_name,
  assay_name = exprs_values,
  exprs_values = "counts",
  FUN = vegdist
)

## S4 method for signature 'SingleCellExperiment'
calculateNMDS(
  x,
  ...,
  assay.type = assay_name,
  assay_name = exprs_values,
  exprs_values = "counts",
  dimred = NULL,
  n_dimred = NULL,
  FUN = vegdist
)

runNMDS(x, ..., altexp = NULL, name = "NMDS")

plotNMDS(x, ..., ncomponents = 2)
```

Arguments

x	For `calculateNMDS`, a numeric matrix of expression values where rows are features and columns are cells. Alternatively, a `TreeSummarizedExperiment` containing such a matrix. For `runNMDS` a SingleCellExperiment
...	additional arguments to pass to `FUN` and `nmdsFUN`.
FUN	a `function` or `character` value with a function name returning a `dist` object
nmdsFUN	a `character` value to choose the scaling implementation, either "isoMDS" for `MASS::isoMDS` or "monoMDS" for `vegan::monoMDS`
ncomponents	Numeric scalar indicating the number of NMDS dimensions to obtain.
ntop	Numeric scalar specifying the number of features with the highest variances to use for dimensionality reduction.
subset_row	Vector specifying the subset of features to use for dimensionality reduction. This can be a character vector of row names, an integer vector of row indices or a logical vector.
scale	Logical scalar, should the expression values be standardized?
transposed	Logical scalar, is x transposed with cells in rows?
keep_dist	Logical scalar indicating whether the `dist` object calculated by `FUN` should be stored as 'dist' attribute of the matrix returned/stored by `calculateNMDS`/ `runNMDS`.

assay.type	a single character value for specifying which assay to use for calculation.
assay_name	a single character value for specifying which assay to use for calculation. (Please use assay.type instead. At some point assay_name will be disabled.)
exprs_values	a single character value for specifying which assay to use for calculation. (Please use assay.type instead.)
dimred	String or integer scalar specifying the existing dimensionality reduction results to use.
n_dimred	Integer scalar or vector specifying the dimensions to use if dimred is specified.
altexp	String or integer scalar specifying an alternative experiment containing the input data.
name	String specifying the name to be used to store the result in the reducedDims of the output.

Details

Either MASS::isoMDS or vegan::monoMDS are used internally to compute the NMDS components. If you supply a custom FUN, make sure that the arguments of FUN and nmdsFUN do not collide.

Value

For calculateNMDS, a matrix is returned containing the MDS coordinates for each sample (row) and dimension (column).

Author(s)

Felix Ernst

See Also

MASS::isoMDS, vegan::monoMDS for NMDS component calculation.

plotMDS, to quickly visualize the results.

Examples

```
# generate some example data
mat <- matrix(1:60, nrow = 6)
df <- DataFrame(n = c(1:6))
tse <- TreeSummarizedExperiment(assays = list(counts = mat),
                                rowData = df)
#
calculateNMDS(tse)

#
data(esophagus)
esophagus <- runNMDS(esophagus, FUN = vegan::vegdist, name = "BC")
esophagus <- runNMDS(esophagus, FUN = vegan::vegdist, name = "euclidean",
                     method = "euclidean")
reducedDims(esophagus)
```

splitByRanks *Split/Unsplit a* SingleCellExperiment *by taxonomic ranks*

Description

splitByRanks takes a SummarizedExperiment, splits it along the taxonomic ranks, aggregates the data per rank, converts the input to a SingleCellExperiment objects and stores the aggregated data as alternative experiments.

Usage

```
splitByRanks(x, ...)

## S4 method for signature 'SummarizedExperiment'
splitByRanks(x, ranks = taxonomyRanks(x), na.rm = TRUE, ...)

## S4 method for signature 'SingleCellExperiment'
splitByRanks(x, ranks = taxonomyRanks(x), na.rm = TRUE, ...)

## S4 method for signature 'TreeSummarizedExperiment'
splitByRanks(x, ranks = taxonomyRanks(x), na.rm = TRUE, ...)

unsplitByRanks(x, ...)

## S4 method for signature 'SingleCellExperiment'
unsplitByRanks(x, ranks = taxonomyRanks(x), keep_reducedDims = FALSE, ...)

## S4 method for signature 'TreeSummarizedExperiment'
unsplitByRanks(x, ranks = taxonomyRanks(x), keep_reducedDims = FALSE, ...)
```

Arguments

x	a SummarizedExperiment object
...	arguments passed to agglomerateByRank function for SummarizedExperiment objects and other functions. See agglomerateByRank for more details.
ranks	a character vector defining taxonomic ranks. Must all be values of taxonomyRanks() function.
na.rm	TRUE or FALSE: Should taxa with an empty rank be removed? Use it with caution, since results with NA on the selected rank will be dropped. This setting can be tweaked by defining empty.fields to your needs. (default: na.rm = TRUE)
keep_reducedDims	TRUE or FALSE: Should the reducedDims(x) be transferred to the result? Please note, that this breaks the link between the data used to calculate the reduced dims. (default: keep_reducedDims = FALSE)

Details

unsplitByRanks takes these alternative experiments and flattens them again into a single SummarizedExperiment.

splitByRanks will use by default all available taxonomic ranks, but this can be controlled by setting ranks manually. NA values are removed by default, since they would not make sense, if the result should be used for unsplitByRanks at some point. The input data remains unchanged in the returned SingleCellExperiment objects.

unsplitByRanks will remove any NA value on each taxonomic rank so that no ambiguous data is created. In additional, a column taxonomicLevel is created or overwritten in the rowData to specify from which alternative experiment this originates from. This can also be used for splitAltExps to split the result along the same factor again. The input data from the base objects is not returned, only the data from the altExp(). Be aware that changes to rowData of the base object are not returned, whereas only the colData of the base object is kept.

Value

For splitByRanks: SummarizedExperiment objects in a SimpleList.

For unsplitByRanks: x, with rowData and assay data replaced by the unsplit data. colData of x is kept as well and any existing rowTree is dropped as well, since existing rowLinks are not valid anymore.

See Also

splitOn unsplitOn mergeRows, sumCountsAcrossFeatures, agglomerateByRank, altExps, splitAltExps

Examples

```
data(GlobalPatterns)
# print the available taxonomic ranks
taxonomyRanks(GlobalPatterns)

# splitByRanks
altExps(GlobalPatterns) <- splitByRanks(GlobalPatterns)
altExps(GlobalPatterns)
altExp(GlobalPatterns,"Kingdom")
altExp(GlobalPatterns,"Species")

# unsplitByRanks
x <- unsplitByRanks(GlobalPatterns)
x
```

---

splitOn — *Split* TreeSummarizedExperiment *column-wise or row-wise based on grouping variable*

Description

Split TreeSummarizedExperiment column-wise or row-wise based on grouping variable

Usage

```
splitOn(x, ...)

## S4 method for signature 'SummarizedExperiment'
splitOn(x, f = NULL, ...)

## S4 method for signature 'SingleCellExperiment'
splitOn(x, f = NULL, ...)

## S4 method for signature 'TreeSummarizedExperiment'
splitOn(x, f = NULL, update_rowTree = FALSE, ...)

unsplitOn(x, ...)

## S4 method for signature 'list'
unsplitOn(x, update_rowTree = FALSE, ...)

## S4 method for signature 'SimpleList'
unsplitOn(x, update_rowTree = FALSE, ...)

## S4 method for signature 'SingleCellExperiment'
unsplitOn(x, altExpNames = names(altExps(x)), keep_reducedDims = FALSE, ...)
```

Arguments

x	A SummarizedExperiment object or a list of SummarizedExperiment objects.
...	Arguments passed to mergeRows/mergeCols function for SummarizedExperiment objects and other functions. See mergeRows for more details. • use_names A single boolean value to select whether to name elements of list by their group names.
f	A single character value for selecting the grouping variable from rowData or colData or a factor or vector with the same length as one of the dimensions. If f matches with both dimensions, MARGIN must be specified. Split by cols is not encouraged, since this is not compatible with storing the results in altExps.
update_rowTree	TRUE or FALSE: Should the rowTree be updated based on splitted data? Option is enabled when x is a TreeSummarizedExperiment object or a list of such objects. (By default: update_rowTree = FALSE)
altExpNames	a character vector specifying the alternative experiments to be unsplit. (By default: altExpNames = names(altExps(x)))
keep_reducedDims	TRUE or FALSE: Should the reducedDims(x) be transferred to the result? Please note, that this breaks the link between the data used to calculate the reduced dims. (By default: keep_reducedDims = FALSE)

Details

splitOn split data based on grouping variable. Splitting can be done column-wise or row-wise.

The returned value is a list of SummarizedExperiment objects; each element containing members of each group.

Value

For splitOn: SummarizedExperiment objects in a SimpleList.

For unsplitOn: x, with rowData and assay data replaced by the unsplit data. colData of x is kept as well and any existing rowTree is dropped as well, since existing rowLinks are not valid anymore.

Author(s)

Leo Lahti and Tuomas Borman. Contact: microbiome.github.io

See Also

splitByRanks unsplitByRanks mergeRows, sumCountsAcrossFeatures, agglomerateByRank, altExps, splitAltExps

Examples

```
data(GlobalPatterns)
tse <- GlobalPatterns
# Split data based on SampleType.
se_list <- splitOn(tse, f = "SampleType")

# List of SE objects is returned.
se_list

# Create arbitrary groups
rowData(tse)$group <- sample(1:10, nrow(tse), replace = TRUE)
colData(tse)$group <- sample(1:10, ncol(tse), replace = TRUE)

# Split based on rows
# Each element is named based on their group name. If you don't want to name
# elements, use use_name = FALSE. Since "group" can be found from rowdata and colData
# you must use MARGIN.
se_list <- splitOn(tse, f = "group", use_names = FALSE, MARGIN = 1)

# When column names are shared between elements, you can store the list to altExps
altExps(tse) <- se_list

altExps(tse)

# If you want to split on columns and update rowTree, you can do
se_list <- splitOn(tse, f = colData(tse)$group, update_rowTree = TRUE)

# If you want to combine groups back together, you can use unsplitBy
unsplitOn(se_list)
```

subsampleCounts *Subsample Counts*

Description

subsampleCounts will randomly subsample counts in SummarizedExperiment and return the a modified object in which each sample has same number of total observations/counts/reads.

Usage

```
subsampleCounts(
  x,
  assay.type = assay_name,
  assay_name = "counts",
  min_size = min(colSums2(assay(x))),
  seed = runif(1, 0, .Machine$integer.max),
  replace = TRUE,
  name = "subsampled",
  verbose = TRUE,
  ...
)

## S4 method for signature 'SummarizedExperiment'
subsampleCounts(
  x,
  assay.type = assay_name,
  assay_name = "counts",
  min_size = min(colSums2(assay(x))),
  seed = runif(1, 0, .Machine$integer.max),
  replace = TRUE,
  name = "subsampled",
  verbose = TRUE,
  ...
)
```

Arguments

x	A SummarizedExperiment object.
assay.type	A single character value for selecting the SummarizedExperiment assay used for random subsampling. Only counts are useful and other transformed data as input will give meaningless output.
assay_name	a single character value for specifying which assay to use for calculation. (Please use assay.type instead. At some point assay_name will be disabled.)
min_size	A single integer value equal to the number of counts being simulated this can equal to lowest number of total counts found in a sample or a user specified number.

seed	A random number seed for reproducibility of sampling.
replace	Logical Default is TRUE. The default is with replacement (replace=TRUE). See phyloseq::rarefy_even_depth for details on implications of this parameter.
name	A single character value specifying the name of transformed abundance table.
verbose	Logical Default is TRUE. When TRUE an additional message about the random number used is printed.
...	additional arguments not used

Details

Although the subsampling approach is highly debated in microbiome research, we include the subsampleCounts function because there may be some instances where it can be useful. Note that the output of subsampleCounts is not the equivalent as the input and any result have to be verified with the original dataset.

Value

subsampleCounts return x with subsampled data.

Author(s)

Sudarshan A. Shetty and Felix G.M. Ernst

References

McMurdie PJ, Holmes S. Waste not, want not: why rarefying microbiome data is inadmissible. PLoS computational biology. 2014 Apr 3;10(4):e1003531.

Gloor GB, Macklaim JM, Pawlowsky-Glahn V & Egozcue JJ (2017) Microbiome Datasets Are Compositional: And This Is Not Optional. Frontiers in Microbiology 8: 2224. doi: 10.3389/fmicb.2017.02224

Weiss S, Xu ZZ, Peddada S, Amir A, Bittinger K, Gonzalez A, Lozupone C, Zaneveld JR, Vázquez-Baeza Y, Birmingham A, Hyde ER. Normalization and microbial differential abundance strategies depend upon data characteristics. Microbiome. 2017 Dec;5(1):1-8.

Examples

```
# When samples in TreeSE are less than specified min_size, they will be removed.
# If after subsampling features are not present in any of the samples,
# they will be removed.
data("GlobalPatterns")
tse <- GlobalPatterns
tse.subsampled <- subsampleCounts(tse,
                                  min_size = 60000,
                                  name = "subsampled",
                                  seed = 123)
tse.subsampled
dim(tse)
dim(tse.subsampled)
```

subsetSamples *Subset functions*

Description

To make a transition from phyloseq easier, the subsetSamples and subsetFeatures functions are implemented. To avoid name clashes they are named differently.

Usage

```
subsetSamples(x, ...)

subsetFeatures(x, ...)

subsetTaxa(x, ...)

## S4 method for signature 'SummarizedExperiment'
subsetSamples(x, ...)

## S4 method for signature 'SummarizedExperiment'
subsetFeatures(x, ...)

## S4 method for signature 'SummarizedExperiment'
subsetTaxa(x, ...)
```

Arguments

x	a SummarizedExperiment object
...	See subset. drop is not supported.

Details

However, the use of these functions is discouraged since subsetting using [works on both dimension at the same time, is more flexible and is used throughout R to subset data with two or more dimension. Therefore, these functions will be removed in Bioconductor release 3.15 (April, 2022).

Value

A subset of x

Examples

```
data(GlobalPatterns)
subsetSamples(GlobalPatterns, colData(GlobalPatterns)$SampleType == "Soil")
# Vector that is used to specify subset must not include NAs
subsetFeatures(GlobalPatterns, rowData(GlobalPatterns)$Phylum == "Actinobacteria" &
               !is.na(rowData(GlobalPatterns)$Phylum))
```

summaries *Summarizing microbiome data*

Description

To query a SummarizedExperiment for interesting features, several functions are available.

Usage

```
getTopTaxa(
  x,
  top = 5L,
  method = c("mean", "sum", "median"),
  assay.type = assay_name,
  assay_name = "counts",
  na.rm = TRUE,
  ...
)

## S4 method for signature 'SummarizedExperiment'
getTopTaxa(
  x,
  top = 5L,
  method = c("mean", "sum", "median", "prevalence"),
  assay.type = assay_name,
  assay_name = "counts",
  na.rm = TRUE,
  ...
)

getTopFeatures(x, ...)

## S4 method for signature 'SummarizedExperiment'
getTopFeatures(x, ...)

getUniqueTaxa(x, ...)

## S4 method for signature 'SummarizedExperiment'
getUniqueTaxa(x, rank = NULL, ...)

getUniqueFeatures(x, ...)

## S4 method for signature 'SummarizedExperiment'
getUniqueFeatures(x, ...)

countDominantTaxa(x, group = NULL, ...)
```

```
## S4 method for signature 'SummarizedExperiment'
countDominantTaxa(x, group = NULL, ...)

countDominantFeatures(x, ...)

## S4 method for signature 'SummarizedExperiment'
countDominantFeatures(x, ...)

## S4 method for signature 'SummarizedExperiment'
summary(object, assay.type = assay_name, assay_name = "counts")
```

Arguments

x	A SummarizedExperiment object.
top	Numeric value, how many top taxa to return. Default return top five taxa.
method	Specify the method to determine top taxa. Either sum, mean, median or prevalence. Default is 'mean'.
assay.type	a character value to select an assayNames By default it expects count data.
assay_name	a single character value for specifying which assay to use for calculation. (Please use assay.type instead. At some point assay_name will be disabled.)
na.rm	For getTopTaxa logical argument for calculation method specified to argument method. Default is TRUE.
...	Additional arguments passed on to agglomerateByRank() when rank is specified for countDominantTaxa.
rank	A single character defining a taxonomic rank. Must be a value of the output of taxonomyRanks().
group	With group, it is possible to group the observations in an overview. Must be one of the column names of colData.
object	A SummarizedExperiment object.

Details

The getTopTaxa extracts the most top abundant "FeatureID"s in a SummarizedExperiment object.

The getUniqueTaxa is a basic function to access different taxa at a particular taxonomic rank.

countDominantTaxa returns information about most dominant taxa in a tibble. Information includes their absolute and relative abundances in whole data set.

The summary will return a summary of counts for all samples and features in SummarizedExperiment object.

Value

The getTopTaxa returns a vector of the most top abundant "FeatureID"s

The getUniqueTaxa returns a vector of unique taxa present at a particular rank

The countDominantTaxa returns an overview in a tibble. It contains dominant taxa in a column named *name* and its abundance in the data set.

The summary returns a list with two tibbles

Author(s)

Leo Lahti, Tuomas Borman and Sudarshan A. Shetty

See Also

getPrevalentTaxa

perCellQCMetrics, perFeatureQCMetrics, addPerCellQC, addPerFeatureQC, quickPerCellQC

Examples

```
data(GlobalPatterns)
top_taxa <- getTopTaxa(GlobalPatterns,
                       method = "mean",
                       top = 5,
                       assay.type = "counts")
top_taxa

# Use 'detection' to select detection threshold when using prevalence method
top_taxa <- getTopTaxa(GlobalPatterns,
                       method = "prevalence",
                       top = 5,
                       assay_name = "counts",
                       detection = 100)
top_taxa

# Top taxa os specific rank
getTopTaxa(agglomerateByRank(GlobalPatterns,
                             rank = "Genus",
                             na.rm = TRUE))

# Gets the overview of dominant taxa
dominant_taxa <- countDominantTaxa(GlobalPatterns,
                                   rank = "Genus")
dominant_taxa

# With group, it is possible to group observations based on specified groups
# Gets the overview of dominant taxa
dominant_taxa <- countDominantTaxa(GlobalPatterns,
                                   rank = "Genus",
                                   group = "SampleType",
                                   na.rm = TRUE)

dominant_taxa

# Get an overview of sample and taxa counts
summary(GlobalPatterns)

# Get unique taxa at a particular taxonomic rank
# sort = TRUE means that output is sorted in alphabetical order
# With na.rm = TRUE, it is possible to remove NAs
# sort and na.rm can also be used in function getTopTaxa
getUniqueTaxa(GlobalPatterns, "Phylum", sort = TRUE)
```

taxonomy-methods *Functions for accessing taxonomic data stored in* rowData.

Description

These function work on data present in rowData and define a way to represent taxonomic data alongside the features of a SummarizedExperiment.

Usage

```
TAXONOMY_RANKS

taxonomyRanks(x)

## S4 method for signature 'SummarizedExperiment'
taxonomyRanks(x)

taxonomyRankEmpty(
  x,
  rank = taxonomyRanks(x)[1L],
  empty.fields = c(NA, "", " ", "\t", "-", "_")
)

## S4 method for signature 'SummarizedExperiment'
taxonomyRankEmpty(
  x,
  rank = taxonomyRanks(x)[1],
  empty.fields = c(NA, "", " ", "\t", "-", "_")
)

checkTaxonomy(x, ...)

## S4 method for signature 'SummarizedExperiment'
checkTaxonomy(x)

getTaxonomyLabels(x, ...)

## S4 method for signature 'SummarizedExperiment'
getTaxonomyLabels(
  x,
  empty.fields = c(NA, "", " ", "\t", "-", "_"),
  with_rank = FALSE,
  make_unique = TRUE,
  resolve_loops = FALSE,
  ...
```

```
)

taxonomyTree(x, ...)

## S4 method for signature 'SummarizedExperiment'
taxonomyTree(x)

addTaxonomyTree(x, ...)

## S4 method for signature 'SummarizedExperiment'
addTaxonomyTree(x)

mapTaxonomy(x, ...)

## S4 method for signature 'SummarizedExperiment'
mapTaxonomy(x, taxa = NULL, from = NULL, to = NULL, use_grepl = FALSE)

IdTaxaToDataFrame(from)
```

Arguments

x	a SummarizedExperiment object
rank	a single character defining a taxonomic rank. Must be a value of taxonomyRanks() function.
empty.fields	a character value defining, which values should be regarded as empty. (Default: c(NA, "", " ", "\t")). They will be removed if na.rm = TRUE before agglomeration.
...	optional arguments not used currently.
with_rank	TRUE or FALSE: Should the level be add as a suffix? For example: "Phylum:Crenarchaeota" (default: with_rank = FALSE)
make_unique	TRUE or FALSE: Should the labels be made unique, if there are any duplicates? (default: make_unique = TRUE)
resolve_loops	TRUE or FALSE: Should resolveLoops be applied to the taxonomic data? Please note that has only an effect, if the data is unique. (default: resolve_loops = TRUE)
taxa	a character vector, which is used for subsetting the taxonomic information. If no information is found,NULL is returned for the individual element. (default: NULL)
from	• For mapTaxonomy: a scalar character value, which must be a valid taxonomic rank. (default: NULL) • otherwise a Taxa object as returned by IdTaxa
to	a scalar character value, which must be a valid taxonomic rank. (default: NULL)
use_grepl	TRUE or FALSE: should pattern matching via grepl be used? Otherwise literal matching is used. (default: FALSE)

Format

a character vector of length 8 containing the taxonomy ranks recognized. In functions this is used as case insensitive.

Details

taxonomyRanks returns, which columns of rowData(x) are regarded as columns containing taxonomic information.

taxonomyRankEmpty checks, if a selected rank is empty of information.

checkTaxonomy checks, if taxonomy information is valid and whether it contains any problems. This is a soft test, which reports some diagnostic and might mature into a data validator used upon object creation.

getTaxonomyLabels generates a character vector per row consisting of the lowest taxonomic information possible. If data from different levels, is to be mixed, the taxonomic level is prepended by default.

taxonomyTree generates a phylo tree object from the available taxonomic information. Internally it uses toTree and resolveLoop to sanitize data if needed.

IdTaxaToDataFrame extracts taxonomic results from results of IdTaxa.

mapTaxonomy maps the given features (taxonomic groups; taxa) to the specified taxonomic level (to argument) in rowData of the SummarizedExperiment data object (i.e. rowData(x)[,taxonomyRanks(x)]). If the argument to is not provided, then all matching taxonomy rows in rowData will be returned. This function allows handy conversions between different

Taxonomic information from the IdTaxa function of DECIPHER package are returned as a special class. With as(taxa,"DataFrame") the information can be easily converted to a DataFrame compatible with storing the taxonomic information a rowData. Please note that the assigned confidence information are returned as metatdata and can be accessed using metadata(df)$confidence.

Value

- taxonomyRanks: a character vector with all the taxonomic ranks found in colnames(rowData(x))
- taxonomyRankEmpty: a logical value
- mapTaxonomy: a list per element of taxa. Each element is either a DataFrame, a character or NULL. If all character results have the length of one, a single character vector is returned.

See Also

agglomerateByRank, toTree, resolveLoop

Examples

```
data(GlobalPatterns)
GlobalPatterns
taxonomyRanks(GlobalPatterns)

checkTaxonomy(GlobalPatterns)
```

```
table(taxonomyRankEmpty(GlobalPatterns,"Kingdom"))
table(taxonomyRankEmpty(GlobalPatterns,"Species"))

getTaxonomyLabels(GlobalPatterns[1:20,])

# mapTaxonomy
## returns the unique taxonomic information
mapTaxonomy(GlobalPatterns)
# returns specific unique taxonomic information
mapTaxonomy(GlobalPatterns, taxa = "Escherichia")
# returns information on a single output
mapTaxonomy(GlobalPatterns, taxa = "Escherichia",to="Family")

# adding a rowTree() based on the available taxonomic information. Please
# note that any tree already stored in rowTree() will be overwritten.
x <- GlobalPatterns
x <- addTaxonomyTree(x)
x
```

`transformCounts` *Transform Counts*

Description

Variety of transformations for abundance data, stored in `assay`. See details for options.

Usage

```
transformSamples(
  x,
  assay.type = "counts",
  assay_name = NULL,
 method = c("alr", "chi.square", "clr", "frequency", "hellinger", "log", "log10",
   "log2", "normalize", "pa", "rank", "rclr", "relabundance", "rrank", "total"),
  name = method,
  ...
)

## S4 method for signature 'SummarizedExperiment'
transformSamples(
  x,
  assay.type = "counts",
  assay_name = NULL,
 method = c("alr", "chi.square", "clr", "frequency", "hellinger", "log", "log10",
   "log2", "normalize", "pa", "rank", "rclr", "relabundance", "rrank", "total"),
  name = method,
  ...
)
```

```
transformCounts(
  x,
  assay.type = "counts",
  assay_name = NULL,
 method = c("alr", "chi.square", "clr", "frequency", "hellinger", "log", "log10",
   "log2", "max", "normalize", "pa", "range", "rank", "rclr", "relabundance", "rrank",
    "standardize", "total", "z"),
  MARGIN = "samples",
  name = method,
  ...
)

## S4 method for signature 'SummarizedExperiment'
transformCounts(
  x,
  assay.type = "counts",
  assay_name = NULL,
 method = c("alr", "chi.square", "clr", "frequency", "hellinger", "log", "log10",
   "log2", "max", "normalize", "pa", "range", "rank", "rclr", "relabundance", "rrank",
    "standardize", "total", "z"),
  MARGIN = "samples",
  name = method,
  ...
)

transformFeatures(
  x,
  assay.type = "counts",
  assay_name = NULL,
 method = c("frequency", "log", "log10", "log2", "max", "pa", "range", "standardize",
    "z"),
  name = method,
  ...
)

## S4 method for signature 'SummarizedExperiment'
transformFeatures(
  x,
  assay.type = "counts",
  assay_name = NULL,
 method = c("frequency", "log", "log10", "log2", "max", "pa", "range", "standardize",
    "z"),
  name = method,
  ...
)

ZTransform(x, MARGIN = "features", ...)
```

```
## S4 method for signature 'SummarizedExperiment'
ZTransform(x, MARGIN = "features", ...)

relAbundanceCounts(x, ...)

## S4 method for signature 'SummarizedExperiment'
relAbundanceCounts(x, ...)
```

Arguments

x	A SummarizedExperiment object.
assay.type	A single character value for selecting the assay to be transformed.
assay_name	a single character value for specifying which assay to use for calculation. (Please use assay.type instead. At some point assay_name will be disabled.)
method	A single character value for selecting the transformation method.
name	A single character value specifying the name of transformed abundance table.
...	additional arguments passed on to vegan:decostand: • ref_vals: A single value which will be used to fill reference sample's column in returned assay when calculating alr. (default: ref_vals = NA) • pseudocount: NULL or numeric value deciding whether pseudocount is added. The numeric value specifies the value of pseudocount. Recommended default choices for counts and relative abundance assay pseudocount = 1 and pseudocount = min(x[x>0]), respectively.
MARGIN	A single character value for specifying whether the transformation is applied sample (column) or feature (row) wise. (By default: MARGIN = "samples")

Details

These transformCount function provides a variety of options for transforming abundance data. The transformed data is calculated and stored in a new assay. The previously available wrappers transformSamples, transformFeatures ZTransform, and relAbundanceCounts have been deprecated.

The transformCounts provides sample-wise (column-wise) or feature-wise (row-wise) transformation to the abundance table (assay) based on specified MARGIN.

The available transformation methods include:

- 'alr' Additive log ratio (alr) transformation, please refer to decostand for details.
- 'chi.square' Chi square transformation, please refer to decostand for details.
- 'clr' Centered log ratio (clr) transformation, please refer to decostand for details.
- 'frequency' Frequency transformation, please refer to decostand for details.
- 'hellinger' Hellinger transformation, please refer to decostand for details.
- 'log' Logarithmic transformation, please refer to decostand for details.
- 'log10' log10 transformation can be used for reducing the skewness of the data.

$$log10 = \log_{10} x$$

where x is a single value of data.

- 'log2' log2 transformation can be used for reducing the skewness of the data.

$$log2 = \log_2 x$$

  where x is a single value of data.
- 'normalize' Make margin sum of squares equal to one. Please refer to decostand for details.
- 'pa' Transforms table to presence/absence table. Please refer to decostand for details.
- 'rank' Rank transformation, please refer to decostand for details.
- 'rclr' Robust clr transformation, please refer to decostand for details.
- 'relabundance' Relative transformation (alias for 'total'), please refer to decostand for details.
- 'rrank' Relative rank transformation, please refer to decostand for details.
- 'standardize' Scale 'x' to zero mean and unit variance (alias for 'z'), please refer to decostand for details.
- 'total' Divide by margin total (alias for 'relabundance'), please refer to decostand for details.
- 'z' Z transformation (alias for 'standardize'), please refer to decostand for details.

Value

transformCounts returns the input object x, with a new transformed abundance table named name added in the assay.

Author(s)

Leo Lahti and Tuomas Borman. Contact: microbiome.github.io

Examples

```
data(esophagus, package="mia")
x <- esophagus

# By specifying 'method', it is possible to apply different transformations,
# e.g. compositional transformation.
x <- transformCounts(x, method = "relabundance")

# The target of transformation can be specified with "assay.type"
# Pseudocount can be added by specifying 'pseudocount'.

# Get pseudocount; here smallest positive value
mat <- assay(x, "relabundance")
pseudonumber <- min(mat[mat>0])
# Perform CLR
x <- transformCounts(x, assay.type = "relabundance", method = "clr",
                     pseudocount = pseudonumber
                     )

head(assay(x, "clr"))

# With MARGIN, you can specify the if transformation is done for samples or
# for features. Here Z-transformation is done feature-wise.
```

```
x <- transformCounts(x, method = "z", MARGIN = "features")
head(assay(x, "z"))

# Name of the stored table can be specified.
x <- transformCounts(x, method="hellinger", name="test")
head(assay(x, "test"))

# pa returns presence absence table.
x <- transformCounts(x, method = "pa")
head(assay(x, "pa"))

# rank returns ranks of taxa.
x <- transformCounts(x, method = "rank")
head(assay(x, "rank"))

# In order to use other ranking variants, modify the chosen assay directly:
assay(x, "rank_average", withDimnames = FALSE) <- colRanks(assay(x, "counts"),
                                                           ties.method="average",
                                                           preserveShape = TRUE)
```

Index