Package 'mia'

October 14, 2021

Type Package

Version 1.0.8

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 (>= 4.1), SummarizedExperiment, SingleCellExperiment, TreeSummarizedExperiment (>= 1.99.3)

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 7.1.1

VignetteBuilder knitr

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

git_branch RELEASE_3_13

git_last_commit 95eeac0

git_last_commit_date 2021-07-30

Date/Publication 2021-10-14

Author Felix G.M. Ernst [aut, cre] (<https://orcid.org/0000-0001-5064-0928>),
Sudarshan A. Shetty [aut] (<https://orcid.org/0000-0001-7280-9915>),
Tuomas Borman [aut] (<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 Felix G.M. Ernst <felix.gm.ernst@outlook.com>

R topics documented:

mia-package . . . 3
agglomerate-methods . . . 3
calculateDistance . . . 5
calculateDMN . . . 6
calculateJSD . . . 9
calculateUniFrac . . . 10
estimateDivergence . . . 12
estimateDiversity . . . 14
estimateDominance . . . 19
estimateEvenness . . . 23
estimateRichness . . . 25
getAbundance . . . 28
getPrevalence . . . 30
isContaminant . . . 34
loadFromMothur . . . 36
loadFromQIIME2 . . . 37
makePhyloseqFromTreeSummarizedExperiment . . . 39
makeSummarizedExperimentFromBiom . . . 40
makeTreeSummarizedExperimentFromDADA2 . . . 41
makeTreeSummarizedExperimentFromphyloseq . . . 42
meltAssay . . . 43
merge-methods . . . 45
mia-datasets . . . 46
perSampleDominantTaxa . . . 47
relabundance . . . 48
runCCA . . . 49
runDPCoA . . . 51
runNMDS . . . 52
splitByRanks . . . 55
subsetSamples . . . 57
summaries . . . 58

taxonomy-methods 60
transformCounts 63

Index **69**

`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 `taxonomicRanks` 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 taxonomicRanks() 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.
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.

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

# removing empty labels by setting na.rm = TRUE
sum(is.na(rowData(GlobalPatterns)$Family))

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

`calculateDistance` *Calculate sample distances with* `vegan`

Description

`calculateDistance` calculates a distance matrix between samples. The type of distance calculated can be modified by setting `FUN`, which expects a function with a matrix input as its first argument.

Usage

```
calculateDistance(x, FUN = stats::dist, ...)

## S4 method for signature 'ANY'
calculateDistance(x, FUN = stats::dist, ...)

## S4 method for signature 'SummarizedExperiment'
calculateDistance(
  x,
  FUN = stats::dist,
  exprs_values = "counts",
  transposed = FALSE,
  ...
)
```

Arguments

`x`	a `SummarizedExperiment` object containing a tree.
`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.

...	other arguments passed onto FUN
exprs_values	a single character value for specifying which assay to use for calculation.
transposed	Logical scalar, is x transposed with cells in rows?

Value

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

Examples

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

---

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, 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,
  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,
  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
exprs_values	a single character value for specifying which assay to use for calculation.
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)

se <- SummarizedExperiment(assays = list(counts = counts),
                           colData = colData)


#
dmn <- calculateDMN(se)
```

```
dmn[[1L]]

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

## End(Not run)
data(dmn_se)
names(metadata(dmn_se))

# return a list of DMN objects
getDMN(dmn_se)
# return, which objects fits best
bestDMNFit(dmn_se, type = "laplace")
# return the model, which fits best
getBestDMNFit(dmn_se, 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, exprs_values = "counts", transposed = FALSE, ...)

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

Arguments

`x`	a numeric matrix or a `SummarizedExperiment`.
`...`	optional arguments not used.
`exprs_values`	a single `character` value for specifying which assay to use for calculation.
`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")
```

---

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

```
## S4 method for signature 'ANY,phylo'
calculateUniFrac(
  x,
  tree,
```

```
  weighted = FALSE,
  normalized = TRUE,
  BPPARAM = SerialParam()
)

## S4 method for signature 'TreeSummarizedExperiment,missing'
calculateUniFrac(x, exprs_values = "counts", transposed = FALSE, ...)

runUniFrac(
  x,
  tree,
  weighted = FALSE,
  normalized = TRUE,
  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).
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.
exprs_values	a single character value for specifying which assay to use for calculation.
transposed	Logical scalar, is x transposed with cells in rows?
...	optional arguments not used.

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

This function estimates a divergence within samples.

Usage

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

## S4 method for signature 'SummarizedExperiment'
estimateDivergence(
  x,
  abund_values = "counts",
  name = "divergence",
  reference = "median",
  FUN = vegan::vegdist,
  method = "bray",
  ...
)
```

Arguments

x	a SummarizedExperiment object
abund_values	the name of the assay used for calculation of the sample-wise estimates
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. For more information, please check calculateDistance. 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
- calculateDistance

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 diversity measures*

Description

Several functions for calculating diversity indices are available via wrapper functions. Some of them are implemented via the vegan package.

Usage

```
estimateDiversity(
  x,
  abund_values = "counts",
  index = c("coverage", "fisher", "gini_simpson", "inverse_simpson",
    "log_modulo_skewness", "shannon"),
  name = index,
  ...
)

## S4 method for signature 'SummarizedExperiment'
estimateDiversity(
  x,
  abund_values = "counts",
  index = c("coverage", "fisher", "gini_simpson", "inverse_simpson",
    "log_modulo_skewness", "shannon"),
  name = index,
  ...,
  BPPARAM = SerialParam()
)

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

estimateFaith(
  x,
  tree = "missing",
  abund_values = "counts",
  name = "faith",
  ...
)

## S4 method for signature 'SummarizedExperiment,phylo'
estimateFaith(
  x,
  tree = "missing",
  abund_values = "counts",
  name = "faith",
  ...
)
```

```
## S4 method for signature 'TreeSummarizedExperiment,missing'
estimateFaith(
  x,
  tree = "missing",
  abund_values = "counts",
  name = "faith",
  ...
)
```

Arguments

x	a SummarizedExperiment object
abund_values	the name of the assay used for calculation of the sample-wise estimates.
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.
...	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	A phylogenetic tree that is used to calculate 'faith' index. If x is a TreeSummarizedExperiment, rowTree(x) is used by default.

Details

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

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\ 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 value represent higher diversity. (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,
  abund_values = "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,
  abund_values = "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
abund_values	A single character value for selecting the assay used for calculation of the sample-wise estimates.
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,
  abund_values = "counts",
  index = c("pielou", "camargo", "simpson_evenness", "evar", "bulla"),
  name = index,
  ...
)

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

Arguments

`x`	a `SummarizedExperiment` object
`abund_values`	A single character value for selecting the `assay` used for calculation of the sample-wise estimates.
`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)
se <- 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
se <- estimateEvenness(se, index = index, name = name)

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

---

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,
  abund_values = "counts",
  index = c("ace", "chao1", "hill", "observed"),
  name = index,
  detection = 0,
  ...,
  BPPARAM = SerialParam()
)

## S4 method for signature 'SummarizedExperiment'
estimateRichness(
  x,
  abund_values = "counts",
  index = c("ace", "chao1", "hill", "observed"),
```

```
  name = index,
  detection = 0,
  ...,
  BPPARAM = SerialParam()
)
```

Arguments

x	a SummarizedExperiment object
abund_values	the name of the assay used for calculation of the sample-wise estimates
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

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, abund_values = "counts")

## S4 method for signature 'SummarizedExperiment'
getAbundanceSample(x, sample_id = NULL, abund_values = "counts")

getAbundanceFeature(x, feature_id, abund_values)

## S4 method for signature 'SummarizedExperiment'
getAbundanceFeature(x, feature_id = NULL, abund_values = "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)`.
`abund_values`	a `character` value to select an `assayNames`
`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',
                   abund_values = 'counts')
# getAbundanceFeature
getAbundanceFeature(GlobalPatterns,
```

```
                    feature_id = '522457',
                    abund_values = 'counts')
```

---

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, abund_values = "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,
  ...
)

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, ...)

subsetByPrevalentTaxa(x, ...)

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

```
subsetByRareTaxa(x, ...)

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

getPrevalentAbundance(x, abund_values = "relabundance", ...)

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

## S4 method for signature 'SummarizedExperiment'
getPrevalentAbundance(x, abund_values = "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)
abund_values	A single character value for selecting the assay to use for prevalence calculation.
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, abund_values = "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,
  abund_values = "counts",
  name = "isContaminant",
  concentration = NULL,
  control = NULL,
  batch = NULL,
  threshold = 0.1,
  normalize = TRUE,
  detailed = TRUE,
  ...
)

## S4 method for signature 'SummarizedExperiment'
isNotContaminant(
  seqtab,
  abund_values = "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
abund_values	A single character value for selecting the assay to use.
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)
```

---

loadFromMothur *Import Mothur results as a* SummarizedExperiment

---

Description

This method creates a SummarizedExperiment 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 SummarizedExperiment 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

makeTreeSummarizedExperimentFromphyloseq makeSummarizedExperimentFromBiom makeTreeSummarizedExperimer 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)
se
```

---

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.

Usage

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

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)

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

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

makeTreeSummarizedExperimentFromphyloseq makeSummarizedExperimentFromBiom makeTreeSummarizedExperimer 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
```

---

makePhyloseqFromTreeSummarizedExperiment

Create a phyloseq object from a TreeSummarizedExperiment object

---

Description

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

Usage

```
makePhyloseqFromTreeSummarizedExperiment(x, ...)

## S4 method for signature 'SummarizedExperiment'
makePhyloseqFromTreeSummarizedExperiment(x, abund_values = "counts")

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

Arguments

x	a TreeSummarizedExperiment object
...	additional arguments
abund_values	A single character value for selecting the assay to be included in the phyloseq object that is created. By default, it is counts table.

Details

makePhyloseqFromTreeSummarizedExperiment 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 <- makePhyloseqFromTreeSummarizedExperiment(tse)
phy

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

# Counts relative abundances table
tse <- transformCounts(tse, method = "relabundance")
phy2 <- makePhyloseqFromTreeSummarizedExperiment(tse, abund_values = "relabundance")
phy2
```

---

makeSummarizedExperimentFromBiom

Loading a biom file

Description

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

Usage

```
loadFromBiom(file)

makeSummarizedExperimentFromBiom(obj)
```

Arguments

file	biom file location
obj	object of type biom

Value

An object of class SummarizedExperiment

See Also

makeTreeSummarizedExperimentFromphyloseq makeTreeSummarizedExperimentFromDADA2 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)

  # load from object
  x1 <- biomformat::read_biom(rich_dense_file)
  se <- makeSummarizedExperimentFromBiom(x1)
  se
}
```

---

makeTreeSummarizedExperimentFromDADA2

Coerce 'DADA2' results to TreeSummarizedExperiment

Description

makeTreeSummarizedExperimentFromDADA2 is a wrapper for the mergePairs function from the dada2 package.

Usage

```
makeTreeSummarizedExperimentFromDADA2(...)
```

Arguments

...	See mergePairs function for more details.

Details

A count matrix is contructed 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

makeTreeSummarizedExperimentFromphyloseq makeSummarizedExperimentFromBiom 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 <- makeTreeSummarizedExperimentFromDADA2(dadaF, fnF, dadaR, fnR)
  tse
}
```

---

makeTreeSummarizedExperimentFromphyloseq

Coerce a phyloseq *object to a* TreeSummarizedExperiment

Description

makeTreeSummarizedExperimentFromphyloseq converts phyloseq objects into TreeSummarizedExperiment objects.

Usage

```
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

makeSummarizedExperimentFromBiom makeTreeSummarizedExperimentFromDADA2 loadFromQIIME2 loadFromMothur

Examples

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

---

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

Description

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

Usage

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

## S4 method for signature 'SummarizedExperiment'
meltAssay(
  x,
  add_row_data = NULL,
  add_col_data = NULL,
```

```
  assay_name = "counts",
  feature_name = "FeatureID",
  sample_name = "SampleID",
  ...
)
```

Arguments

x	A numeric matrix or a SummarizedExperiment
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)
assay_name	a character value to select an assayNames
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 currently not used.

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_name. 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_se <- meltAssay(GlobalPatterns,
                       add_row_data = TRUE,
                       add_col_data = TRUE,
                       assay_name = "counts")
molten_se
```

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 interger 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

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)
plot(rowTree(merged))
#
data(GlobalPatterns)
GlobalPatterns
merged <- mergeCols(GlobalPatterns,colData(GlobalPatterns)$SampleType)
merged
```

---

mia-datasets *mia datasets*

Description

These datasets are conversions of the phyloseq datasets GlobalPatterns enterotype, esophagus and soilrep.

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

Usage

```
data(GlobalPatterns)

data(enterotype)

data(esophagus)
```

```
data(soilrep)

data(dmn_se)
```

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 `TreeSummarizedExperiment` with 16825 rows and 56 columns.

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

---

perSampleDominantTaxa *Get dominant taxa*

Description

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

Usage

```
perSampleDominantTaxa(x, abund_values = "counts", rank = NULL, ...)

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

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

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

Arguments

`x`	A `SummarizedExperiment` object.
`abund_values`	A single character value for selecting the `assay` to use for identifying dominant taxa.
`rank`	A single character defining a taxonomic rank. Must be a value of the output of `taxonomicRanks()`.
`...`	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

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 <- relAbundanceCounts(GlobalPatterns)
# Fetches calculated relative abundances
head(relabundance(GlobalPatterns))
```

---

runCCA *Canonical Correspondance Analysis*

Description

These functions perform Canonical Correspondance 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, ..., exprs_values = "counts")

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

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

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

```
calculateRDA(x, formula, ..., exprs_values = "counts")

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

Arguments

x	For calculateCCA a numeric matrix with columns as samples or a SummarizedExperiment. For runCCA a SingleCellExperiment or a derived object.
...	additional arguments not used.
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	a data.frame or an object coercible to one containing the variables to use. Can be missing, which turns the CCA analysis into a CA analysis. All variables are used. Please subset, if you want to consider only some of them.
scale	Logical scalar, should the expression values be standardized?
exprs_values	a single character value for specifying which assay to use for calculation.
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.

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 rda

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")
```

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, ..., exprs_values = "counts", dimred = NULL, n_dimred = NULL)

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?
exprs_values	a single character value for specifying which assay to use for calculation.
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

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, ..., exprs_values = "counts", FUN = vegdist)

## S4 method for signature 'SingleCellExperiment'
calculateNMDS(
  x,
  ...,
  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.
exprs_values	a single character value for specifying which assay to use for calculation.
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))
se <- SummarizedExperiment(assays = list(counts = mat),
                           rowData = df)
#
calculateNMDS(se)

#
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 taxonomicRanks() 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: x, with objects of x agglomerated for selected ranks as altExps.

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

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
```

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)$SampleType == "Soil")
subsetFeatures(GlobalPatterns, rowData(GlobalPatterns)$Kingdom == "Bacteria")
```

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"),
  abund_values = "counts"
)

## S4 method for signature 'SummarizedExperiment'
getTopTaxa(
  x,
  top = 5L,
  method = c("mean", "sum", "median", "prevalence"),
  abund_values = "counts"
)

getUniqueTaxa(x, ...)

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

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

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

## S4 method for signature 'SummarizedExperiment'
summary(object, abund_values = "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'.
abund_values	a character value to select an assayNames By default it expects count data.
...	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 taxonomicRanks().
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,
                       abund_values = "counts")
top_taxa

# 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
getUniqueTaxa(GlobalPatterns, "Phylum")
```

---

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 taxonomicRanks() 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.

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

These functions provide a variety of options for transforming abundance data. By using these functions, transformed table is calculated and stored in `assay`. `transformSamples` does the transformation sample-wise, i.e., column-wise. It is alias for `transformCounts`. `transformFeatures` does the transformation feature-wise, i.e., row-wise. `ZTransform` is a shortcut for Z-transformation. `relAbundanceCounts` is a shortcut for fetching relative abundance table.

Usage

```
transformSamples(
  x,
  abund_values = "counts",
  method = c("clr", "hellinger", "log10", "pa", "rank", "relabundance"),
  name = method,
  pseudocount = FALSE,
  threshold = 0
)
```

```
## S4 method for signature 'SummarizedExperiment'
transformSamples(
  x,
  abund_values = "counts",
  method = c("clr", "hellinger", "log10", "pa", "rank", "relabundance"),
  name = method,
  pseudocount = FALSE,
  threshold = 0
)

transformCounts(
  x,
  abund_values = "counts",
  method = c("clr", "hellinger", "log10", "pa", "rank", "relabundance"),
  name = method,
  pseudocount = FALSE,
  threshold = 0
)

## S4 method for signature 'SummarizedExperiment'
transformCounts(
  x,
  abund_values = "counts",
  method = c("clr", "hellinger", "log10", "pa", "rank", "relabundance"),
  name = method,
  pseudocount = FALSE,
  threshold = 0
)

transformFeatures(
  x,
  abund_values = "counts",
  method = c("log10", "pa", "z"),
  name = method,
  pseudocount = FALSE,
  threshold = 0
)

## S4 method for signature 'SummarizedExperiment'
transformFeatures(
  x,
  abund_values = "counts",
  method = c("log10", "pa", "z"),
  name = method,
  pseudocount = FALSE,
  threshold = 0
)
```

```
ZTransform(x, ...)

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

relAbundanceCounts(x, ...)

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

Arguments

x	A SummarizedExperiment object.
abund_values	A single character value for selecting the assay to be transformed.
method	A single character value for selecting the transformation method.
name	A single character value specifying the name of transformed abundance table.
pseudocount	FALSE or numeric value deciding whether pseudocount is added. Numerical value specifies the value of pseudocount. (Only used for methods method = "clr", method = "hellinger", or method = "log10")
threshold	A numeric value for setting threshold for pa transformation. By default it is 0. (Only used for method = "pa")
...	additional arguments

Details

transformCounts or transformSamples and transformFeatures applies transformation to abundance table. Provided transformation methods include:

- 'clr' Centered log ratio (clr) transformation can be used for reducing the skewness of data and for centering it. (See e.g. Gloor et al. 2017.)

$$clr = log_{10}x_r - log_{10}r$$

  where x_r is a single relative value, μ_r is mean relative value".

- 'hellinger' Hellinger transformation can be used to reduce the impact of extreme data points. It can be utilize for clustering or ordination analysis. (See e.g. Legendre & Gallagher 2001.)

$$hellinger = \sqrt{\frac{x}{x_{tot}}}$$

  where x is a single value and x_{tot} is the sum of all values

- 'log10' log10 transformation can be used for reducing the skewness of the data.

$$log10 = \log_1 0x$$

  where x is a single value of data.

- 'pa' Transforms table to presence/absence table. All abundances higher than ϵ are transformed to 1 (present), otherwise 0 (absent). By default, threshold is 0.
- 'rank' Rank returns ranks of taxa. For each sample, the least abundant taxa get lower value and more abundant taxa bigger value. The implementation is based on the colRanks function with ties.method="first".
- 'relabundance' Transforms abundances to relative. Generally, all microbiome data are compositional. That is, e.g., because all measuring instruments have their capacity limits. To make results comparable with other results, values must be relative. (See e.g. Gloor et al. 2017.)

$$relabundance = \frac{x}{x_{tot}}$$

where x is a single value and x_{tot} is the sum of all values.

- 'z' Z-transformation, Z score transformation, or Z-standardization normalizes the data by shifting (to mean μ) and scaling (to standard deviation σ). Z-transformation can be done with function `ZTransform`. It is done per rows (features / taxa), unlike most other transformations. This is often preceded by log10p or clr transformation. In other words, single value is standardized with respect of feature's values.

$$z = \frac{x + \mu}{\sigma}$$

where x is a single value, μ is the mean of the feature, and σ is the standard deviation of the feature.

Value

`transformCounts`, `transformSamples`, `transformFeatures`, `relAbundanceCounts`, and `ZTransform` return `x` with additional, transformed abundance table named `name` in the `assay`.

Author(s)

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

References

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

Legendre P & Gallagher ED (2001) Ecologically meaningful transformations for ordination of species data. Oecologia 129: 271-280.

Examples

```
data(esophagus)
x <- esophagus

# By specifying, it is possible to apply different transformations, e.g. clr transformation.
# Pseudocount can be added by specifying 'pseudocount'.
x <- transformSamples(x, method="clr", pseudocount=1)
head(assay(x, "clr"))
```

```
# Also, the target of transformation
# can be specified with "abund_values".
x <- transformSamples(x, method="relabundance")
x <- transformSamples(x, method="clr", abund_values="relabundance",
                pseudocount = min(assay(x, "relabundance")[assay(x, "relabundance")>0]))
x2 <- transformSamples(x, method="clr", abund_values="counts", pseudocount = 1)
head(assay(x, "clr"))

# Different pseudocounts used by default for counts and relative abundances
x <- transformSamples(x, method="relabundance")
mat <- assay(x, "relabundance");
pseudonumber <- min(mat[mat>0])
x <- transformSamples(x, method="clr", abund_values = "relabundance", pseudocount=pseudonumber)
x <- transformSamples(x, method="clr", abund_values = "counts", pseudocount=1)

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

# pa returns presence absence table. With 'threshold', it is possible to set the
# threshold to a desired level. By default, it is 0.
x <- transformSamples(x, method="pa", threshold=35)
head(assay(x, "pa"))

# rank returns ranks of taxa. It is calculated column-wise, i.e., per sample
# and using the ties.method="first" from the colRanks function
x <- transformSamples(x, method="rank")
head(assay(x, "rank"))

# transformCounts is an alias for transformSamples
x <- transformCounts(x, method="relabundance", name="test2")
head(assay(x, "test2"))

# 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)

# If you want to do the transformation for features, you can do that by using
x <- transformFeatures(x, method="log10", name="log10_features", pseudocount = 1)
head(assay(x, "log10_features"))

# Z-transform can be done for features by using shortcut function
x <- ZTransform(x)
head(assay(x, "z"))

# For visualization purposes it is sometimes done by applying CLR for samples,
# followed by Z transform for taxa
x <- ZTransform(transformCounts(x, method="clr", abund_values = "counts", pseudocount = 1))

# Relative abundances can be also calculated with the dedicated
# relAbundanceCounts function.
```

```
x <- relAbundanceCounts(x)
head(assay(x, "relabundance"))
```

Index

∗ **datasets**
    mia-datasets, 46
    taxonomy-methods, 60
?agglomerateByRank, *31*
[, *57*

accessors for DMN objects, *8*
addContaminantQC *(*isContaminant*)*, 34
addContaminantQC,SummarizedExperiment-method *(*isContaminant*)*, 34
addNotContaminantQC *(*isContaminant*)*, 34
addNotContaminantQC,SummarizedExperiment-method *(*isContaminant*)*, 34
addPerCellQC, *59*
addPerFeatureQC, *59*
addPerSampleDominantTaxa *(*perSampleDominantTaxa*)*, 47
addPerSampleDominantTaxa,SummarizedExperiment-method *(*perSampleDominantTaxa*)*, 47
addTaxonomyTree *(*taxonomy-methods*)*, 60
addTaxonomyTree,SummarizedExperiment-method *(*taxonomy-methods*)*, 60
agglomerate-methods, 3
agglomerateByPrevalence *(*getPrevalence*)*, 30
agglomerateByPrevalence,SummarizedExperiment-method *(*getPrevalence*)*, 30
agglomerateByRank, *31, 32*, *55, 56*, *62*
agglomerateByRank *(*agglomerate-methods*)*, 3
agglomerateByRank,SingleCellExperiment-method *(*agglomerate-methods*)*, 3
agglomerateByRank,SummarizedExperiment-method *(*agglomerate-methods*)*, 3
agglomerateByRank,TreeSummarizedExperiment-method *(*agglomerate-methods*)*, 3
altExps, *56*
assay, *20*, *23*, *31*, *35*, *40*, *47*, *65, 66*
assayNames, *29*, *44*, *58*

bestDMNFit *(*calculateDMN*)*, 6
bestDMNFit,SummarizedExperiment-method *(*calculateDMN*)*, 6
BiocParallelParam, *8, 9*, *11*, *16*, *20*, *24*, *26*
biom, *41*

calculateCCA *(*runCCA*)*, 49
calculateCCA,ANY-method *(*runCCA*)*, 49
calculateCCA,SummarizedExperiment-method *(*runCCA*)*, 49
calculateDistance, 5, *14*
calculateDistance,ANY-method *(*calculateDistance*)*, 5
calculateDistance,SummarizedExperiment-method *(*calculateDistance*)*, 5
calculateDMN, 6
calculateDMN,ANY-method *(*calculateDMN*)*, 6
calculateDMN,SummarizedExperiment-method *(*calculateDMN*)*, 6
calculateDMNgroup *(*calculateDMN*)*, 6
calculateDMNgroup,ANY-method *(*calculateDMN*)*, 6
calculateDMNgroup,SummarizedExperiment-method *(*calculateDMN*)*, 6
calculateDPCoA *(*runDPCoA*)*, 51
calculateDPCoA,ANY,ANY-method *(*runDPCoA*)*, 51
calculateDPCoA,TreeSummarizedExperiment,missing-method *(*runDPCoA*)*, 51
calculateJSD, 9
calculateJSD,ANY-method *(*calculateJSD*)*, 9
calculateJSD,SummarizedExperiment-method *(*calculateJSD*)*, 9
calculateNMDS *(*runNMDS*)*, 52
calculateNMDS,ANY-method *(*runNMDS*)*, 52
calculateNMDS,SingleCellExperiment-method *(*runNMDS*)*, 52

calculateNMDS,SummarizedExperiment-method (runNMDS), 52
calculateRDA (runCCA), 49
calculateRDA,ANY-method (runCCA), 49
calculateRDA,SummarizedExperiment-method (runCCA), 49
calculateUniFrac, 10
calculateUniFrac,ANY,phylo-method (calculateUniFrac), 10
calculateUniFrac,TreeSummarizedExperiment,missing-method (calculateUniFrac), 10
cca, *50*
checkTaxonomy (taxonomy-methods), 60
checkTaxonomy,SummarizedExperiment-method (taxonomy-methods), 60
colData, *13*, *17*, *21*, *24*, *27*, *48*
cophenetic.phylo, *51*
countDominantTaxa (summaries), 58
countDominantTaxa,SummarizedExperiment-method (summaries), 58
cvdmngroup , *8*

DECIPHER::ConsensusSequence, *45*
decontam::isContaminant, *35, 36*
decontam::isNotContaminant, *35, 36*
detectLoop, *4*
DirichletMultinomial, *6*
dist, *11*, *53*
diversity, *18*
dmn, *8*
dmn_se (mia-datasets), 46
DMNGroup, *8*
dmngroup, *8*

enterotype (mia-datasets), 46
esophagus (mia-datasets), 46
estimateDivergence, 12
estimateDivergence,SummarizedExperiment-method (estimateDivergence), 12
estimateDiversity, 14, *22*, *25*
estimateDiversity,SummarizedExperiment-method (estimateDiversity), 14
estimateDiversity,TreeSummarizedExperiment-method (estimateDiversity), 14
estimateDominance, *14*, *18*, 19, *25*
estimateDominance,SummarizedExperiment-method (estimateDominance), 19
estimateEvenness, *14*, *17*, *22*, 23
estimateEvenness,SummarizedExperiment-method (estimateEvenness), 23
estimateFaith (estimateDiversity), 14
estimateFaith,SummarizedExperiment,phylo-method (estimateDiversity), 14
estimateFaith,TreeSummarizedExperiment,missing-method (estimateDiversity), 14
estimateR, *18*, *26, 27*
estimateRichness, *14*, *17*, *22*, *25*, 25
estimateRichness,SummarizedExperiment-method (estimateRichness), 25

getAbundance, 28
getAbundanceFeature (getAbundance), 28
getAbundanceFeature,SummarizedExperiment-method (getAbundance), 28
getAbundanceSample (getAbundance), 28
getAbundanceSample,SummarizedExperiment-method (getAbundance), 28
getBestDMNFit (calculateDMN), 6
getBestDMNFit,SummarizedExperiment-method (calculateDMN), 6
getDMN (calculateDMN), 6
getDMN,SummarizedExperiment-method (calculateDMN), 6
getPrevalence, 30
getPrevalence,ANY-method (getPrevalence), 30
getPrevalence,SummarizedExperiment-method (getPrevalence), 30
getPrevalentAbundance (getPrevalence), 30
getPrevalentAbundance,ANY-method (getPrevalence), 30
getPrevalentAbundance,SummarizedExperiment-method (getPrevalence), 30
getPrevalentTaxa, *59*
getPrevalentTaxa (getPrevalence), 30
getPrevalentTaxa,ANY-method (getPrevalence), 30
getPrevalentTaxa,SummarizedExperiment-method (getPrevalence), 30
getRareTaxa (getPrevalence), 30
getRareTaxa,ANY-method (getPrevalence), 30
getRareTaxa,SummarizedExperiment-method (getPrevalence), 30
getTaxonomyLabels (taxonomy-methods), 60

getTaxonomyLabels,SummarizedExperiment-method
(taxonomy-methods), 60
getTopTaxa, *32*
getTopTaxa (summaries), 58
getTopTaxa,SummarizedExperiment-method
(summaries), 58
getUniqueTaxa (summaries), 58
getUniqueTaxa,SummarizedExperiment-method
(summaries), 58
GlobalPatterns (mia-datasets), 46

IdTaxa, *62*
IdTaxaToDataFrame (taxonomy-methods), 60
isContaminant, 34
isContaminant,SummarizedExperiment-method
(isContaminant), 34
isNotContaminant,SummarizedExperiment-method
(isContaminant), 34

loadFromBiom
(makeSummarizedExperimentFromBiom),
40
loadFromMothur, 36, *39*, *41–43*
loadFromQIIME2, *37*, 37, *41–43*

makePhyloseqFromTreeSummarizedExperiment,
39
makePhyloseqFromTreeSummarizedExperiment,SummarizedExperiment-method
(makePhyloseqFromTreeSummarizedExperiment),
39
makePhyloseqFromTreeSummarizedExperiment,TreeSummarizedExperiment-method
(makePhyloseqFromTreeSummarizedExperiment),
39
makeSummarizedExperimentFromBiom, *37*,
39, 40, *42, 43*
makeTreeSummarizedExperimentFromDADA2,
37, *39*, *41*, 41, *43*
makeTreeSummarizedExperimentFromphyloseq,
37, *39*, *41, 42*, 42
mapTaxonomy (taxonomy-methods), 60
mapTaxonomy,SummarizedExperiment-method
(taxonomy-methods), 60
MASS::isoMDS, *53, 54*
meltAssay, 43
meltAssay,SummarizedExperiment-method
(meltAssay), 43
merge-methods, 45
mergeCols (merge-methods), 45
mergeCols,SummarizedExperiment-method
(merge-methods), 45
mergeCols,TreeSummarizedExperiment-method
(merge-methods), 45
mergeRows, *4*, *56*
mergeRows (merge-methods), 45
mergeRows,SummarizedExperiment-method
(merge-methods), 45
mergeRows,TreeSummarizedExperiment-method
(merge-methods), 45
metadata, *8*
mia-datasets, 46
mia-package, 3

name, *26*

perCellQCMetrics, *59*
perFeatureQCMetrics, *59*
performDMNgroupCV (calculateDMN), 6
performDMNgroupCV,ANY-method
(calculateDMN), 6
performDMNgroupCV,SummarizedExperiment-method
(calculateDMN), 6
perSampleDominantTaxa, 47
perSampleDominantTaxa,SummarizedExperiment-method
(perSampleDominantTaxa), 47
phylo, *11*
plotColData, *14*, *17*, *25*, *27*
plotMDS, *54*
plotNMDS (runNMDS), 52
plotReducedDim, *52*
quickPerCellQC, *59*

rda, *50*
reducedDims, *52*
relabundance, 48
relabundance,SummarizedExperiment-method
(relabundance), 48
relabundance<- (relabundance), 48
relabundance<-,SummarizedExperiment-method
(relabundance), 48
relAbundanceCounts (transformCounts), 63
relAbundanceCounts,SummarizedExperiment-method
(transformCounts), 63
resolveLoop, *62*
runCCA, 49
runCCA,SingleCellExperiment-method
(runCCA), 49

runDMN (calculateDMN), 6
runDPCoA, 51
runJSD (calculateJSD), 9
runNMDS, 52
runRDA (runCCA), 49
runRDA,SingleCellExperiment-method (runCCA), 49
runUniFrac (calculateUniFrac), 10

SingleCellExperiment, *50*, *53*
soilrep (mia-datasets), 46
splitAltExps, *56*
splitByRanks, 55
splitByRanks,SingleCellExperiment-method (splitByRanks), 55
splitByRanks,SummarizedExperiment-method (splitByRanks), 55
splitByRanks,TreeSummarizedExperiment-method (splitByRanks), 55
subset, *57*
subsetByPrevalentTaxa (getPrevalence), 30
subsetByPrevalentTaxa,SummarizedExperiment-method (getPrevalence), 30
subsetByRareTaxa (getPrevalence), 30
subsetByRareTaxa,SummarizedExperiment-method (getPrevalence), 30
subsetFeatures (subsetSamples), 57
subsetFeatures,SummarizedExperiment-method (subsetSamples), 57
subsetSamples, 57
subsetSamples,SummarizedExperiment-method (subsetSamples), 57
subsetTaxa (subsetSamples), 57
subsetTaxa,SummarizedExperiment-method (subsetSamples), 57
sumCountsAcrossFeatures, *4*, *45*, *46*, *56*
summaries, 58
SummarizedExperiment, *4*, *5*, *8*, *9*, *13*, *16*, *20*, *23*, *26*, *29*, *31*, *34*, *35*, *37*, *40*, *41*, *43–45*, *47*, *48*, *50*, *55*, *57–59*, *61*, *65*
summary,SummarizedExperiment-method (summaries), 58

taxonomicRanks, *3*
taxonomicRanks (taxonomy-methods), 60
taxonomy-methods, 60
TAXONOMY_RANKS (taxonomy-methods), 60
taxonomyRankEmpty (taxonomy-methods), 60
taxonomyRankEmpty,SummarizedExperiment-method (taxonomy-methods), 60
taxonomyRanks (taxonomy-methods), 60
taxonomyRanks,SummarizedExperiment-method (taxonomy-methods), 60
taxonomyTree (taxonomy-methods), 60
taxonomyTree,SummarizedExperiment-method (taxonomy-methods), 60
toTree, *62*
transformCounts, 63
transformCounts,SummarizedExperiment-method (transformCounts), 63
transformFeatures (transformCounts), 63
transformFeatures,SummarizedExperiment-method (transformCounts), 63
transformSamples (transformCounts), 63
transformSamples,SummarizedExperiment-method (transformCounts), 63
TreeSummarizedExperiment, *3*, *10*, *11*, *38*, *45*, *48*, *49*, *51*

unsplitByRanks (splitByRanks), 55
unsplitByRanks,SingleCellExperiment-method (splitByRanks), 55
unsplitByRanks,TreeSummarizedExperiment-method (splitByRanks), 55

vegan::diversity, *17*
vegan::fisher.alpha, *16*
vegan::monoMDS, *53*, *54*

ZTransform (transformCounts), 63
ZTransform,SummarizedExperiment-method (transformCounts), 63