Package ‘ecodive’

January 16, 2026

Type Package

Title Parallel and Memory-Efficient Ecological Diversity Metrics

Version 2.2.2

Description Computes alpha and beta diversity metrics using concurrent 'C' threads. Metrics include 'UniFrac', Faith's phylogenetic diversity, Bray-Curtis dissimilarity, Shannon diversity index, and many others.
Also parses newick trees into 'phylo' objects and rarefies feature tables.

URL `https://cmmr.github.io/ecodive/`, `https://github.com/cmmr/ecodive`

BugReports `https://github.com/cmmr/ecodive/issues`

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 3.6.0)

RoxygenNote 7.3.3

Config/Needs/website rmarkdown

Config/testthat/edition 3

Imports parallel, utils

Suggests knitr, Matrix, parallelly, rmarkdown, slam, testthat (>= 3.0.0)

VignetteBuilder knitr

NeedsCompilation yes

Author Daniel P. Smith [aut, cre] (ORCID: <`https://orcid.org/0000-0002-2479-2044`>),
Alkek Center for Metagenomics and Microbiome Research [cph, fnd]

Maintainer Daniel P. Smith <`dansmith01@gmail.com`>

Repository CRAN

Date/Publication 2026-01-16 15:10:02 UTC

Contents

`adiv_functions` *Alpha Diversity Metrics*

Description

Alpha Diversity Metrics

Usage

```
ace(counts, cutoff = 10L, margin = 1L, cpus = n_cpus())

berger(counts, norm = "percent", margin = 1L, cpus = n_cpus())

brillouin(counts, margin = 1L, cpus = n_cpus())

chao1(counts, margin = 1L, cpus = n_cpus())

faith(counts, tree = NULL, margin = 1L, cpus = n_cpus())

fisher(counts, digits = 3L, margin = 1L, cpus = n_cpus())

inv_simpson(counts, norm = "percent", margin = 1L, cpus = n_cpus())

margalef(counts, margin = 1L, cpus = n_cpus())

mcintosh(counts, margin = 1L, cpus = n_cpus())

menhinick(counts, margin = 1L, cpus = n_cpus())

observed(counts, margin = 1L, cpus = n_cpus())

shannon(counts, norm = "percent", margin = 1L, cpus = n_cpus())
```

```
simpson(counts, norm = "percent", margin = 1L, cpus = n_cpus())

squares(counts, margin = 1L, cpus = n_cpus())
```

Arguments

`counts`	A numeric matrix of count data where each column is a feature, and each row is a sample. Any object coercible with `as.matrix()` can be given here, as well as `phyloseq`, `rbiom`, `SummarizedExperiment`, and `TreeSummarizedExperiment` objects. For optimal performance with very large datasets, see the guide in `vignette('performance')`.
`cutoff`	The maximum number of observations to consider "rare". Default: `10`.
`margin`	If your samples are in the matrix's rows, set to `1L`. If your samples are in columns, set to `2L`. Ignored when `counts` is a `phyloseq`, `rbiom`, `SummarizedExperiment`, or `TreeSummarizedExperiment` object. Default: `1L`
`cpus`	How many parallel processing threads should be used. The default, `n_cpus()`, will use all logical CPU cores.
`norm`	Normalize the incoming counts. Options are: `norm = "percent"` **-** Relative abundance (sample abundances sum to 1). `norm = "binary"` **-** Unweighted presence/absence (each count is either 0 or 1). `norm = "clr"` **-** Centered log ratio. `norm = "none"` **-** No transformation. Default: `'percent'`, which is the expected input for these formulas.
`tree`	A `phylo`-class object representing the phylogenetic tree for the OTUs in `counts`. The OTU identifiers given by `colnames(counts)` must be present in `tree`. Can be omitted if a tree is embedded with the `counts` object or as `attr(counts, 'tree')`.
`digits`	Precision of the returned values, in number of decimal places. E.g. the default `digits=3` could return `6.392`.

Value

A numeric vector.

Formulas

Prerequisite: all counts are whole numbers.

Given:

- n : The number of features (e.g. species, OTUs, ASVs, etc).
- X_i : Integer count of the i-th feature.
- X_T : Total of all counts (i.e. sequencing depth). $X_T = \sum_{i=1}^{n} X_i$
- P_i : Proportional abundance of the i-th feature. $P_i = X_i / X_T$
- F_1 : Number of features where $X_i = 1$ (i.e. singletons).
- F_2 : Number of features where $X_i = 2$ (i.e. doubletons).

Abundance-based Coverage Estimator (ACE) `ace()`	See below.
Berger-Parker Index `berger()`	$\max(P_i)$
Brillouin Index `brillouin()`	$\dfrac{\ln{[(\sum_{i=1}^{n} X_i)!]} - \sum_{i=1}^{n} \ln{(X_i!)}}{\sum_{i=1}^{n} X_i}$
Chao1 `chao1()`	$n + \dfrac{(F_1)^2}{2F_2}$
Faith's Phylogenetic Diversity `faith()`	See below.
Fisher's Alpha (α) `fisher()`	$\dfrac{n}{\alpha} = \ln{\left(1 + \dfrac{X_T}{\alpha}\right)}$ The value of α must be solved for iteratively.
Gini-Simpson Index `simpson()`	$1 - \sum_{i=1}^{n} P_i^2$
Inverse Simpson Index `inv_simpson()`	$1 / \sum_{i=1}^{n} P_i^2$
Margalef's Richness Index `margalef()`	$\dfrac{n - 1}{\ln{X_T}}$
McIntosh Index `mcintosh()`	$\dfrac{X_T - \sqrt{\sum_{i=1}^{n} (X_i)^2}}{X_T - \sqrt{X_T}}$
Menhinick's Richness Index `menhinick()`	$\dfrac{n}{\sqrt{X_T}}$
Observed Features `observed()`	n
Shannon Diversity Index `shannon()`	$-\sum_{i=1}^{n} P_i \times \ln(P_i)$
Squares Richness Estimator `squares()`	$n + \dfrac{(F_1)^2 \sum_{i=1}^{n} (X_i)^2}{X_T^2 - nF_1}$

Abundance-based Coverage Estimator (ACE):

Given:

- n : The number of features (e.g. species, OTUs, ASVs, etc).
- r : Rare cutoff. Features with $\le r$ counts are considered rare.
- X_i : Integer count of the i-th feature.
- F_i : Number of features with exactly i counts.
- F_1 : Number of features where $X_i = 1$ (i.e. singletons).
- F_{rare} : Number of rare features where $X_i \le r$.
- F_{abund} : Number of abundant features where $X_i > r$.
- X_{rare} : Total counts belonging to rare features.
- C_{ace} : The sample abundance coverage estimator, defined below.
- γ_{ace}^2 : The estimated coefficient of variation, defined below.
- D_{ace} : Estimated number of features in the sample.

$$C_{ace} = 1 - \frac{F_1}{X_{rare}}$$

$$\gamma_{ace}^2 = \max\left[\frac{F_{rare} \sum_{i=1}^{r} i(i-1)F_i}{C_{ace}X_{rare}(X_{rare} - 1)} - 1, 0\right]$$

$$D_{ace} = F_{abund} + \frac{F_{rare}}{C_{ace}} + \frac{F_1}{C_{ace}} \gamma_{ace}^2$$

Faith's Phylogenetic Diversity (Faith's PD):

Given n branches with lengths L and a sample's abundances A on each of those branches coded as 1 for present or 0 for absent:

$\sum_{i=1}^{n} L_i A_i$

Examples

```
# Example counts matrix
t(ex_counts)

ace(ex_counts)

chao1(ex_counts)

squares(ex_counts)
```

`alpha_div` *Alpha Diversity Wrapper Function*

Description

Alpha Diversity Wrapper Function

Usage

```
alpha_div(
  counts,
  metric,
  norm = "percent",
  cutoff = 10L,
  digits = 3L,
  tree = NULL,
  margin = 1L,
  cpus = n_cpus()
)
```

Arguments

`counts`	A numeric matrix of count data where each column is a feature, and each row is a sample. Any object coercible with `as.matrix()` can be given here, as well as `phyloseq`, `rbiom`, `SummarizedExperiment`, and `TreeSummarizedExperiment` objects. For optimal performance with very large datasets, see the guide in `vignette('performance')`.
`metric`	The name of an alpha diversity metric. One of `c('ace', 'berger', 'brillouin', 'chao1', 'faith', 'fisher', 'inv_simpson', 'margalef', 'mcintosh', 'menhinick', 'observed', 'shannon', 'simpson', 'squares')`. Case-insensitive and partial name matching is supported. Programmatic access via `list_metrics('alpha')`.
`norm`	Normalize the incoming counts. Options are: `norm = "percent"` - Relative abundance (sample abundances sum to 1). `norm = "binary"` - Unweighted presence/absence (each count is either 0 or 1). `norm = "clr"` - Centered log ratio.

	`norm = "none"` - No transformation.
	Default: `'percent'`, which is the expected input for these formulas.
`cutoff`	The maximum number of observations to consider "rare". Default: `10`.
`digits`	Precision of the returned values, in number of decimal places. E.g. the default `digits=3` could return `6.392`.
`tree`	A `phylo`-class object representing the phylogenetic tree for the OTUs in `counts`. The OTU identifiers given by `colnames(counts)` must be present in `tree`. Can be omitted if a tree is embedded with the `counts` object or as `attr(counts, 'tree')`.
`margin`	If your samples are in the matrix's rows, set to `1L`. If your samples are in columns, set to `2L`. Ignored when `counts` is a `phyloseq`, `rbiom`, `SummarizedExperiment`, or `TreeSummarizedExperiment` object. Default: `1L`
`cpus`	How many parallel processing threads should be used. The default, `n_cpus()`, will use all logical CPU cores.

Details

Integer Count Requirements:

A frequent and critical error in alpha diversity analysis is providing the wrong type of data to a metric's formula. Some indices are mathematically defined based on counts of individuals and require raw, integer abundance data. Others are based on proportional abundances and can accept either integer counts (which are then converted to proportions) or pre-normalized proportional data. Using proportional data with a metric that requires integer counts will return an error message.

Requires Integer Counts Only:

- Chao1
- ACE
- Squares Richness Estimator
- Margalef's Index
- Menhinick's Index
- Fisher's Alpha
- Brillouin Index

Can Use Proportional Data:

- Observed Features
- Shannon Index
- Gini-Simpson Index
- Inverse Simpson Index
- Berger-Parker Index
- McIntosh Index
- Faith's PD

Value

A numeric vector.

Examples

```
# Example counts matrix
ex_counts

# Shannon diversity values
alpha_div(ex_counts, 'Shannon')

# Chao1 diversity values
alpha_div(ex_counts, 'c')

# Faith PD values
alpha_div(ex_counts, 'faith', tree = ex_tree)
```

`bdiv_functions` *Beta Diversity Metrics*

Description

Beta Diversity Metrics

Usage

```
aitchison(
  counts,
  pseudocount = NULL,
  margin = 1L,
  pairs = NULL,
  cpus = n_cpus()
)

bhattacharyya(
  counts,
  norm = "percent",
  margin = 1L,
  pairs = NULL,
  cpus = n_cpus()
)

bray(counts, norm = "percent", margin = 1L, pairs = NULL, cpus = n_cpus())

canberra(counts, norm = "percent", margin = 1L, pairs = NULL, cpus = n_cpus())

chebyshev(counts, norm = "percent", margin = 1L, pairs = NULL, cpus = n_cpus())

chord(counts, margin = 1L, pairs = NULL, cpus = n_cpus())
```

```
clark(counts, norm = "percent", margin = 1L, pairs = NULL, cpus = n_cpus())

divergence(
  counts,
  norm = "percent",
  margin = 1L,
  pairs = NULL,
  cpus = n_cpus()
)

euclidean(counts, norm = "percent", margin = 1L, pairs = NULL, cpus = n_cpus())

gower(counts, norm = "percent", margin = 1L, pairs = NULL, cpus = n_cpus())

hellinger(counts, norm = "percent", margin = 1L, pairs = NULL, cpus = n_cpus())

horn(counts, norm = "percent", margin = 1L, pairs = NULL, cpus = n_cpus())

jensen(counts, norm = "percent", margin = 1L, pairs = NULL, cpus = n_cpus())

jsd(counts, norm = "percent", margin = 1L, pairs = NULL, cpus = n_cpus())

lorentzian(
  counts,
  norm = "percent",
  margin = 1L,
  pairs = NULL,
  cpus = n_cpus()
)

manhattan(counts, norm = "percent", margin = 1L, pairs = NULL, cpus = n_cpus())

matusita(counts, norm = "percent", margin = 1L, pairs = NULL, cpus = n_cpus())

minkowski(
  counts,
  norm = "percent",
  power = 1.5,
  margin = 1L,
  pairs = NULL,
  cpus = n_cpus()
)

morisita(counts, margin = 1L, pairs = NULL, cpus = n_cpus())

motyka(counts, norm = "percent", margin = 1L, pairs = NULL, cpus = n_cpus())
```

```
psym_chisq(
  counts,
  norm = "percent",
  margin = 1L,
  pairs = NULL,
  cpus = n_cpus()
)

soergel(counts, norm = "percent", margin = 1L, pairs = NULL, cpus = n_cpus())

squared_chisq(
  counts,
  norm = "percent",
  margin = 1L,
  pairs = NULL,
  cpus = n_cpus()
)

squared_chord(
  counts,
  norm = "percent",
  margin = 1L,
  pairs = NULL,
  cpus = n_cpus()
)

squared_euclidean(
  counts,
  norm = "percent",
  margin = 1L,
  pairs = NULL,
  cpus = n_cpus()
)

topsoe(counts, norm = "percent", margin = 1L, pairs = NULL, cpus = n_cpus())

wave_hedges(
  counts,
  norm = "percent",
  margin = 1L,
  pairs = NULL,
  cpus = n_cpus()
)

hamming(counts, margin = 1L, pairs = NULL, cpus = n_cpus())

jaccard(counts, margin = 1L, pairs = NULL, cpus = n_cpus())
```

```
ochiai(counts, margin = 1L, pairs = NULL, cpus = n_cpus())

sorensen(counts, margin = 1L, pairs = NULL, cpus = n_cpus())

unweighted_unifrac(
  counts,
  tree = NULL,
  margin = 1L,
  pairs = NULL,
  cpus = n_cpus()
)

weighted_unifrac(
  counts,
  tree = NULL,
  margin = 1L,
  pairs = NULL,
  cpus = n_cpus()
)

normalized_unifrac(
  counts,
  tree = NULL,
  margin = 1L,
  pairs = NULL,
  cpus = n_cpus()
)

generalized_unifrac(
  counts,
  tree = NULL,
  alpha = 0.5,
  margin = 1L,
  pairs = NULL,
  cpus = n_cpus()
)

variance_adjusted_unifrac(
  counts,
  tree = NULL,
  margin = 1L,
  pairs = NULL,
  cpus = n_cpus()
)
```

Arguments

`counts`	A numeric matrix of count data where each column is a feature, and each row is a sample. Any object coercible with `as.matrix()` can be given here, as well as

	`phyloseq`, `rbiom`, `SummarizedExperiment`, and `TreeSummarizedExperiment` objects. For optimal performance with very large datasets, see the guide in `vignette('performance')`.
`pseudocount`	The value to add to all counts in `counts` to prevent taking `log(0)` for unobserved features. The default, `NULL`, selects the smallest non-zero value in `counts`.
`margin`	If your samples are in the matrix's rows, set to `1L`. If your samples are in columns, set to `2L`. Ignored when `counts` is a `phyloseq`, `rbiom`, `SummarizedExperiment`, or `TreeSummarizedExperiment` object. Default: `1L`
`pairs`	Which combinations of samples should distances be calculated for? The default value (`NULL`) calculates all-vs-all. Provide a numeric or logical vector specifying positions in the distance matrix to calculate. See examples.
`cpus`	How many parallel processing threads should be used. The default, `n_cpus()`, will use all logical CPU cores.
`norm`	Normalize the incoming counts. Options are: `norm = "percent"` **-** Relative abundance (sample abundances sum to 1). `norm = "binary"` **-** Unweighted presence/absence (each count is either 0 or 1). `norm = "clr"` **-** Centered log ratio. `norm = "none"` **-** No transformation. Default: `'percent'`, which is the expected input for these formulas.
`power`	Scaling factor for the magnitude of differences between communities (p). Default: `1.5`
`tree`	A `phylo`-class object representing the phylogenetic tree for the OTUs in `counts`. The OTU identifiers given by `colnames(counts)` must be present in `tree`. Can be omitted if a tree is embedded with the `counts` object or as `attr(counts, 'tree')`.
`alpha`	How much weight to give to relative abundances; a value between 0 and 1, inclusive. Setting `alpha=1` is equivalent to `normalized_unifrac()`.

Value

A `dist` object.

Formulas

Given:

- n : The number of features.
- X_i, Y_i : Absolute counts for the i-th feature in samples X and Y.
- X_T, Y_T : Total counts in each sample. $X_T = \sum_{i=1}^{n} X_i$
- P_i, Q_i : Proportional abundances of X_i and Y_i. $P_i = X_i / X_T$
- X_L, Y_L : Mean log of abundances. $X_L = \frac{1}{n} \sum_{i=1}^{n} \ln X_i$
- R_i : The range of the i-th feature across all samples (max - min).

Aitchison distance `aitchison()`	$\sqrt{\sum_{i=1}^{n}[(\ln X_i - X_L) - (\ln Y_i - Y_L)]^2}$
Bhattacharyya distance `bhattacharyya()`	$-\ln \sum_{i=1}^{n} \sqrt{P_i Q_i}$
Bray-Curtis dissimilarity `bray()`	$\frac{\sum_{i=1}^{n} \lvert P_i - Q_i \rvert}{\sum_{i=1}^{n}(P_i + Q_i)}$
Canberra distance `canberra()`	$\sum_{i=1}^{n} \frac{\lvert P_i - Q_i \rvert}{P_i + Q_i}$
Chebyshev distance `chebyshev()`	$\max(\lvert P_i - Q_i \rvert)$
Chord distance `chord()`	$\sqrt{\sum_{i=1}^{n} \left(\frac{X_i}{\sqrt{\sum_{j=1}^{n} X_j^2}} - \frac{Y_i}{\sqrt{\sum_{j=1}^{n} Y_j^2}} \right)^2}$
Clark's divergence distance `clark()`	$\sqrt{\sum_{i=1}^{n} \left(\frac{P_i - Q_i}{P_i + Q_i} \right)^2}$
Divergence `divergence()`	$2 \sum_{i=1}^{n} \frac{(P_i - Q_i)^2}{(P_i + Q_i)^2}$
Euclidean distance `euclidean()`	$\sqrt{\sum_{i=1}^{n}(P_i - Q_i)^2}$
Gower distance `gower()`	$\frac{1}{n} \sum_{i=1}^{n} \frac{\lvert P_i - Q_i \rvert}{R_i}$
Hellinger distance `hellinger()`	$\sqrt{\sum_{i=1}^{n}(\sqrt{P_i} - \sqrt{Q_i})^2}$
Horn-Morisita dissimilarity `horn()`	$1 - \frac{2 \sum_{i=1}^{n} P_i Q_i}{\sum_{i=1}^{n} P_i^2 + \sum_{i=1}^{n} Q_i^2}$
Jensen-Shannon distance `jensen()`	$\sqrt{\frac{1}{2} \left[\sum_{i=1}^{n} P_i \ln \left(\frac{2P_i}{P_i + Q_i} \right) + \sum_{i=1}^{n} Q_i \ln \left(\frac{2Q_i}{P_i + Q_i} \right) \right]}$
Jensen-Shannon divergence (JSD) `jsd()`	$\frac{1}{2} \left[\sum_{i=1}^{n} P_i \ln \left(\frac{2P_i}{P_i + Q_i} \right) + \sum_{i=1}^{n} Q_i \ln \left(\frac{2Q_i}{P_i + Q_i} \right) \right]$
Lorentzian distance `lorentzian()`	$\sum_{i=1}^{n} \ln(1 + \lvert P_i - Q_i \rvert)$
Manhattan distance `manhattan()`	$\sum_{i=1}^{n} \lvert P_i - Q_i \rvert$
Matusita distance `matusita()`	$\sqrt{\sum_{i=1}^{n} \left(\sqrt{P_i} - \sqrt{Q_i} \right)^2}$
Minkowski distance `minkowski()`	$\sqrt[p]{\sum_{i=1}^{n}(P_i - Q_i)^p}$ Where p is the geometry of the space.
Morisita dissimilarity * Integers Only `morisita()`	$1 - \frac{2 \sum_{i=1}^{n} X_i Y_i}{\left(\frac{\sum_{i=1}^{n} X_i(X_i - 1)}{X_T(X_T - 1)} + \frac{\sum_{i=1}^{n} Y_i(Y_i - 1)}{Y_T(Y_T - 1)} \right) X_T Y_T}$
Motyka dissimilarity `motyka()`	$\frac{\sum_{i=1}^{n} \max(P_i, Q_i)}{\sum_{i=1}^{n}(P_i + Q_i)}$
Probabilistic Symmetric χ^2 distance `psym_chisq()`	$2 \sum_{i=1}^{n} \frac{(P_i - Q_i)^2}{P_i + Q_i}$
Soergel distance `soergel()`	$\frac{\sum_{i=1}^{n} \lvert P_i - Q_i \rvert}{\sum_{i=1}^{n} \max(P_i, Q_i)}$
Squared χ^2 distance `squared_chisq()`	$\sum_{i=1}^{n} \frac{(P_i - Q_i)^2}{P_i + Q_i}$
Squared Chord distance `squared_chord()`	$\sum_{i=1}^{n} \left(\sqrt{P_i} - \sqrt{Q_i} \right)^2$

Squared Euclidean distance `squared_euclidean()` $\sum_{i=1}^{n}(P_i - Q_i)^2$

Topsoe distance `topsoe()`
$$\sum_{i=1}^{n} P_i \ln\left(\frac{2P_i}{P_i + Q_i}\right) + \sum_{i=1}^{n} Q_i \ln\left(\frac{2Q_i}{P_i + Q_i}\right)$$

Wave Hedges distance `wave_hedges()`
$$\frac{\sum_{i=1}^{n} |P_i - Q_i|}{\sum_{i=1}^{n} \max(P_i, Q_i)}$$

Presence / Absence:

Given:

- A, B : Number of features in each sample.
- J : Number of features in common.

Dice-Sorensen dissimilarity `sorensen()`
$$\frac{2J}{(A + B)}$$

Hamming distance `hamming()` $(A + B) - 2J$

Jaccard distance `jaccard()`
$$1 - \frac{J}{(A + B - J)]}$$

Otsuka-Ochiai dissimilarity `ochiai()`
$$1 - \frac{J}{\sqrt{AB}}$$

Phylogenetic:

Given n branches with lengths L and a pair of samples' binary (A and B) or proportional abundances (P and Q) on each of those branches.

Unweighted UniFrac `unweighted_unifrac()`
$$\frac{1}{n}\sum_{i=1}^{n} L_i|A_i - B_i|$$

Weighted UniFrac `weighted_unifrac()`
$$\sum_{i=1}^{n} L_i|P_i - Q_i|$$

Normalized Weighted UniFrac `normalized_unifrac()`
$$\frac{\sum_{i=1}^{n} L_i|P_i - Q_i|}{\sum_{i=1}^{n} L_i(P_i + Q_i)}$$

Generalized UniFrac (GUniFrac) `generalized_unifrac()`
$$\frac{\sum_{i=1}^{n} L_i(P_i + Q_i)^{\alpha} \left|\frac{P_i - Q_i}{P_i + Q_i}\right|}{\sum_{i=1}^{n} L_i(P_i + Q_i)^{\alpha}}$$ Where α is a sc

Variance-Adjusted Weighted UniFrac `variance_adjusted_unifrac()`
$$\frac{\sum_{i=1}^{n} L_i \frac{|P_i - Q_i|}{\sqrt{(P_i + Q_i)(2 - P_i - Q_i)}}}{\sum_{i=1}^{n} L_i \frac{P_i + Q_i}{\sqrt{(P_i + Q_i)(2 - P_i - Q_i)}}}$$

See `vignette('unifrac')` for detailed example UniFrac calculations.

References

Levy, A., Shalom, B. R., & Chalamish, M. (2024). A guide to similarity measures. *arXiv*.

Cha, S.-H. (2007). Comprehensive survey on distance/similarity measures between probability density functions. *International Journal of Mathematical Models and Methods in Applied Sciences*, 1(4), 300–307.

Examples

```
# Example counts matrix
t(ex_counts)

bray(ex_counts)

jaccard(ex_counts)

generalized_unifrac(ex_counts, tree = ex_tree)

# Only calculate distances for Saliva vs all.
bray(ex_counts, pairs = 1:3)
```

`beta_div` *Beta Diversity Wrapper Function*

Description

Beta Diversity Wrapper Function

Usage

```
beta_div(
  counts,
  metric,
  norm = "percent",
  power = 1.5,
  pseudocount = NULL,
  alpha = 0.5,
  tree = NULL,
  pairs = NULL,
  margin = 1L,
  cpus = n_cpus()
)
```

Arguments

`counts`	A numeric matrix of count data where each column is a feature, and each row is a sample. Any object coercible with `as.matrix()` can be given here, as well as `phyloseq`, `rbiom`, `SummarizedExperiment`, and `TreeSummarizedExperiment` objects. For optimal performance with very large datasets, see the guide in `vignette('performance')`.
`metric`	The name of a beta diversity metric. One of `c('aitchison', 'bhattacharyya', 'bray', 'canberra', 'chebyshev', 'chord', 'clark', 'divergence', 'euclidean', 'generalized_unifrac', 'gower', 'hamming', 'hellinger', 'horn', 'jaccard', 'jensen', 'jsd', 'lorentzian', 'manhattan', 'matusita', 'minkowski', 'morisita', 'motyka', 'normalized_unifrac', 'ochiai', 'psym_chisq', 'soergel', 'sorensen', 'squared_chisq', 'squared_chord', 'squared_euclidean', 'topsoe', 'unweighted_unifrac', 'variance_adjusted_unifrac', 'wave_hedges', 'weighted_unifrac')`. Flexible matching is supported (see below). Programmatic access via `list_metrics('beta')`.
`norm`	Normalize the incoming counts. Options are: `norm = "percent"` **-** Relative abundance (sample abundances sum to 1). `norm = "binary"` **-** Unweighted presence/absence (each count is either 0 or 1). `norm = "clr"` **-** Centered log ratio. `norm = "none"` **-** No transformation. Default: `'percent'`, which is the expected input for these formulas.
`power`	Scaling factor for the magnitude of differences between communities (p). Default: `1.5`
`pseudocount`	The value to add to all counts in `counts` to prevent taking `log(0)` for unobserved features. The default, `NULL`, selects the smallest non-zero value in `counts`.
`alpha`	How much weight to give to relative abundances; a value between 0 and 1, inclusive. Setting `alpha=1` is equivalent to `normalized_unifrac()`.
`tree`	A `phylo`-class object representing the phylogenetic tree for the OTUs in `counts`. The OTU identifiers given by `colnames(counts)` must be present in `tree`. Can be omitted if a tree is embedded with the `counts` object or as `attr(counts, 'tree')`.
`pairs`	Which combinations of samples should distances be calculated for? The default value (`NULL`) calculates all-vs-all. Provide a numeric or logical vector specifying positions in the distance matrix to calculate. See examples.
`margin`	If your samples are in the matrix's rows, set to `1L`. If your samples are in columns, set to `2L`. Ignored when `counts` is a `phyloseq`, `rbiom`, `SummarizedExperiment`, or `TreeSummarizedExperiment` object. Default: `1L`
`cpus`	How many parallel processing threads should be used. The default, `n_cpus()`, will use all logical CPU cores.

Details

List of Beta Diversity Metrics

Option / Function Name	Metric Name
`aitchison`	Aitchison distance
`bhattacharyya`	Bhattacharyya distance
`bray`	Bray-Curtis dissimilarity
`canberra`	Canberra distance
`chebyshev`	Chebyshev distance
`chord`	Chord distance
`clark`	Clark's divergence distance
`divergence`	Divergence
`euclidean`	Euclidean distance
`generalized_unifrac`	Generalized UniFrac (GUniFrac)
`gower`	Gower distance
`hamming`	Hamming distance
`hellinger`	Hellinger distance
`horn`	Horn-Morisita dissimilarity
`jaccard`	Jaccard distance
`jensen`	Jensen-Shannon distance
`jsd`	Jesen-Shannon divergence (JSD)
`lorentzian`	Lorentzian distance
`manhattan`	Manhattan distance
`matusita`	Matusita distance
`minkowski`	Minkowski distance
`morisita`	Morisita dissimilarity
`motyka`	Motyka dissimilarity
`normalized_unifrac`	Normalized Weighted UniFrac
`ochiai`	Otsuka-Ochiai dissimilarity
`psym_chisq`	Probabilistic Symmetric Chi-Squared distance
`soergel`	Soergel distance
`sorensen`	Dice-Sorensen dissimilarity
`squared_chisq`	Squared Chi-Squared distance
`squared_chord`	Squared Chord distance
`squared_euclidean`	Squared Euclidean distance
`topsoe`	Topsoe distance
`unweighted_unifrac`	Unweighted UniFrac
`variance_adjusted_unifrac`	Variance-Adjusted Weighted UniFrac (VAW-UniFrac)
`wave_hedges`	Wave Hedges distance
`weighted_unifrac`	Weighted UniFrac

Flexible name matching

Case insensitive and partial matching. Any runs of non-alpha characters are converted to underscores. E.g. `metric = 'Weighted UniFrac` selects `weighted_unifrac`.

UniFrac names can be shortened to the first letter plus "unifrac". E.g. `uunifrac`, `w_unifrac`, or `V UniFrac`. These also support partial matching.

Finished code should always use the full primary option name to avoid ambiguity with future additions to the metrics list.

Value

A numeric vector.

Examples

```
# Example counts matrix
ex_counts

# Bray-Curtis distances
beta_div(ex_counts, 'bray')

# Generalized UniFrac distances
beta_div(ex_counts, 'GUniFrac', tree = ex_tree)
```

`ex_counts` *Example counts matrix*

Description

Genera found on four human body sites.

Usage

```
ex_counts
```

Format

A matrix of 4 samples (columns) x 6 genera (rows).

Source

Derived from The Human Microbiome Project dataset. `https://commonfund.nih.gov/hmp`

`ex_tree` *Example phylogenetic tree*

Description

Companion tree for `ex_counts`.

Usage

```
ex_tree
```

Format

A `phylo` object.

Details

`ex_tree` encodes this tree structure:

```
      +----------44---------- Haemophilus
  +-2-|
  |   +----------------68---------------- Bacteroides
  |
  |             +---18---- Streptococcus
  |      +--12--|
  |      |      +--11-- Staphylococcus
  +--11--|
         |      +-----24----- Corynebacterium
         +--12--|
                +--13-- Propionibacterium
```

`list_metrics` *Find and Browse Available Metrics*

Description

Programmatic access to the lists of available metrics, and their associated functions.

Usage

```
list_metrics(
  div = c(NA, "alpha", "beta"),
  val = c("data.frame", "list", "func", "id", "name", "div", "phylo", "weighted",
    "int_only", "true_metric"),
  nm = c(NA, "id", "name"),
  phylo = NULL,
  weighted = NULL,
  int_only = NULL,
  true_metric = NULL
)

match_metric(
  metric,
  div = NULL,
  phylo = NULL,
  weighted = NULL,
  int_only = NULL,
  true_metric = NULL
)
```

Arguments

`div`, `phylo`, `weighted`, `int_only`, `true_metric`	Consider only metrics matching specific criteria. For example, `div = "alpha"` will only return alpha diversity metrics. Default: `NULL`
`val`	Sets the return value for this function call. See "Value" section below. Default: `"data.frame"`
`nm`	What value to use for the names of the returned object. Default is `"id"` when `val` is `"list"` or `"func"`, otherwise the default is `NA` (no name).
`metric`	The name of an alpha/beta diversity metric to search for. Supports partial matching. All non-alpha characters are ignored.

Value

`match_metric()`

A `list` with the following elements.

- `name` : Metric name, e.g. `"Faith's Phylogenetic Diversity"`
- `id` : Metric ID - also the name of the function, e.g. `"faith"`
- `div` : Either `"alpha"` or `"beta"`.
- `phylo` : `TRUE` if metric requires a phylogenetic tree; `FALSE` otherwise.
- `weighted` : `TRUE` if metric takes relative abundance into account; `FALSE` if it only uses presence/absence.
- `int_only` : `TRUE` if metric requires integer counts; `FALSE` otherwise.
- `true_metric` : `TRUE` if metric is a true metric and satisfies the triangle inequality; `FALSE` if it is a non-metric dissimilarity; `NA` for alpha diversity metrics.
- `func` : The function for this metric, e.g. `ecodive::faith`
- `params` : Formal args for `func`, e.g. `c("counts", "tree", "cpus")`

`list_metrics()`

The returned object's type and values are controlled with the `val` and `nm` arguments.

- `val = "data.frame"` : The data.frame from which the below options are sourced.
- `val = "list"` : A list of objects as returned by `match_metric()` (above).
- `val = "func"` : A list of functions.
- `val = "id"` : A character vector of metric IDs.
- `val = "name"` : A character vector of metric names.
- `val = "div"` : A character vector `"alpha"` and/or `"beta"`.
- `val = "phylo"` : A logical vector indicating which metrics require a tree.
- `val = "weighted"` : A logical vector indicating which metrics take relative abundance into account (as opposed to just presence/absence).
- `val = "int_only"` : A logical vector indicating which metrics require integer counts.
- `val = "true_metric"` : A logical vector indicating which metrics are true metrics and satisfy the triangle inequality, which work better for ordinations such as PCoA.

If `nm` is set, then the names of the vector or list will be the metric ID (`nm="id"`) or name (`nm="name"`). When `val="data.frame"`, the names will be applied to the `rownames()` property of the `data.table`.

Examples

```
# A data.frame of all available metrics.
head(list_metrics())

# All alpha diversity function names.
list_metrics('alpha', val = 'id')

# Try to find a metric named 'otus'.
m <- match_metric('otus')

# The result is a list that includes the function.
str(m)
```

n_cpus *Number of CPU Cores*

Description

A thin wrapper around `parallely::availableCores()`. If the `parallely` package is not installed, then it falls back to `parallel::detectCores(all.tests = TRUE, logical = TRUE)`. Returns `1` if `pthread` support is unavailable or when the number of cpus cannot be determined.

Usage

```
n_cpus()
```

Value

A scalar integer, guaranteed to be at least `1`.

Examples

```
n_cpus()
```

rarefy *Rarefy OTU counts.*

Description

Sub-sample OTU observations such that all samples have an equal number. If called on data with non-integer abundances, values will be re-scaled to integers between 1 and `depth` such that they sum to `depth`.

Usage

```
rarefy(
  counts,
  depth = 0.1,
  n_samples = NULL,
  seed = 0,
  times = NULL,
  drop = TRUE,
  margin = 1L,
  cpus = n_cpus()
)
```

Arguments

`counts`	A numeric matrix of count data where each column is a feature, and each row is a sample. Any object coercible with `as.matrix()` can be given here, as well as `phyloseq`, `rbiom`, `SummarizedExperiment`, and `TreeSummarizedExperiment` objects. For optimal performance with very large datasets, see the guide in `vignette('performance')`.
`depth`	How many observations to keep per sample. When `0 < depth < 1`, it is taken as the minimum percentage of the dataset's observations to keep. Ignored when `n_samples` is specified. Default: `0.1`
`n_samples`	The number of samples to keep. When `0 < n_samples < 1`, it is taken as the percentage of samples to keep. If negative, that number of samples is dropped. If `0`, all samples are kept. If `NULL`, then `depth` is used instead. Default: `NULL`
`seed`	An integer seed for randomizing which observations to keep or drop. If you need to create different random rarefactions of the same data, set the seed to a different number each time. Default: `0`
`times`	How many independent rarefactions to perform. If set, `rarefy()` will return a list of matrices. The seeds for each matrix will be sequential, starting from `seed`. Default: `NULL`
`drop`	Drop rows and columns with zero observations after rarefying. Default: `TRUE`
`margin`	If your samples are in the matrix's rows, set to `1L`. If your samples are in columns, set to `2L`. Ignored when `counts` is a `phyloseq`, `rbiom`, `SummarizedExperiment`, or `TreeSummarizedExperiment` object. Default: `1L`
`cpus`	How many parallel processing threads should be used. The default, `n_cpus()`, will use all logical CPU cores.

Value

A rarefied matrix. `Matrix` and `slam` objects will be returned with the same type; otherwise a base R `matrix` will be returned.

Examples

```
    # A 4-sample x 5-OTU matrix with samples in rows.
    counts <- matrix(c(0,0,0,0,0,8,9,10,5,5,5,5,2,0,0,0,6,5,7,0), 4, 5,
      dimnames = list(LETTERS[1:4], paste0('OTU', 1:5)))
    counts
    rowSums(counts)

    # Rarefy all samples to a depth of 13.
    # Note that sample 'A' has 0 counts and is dropped.
    r_mtx <- rarefy(counts, depth = 13, seed = 1)
    r_mtx
    rowSums(r_mtx)

    # Keep zero-sum rows and columns by setting `drop = FALSE`.
    rarefy(counts, depth = 13, drop = FALSE, seed = 1)

    # Rarefy to the depth of the 2nd most abundant sample (B, depth=22).
    rarefy(counts, n_samples = 2, seed = 1)

    # Perform 3 independent rarefactions.
    r_list <- rarefy(counts, depth = 13, times = 3, seed = 1)
    length(r_list)
    r_list[[1]]

    # The class of the input matrix is preserved.
    if (requireNamespace('Matrix', quietly = TRUE)) {
      counts_dgC <- Matrix::Matrix(counts, sparse = TRUE)
      class(counts_dgC)
      r_dgC <- rarefy(counts_dgC, depth = 13, seed = 1)
      class(r_dgC)
    }
```

read_tree — *Read a newick formatted phylogenetic tree.*

Description

A phylogenetic tree is required for computing UniFrac distance matrices. You can load a tree from a file or by providing the tree string directly. This tree must be in Newick format, also known as parenthetic format and New Hampshire format.

Usage

```
read_tree(newick, underscores = FALSE)
```

Arguments

`newick`	Input data as either a file path, URL, or Newick string. Compressed (gzip or bzip2) files are also supported.
`underscores`	If `TRUE`, underscores in unquoted names will remain underscores. If `FALSE`, underscores in unquoted named will be converted to spaces.

Value

A `phylo` class object representing the tree.

Examples

```
tree <- read_tree("
    (A:0.99,((B:0.87,C:0.89):0.51,(((D:0.16,(E:0.83,F:0.96)
    :0.94):0.69,(G:0.92,(H:0.62,I:0.85):0.54):0.23):0.74,J:0.1
    2):0.43):0.67);")
class(tree)
```

Index