Package 'linf'

August 21, 2026

Type Package

Title L-Infinity Normalization and Dominant Community State Types

Version 0.2.0

Author Pawel Gajer [aut, cre]

Maintainer Pawel Gajer <pgajer@gmail.com>

Description Implements L-infinity normalization for compositional matrices, assigns samples to dominant features, constructs truncated and hierarchically refined dominant community state types, and computes representative landmark profiles. The methods are described in the accompanying publication <doi:10.48550/arXiv.2503.21543>. Bundled vaginal and gut microbiome data support reproducible demonstrations of the package interface; phenotype fields in the stratified gut subset are illustrative and are not suitable for population-level inference.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

LazyData true

Depends R (>= 4.0)

Imports Matrix, methods

Suggests testthat (>= 3.2.0), htmltools, knitr, plotly, rmarkdown

VignetteBuilder knitr

Config/testthat/edition 3

URL https://github.com/pgajer/linf, https://arxiv.org/abs/2503.21543

BugReports https://github.com/pgajer/linf/issues

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.3

NeedsCompilation no

Repository CRAN

Date/Publication 2026-08-21 05:42:17 UTC

Contents

`agp_gut` *American Gut Project gut microbiome dataset*

Description

A stratified subsample of 766 gut microbiome samples from the American Gut Project (PRJEB11419, AGP-US-2015), bundled for demonstrating L-infinity dCST construction in a gut ecosystem.

Usage

```
agp_gut
```

Format

A list with four components:

counts Integer matrix (766 x 314). Raw 16S V4 read counts. Rows are samples, columns are SILVA species-level taxa.

meta Data frame (766 rows) with columns: `Run` (SRA run accession), `dcst_depth1`, `dcst_depth2` (pre-computed dCST labels), `IBS`, `IBD`, `Diabetes`, `Autoimmune`, `Seasonal_allergies`, `Migraine`, `Acid_reflux`, `Lung_disease`, `Cardiovascular_disease`, `Skin_condition`, `Obesity` (binary disease indicators from self-reported AGP metadata), `BMI` (numeric, self-reported), and `selection_reason` (target-dCST inclusion or seeded background sampling).

taxa Character vector of 314 SILVA taxonomy strings.

source Character string documenting provenance.

Details

The subsample includes all samples assigned to four uncommon demonstration dCSTs (Prevotella_7, Pasteurellaceae, Akkermansia, and Staphylococcus). The remaining slots are a simple random sample, drawn with seed 42, from the eligible background after excluding Eukaryota and Unassigned labels. Phenotype fields are joined only after membership is fixed and do not influence selection. Because inclusion probabilities differ by dCST, this object is a computational demonstration dataset rather than a probability sample of the underlying cohort.

The American Gut Project is a large citizen-science 16S rRNA survey of the human microbiome. Health conditions are self-reported via questionnaire and should be interpreted with appropriate caution.

The phenotype fields must not be used with this dCST-stratified subset for population prevalence estimates, effect-size estimation, or association testing. The exact selection is generated by `data-raw/create_agp_gut_subse` run-ID membership, selection reasons, and derived annotations are retained in `inst/extdata/agp_gut_meta.csv`.

The count matrix can be used directly with `filter.asv`, `normalize.linf`, and downstream dCST functions.

Note on Escherichia-Shigella: this genus is inflated in 16S V4 data due to primer cross-reactivity and should be interpreted with caution.

Source

Derived from the public American Gut Project records under ENA accession PRJEB11419 via the PRIME pipeline. See `data-raw/create_agp_gut_subset.py`, `data-raw/build_agp_gut.R`, and the installed `DATA_PROVENANCE.md` file.

References

McDonald, D., Hyde, E., Debelius, J. W., et al. (2018). American Gut: an Open Platform for Citizen Science Microbiome Research. *mSystems*, 3(3), e00031-18. doi:10.1128/mSystems.0003118

Examples

```
data(agp_gut)
dim(agp_gut$counts)                    # 766 x 314
table(agp_gut$meta$dcst_depth1)        # dCST distribution
sum(agp_gut$meta$IBS)                  # IBS cases
```

`asv.to.linf.csts` *From counts to filtered, normalized, truncated dCSTs*

Description

Convenience pipeline: runs `filter.asv()` on counts, computes L-infinity-normalized relatives on the filtered counts, and assigns truncated dCSTs.

Usage

```
asv.to.linf.csts(S.counts, ..., backend = c("auto", "dense", "sparse"))
```

Arguments

`S.counts`	Numeric matrix of counts (samples x features).
`...`	Arguments passed to `filter.asv()` (e.g., `min.lib`, `prev.prop`, `min.count`, `min.rel`).
`backend`	Character. Matrix backend to use: `"auto"`, `"dense"`, or `"sparse"`.

Value

A list with:

- `filter`: the full result from `filter.asv()`.
- `linf.rel`: L-infinity-normalized matrix on filtered counts.
- `csts`: result from `linf.csts()` on `linf.rel`.

Examples

```
set.seed(1)
S <- matrix(rpois(100, lambda = 5), nrow = 20, ncol = 5)
res <- asv.to.linf.csts(S, min.lib = 10, prev.prop = 0.1, min.count = 1)
names(res)
apply(res$linf.rel, 1, max)  # should be 1 (or 0 for all-zero rows)
```

dcst.view — *Select a stored dCST policy view*

Description

Returns a copy of a "linf.csts" object using either the stored "pure" or "absorb" hierarchy. This does not recompute dCSTs.

Usage

```
dcst.view(csts, view = c("absorb", "pure"))
```

Arguments

csts	A "linf.csts" object produced by linf.csts and optionally refined by refine.linf.csts / refine.linf.csts.iter.
view	Character. The stored policy view to activate: "pure" or "absorb".

Value

A "linf.csts" object using the requested view.

filter.asv — *Filter ASV count matrix by library size (samples) and prevalence (features)*

Description

Filters an ASV **count** matrix by: (1) removing samples with library size below min.lib; (2) keeping features (taxa) that are "present" in at least ceiling(prev.prop * nrow(S.counts)) samples, where presence is defined by either min.count (counts) or min.rel (relative). After feature filtering, zero-total samples are dropped and a row-normalized relative-abundance matrix is returned alongside the filtered counts.

Usage

```
filter.asv(
  S.counts,
  min.lib = 1000,
  prev.prop = 0.05,
  min.count = 2,
  min.rel = NULL,
  min.feat.total = NULL,
  verbose = TRUE
)
```

Arguments

`S.counts`	Numeric matrix (samples x features) of nonnegative counts.
`min.lib`	Integer. Minimum library size (row sum of counts) to keep a sample. Default: 1000.
`prev.prop`	Numeric in (0,1]. Minimum fraction of samples where a feature must be present. Default: 0.05.
`min.count`	Integer (>=1) or NULL. Reads to call "present" (ignored if `min.rel` set). Default: 2.
`min.rel`	Numeric in (0,1) or NULL. Relative abundance to call "present" (overrides `min.count`). Default: NULL.
`min.feat.total`	Integer (>=0) or NULL. Optional minimum total reads across all samples per feature. Default: NULL.
`verbose`	Logical. Print keep/drop summaries. Default: TRUE.

Details

Sample filtering is applied on raw counts first. Prevalence is computed on the post-sample-filter matrix using either a count or relative rule. A feature is retained if prevalence $\geq \lceil \text{prev.prop} \times n_{\text{samples}} \rceil$. After feature filtering, samples with zero remaining counts are dropped and the relative matrix `rel` is row-normalized.

Value

A list with elements:

counts Filtered count matrix (samples x features).

rel Row-normalized relative-abundance matrix.

kept.sample.idx Kept sample indices (original order).

kept.feature.idx Kept feature indices (original order).

prevalence Per-feature prevalence counts for kept features.

thresholds List of thresholds actually used.

Conventions

Dot-delimited names; rows are samples, columns are features. Supply raw counts.

Examples

```
set.seed(1)
S <- matrix(rpois(100 * 20, lambda = 5), nrow = 100, ncol = 20)
res <- filter.asv(S, min.lib = 50, prev.prop = 0.1, min.count = 2)
dim(res$counts); dim(res$rel)
range(rowSums(res$rel))
```

`latex.linf.csts` *Render dCSTs as a LaTeX table*

Description

Generates LaTeX code summarizing dCSTs at a specified hierarchy depth. Uses the explicit dCST hierarchy stored in `lineage.labels` when available.

Usage

```
latex.linf.csts(
  csts,
  depth = NULL,
  caption = NULL,
  label = NULL,
  digits = 1,
  include.percent = TRUE
)
```

Arguments

`csts`	A dCST object produced by `linf.csts()` and optionally refined by `refine.linf.csts()` or `refine.linf.csts.iter()`.
`depth`	Integer. dCST depth to render. Default is the leaf level (`csts$depth`). If `lineage.labels` is missing, depth is ignored.
`caption`	Character or NULL. LaTeX table caption.
`label`	Character or NULL. LaTeX label for referencing the table.
`digits`	Integer. Digits to display for percentages. Default: 1.
`include.percent`	Logical. If TRUE, include percent of total samples.

Details

This function is hierarchy-aware. It does not infer dCST depth from label strings. If the dCST object does not contain `lineage.labels`, the function falls back to using `lineage.label` as a single-level dCST.

Value

Character vector containing LaTeX table code.

`linf.csts` *Truncated dominant community state types with configurable low-support handling*

Description

Forms provisional depth-1 dominance sample sets from the dominant feature of each sample and then applies the minimum support threshold `n0`. Sets with fewer than `n0` samples are handled according to `low.freq.policy`:

- `"pure"`: retain only sets with support >= `n0` as named dCSTs and collapse all low-support sets into `rare.label`.
- `"absorb"`: reassign each low-support sample to the retained state with the largest value among the retained features (ties handled by `tie.method`).

Usage

```
linf.csts(
  S,
  feature.ids = NULL,
  feature.labels = NULL,
  n0 = 50,
  low.freq.policy = c("pure", "absorb"),
  rare.label = "RARE_DOMINANT",
  tie.method = c("first", "random", "error"),
  return.diagnostics = FALSE,
  return.landmarks = FALSE,
  landmark.types = c("endpoint.max", "endpoint.min"),
  landmark.view = c("active", "pure", "absorb"),
  backend = c("auto", "dense", "sparse")
)
```

Arguments

`S`	Numeric matrix (samples x features), typically L-infinity relatives.
`feature.ids`	Optional character vector of stable feature identifiers, length `ncol(S)`.
`feature.labels`	Optional character vector of display labels, length `ncol(S)`.
`n0`	Integer >= 1. Minimum support required to retain a dominance sample set.
`low.freq.policy`	Character. One of `"pure"` or `"absorb"`. Default: `"pure"`.
`rare.label`	Character scalar used when `low.freq.policy = "pure"`. Default: `"RARE_DOMINANT"`.
`tie.method`	Character. Tie handling passed to `linf.dominant.features()` and used during absorb reassignment ("first", "random", "error").
`return.diagnostics`	Logical. If TRUE, return reassignment diagnostics.

`return.landmarks`
: Logical. If TRUE, attach a depth-1 landmark summary computed by `linf.landmarks`.

`landmark.types`
: Character vector of landmark types passed to `linf.landmarks` when `return.landmarks = TRUE`.

`landmark.view`
: Character. Landmark view passed to `linf.landmarks` when `return.landmarks = TRUE`.

`backend`
: Character. Matrix backend to use: `"auto"`, `"dense"`, or `"sparse"`. The default `"auto"` preserves sparse input and otherwise uses the dense path.

Value

List with:

- `depth1.feature.index`, `depth1.feature.id`, `depth1.feature.label`: active depth-1 assignment
- `lineage.id`, `lineage.label`: active leaf-lineage assignment
- policy-specific variants of the depth-1 and leaf-lineage fields, ending in `.pure` or `.absorb`
- `lineage.ids`, `lineage.labels`: active hierarchy, plus policy-specific `.pure` and `.absorb` hierarchies
- `depth`: current hierarchy depth
- `retained.feature.indices`, `retained.feature.ids`, `retained.feature.labels`
- `provisional.feature.index`, `provisional.feature.id`, `provisional.feature.label`
- `feature.ids`, `feature.labels`
- `size.table`, `size.table.id`
- `n0`, `low.freq.policy`, `rare.label`
- `diagnostics` (if `return.diagnostics = TRUE`)
- `landmarks` (if `return.landmarks = TRUE`)

`linf.dcst.landmark.pipeline` *Landmark-aware dCST pipeline*

Description

Small wrapper that normalizes a nonnegative matrix with `normalize.linf`, computes depth-1 dCSTs with `linf.csts`, refines once to depth 2 with `refine.linf.csts`, and then computes landmark points for both depths with `linf.landmarks`.

The function is intentionally explicit rather than highly abstracted. It keeps the intermediate objects visible and returns them together in one list so downstream workflows can inspect the normalized matrix, the depth-1 dCSTs, the depth-2 dCSTs, and the landmark tables without re-running the pipeline.

Usage

```
linf.dcst.landmark.pipeline(
  X,
  feature.ids = NULL,
  feature.labels = NULL,
  n0.depth1 = 50,
  n0.depth2 = 25,
  refinement.factor = 2,
  sep = "__",
  low.freq.policy = c("pure", "absorb"),
  rare.label = "RARE_DOMINANT",
 depth1.landmark.types = c("endpoint.max", "endpoint.min", "mean.rep", "median.rep"),
 depth2.landmark.types = c("endpoint.max", "endpoint.min", "mean.rep", "median.rep"),
  landmark.view = c("absorb", "active", "pure"),
  tie.method = c("first", "random", "error"),
  verbose = FALSE,
  backend = c("auto", "dense", "sparse")
)
```

Arguments

X	Numeric matrix (samples x features). Must be finite and nonnegative.
feature.ids	Optional character vector of stable feature identifiers, length ncol(X).
feature.labels	Optional character vector of display labels, length ncol(X).
n0.depth1	Integer >= 1. Minimum support for a depth-1 dominance sample set.
n0.depth2	Integer >= 1. Minimum support for a depth-2 child lineage.
refinement.factor	Numeric > 0. Auto-refinement threshold multiplier passed to refine.linf.csts.
sep	Character scalar used to join depth-refined dCST path tokens.
low.freq.policy	Character. One of "pure" or "absorb". Controls the active dCST view while preserving both pure and absorb views inside the returned dCST objects.
rare.label	Character scalar for rare buckets.
depth1.landmark.types	Character vector of landmark types for depth 1.
depth2.landmark.types	Character vector of landmark types for depth 2.
landmark.view	Character. One of "absorb", "active", or "pure". Defaults to "absorb" because landmark reporting is typically requested on the absorb view.
tie.method	Character. Tie handling passed through to linf.csts and linf.landmarks.
verbose	Logical. Passed to refine.linf.csts.
backend	Character. Matrix backend to use: "auto", "dense", or "sparse".

Value

A named list with components:

- `linf.rel`: L-infinity-normalized matrix.
- `dcst.depth1`: depth-1 `"linf.csts"` object.
- `dcst.depth2`: depth-2 `"linf.csts"` object.
- `landmarks.depth1`: `"linf.landmarks"` object at depth 1.
- `landmarks.depth2`: `"linf.landmarks"` object at depth 2.
- `params`: compact record of the pipeline arguments used.

Examples

```
X <- rbind(
  s1 = c(10, 8, 1),
  s2 = c(9, 7, 2),
  s3 = c(8, 2, 7),
  s4 = c(7, 1, 8),
  s5 = c(1, 10, 2),
  s6 = c(2, 9, 1)
)
ids <- c("asv_1", "asv_2", "asv_3")
labels <- c("L. iners 1", "Gard. vaginalis 2", "BVAB1 3")

out <- linf.dcst.landmark.pipeline(
  X,
  feature.ids = ids,
  feature.labels = labels,
  n0.depth1 = 2,
  n0.depth2 = 2,
  refinement.factor = 2,
  low.freq.policy = "pure",
  landmark.view = "absorb",
  verbose = FALSE
)

names(out)
out$dcst.depth2$depth
out$landmarks.depth2$view
```

`linf.dominant.features`

Dominant-feature assignment

Description

Assigns each row to the column achieving its maximum.

For each sample (row) of a nonnegative matrix, identifies the dominant feature as the column with the maximum value. Samples with the same dominant feature form a depth-1 dominance sample set. Ties are broken by the first maximum (as in `max.col(..., ties.method = "first")`). Rows that are all zero are assigned `NA`.

Feature IDs default to `colnames(S)`; if absent, synthetic IDs `"V1"`, `"V2"`, `...`, `"Vp"` are generated. Display labels default to the feature IDs unless `feature.labels` is supplied. To guarantee a 1-1 mapping between columns and both IDs and labels, duplicates are disambiguated via `make.unique()`.

Usage

```
linf.dominant.features(
  S,
  feature.ids = NULL,
  feature.labels = NULL,
  tie.method = c("first", "random", "error"),
  return.value = FALSE,
  backend = c("auto", "dense", "sparse")
)
```

Arguments

`S`	Numeric matrix (samples x features), typically L-infinity-normalized.
`feature.ids`	Optional character vector of stable feature identifiers, length `ncol(S)`.
`feature.labels`	Optional character vector of display labels, length `ncol(S)`.
`tie.method`	Character. How to resolve ties during dominant-feature assignment.
`return.value`	Logical. If `TRUE`, include a `value` vector with row maxima.
`backend`	Character. Matrix backend to use: `"auto"`, `"dense"`, or `"sparse"`. The default `"auto"` preserves sparse input and otherwise uses the dense path.

Value

A list with components:

- `index`: integer index of the dominant column per sample (`NA` for all-zero rows)
- `id`: dominant feature ID per sample (`NA` for all-zero rows)
- `label`: dominant column label per sample (`NA` for all-zero rows)
- `id.levels`: full feature ID set after `make.unique(..., sep = "_")`
- `levels`: full column label set after `make.unique(..., sep = "_")`
- `observed.id.levels`: subset of `id.levels` that appear in `id`
- `observed.levels`: subset of `levels` that appear in `label`
- `value`: row maxima (only when `return.value = TRUE`)

See Also

normalize.linf, linf.csts

Examples

```
# Basic example with named columns
S <- rbind(
  a = c(A = 10, B = 5,  C = 0),   # -> A
  b = c(A = 0,  B = 0,  C = 0),   # -> NA
  c = c(A = 1,  B = 4,  C = 4)    # tie -> first max: B
)
out <- linf.dominant.features(S)
out$index
out$label
out$levels
out$observed.levels

# Unnamed columns (synthetic labels V1..Vp), duplicate names disambiguated
T <- matrix(c(0,2,  3,1,  0,0), nrow = 3, byrow = TRUE)
colnames(T) <- c("X", "X")  # duplicates -> X, X_1
linf.dominant.features(T)$levels

# With L-infinity normalization in a pipeline
M <- normalize.linf(S)
linf.dominant.features(M)$label
```

---

linf.feature.labels *Format feature labels for dominant-feature assignments and dCSTs*

Description

Builds unique display labels from stable feature IDs and taxonomy strings. This is useful when dCST computation should operate on stable internal feature identifiers (for example asv_4) while reports and figures should use human-readable labels such as L. iners 4.

Underscores in taxonomy strings are converted to spaces before abbreviation and aliasing. If multiple features share the same display taxon, an index can be appended either from the global feature ID (for example asv_4 -> 4) or by within-taxon order.

Usage

```
linf.feature.labels(
  feature.ids,
  taxonomy,
  abbreviations = NULL,
  aliases = NULL,
  duplicate.index = c("global", "within_taxon", "none"),
  fallback.to.id = TRUE
)
```

Arguments

feature.ids	Character vector of stable feature identifiers.
taxonomy	Character vector of taxonomy strings, same length as feature.ids.
abbreviations	Optional named character vector mapping genus tokens to display abbreviations, for example c(Lactobacillus = "L.").
aliases	Optional named character vector mapping full taxonomy strings to alternate labels, for example c("Ca. Lachnocurva vaginae" = "BVAB1").
duplicate.index	One of "global", "within_taxon", or "none". Controls how duplicate display taxa are disambiguated.
fallback.to.id	Logical. If TRUE, missing taxonomy values fall back to the feature ID.

Value

Character vector of unique display labels.

Examples

```
ids <- c("asv_1", "asv_4", "asv_5", "asv_6")
tax <- c(
  "Lactobacillus iners",
  "Lactobacillus iners",
  "Megasphaera lornae",
  "Ca_Lachnocurva_vaginae"
)
abbr <- c(
  Lactobacillus = "L.",
  Gardnerella = "Gard.",
  Megasphaera = "Mega."
)
aliases <- c(
  "Ca. Lachnocurva vaginae" = "BVAB1",
  "Ca_Lachnocurva_vaginae" = "BVAB1"
)
linf.feature.labels(ids, tax, abbreviations = abbr, aliases = aliases)
```

---

linf.hypercube.embedding

Extended homogeneous-coordinate hypercube embedding

Description

Computes the zero-aware hypercube embedding associated with one reference component of a non-negative compositional matrix. For rows with positive reference component, the function forms the ordinary homogeneous ratios against that reference and radially maps them into the unit cube. For rows whose reference component is zero, it uses the L-infinity boundary extension so that the embedding remains defined.

Usage

```
linf.hypercube.embedding(
  X,
  reference,
  lambda = NULL,
  sigma.quantile = 0.95,
  sigma.target = 0.95,
  feature.ids = NULL,
  feature.labels = NULL,
  tol = 0,
  backend = c("auto", "dense", "sparse")
)
```

Arguments

X	Nonnegative numeric matrix with samples in rows and features in columns.
reference	Reference component. May be a column index, feature ID, or feature label.
lambda	Positive numeric scalar. If NULL, choose a data-scaled value using sigma.quantile and sigma.target.
sigma.quantile	Quantile of positive finite-reference $\|z\|_1$ values used when lambda = NULL.
sigma.target	Target value of $\sigma_\lambda(t)$ at the selected quantile when lambda = NULL.
feature.ids	Optional stable feature identifiers, length ncol(X).
feature.labels	Optional display labels, length ncol(X).
tol	Nonnegative tolerance. Reference entries <= tol are treated as zero, and L-infinity norms <= tol are treated as zero.
backend	Matrix backend: "auto", "dense", or "sparse". Sparse inputs are accepted, but the returned embedding is a dense matrix because homogeneous-coordinate embeddings are generally dense.

Details

Let $x = (x_1, \ldots, x_p)$ be a nonnegative row and let k be the reference component. When $x_k > 0$, define $z = x_{-k}/x_k$. The embedded row is

$$\sigma_\lambda(\|z\|_1)\frac{z}{\|z\|_\infty}, \qquad \sigma_\lambda(t) = 1 - \exp(-\lambda t).$$

When $x_k = 0$, the embedded row is the L-infinity-normalized boundary vector

$$x_{-k}/\|x_{-k}\|_\infty.$$

All-zero rows are mapped to all-zero embedded rows by convention.

If lambda is not supplied, it is chosen from the positive finite-reference rows so that sigma.target is attained at the sigma.quantile quantile of $\|z\|_1$. This is a numerical scaling convention for finite datasets; it does not change the reference component or the boundary extension rule.

Value

A numeric matrix with `nrow(X)` rows and `ncol(X) - 1` columns. The columns correspond to the non-reference components. Attributes record the reference component, lambda choice, and finite/boundary row counts.

Examples

```
X <- rbind(
  c(A = 2, B = 1, C = 1),
  c(A = 0, B = 2, C = 1)
)
linf.hypercube.embedding(X, reference = "A", lambda = log(2))
```

`linf.landmarks` *Landmark points for dCST dominance-lineages*

Description

Computes representative landmark points for the dominance-lineages of a `"linf.csts"` object at a chosen depth and view.

Landmark types are defined with respect to the leaf feature of the dCST path: the last feature ID in the lineage ID path. Lineages whose leaf token is `rare.label` are reported but skipped for landmark computation because they do not correspond to a unique target feature.

Usage

```
linf.landmarks(
  M,
  csts,
  depth = NULL,
  view = c("active", "pure", "absorb"),
  landmark.types = c("endpoint.max", "endpoint.min", "mean.rep", "median.rep"),
  tie.method = c("first", "random", "error"),
  backend = c("auto", "dense", "sparse")
)
```

Arguments

`M`	Numeric matrix (samples x features) used to build or refine the dCSTs.
`csts`	A `"linf.csts"` object.
`depth`	Integer. dCST depth to inspect. Defaults to the leaf depth `csts$depth`.
`view`	Character. One of `"active"`, `"pure"`, or `"absorb"`.
`landmark.types`	Character vector containing any of `"endpoint.max"`, `"endpoint.min"`, `"mean.rep"`, or `"median.rep"`.

tie.method	Character. Tie handling for landmark selection: "first", "random", or "error".
backend	Character. Matrix backend to use: "auto", "dense", or "sparse". The default "auto" preserves sparse input and otherwise uses the dense path.

Value

A list of class "linf.landmarks" with components:

- depth, view, sep, rare.label
- feature.ids, feature.labels
- lineages: one row per dominance-lineage with computability metadata
- landmarks: one row per computed landmark point

---

normalize.linf *L-infinity normalization (row-wise)*

Description

Scales each row of a numeric matrix by its L-infinity norm (row maximum). Rows whose maximum is zero (or below tolerance) are left unchanged and remain all-zero.

Usage

```
normalize.linf(X, tol = 0, backend = c("auto", "dense", "sparse"))
```

Arguments

X	Numeric matrix (samples x features).
tol	Numeric >= 0. Values with row max <= tol are treated as zero rows. Default: 0 (exact zero only).
backend	Character. Matrix backend to use: "auto", "dense", or "sparse". The default "auto" preserves sparse input and otherwise uses the dense path.

Details

Zero rows have undefined L-infinity direction. By convention, they are preserved as all-zero rows and will yield NA labels in downstream dominant-feature or dCST assignment.

Value

Numeric matrix of same dimensions as X, L-infinity normalized.

print.linf.csts *Print method for "linf.csts"*

Description

Print method for "linf.csts"

Usage

```
## S3 method for class 'linf.csts'
print(x, ...)
```

Arguments

x	A "linf.csts" object.
...	Unused.

Value

The input object, invisibly.

refine.linf.csts *Refine a dCST hierarchy by one level*

Description

Selects well-supported leaf dominance-lineages and refines them by dropping the dominant feature(s) encoded in the parent label path and re-applying linf.csts to the remaining features. The resulting child labels are appended to the parent label using sep.

Low-support child lineages are handled by low.freq.policy. When low.freq.policy = "pure", rare buckets at depth >= 2 become parent-prefixed automatically via the hierarchical paste(parent, child, sep = sep).

Usage

```
refine.linf.csts(
  M,
  csts,
  n0 = 50,
  refinement.factor = 2,
  sep = "__",
  low.freq.policy = c("pure", "absorb"),
  rare.label = "RARE_DOMINANT",
  verbose = TRUE,
  backend = c("auto", "dense", "sparse")
)
```

Arguments

M	Numeric matrix (samples x features) used for refinement. Column names should match feature labels used in dCST names.
csts	A "linf.csts" object.
n0	Integer >= 1. Minimum support required to retain a child lineage (passed to linf.csts).
refinement.factor	Numeric > 0. Auto-refine parent lineages with support >= refinement.factor * n0.
sep	Character scalar used to concatenate hierarchical labels.
low.freq.policy	Character. One of "pure" or "absorb". Default: "pure".
rare.label	Character scalar for rare buckets when low.freq.policy = "pure".
verbose	Logical. If TRUE, print progress information.
backend	Character. Matrix backend to use: "auto", "dense", or "sparse". The default "auto" inherits the backend from M or from csts when available.

Value

Updated "linf.csts" object with depth increased by one and updated lineage.label. Policy-specific views are stored in lineage.label.pure and lineage.label.absorb.

---

refine.linf.csts.iter *Iteratively refine dCSTs by one additional level*

Description

Appends one refinement level to an existing dCST hierarchy produced by refine.linf.csts or refine.linf.csts.iter. Dominance-lineages to refine can be provided explicitly or selected automatically based on refinement.factor * n0.

Usage

```
refine.linf.csts.iter(
  M,
  refined,
  lineages.to.refine = NULL,
  n0 = 50,
  refinement.factor = 5,
  sep = "__",
  low.freq.policy = c("pure", "absorb"),
  rare.label = "RARE_DOMINANT",
  verbose = TRUE,
  backend = c("auto", "dense", "sparse")
)
```

Arguments

M	Numeric matrix used for refinement.
refined	A "linf.csts" object with an existing hierarchy.
lineages.to.refine	Character vector of leaf dominance-lineage IDs to refine, or NULL for auto-selection.
n0	Integer >= 1. Minimum support required to retain a child lineage.
refinement.factor	Numeric > 0. Auto-selection threshold multiplier.
sep	Character scalar used to concatenate hierarchical labels.
low.freq.policy	Character. One of "pure" or "absorb". Default: "pure".
rare.label	Character scalar for rare buckets when low.freq.policy = "pure".
verbose	Logical. If TRUE, print progress information.
backend	Character. Matrix backend to use: "auto", "dense", or "sparse". The default "auto" inherits the backend from M or from refined when available.

Value

Updated "linf.csts" object with depth increased by one.

---

summary.linf.csts *Summarize dCST hierarchy statistics*

Description

Summarize dCST hierarchy statistics

Usage

```
## S3 method for class 'linf.csts'
summary(object, ...)
```

Arguments

object	A "linf.csts" object.
...	Unused.

Value

Data frame with depth-wise statistics

transfer.dcsts *Transfer samples into a frozen dCST hierarchy*

Description

Assigns rows of a new sample-by-feature matrix into the realized lineage sets of a fitted "linf.csts" hierarchy. The hierarchy is not refit: each sample is walked through the frozen tree by choosing, at each depth, the realized child lineage whose newly added feature has the largest abundance in that sample.

Usage

```
transfer.dcsts(
  X,
  csts,
  depth = NULL,
  view = c("absorb", "active", "pure"),
  match.by = c("feature.ids", "feature.labels"),
  feature.ids = NULL,
  feature.labels = NULL,
  tie.method = c("support", "first", "random", "error"),
  carry.forward.terminal.depths = TRUE,
  sep = NULL,
  backend = c("auto", "dense", "sparse")
)
```

Arguments

X	Nonnegative sample-by-feature matrix to transfer. Columns may be in any order and may include features not present in csts; missing retained features are treated as zero.
csts	A fitted "linf.csts" object produced by linf.csts and optionally refined by refine.linf.csts or refine.linf.csts.iter.
depth	Integer vector of requested depths. Defaults to all fitted depths in csts.
view	Which fitted hierarchy view to transfer into. "absorb" uses lineage.labels.absorb; "pure" uses lineage.labels.pure; "active" uses lineage.labels.
match.by	Whether columns in X are aligned to csts$feature.ids or csts$feature.labels.
feature.ids	Optional feature identifiers for columns of X. Defaults to colnames(X).
feature.labels	Optional feature labels for columns of X. Defaults to feature.ids.
tie.method	How to resolve ties among frozen child lineages with equal sample abundance. "support" chooses the tied lineage with largest reference support and then lexical order; "first" uses frozen child order; "random" samples one tied lineage; "error" stops.

`carry.forward.terminal.depths`
: Logical. If `TRUE`, an unmatched depth-1 sample may fall back to its dominant retained feature. Terminal lineages at later depths are assigned only when they appear as realized stable lineage sets in the fitted hierarchy.

`sep`
: Separator used in lineage labels. Defaults to `csts$sep` or `"__"`.

`backend`
: Matrix backend; passed to the internal matrix preparer.

Value

A list of class `"linf.dcst.transfer"` with components:

- `assignment`: character matrix of transferred labels for the requested depths.
- `all.depths`: character matrix for all fitted depths.
- `depth`: requested depth vector.
- `view`, `match.by`, `tie.method`: settings used.

Examples

```
X <- rbind(
  s1 = c(A = 10, B = 2, C = 1),
  s2 = c(A = 9, B = 3, C = 1),
  s3 = c(A = 1, B = 10, C = 2),
  s4 = c(A = 1, B = 9, C = 3)
)
M <- normalize.linf(X)
fit <- linf.csts(M, n0 = 2, low.freq.policy = "absorb")
transfer.dcsts(X, fit)$assignment
```

---

valencia13k_dcst_depth2_merged

Valencia 13k merged depth-2 dCST assignments

Description

A lightweight bundled assignment asset containing merged depth-2 Dominant community state type (dCST) labels for the full Valencia 13k vaginal microbiome training set.

Usage

```
valencia13k_dcst_depth2_merged
```

Format

A list with five components:

assignments Data frame with one row per source sample and columns `sample_id` (anonymized bundled-data identifier), `source_row` (row number in the filtered Valencia 13k source), `Val_CST`, `Val_subCST`, `dcst_depth1`, and `dcst_depth2`.

summaries Named list of per-depth summary tables. Each table contains `depth`, `dcst_label`, `n`, `prop`, and `path_length`.

feature_labels Character vector of source taxon labels.

params List recording the construction parameters.

source Character string documenting provenance.

Details

The asset is computed from the Valencia 13k compositional matrix after L-infinity normalization. dCSTs use `n0 = 50` and the merged `low.freq.policy = "absorb"` view, so samples from low-support provisional states are reassigned to retained states rather than stored as explicit rare buckets.

Source

Generated from the VALENCIA training data at https://github.com/ravel-lab/VALENCIA. See `data-raw/build_valencia13k_merged_dcst_depths.R` and the installed `DATA_PROVENANCE.md` file.

Examples

```
data(valencia13k_dcst_depth2_merged)
nrow(valencia13k_dcst_depth2_merged$assignments)
head(valencia13k_dcst_depth2_merged$summaries$depth2)
```

---

valencia13k_dcst_depth3_merged

Valencia 13k merged depth-3 dCST assignments

Description

A lightweight bundled assignment asset containing merged depth-3 Dominant community state type (dCST) labels for the full Valencia 13k vaginal microbiome training set.

Usage

```
valencia13k_dcst_depth3_merged
```

Format

A list with five components:

assignments Data frame with one row per source sample and columns `sample_id` (anonymized bundled-data identifier), `source_row` (row number in the filtered Valencia 13k source), `Val_CST`, `Val_subCST`, `dcst_depth1`, `dcst_depth2`, and `dcst_depth3`.

summaries Named list of per-depth summary tables. Each table contains `depth`, `dcst_label`, `n`, `prop`, and `path_length`.

feature_labels Character vector of source taxon labels.

params List recording the construction parameters.

source Character string documenting provenance.

Details

The depth-3 asset extends `valencia13k_dcst_depth2_merged` by one additional hierarchical dCST refinement. All levels use `n0 = 50` and `low.freq.policy = "absorb"`. This object is intended as a reusable source for selecting richer VALENCIA-derived component sets without recomputing the full hierarchy from the 13k source matrix.

Source

Generated from the VALENCIA training data at https://github.com/ravel-lab/VALENCIA. See `data-raw/build_valencia13k_merged_dcst_depths.R` and the installed `DATA_PROVENANCE.md` file.

Examples

```
data(valencia13k_dcst_depth3_merged)
nrow(valencia13k_dcst_depth3_merged$assignments)
head(valencia13k_dcst_depth3_merged$summaries$depth3)
```

`valencia2k` *Valencia 2k vaginal microbiome dataset*

Description

A stratified subsample of 2,000 vaginal samples from the Valencia 13k CST-classifier training set (France et al. 2020), bundled as an example dataset for demonstrating dCST construction after L-infinity normalization.

Usage

```
valencia2k
```

Format

A list with five components:

rel Numeric matrix (2000 x 178). Compositional relative abundances; each row sums to 1. Rows are samples, columns are taxonomic features.

cst Data frame (2000 x 3) with columns: `sample_id` (character), `Val_CST` (Valencia CST assignment: I, II, III, IV-A, IV-B, IV-C, V), `Val_subCST` (Valencia sub-CST assignment: I-A, I-B, II, III-A, III-B, IV-A, IV-B, IV-C0, IV-C1, IV-C2, IV-C3, IV-C4, V).

reads Integer vector of length 2000. Per-sample read counts after taxonomic filtering. Use `sweep(valencia2k$rel, 1, valencia2k$reads, "*")` to reconstruct a count-like matrix.

taxa Character vector of 178 taxon names (column names of `rel`).

source Character string documenting provenance.

Details

The subsample preserves proportional representation of all 13 Valencia sub-CSTs and was drawn with `set.seed(42)`.

The Valencia CST classifier (France et al. 2020) assigns vaginal microbiome samples to community state types (CSTs) based on nearest-centroid classification in relative-abundance space. The original training set contains 12,881 samples and 178 taxonomic features after filtering.

This 2,000-sample subsample is intended for vignette demonstrations. The compositional matrix can be used directly with `normalize.linf` and downstream dCST functions. For workflows that require count-like input, reconstruct approximate counts using the `reads` vector.

Source

Subsampled from the VALENCIA training data at https://github.com/ravel-lab/VALENCIA. See `data-raw/build_valencia2k.R` and the installed `DATA_PROVENANCE.md` file for construction and licensing details.

References

France, M. T., Ma, B., Gajer, P., Brown, S., Humphrys, M. S., Holm, J. B., Waetjen, L. E., Brotman, R. M., & Ravel, J. (2020). VALENCIA: a nearest centroid classification method for vaginal microbial communities based on composition. *Microbiome*, 8(1), 166. doi:10.1186/s40168020009346

Examples

```
data(valencia2k)
dim(valencia2k$rel)           # 2000 x 178
table(valencia2k$cst$Val_CST) # CST distribution
head(valencia2k$taxa, 10)     # first 10 taxon names
```

`valencia_linf_hypercube_1k` *Valencia 1k four-component hypercube embedding example*

Description

A stratified 1,000-sample subset of the Valencia 13k vaginal microbiome training set, reduced to four selected phylotype coordinates and L1-normalized over those coordinates. The object is bundled as a lightweight example for the zero-aware homogeneous-coordinate hypercube embedding in `linf.hypercube.embedding`.

Usage

```
valencia_linf_hypercube_1k
```

Format

A list with four components:

rel4 Numeric matrix (1000 x 4). Rows are samples and columns are `Li`, `Lc`, `Gv`, and `Bv`. Each row sums to 1 after restricting the original Valencia profile to the four mapped taxa.

meta Data frame (1000 rows) with columns: `sample_id` (anonymized bundled-data identifier), `source_row` (row number in the filtered Valencia 13k source), `Val_CST`, `Val_subCST`, `selected_mass` (original relative-abundance mass carried by the four selected taxa), and `dominant_component` (largest of `Li`, `Lc`, `Gv`, `Bv` after renormalization).

component_map Named character vector mapping `Li`, `Lc`, `Gv`, and `Bv` to the original Valencia taxon names: `Lactobacillus_iners`, `Lactobacillus_crispatus`, `Gardnerella_vaginalis`, and `BVAB1`.

source Character string documenting provenance.

Details

Rows with zero total mass in the four selected taxa are removed before renormalization. Sampling is stratified by the dominant selected component, using `set.seed(20261604)`. The object is not intended to replace the full Valencia matrix; it is a compact reproducible example for visualizing compositional projective-space coordinate charts.

Source

Generated from the VALENCIA training data at https://github.com/ravel-lab/VALENCIA. See `data-raw/build_valencia_linf_hypercube_1k.R` and the installed `DATA_PROVENANCE.md` file.

Examples

```
data(valencia_linf_hypercube_1k)
dim(valencia_linf_hypercube_1k$rel4)
table(valencia_linf_hypercube_1k$meta$dominant_component)
emb <- linf.hypercube.embedding(
  valencia_linf_hypercube_1k$rel4,
  reference = "Li"
)
dim(emb)
```

Index