Package 'linf'

November 7, 2026

Type Package

Title L-Infinity Normalization and Dominant Community State Types

Version 0.4.0

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, constructs unordered dominance sets with support-based sample filtering, and computes representative landmark profiles. The methods are described by Gajer and Ravel (2025) <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://pgajer.github.io/linf/, 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

Config/Needs/website pgajer/ivue, rgl

NeedsCompilation no

Repository CRAN

Date/Publication 2026-11-07 19:10:02 UTC

Contents

linf-package . . . 2
agp_gut . . . 3
dcst.view . . . 5
filter.asv . . . 6
linf.csts . . . 7
linf.dcst.landmark.pipeline . . . 10
linf.dominant.features . . . 12
linf.feature.labels . . . 14
linf.hypercube.embedding . . . 15
linf.landmarks . . . 17
linf.udcst.graph . . . 18
linf.udcsts . . . 20
normalize.linf . . . 23
print.linf.csts . . . 24
refine.linf.csts . . . 24
summary.linf.csts . . . 26
transfer.dcsts . . . 27
valencia13k_dcst_depth2_merged . . . 29
valencia13k_dcst_depth3_merged . . . 30
valencia2k . . . 32
valencia_linf_hypercube_1k . . . 33

Index **35**

`linf-package` *L-Infinity Normalization and Dominant Community State Types*

Description

Work with nonnegative samples-by-features matrices: filter counts, normalize by each row's maximum, fit supported dominant-feature groups, refine a hierarchy, and transfer new samples to a frozen reference.

Start with your task

Prepare `filter.asv` filters counts; `normalize.linf` scales rows above tolerance.

Assign and fit `linf.dominant.features` identifies individual maxima; `linf.csts` applies a support threshold.

Ignore dominance order `linf.udcsts` pools top-feature sets before retaining, absorbing, or filtering low-support samples.

Refine and inspect `refine.linf.csts` adds a level; `dcst.view` activates a stored policy. Use `print()` and `summary()` on fitted objects and `linf.landmarks` to select observed representatives.

Reuse `transfer.dcsts` assigns new samples without learning new states.

Display `linf.feature.labels` formats taxonomy; `linf.hypercube.embedding` prepares coordinate charts.

`linf.dcst.landmark.pipeline` combines normalization, depth-1 fitting, one refinement and landmark selection when all stages are wanted.

Guides and datasets

Run `vignette("function-guide", package = "linf")` for the complete function catalog and `vignette("example-datasets", package = "linf")` for data selection, alignment and reference/query examples. Longer installed workflows are `vignette("linf-intro", package = "linf")` (gut) and `vignette("linf-vaginal", package = "linf")` (vaginal).

`agp_gut` supplies counts; `valencia2k` supplies relative abundances. `valencia13k_dcst_depth2_merged` and `valencia13k_dcst_depth3_merged` are saved assignment tables, not fitted hierarchy objects. `valencia_linf_hypercube_1k` supplies four-component input for embedding, not precomputed coordinates. The interactive embedding article on the package website is website-only.

Author(s)

Maintainer: Pawel Gajer <pgajer@gmail.com>

Authors:

- Pawel Gajer <pgajer@gmail.com>

See Also

`browseVignettes("linf")`

`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_subset.py; 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.

The Escherichia-Shigella values are processed 16S read counts. McDonald et al. (2018), cited below, describe microbial overgrowth during room-temperature shipping in American Gut, including E. coli blooms. The bundled subset does not identify the cause of its Escherichia-Shigella abundance or establish primer cross-reactivity. Do not infer a biological abundance or a particular technical cause from the displayed dominance count alone.

Reading provenance offline

Read the installed source and preparation record at the console: `cat(readLines(system.file("DATA_PROVENANCE.md", package = "linf", mustWork = TRUE)), sep = "\n")`. The installed `DATA_MANIFEST.csv` records pinned input checksums; see the reproduction recipe.

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
```

`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 repeated calls to `refine.linf.csts`.
`view`	Character. The stored policy view to activate: `"pure"` or `"absorb"`.

Value

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

Examples

```
M <- rbind(
  s1 = c(A = 1.0, B = 0.2, C = 0.1),
  s2 = c(A = 0.9, B = 0.3, C = 0.1),
  s3 = c(A = 0.2, B = 1.0, C = 0.1),
  s4 = c(A = 0.2, B = 0.1, C = 1.0)
)
fit <- linf.csts(M, n0 = 2, low.freq.policy = "pure")
table(fit$lineage.label)
table(dcst.view(fit, view = "absorb")$lineage.label)
```

`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. Fractional count-like values are accepted, but entries must be finite and nonnegative. Thresholds are validated even when overridden by another rule.

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 >= 0. 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. Zero-total rows have no present features under a relative threshold, and are removed at the final sample-filtering stage. They still count in the prevalence denominator if retained by `min.lib = 0`.

Empty results retain all documented fields. If no samples survive the initial filter, no features are retained and the returned matrices are 0 by 0. Otherwise, features retained by prevalence can remain as columns of a zero-row result; the returned indices always describe the returned matrices.

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))
```

---

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"). Positive ties use original matrix column order for `"first"` on both backends. If all retained values are zero, absorption uses the retained state with greatest reference support (then lexical label order), regardless of `tie.method`.
`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.

Details

New fits retain the original input dimnames in `input.dimnames`, separately from custom feature IDs. Refinement and landmarks require the same dimensions and reject mismatching names/order

when both the fitted and supplied axis are named. They never reorder automatically. Unnamed axes and older fits lacking this metadata retain positional matching; names alone cannot detect changed values or permutations within duplicated names.

The `history` list records settings for each stored depth. The first entry records fitting support, policy and ties. Refinement entries additionally record selection mode, selected lineage/node IDs, nodes actually refined, source policy, separator and refinement factor. `available = FALSE` marks older levels whose history was not saved when a legacy fit is refined. The top-level `n0` remains the original depth-1 threshold.

Fitted objects store explicit feature paths and node metadata in `nodes` and `node.ids`, with `.pure` and `.absorb` variants. A node path contains original column indices; zero marks a synthetic rare component. Node keys are local to the fitted feature mapping. Metadata records parent keys, leaf-feature index, rare status and whether the node was carried unchanged from the preceding stored level.

Readable lineage IDs and labels are preserved when unambiguous. If different paths produce the same string (for example, feature `A__B` and path `A` then `B`), colliding strings receive a `[node ...]` suffix. Use the returned IDs for explicit refinement; never split them to recover features. Real features named like `rare.label` remain distinct from synthetic rare groups. Consumers upgrade unambiguous older objects on use and reject ambiguous legacy hierarchies with a rebuild message.

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`)

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),
```

```
  s5 = c(A = 1, B = 2, C = 10)
)
fit <- linf.csts(normalize.linf(X), n0 = 2, low.freq.policy = "pure")
table(fit$lineage.label)
fit$retained.feature.ids
```

`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 nonnegative 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"),
  log.lambda = NULL
)
```

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 for automatic scaling.
`sigma.target`	Target value of $\sigma_\lambda(t)$ at the selected quantile during automatic scaling.
`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 this implementation returns a dense matrix. Non-reference zeros remain zero; account for dense storage in large inputs.
`log.lambda`	Optional finite natural logarithm of the radial scale, as an alternative to `lambda`. Supply at most one of these two arguments.

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 neither `lambda` nor `log.lambda` is supplied, the scale is chosen from positive-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. Direction is computed from scaled non-reference values, and radial products and the type-7 quantile are evaluated using logarithms to avoid overflow. The radial map uses `expm1` to preserve very small positive coordinates. Norms beyond the ordinary numeric range are included in automatic calibration.

The `log.lambda` attribute always records the natural logarithm of the scale. The `lambda` attribute is `NA_real_` if that scale cannot be represented as a positive finite number. Reuse any fitted scale with `log.lambda = attr(reference.embedding, "log.lambda")`; this also works for extreme calibration data. The all-zero/reference-only fallback scale is one.

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 observed feature in the stored node path. Synthetic rare nodes 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, in the same row and column order; no name-based alignment is performed.
`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.

Details

All four methods select observed rows, not averaged profiles. Endpoints maximize/minimize the leaf-feature value; mean and median representatives minimize absolute deviation from that feature's mean or median. Values use the scale of `M`; L-infinity-normalized values are relative to the row maximum, not fractions of total abundance. Depth-1 pure dominant values can all equal one, leaving tied selections.

Target features come from explicit node paths, preserving separators inside IDs and separating literal feature names from concatenated paths. Terminal lineages retain their last feature at later stored

depths. Inspect `lineages$landmarks.computable` to identify synthetic rare categories without a target feature.

Both matrix dimensions must match the fit. For new fits, named axes must match the original input names/order saved in `csts$input.dimnames`, which are independent of custom feature IDs. No automatic alignment is performed. Unnamed axes and legacy fits without original dimnames use positional order.

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

Examples

```
M <- rbind(
  s1 = c(A = 1.0, B = 0.2),
  s2 = c(A = 0.9, B = 0.4),
  s3 = c(A = 0.3, B = 1.0),
  s4 = c(A = 0.1, B = 0.9)
)
fit <- linf.csts(M, n0 = 2, low.freq.policy = "absorb")
landmarks <- linf.landmarks(
  M,
  fit,
  landmark.types = c("endpoint.max", "mean.rep")
)
landmarks$landmarks
```

`linf.udcst.graph` *Triple-Supported Graph of Unordered Pair States*

Description

Construct a graph whose vertices are pure level-two unordered dominance states. Edges join overlapping pairs, optionally requiring both pair configurations to occur within their unique common leading triple.

Usage

```
linf.udcst.graph(
  X = NULL,
  pair.fit = NULL,
  triple.fit = NULL,
  sample.ids = NULL,
  coverage = NULL,
  n0 = 1L,
  states = NULL,
  adjacency = c("observed_triple", "shared_feature"),
  min.support = 1L,
  feature.ids = NULL,
  feature.labels = NULL,
  backend = c("auto", "dense", "sparse")
)
```

Arguments

`X`	Samples-by-features input accepted by `linf.udcsts()`, with at least three features. Supply either `X` or both fitted assignments below.
`pair.fit`, `triple.fit`	Results of `linf.udcsts()` at levels two and three, respectively, fitted to the same rows and dictionary. Pure IDs are used regardless of the fitted rare-state policy. Row identities, ordered ranks and dictionary agreement are checked. Anonymous fitted rows are matched by position; supply stable row names when fitting.
`sample.ids`	Unique, nonempty sample IDs in input order. Defaults to fitted row names, or character row numbers for unnamed inputs.
`coverage`	Optional target percentage in `(0, 100]`. Retain complete frequency tiers until at least this coverage is reached. Mutually exclusive with `states` and a nondefault `n0`.
`n0`	Minimum pair-state frequency, default one. Applied after counts on all input rows have been computed.
`states`	Optional character vector of observed pair IDs to retain. `character(0)` selects an empty graph. Mutually exclusive with `coverage` and a nondefault `n0`.
`adjacency`	`"observed_triple"` (default) or `"shared_feature"`.
`min.support`	Positive integer minimum incidence on each side of an observed-triple edge. Also sets the threshold for witnessed faces.
`feature.ids`, `feature.labels`, `backend`	Passed to `linf.udcsts()` when `X` is supplied. For existing fits, their feature metadata are used.

Details

For pairs A and B sharing exactly one feature, the unique witness is T = union(A, B). An observed-triple edge requires both incidences I(A,T) and I(B,T) to be at least `min.support`. All marginals

and similarities refer to the full input sample set, including excluded pair states. Selecting vertices never renormalizes these counts.

All graph edges have `length = 1`. Jaccard and support columns are evidence/display attributes, not path lengths. A witnessed face requires all three pairs of one triple to be selected and pass the incidence threshold. A triangle with three different witnesses is not a face. Isolated vertices and disconnected components are preserved. No MST repair or embedding is performed. Pure IDs use column indices and must travel with the returned feature dictionary; realign columns to that dictionary before fitting changed input column orders.

Value

A list of class `linf_udcst_graph` with `nodes` (pair ID, label, frequency, feature indices, component), `edges` (admitted edges), `candidate.edges` (all shared-feature edges on retained nodes), `faces`, full nonzero `incidence`, `pair.frequencies`, `triple.frequencies`, `membership` (sample ID and pure pair/triple IDs), `dictionary`, and `metadata`. Edges use integer `from`/`to` indices into `nodes` and retain pair IDs, witness ID, full marginal/incidence counts, minimum support, endpoint Jaccards, their mean/minimum/geometric mean, pair-conditional fractions and triple-allocation fractions. Faces use node indices `a`, `b`, `c` and retain their witness and minimum incidence. Metadata records settings, sample coverage, component sizes and isolate count.

Examples

```
X <- rbind(c(A=6, B=3, C=1), c(A=6, B=1, C=3), c(A=1, B=6, C=3))
g <- linf.udcst.graph(X, coverage=60)
g$nodes
g$edges
g$faces
```

---

linf.udcsts *Unordered Dominance Community State Types*

Description

Group samples by their top-feature sets, ignoring dominance order, and then apply a minimum support threshold to the pooled sets.

Usage

```
linf.udcsts(
  X,
  level = 2L,
  n0 = 50L,
  low.freq.policy = c("pure", "absorb", "filter"),
  feature.ids = NULL,
  feature.labels = NULL,
  backend = c("auto", "dense", "sparse")
)
```

Arguments

X	Nonnegative numeric samples-by-features matrix or numeric data frame. Numeric dense and sparse Matrix objects are also supported. Every row must contain at least one positive value.
level	Integer between 1 and ncol(X). Number of features in each set.
n0	Integer >= 1. Minimum pure set frequency, evaluated after pooling all dominance orders belonging to the same set.
low.freq.policy	One of "pure", "absorb", or "filter". "pure" marks low-support samples as rare without reassigning them; "filter" excludes them through the returned keep mask; "absorb" assigns them to retained sets using the rule below.
feature.ids, feature.labels	Optional stable feature IDs and display labels, respectively. Each has length ncol(X). Defaults and name disambiguation follow linf.csts().
backend	One of "auto", "dense", or "sparse". Automatic mode preserves sparse inputs. Sparse mode never densifies the whole matrix; absorption materializes only the current sample row.

Details

For each row, decreasing abundance and then original column order determine the ordered top level features. Sorting their column indices gives the unordered set. Set IDs join these indices with commas, independently of feature labels, so IDs are local to the fitted feature dictionary. Exact ties, including zero-valued ranks, always use original column order. Rows with fewer than level positive entries still receive a complete set; positive.features identifies these rows. All-zero rows are rejected.

The pure top-feature sets are formed on all input rows before thresholding. For example, three (A, B) rows and three (B, A) rows form one set of size six even when neither ordered group meets n0 = 5. At level 1, pure set membership agrees with dominant-feature grouping under first-column ties. Across levels the unordered partitions need not form a refinement hierarchy. The result is not a linf.csts hierarchy and cannot be passed to its refinement, policy-view, landmark, or frozen-transfer functions.

Under "pure", low-support samples have assignment = NA and status = "rare"; they remain included in keep. Under "filter", they instead have status = "filtered" and keep = FALSE. Neither policy alters pure.id, support, or the input matrix. An empty retained family is valid for both policies. The mask selects observations, not features: use X[fit$keep, , drop = FALSE] to obtain the filtered matrix.

Under "absorb", each low-support row goes to the retained set maximizing the sum of its feature abundances. Scores are evaluated after dividing the row by its positive maximum to avoid overflow. This is equivalent to maximizing the unscaled sum in exact arithmetic. Ties use lexicographic order of the sets' sorted numerical column-index tuples, including when all candidate scores are zero. If no set survives, absorption is undefined and raises an error. This symmetric set-based rule is distinct from parent-local ordered dCST absorption, and can change under nonlinear rank-preserving transformations even though the pure labels do not.

Frequencies always refer to the pure sets before policy application. For a set with frequency m, `cumm.pct` is

$$100 \sum_{B:m_B \geq m} m_B/N,$$

where m_B is another pure set's frequency and N is the full input row count. Equal-frequency rows have equal cumulative percentages.

Value

A list with:

pure.id Original pure set ID for every input row.

assignment Active set ID, or `NA` for rare/filtered rows.

status Per-row `"retained"`, `"rare"`, `"filtered"`, or `"absorbed"`. Retained means the row's original pure set meets `n0`.

keep Logical mask in input row order. All `TRUE` except for excluded rows under `"filter"`.

support Table of all pure set frequencies, including small sets.

frequency.table Three-column data frame: `ID`, `freq`, and `cumm.pct`. Sorted by decreasing frequency, then lexical set ID.

retained.ids Set IDs meeting `n0`, in numerical tuple tie order.

ordered.features,feature.sets Integer matrices in input row order, with `level` columns: dominance-ordered and sorted feature indices.

positive.features Number of positive features in each input row.

feature.ids,feature.labels Resolved column identities and labels. Index these with `feature.sets` to obtain readable set members.

input.dimnames Original row and column names, for alignment.

level,n0,low.freq.policy,matrix.backend Construction settings.

See Also

`linf.csts()`, `linf.dominant.features()`

Examples

```
X <- rbind(c(A = 9, B = 5, C = 1), c(A = 8, B = 4, C = 1),
           c(A = 4, B = 9, C = 1), c(A = 3, B = 8, C = 1),
           c(A = 1, B = 3, C = 9))
fit <- linf.udcsts(X, level = 2, n0 = 3, low.freq.policy = "filter")
fit$frequency.table
X[fit$keep, , drop = FALSE]
fit$feature.labels[fit$feature.sets[1, ]]
```

`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 at or below tolerance are left unchanged; nonzero entries in these rows are not replaced by zeros.

Usage

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

Arguments

`X`	Numeric matrix (samples x features).
`tol`	Finite numeric >= 0. Rows with maximum <= tol are not scaled. 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 yield `NA` in dominant-feature assignment. The pure dCST view places them in the rare category; the absorb view assigns them to its fallback retained state, if any. A nonzero row left unscaled because of `tol` still has a dominant feature.

Value

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

Examples

```
X <- rbind(
  sample1 = c(A = 2, B = 1, C = 0),
  sample2 = c(A = 0, B = 0, C = 0)
)
Z <- normalize.linf(X)
Z
apply(Z, 1, max)
```

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. Printing shows sample accounting and up to ten groups per depth. Use table(x$lineage.labels[[d]], useNA = "ifany") for the complete counts at depth d.

Examples

```
M <- rbind(
  s1 = c(A = 1.0, B = 0.2),
  s2 = c(A = 0.8, B = 1.0)
)
fit <- linf.csts(M, n0 = 1)
print(fit)
```

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

Description

Selects leaf dominance-lineages and refines them by dropping the dominant feature(s) stored in explicit node paths and re-applying linf.csts to the remaining features. The resulting child labels are joined using sep; colliding readable paths receive a node suffix.

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,
  lineages.to.refine = NULL,
  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. Columns must correspond, in order, to the stable feature IDs stored in csts. Rows must remain in fitted sample order. Both dimensions are checked; named axes are checked against csts$input.dimnames when available. Unnamed axes/legacy fits use positional matching. No automatic reordering.
csts	A "linf.csts" object.
lineages.to.refine	Optional character vector of leaf dominance-lineage IDs to refine. When NULL, lineages are selected automatically using refinement.factor * n0.
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, emit progress messages.
backend	Character. Matrix backend to use: "auto", "dense", or "sparse". The default "auto" inherits the backend from M or from csts when available.

Details

Selection and child fitting use active leaf membership. Explicit selection bypasses the automatic parent-support rule but not the child threshold. Every call appends a stored level; unselected parents and parents without retained children keep their labels there. Child fits use first-column tie handling; the parent fit's random-tie setting is not inherited. Keep policy, rare label and separator consistent across repeated refinements. Stored synthetic or undefined parents remain terminal in their own policy view, even when the other view is refined. A real feature whose ID equals the rare label is not a synthetic parent and can be refined.

Value

Updated "linf.csts" object with depth increased by one and updated lineage.label and a new history entry recording the settings and selected/refined nodes at this depth. Policy-specific views are stored in lineage.label.pure and lineage.label.absorb.

Examples

```
M <- rbind(
  s1 = c(A = 1.0, B = 0.8, C = 0.1),
  s2 = c(A = 1.0, B = 0.7, C = 0.2),
  s3 = c(A = 1.0, B = 0.6, C = 0.3),
  s4 = c(A = 0.2, B = 1.0, C = 0.8),
  s5 = c(A = 0.1, B = 1.0, C = 0.7),
  s6 = c(A = 0.3, B = 1.0, C = 0.6)
)
depth1 <- linf.csts(M, n0 = 2, low.freq.policy = "absorb")
depth2 <- refine.linf.csts(
  M,
  depth1,
  lineages.to.refine = "A",
  n0 = 2,
  low.freq.policy = "absorb",
  verbose = FALSE
)
depth2$lineage.labels[[2]]
```

---

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 one row per stored depth. `input.samples`, `assigned.samples`, `unassigned.samples` and `rare.samples` expose the denominators. `n.lineages` includes synthetic rare groups; `retained.lineages` excludes them. For compatibility, `total.samples` remains an alias of `assigned.samples`. Size statistics include rare groups and are `NA` when no assignments exist; valid empty fits do not produce warnings.

Examples

```
M <- rbind(
  s1 = c(A = 1.0, B = 0.2),
  s2 = c(A = 0.9, B = 0.3),
  s3 = c(A = 0.2, B = 1.0)
)
fit <- linf.csts(M, n0 = 1)
summary(fit)
```

---

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. Scoring uses explicit node feature indices, independently of display labels and separator-containing feature IDs.

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 repeated calls to refine.linf.csts.
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). When used for matching, keys must be unique, nonmissing and nonempty; ambiguous query keys are rejected.
feature.labels	Optional feature labels for columns of X. Defaults to feature.ids. The same key requirements apply when matching by labels; repeated display labels are allowed when matching by 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, retained for call compatibility. Under either setting, terminal lineages are assigned only where they occur in the fitted hierarchy. No out-of-hierarchy fallback is performed.
sep	Separator recorded with the result for call compatibility. Node identity and feature lookup use the fitted metadata, not this string. Defaults to csts$sep or "__".
backend	Matrix backend; passed to the internal matrix preparer.

Details

Diagnostic reasons are complete (every fitted depth assigned), no_candidates (no realized children), synthetic_only (only unscorable rare categories), missing_candidate_features (all real candidate features absent from the query), or no_positive_candidate_values (some real candidate features are present but none has positive abundance). A partially missing candidate set can have the last reason; inspect missing.candidates too. Completed walks have NA stopping depth and candidate counts. A terminal lineage carried through later stored levels counts as an assignment at each such level; assigned.depth is not the number of distinct feature choices. ID output does not change the existing tie rule: lexical display-label order still resolves equal-support ties. Relabeling and refitting can therefore change tied assignments; inspect n.tied.depths when comparing results.

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.
- `assignment.ids`, `all.depths.ids`: corresponding lineage-ID matrices whose identifiers do not encode display labels. IDs belong to the fitted feature mapping; they are not universal across fits.
- `feature.match`: one row per fitted feature, with ID, label, query column index (NA if absent) and a logical matched flag.
- `diagnostics`: one row per query, with sample index/name, deepest assigned depth, stopping depth/reason, counts of present and absent candidate features at that stop, and number of depths with ties. Diagnostics describe the full walk, regardless of requested depths.
- `depth`: requested depth vector.
- `view`, `match.by`, `tie.method`: settings used.

At any depth with no realized candidate or no positive abundance for its candidate features, that depth and all subsequent depths are `NA`. The original assignment matrices retain display labels regardless of `match.by`; use their `.ids` counterparts for joins.

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 Construction parameters, including `generator` with the linf version, R version and pinned upstream revision used to rebuild these assignments.

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. This list is an assignment asset, not a fitted `"linf.csts"` hierarchy; it cannot be passed directly to refinement, landmark or transfer functions.

Reading provenance offline

Read the installed source and preparation record at the console: `cat(readLines(system.file("DATA_PROVENANCE.md", package = "linf", mustWork = TRUE)), sep = "\n")`. The installed `DATA_MANIFEST.csv` records pinned input checksums; see the reproduction recipe.

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 Construction parameters, including `generator` with the linf version, R version and pinned upstream revision used to rebuild these assignments.

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. It is not a fitted `"linf.csts"` object and does not contain the source abundance matrix or complete policy hierarchies.

Reading provenance offline

Read the installed source and preparation record at the console: `cat(readLines(system.file("DATA_PROVENANCE.md", package = "linf", mustWork = TRUE)), sep = "\n")`. The installed `DATA_MANIFEST.csv` records pinned input checksums; see the reproduction recipe.

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 4) with columns: `sample_id` (character), `source_row` (verified row index in the pinned prepared 13k input; shared with other Valencia assets), `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, not original observed counts.

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.

Reading provenance offline

Read the installed source and preparation record at the console: `cat(readLines(system.file("DATA_PROVENANCE.md", package = "linf", mustWork = TRUE)), sep = "\n")`. The installed `DATA_MANIFEST.csv` records pinned input checksums; see the reproduction recipe.

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. The stored object contains input abundances, not precomputed embedding coordinates; compute those with `linf.hypercube.embedding`.

Reading provenance offline

Read the installed source and preparation record at the console: `cat(readLines(system.file("DATA_PROVENANCE.md", package = "linf", mustWork = TRUE)), sep = "\n")`. The installed `DATA_MANIFEST.csv` records pinned input checksums; see the reproduction recipe.

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

∗ **datasets**
agp_gut, 3
valencia13k_dcst_depth2_merged, 29
valencia13k_dcst_depth3_merged, 30
valencia2k, 32
valencia_linf_hypercube_1k, 33
∗ **package**
linf-package, 2

agp_gut, *3*, 3

dcst.view, *3*, 5

filter.asv, *2*, *4*, 6

linf *(*linf-package*)*, 2
linf-package, 2
linf.csts, *2*, *5*, 7, *10, 11*, *13*, *24*, *28*
linf.csts(), *21, 22*
linf.dcst.landmark.pipeline, *3*, 10
linf.dominant.features, *2*, 12
linf.dominant.features(), *22*
linf.feature.labels, *3*, 14
linf.hypercube.embedding, *3*, 15, *33, 34*
linf.landmarks, *3*, *8*, *10, 11*, 17
linf.udcst.graph, 18
linf.udcsts, *2*, 20
linf.udcsts(), *19*

normalize.linf, *2*, *4*, *10*, *13*, 23, *32*

print.linf.csts, 24

refine.linf.csts, *3*, *5*, *10, 11*, 24, *28*

summary.linf.csts, 26

transfer.dcsts, *3*, 27

valencia13k_dcst_depth2_merged, *3*, 29, *31*
valencia13k_dcst_depth3_merged, *3*, 30
valencia2k, *3*, 32
valencia_linf_hypercube_1k, *3*, 33