Package: transitiontrees (via r-universe)

June 18, 2026

Type Package

Title Transition Trajectories and Dynamics of Variable-Length Pathways or Sequences

Version 0.1.2

Description Analyzes transition trajectories in event, sequence, and ordered data, focusing on how states follow one another, how far processes unfold, and where pathways branch or converge. Trajectories are modeled using variable-order prediction suffix trees (Ron, Singer, & Tishby, 1996) <doi:10.1023/A:1026490906255>, implemented in both frequency-based and prediction-based forms. The framework includes multiple pruning, validation, and smoothing techniques to ensure model robustness. Visualization options include transition trees, radial sunburst diagrams, transition heatmaps, and forward trajectory trees.

License GPL-3

URL https://github.com/mohsaqr/transitiontrees

BugReports https://github.com/mohsaqr/transitiontrees/issues

Encoding UTF-8

Language en-US

LazyData true

LazyDataCompression xz

Depends R (>= 4.1)

Imports ggplot2, grDevices, stats, utils

Suggests testthat (>= 3.0.0), knitr, rmarkdown, visNetwork, tna, Nestimate

VignetteBuilder knitr

Config/testthat/edition 3

RoxygenNote 7.3.3

NeedsCompilation no

Author Mohammed Saqr [aut, cre, cph], Sonsoles López-Pernas [aut]

Maintainer Mohammed Saqr <saqr@saqr.me>

Repository https://cran.r-universe.dev

Date/Publication 2026-06-18 17:10:08 UTC

RemoteUrl https://github.com/cran/transitiontrees

RemoteRef HEAD

RemoteSha 037bd79833b65d3fa7fc936bdb0411a946a46dbb

Contents

ai_long . . . 3
as.data.frame.transitiontrees . . . 4
as.data.frame.transitiontrees_bootstrap . . . 5
as.data.frame.transitiontrees_comparison . . . 5
as.data.frame.transitiontrees_group . . . 6
as.data.frame.transitiontrees_group_comparison . . . 6
bootstrap_pathways . . . 7
common_pathways . . . 9
compare_groups . . . 10
compare_pruning . . . 11
compare_smoothing . . . 12
compare_trees . . . 14
context_tree . . . 15
divergent_pathways . . . 17
engagement . . . 18
generate_sequences . . . 19
group_regulation_long . . . 20
impute_sequences . . . 20
logLik.transitiontrees . . . 21
mine_contexts . . . 22
mine_sequences . . . 23
model_fit . . . 24
n_nodes . . . 25
nobs.transitiontrees . . . 25
pathway_exists . . . 26
perplexity . . . 26
plot.transitiontrees . . . 27
plot.transitiontrees_bootstrap . . . 28
plot.transitiontrees_comparison . . . 29
plot.transitiontrees_group . . . 30
plot.transitiontrees_group_comparison . . . 30
plot.transitiontrees_tune . . . 31
plot_difference . . . 32
plot_distributions . . . 33
plot_divergence . . . 34
plot_pathway_resamples . . . 35

plot_pathways . . . 36
plot_predictive . . . 37
plot_pruning . . . 38
plot_trajectories . . . 39
predict.transitiontrees . . . 40
prepare_input . . . 41
print.summary.transitiontrees . . . 42
print.transitiontrees . . . 43
print.transitiontrees_bootstrap . . . 43
print.transitiontrees_comparison . . . 44
print.transitiontrees_group . . . 45
print.transitiontrees_group_comparison . . . 45
print.transitiontrees_tune . . . 46
prune_tree . . . 46
query_pathway . . . 47
score_positions . . . 48
score_sequences . . . 49
sharp_pathways . . . 50
simulate.transitiontrees . . . 50
smooth_tree . . . 51
subtree . . . 52
summary.transitiontrees . . . 53
summary.transitiontrees_bootstrap . . . 53
summary.transitiontrees_group_comparison . . . 54
trajectories . . . 54
tree_dependence . . . 55
tree_distance . . . 56
tree_pathways . . . 57
tune_tree . . . 58

Index **60**

`ai_long` *AI-collaboration messages (long format)*

Description

A long, one-row-per-message log from an AI-assisted collaboration study, with Unix timestamps and an explicit session id. Used to demonstrate long-format loading with Unix time and sessions. Bundled example dataset.

Usage

```
ai_long
```

Format

A data.frame with 8551 rows and 9 columns, including project, session_id, timestamp (Unix seconds), code / cluster (the state at two granularities), and code_order / order_in_session (within-sequence order).

Source

Bundled example dataset.

Examples

```
data(ai_long)
context_tree(ai_long, actor = "project", time = "timestamp",
             action = "code", max_depth = 2L)
```

---

as.data.frame.transitiontrees

Coerce a context tree to a Tidy Data Frame

Description

Returns the canonical tidy node table — identical to tree_pathways(tree). Lets users do as.data.frame(tree) and immediately filter, sort, or export with base-R idioms.

Usage

```
## S3 method for class 'transitiontrees'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	A transitiontrees.
row.names, optional	Ignored.
...	Forwarded to tree_pathways().

Value

A data.frame with columns pathway, depth, count, likely_next, next_probability, divergence, changes_prediction. See tree_pathways.

as.data.frame.transitiontrees_bootstrap

Coerce a context tree Bootstrap to a Tidy Data Frame

Description

Uniform tidy-extract: returns the per-pathway summary table (`object$summary`), so `as.data.frame(boot)` and `summary(boot)` are interchangeable extractors.

Usage

```
## S3 method for class 'transitiontrees_bootstrap'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	A `transitiontrees_bootstrap`.
`row.names, optional`	Ignored.
`...`	Ignored.

Value

A data.frame; see `bootstrap_pathways` for the full column vocabulary.

as.data.frame.transitiontrees_comparison

Coerce a context tree Comparison to a Tidy Data Frame

Description

Uniform tidy-extract: returns the per-pathway divergence breakdown (`object$pathways`) — the consumer-facing detail behind the scalar `pdist` and the permutation `p_value`.

Usage

```
## S3 method for class 'transitiontrees_comparison'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	A `transitiontrees_comparison`.
`row.names, optional`	Ignored.
`...`	Ignored.

Value

A data.frame with columns `pathway`, `count_a`, `count_b`, `divergence_ab`, `divergence_ba`, `divergence_sym`.

---

as.data.frame.transitiontrees_group

Coerce a Group of Trees to One Tidy Data Frame

Description

Row-binds each group's `tree_pathways` table, tagged with a leading `group` column, so the whole batch is one tidy frame ready to filter, facet, or join.

Usage

```
## S3 method for class 'transitiontrees_group'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	A `transitiontrees_group`.
`row.names, optional`	Ignored.
`...`	Forwarded to `tree_pathways()`.

Value

A data.frame: the canonical pathway columns with a leading `group` column identifying the source tree.

---

as.data.frame.transitiontrees_group_comparison

Coerce a Group Comparison to a Tidy Data Frame

Description

Uniform tidy-extract: returns the per-pathway comparison table (`object$pathways`) so `as.data.frame(cmp)` yields the full divergence / usage breakdown as a base `data.frame`.

Usage

```
## S3 method for class 'transitiontrees_group_comparison'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	A transitiontrees_group_comparison.
row.names, optional	Ignored.
...	Ignored.

Value

A data.frame of per-pathway results; see compare_groups for the column vocabulary.

---

bootstrap_pathways *Bootstrap Pathway Stability and Informativeness*

Description

Non-parametric sequence bootstrap for a fitted transitiontrees. Methodologically built on Saqr, Tikka & López-Pernas (2025), extending an edge-level bootstrap framework to variable-depth pathways.

The bootstrap tracks every pathway in the original tree. Each iteration resamples whole sequences with replacement, aggregates raw counts per depth, and reads each pathway's count vector directly from the resample. **No smoothing, no** nmin **filter, no extra parameters** inside the loop — the bootstrap operates on counts analogously to an edge-weight bootstrap.

Two complementary measures are reported per pathway:

p_stability Bootstrap-estimated probability that the chosen stat (default count) falls outside [cr[1] * observed, cr[2] * observed], with a +1 correction. This is a stability p-value: small values mean the pathway rarely fails the chosen reproducibility criterion under sequence-level resampling.

stability_rate Uncorrected descriptive companion: the fraction of resamples where the chosen stat lies inside the consistency band. A pathway whose count drops to zero in a resample fails the band test automatically.

informative_rate Fraction of resamples where the pathway's empirical G^2 likelihood-ratio statistic against its parent context exceeds the chi-square critical value at level alpha_g2 (df = |alphabet| - 1). Tests *reproducibly significant divergence from the shorter-history baseline*.

Read stable and informative together:

- stable && informative: reproducible and predictively distinctive pathway.
- stable && !informative: reproducible pathway count / statistic, but not predictively distinctive from its parent.
- !stable && informative: sharp or divergent pathway carried by an unstable subset of sequences.
- !stable && !informative: weak or sample-fragile pathway.

Usage

```
bootstrap_pathways(
  tree,
  iter = 1000L,
  stat = c("count", "next_probability", "divergence"),
  consistency_range = c(0.5, 1.5),
  stability_threshold = 0.95,
  informative_threshold = 0.8,
  alpha = 0.05,
  ci_level = 0.05,
  seed = 1L,
  keep_resamples = TRUE,
  progress = FALSE
)
```

Arguments

tree	A fitted transitiontrees carrying tree$data.
iter	Integer. Number of bootstrap iterations. Default 1000.
stat	Character. Pathway statistic on which p_stability is measured. One of "count" (default; the edge-weight analogue), "next_probability" (the most-likely next-state probability), or "divergence".
consistency_range	Numeric vector of length 2. Multiplicative tolerance band around the observed value. Default c(0.5, 1.5): a resample counts as consistent when its statistic stays within half-to-one-and-a-half times the observed value.
stability_threshold	Numeric in $(0, 1)$. Backward- compatible stability-rate threshold. A pathway is stable = TRUE when p_stability < 1 - stability_threshold. Default 0.95: at least 95% of resamples must fall inside the (wide) consistency_range band.
informative_threshold	Numeric in $(0, 1)$. A pathway is informative = TRUE when informative_rate >= informative_threshold. Default 0.80: a pathway must clear the G^2 critical value in at least 80% of resamples to be called informative, relaxing the earlier 0.95 gate that suppressed many genuinely informative deeper pathways.
alpha	Numeric in $(0, 1)$. Significance level for the G^2 test against parent. Default 0.05.
ci_level	Numeric in $(0, 1)$. Tail probability for the bootstrap CIs on count, next_probability, divergence, G2. Default 0.05 (95% CI).
seed	Integer or NULL. RNG seed. Default 1L.
keep_resamples	Logical. If TRUE (default), the per-iteration resample matrices M_count, M_next_probability, M_divergence, M_G2, M_changes_prediction are retained on the returned object. Set to FALSE to drop them (each is iter x n_pathways) when memory matters; the summary table is unaffected.
progress	Logical. Show a progress bar. Default FALSE.

Value

A `transitiontrees_bootstrap` object: a list with

summary Per-pathway data.frame, sorted so that `stable` & `informative` pathways come first then by `stability_rate` descending.

pathways_orig Empirical original-pathway statistics (no smoothing) as a tidy data.frame.

M_count, M_next_probability, M_divergence, M_G2, M_changes_prediction Raw resample matrices: `iter` x `n_pathways`, columns named by pathway.

iter, stat, consistency_range, stability_threshold, informative_threshold, alpha_g2, ci_level, seed, g2_critical_value Configuration.

References

Saqr, M., Tikka, S., & López-Pernas, S. (2025). Transition Network Analysis. *LAK '25*, doi:10.1145/3706468.3706513.

Examples

```
seqs <- replicate(40, sample(c("A", "B", "C"), 10, replace = TRUE),
                  simplify = FALSE)
tree <- context_tree(seqs, max_depth = 1L)
boot <- bootstrap_pathways(tree, iter = 50L)
summary(boot)
```

`common_pathways` *Most Common Pathways in a Fitted Tree*

Description

Returns the top `n` pathways by occurrence count – the trajectories the data actually contains many copies of.

Usage

```
common_pathways(tree, top = 10L, depth = NULL, min_count = 1L)
```

Arguments

`tree`	A `transitiontrees`.
`top`	Integer. Number of pathways to return. Default 10.
`depth`	Integer or NULL. Restrict to pathways of this exact depth. `NULL` (default) keeps all depths.
`min_count`	Integer. Minimum count cut-off. Default 1.

Value

A data.frame, same columns as `tree_pathways`, sorted by count descending.

Examples

```
seqs <- replicate(50, sample(c("A","B","C"), 12, replace = TRUE),
                  simplify = FALSE)
tree <- context_tree(seqs, max_depth = 3)
common_pathways(tree, top = 8)
common_pathways(tree, top = 8, depth = 3L)   # restrict to depth-3
```

`compare_groups` *Compare Groups of Sequences for Structural Differences*

Description

Test how a set of fitted group trees (a `transitiontrees_group` from `context_tree(..., group =)`) differ, on two complementary axes:

behavioral given the *same* context, do the groups predict a different next state? Measured per context by the count-weighted Jensen-Shannon divergence (bits) across the groups' next-state distributions.

usage do the groups *reach* a context at different rates? Measured per context by a G^2 homogeneity statistic on its prevalence (its share of each group's positions).

Significance is assessed by a label-permutation null throughout (per-pathway and omnibus), with Benjamini-Hochberg FDR on the per-pathway p-values.

Usage

```
compare_groups(group, iter = 999L, min_count = 1L, seed = 1L, block = NULL)
```

Arguments

`group`	A `transitiontrees_group`.
`iter`	Integer. Number of label permutations. Default 999.
`min_count`	Integer. Drop contexts whose total count across all groups is below this. Default 1.
`seed`	Integer or `NULL`. RNG seed. Default 1.
`block`	Block ids for a **stratified** permutation: group labels are shuffled only *within* each block, so the null respects nested / repeated-measures structure (e.g. several sequences from one subject) and holds any between-block difference fixed. Normally you do not pass this — fit with `context_tree(..., block = )` and it is carried on the object and used automatically. Passing a vector here (one id per sequence, in pooled group-then-row order) overrides that. `NULL` with no stored block shuffles labels freely.

Details

The permutation pools every sequence, shuffles the group labels (preserving group sizes), and recomputes the statistics from raw counts using the same counting routine as the fit. The tested context set is the union of the contexts the groups' trees actually represent. For two groups the behavioral measure is JSD, which is **not** the symmetric-KL distance used by `compare_trees()`; the `distance_matrix` component does use `tree_distance()` (symmetric KL) for consistency with the pairwise function.

Value

A `transitiontrees_group_comparison`: a list with

pathways Per-context data.frame sorted by `jsd_bits` descending, with columns `pathway`, `depth`, `count_total`, one `count_<group>` and one `modal_<group>` column per group (most likely next state, ties broken by alphabet order), `flips` (do the groups' modal next states disagree?), `jsd_bits`, `jsd_p`, `jsd_padj`, `usage_g2`, `usage_p`, `usage_padj`. `usage_*` is `NA` for the root, which has no prevalence test.

omnibus Two-row data.frame: the behavioral and usage global statistics with permutation p-values.

distance_matrix K x K symmetric-KL distance matrix between the groups (from `tree_distance()`).

groups, iter, seed, n_contexts Configuration.

See Also

`compare_trees` for the pairwise permutation test.

Examples

```
gx <- replicate(40, sample(c("A","B","C"), 8, replace = TRUE,
                           prob = c(.2,.6,.2)), simplify = FALSE)
gy <- replicate(40, sample(c("A","B","C"), 8, replace = TRUE,
                           prob = c(.2,.2,.6)), simplify = FALSE)
grp <- context_tree(c(gx, gy), group = rep(c("x","y"), each = 40),
                    max_depth = 1L)
cmp <- compare_groups(grp, iter = 199L)
cmp
```

`compare_pruning` *Compare Pruning Criteria on One Tree*

Description

Prunes a fitted tree under several criteria — holding `alpha` and `threshold` fixed — and returns a tidy one-row-per-criterion summary of how aggressively each trims the tree. A convenience wrapper over repeated `prune_tree` calls that collapses the usual `vapply()` criterion loop into one call.

Usage

```
compare_pruning(
  tree,
  criterion = c("G2", "KL", "AIC", "BIC"),
  alpha = 0.05,
  threshold = 0.005
)
```

Arguments

tree	A transitiontrees (typically unpruned).
criterion	Character vector of criteria to compare. Defaults to all four: "G2", "KL", "AIC", "BIC".
alpha	Significance level for "G2" (and the AIC/BIC penalties' chi-square cutoff). Default 0.05.
threshold	Minimum information gain in nats for "KL". Default 0.005.

Value

A data.frame with one row per criterion (in the order given by criterion) and columns criterion, n_nodes (post-prune size) and reduction_pct (percent of the original nodes removed).

See Also

prune_tree to apply one criterion, tune_tree for cross-validated selection.

Examples

```
set.seed(1)
seqs <- replicate(80, sample(c("A", "B", "C"), 14, replace = TRUE),
                  simplify = FALSE)
tree <- context_tree(seqs, max_depth = 4L, min_count = 3L)
compare_pruning(tree)
compare_pruning(tree, criterion = c("G2", "BIC"), alpha = 0.01)
```

---

compare_smoothing *Compare Smoothing Schemes on One Dataset*

Description

Fits a context tree under several smoothing schemes — holding max_depth, nmin and every other argument fixed — and returns a tidy one-row-per-scheme comparison of tree size and in-sample perplexity. A convenience wrapper over repeated context_tree calls that collapses the usual five-line lapply() loop into a single call.

Usage

```
compare_smoothing(
  data,
 smoothing = c("floor", "laplace", "kneser_ney", "witten_bell", "jelinek_mercer"),
  ...
)
```

Arguments

data	Either sequence data in any form accepted by context_tree (wide matrix / data.frame, list of character vectors, or a transition/network object) — fitted afresh under each scheme — **or** an already-fitted transitiontrees, which is *re-smoothed* under each scheme (topology frozen, no re-count; e.g. to sweep smoothers on a pruned tree).
smoothing	Character vector of smoothing-method names to compare. Defaults to all five: "floor", "laplace", "kneser_ney", "witten_bell", "jelinek_mercer".
...	Further arguments passed to context_tree (e.g. max_depth, nmin, alphabet), held fixed across every scheme. Ignored when data is a fitted tree.

Details

The perplexity reported is **in-sample** (computed on the fitting data), so it rewards memorisation and must *not* be used to pick a smoother — use tune_tree() for out-of-sample selection. The point of this table is the side-by-side view and the invariance of n_nodes across schemes: smoothing changes the *probabilities* inside the tree, never *which* contexts exist (topology is set by nmin, not by the smoother).

Value

A data.frame with one row per scheme (in the order given by smoothing) and columns smoothing (method name), n_nodes (tree size) and perplexity (in-sample).

See Also

smooth_tree to re-smooth a fitted tree without re-counting; tune_tree for cross-validated selection.

Examples

```
set.seed(1)
seqs <- replicate(50, sample(c("A", "B", "C"), 12, replace = TRUE),
                  simplify = FALSE)
compare_smoothing(seqs, max_depth = 3L, min_count = 5L)
compare_smoothing(seqs, smoothing = c("floor", "kneser_ney"),
                  max_depth = 2L)
```

compare_trees *Compare Two context trees by Symmetric Divergence with Permutation Test*

Description

Computes the count-weighted symmetric Kullback-Leibler divergence between two fitted transitiontrees, then provides a reference distribution by permuting sequence-to-tree assignments.

Use this to ask: do two cohorts (group A vs. group B, baseline vs. intervention) generate significantly different pathway dynamics?

Usage

```
compare_trees(tree_a, tree_b = NULL, iter = 200L, seed = 1L, symmetric = TRUE)
```

Arguments

tree_a, tree_b	context trees fit on data subsets A and B. Alternatively, pass a two-element transitiontrees_group (from context_tree(..., group =)) as tree_a and leave tree_b = NULL; its two trees are compared in key order.
iter	Integer. Number of permutations. Default 200.
seed	Integer. RNG seed for reproducibility. Default 1.
symmetric	Logical. Default TRUE.

Value

A transitiontrees_comparison object with components:

pdist observed scalar distance

null_dist numeric vector, length iter

p_value one-sided p-value (proportion of null at least as extreme as observed)

pathways per-pathway breakdown data.frame

Examples

```
set.seed(1)
m1 <- matrix(sample(c("A","B","C"), 200, TRUE), 20)
m2 <- matrix(sample(c("A","B","C"), 200, TRUE), 20)
tr1 <- context_tree(m1, max_depth = 2L, min_count = 3L)
tr2 <- context_tree(m2, max_depth = 2L, min_count = 3L)
compare_trees(tr1, tr2, iter = 50)
```

`context_tree` *Fit a Prediction Suffix Tree from Categorical Sequence Data*

Description

Estimates a variable-depth context tree (prediction suffix tree; Ron, Singer & Tishby 1996) from a collection of sequences. Each internal node represents a context (string of recent states); each leaf carries a smoothed conditional distribution over the next state. The tree is grown to `max_depth`, then optionally pruned via `prune_tree()`.

Usage

```
context_tree(
  data,
  max_depth = 5L,
  min_count = 5L,
  smoothing = "floor",
  alphabet = NULL,
  weights = NULL,
  group = NULL,
  block = NULL,
  actor = NULL,
  time = NULL,
  action = NULL,
  order = NULL,
  session = NULL,
  time_threshold = 900
)
```

Arguments

`data` Sequence data in any of these forms: a wide data.frame / character matrix (rows = trajectories, columns = time-steps), a list of character vectors, or a **supported network/transition model object**, taken directly: a fitted network object carrying its sequences, or a transition-network object. Any such object that follows the same convention (a `$data`/`$sequences`/ `$seqdata` sequence slot) is detected structurally and works with no special-casing. For these objects the sequence frame is extracted from wherever the upstream stored it; an integer-coded frame is decoded through the object's label map (`$nodes` id/label table, positional `$labels`, or a `labels`/`alphabet` attribute), and that label set becomes the default alphabet so the tree shares the model's symbol space. A *pure graph* object that carries no sequences anywhere (an aggregated transition network) is rejected with guidance — sequences cannot be recovered from edge weights, the same reason numeric transition matrices are rejected; route to the sequence-bearing object explicitly. In wide / list input, `NA`, the empty string, and the TraMineR codes `"%"` (void) and `"*"` (missing) all mark "no observation": the

cell is dropped (gaps closed, trailing padding trimmed) rather than treated as a state.

`max_depth` Integer. Maximum context length the tree may represent. Default 5.

`min_count` Integer. Minimum number of times a context must occur to receive its own node. Default 5. Contexts seen fewer than `min_count` times are absorbed into their parent.

`smoothing` Smoothing specification: a method name as a string (uses defaults for that method's hyperparameters) or a list of the form `list(method, ...kwargs)` for explicit hyperparameters. Methods: `"floor"` (default; `ymin = 0.001`), `"laplace"` (`alpha = 1`), `"kneser_ney"` (`discount = 0.75`), `"witten_bell"`, `"jelinek_mercer"` (`lambda = 0.5`). The `"floor"` method also takes `rule`: `"interpolate"` (default, the interpolating floor — a distribution with a zero-count state is shifted toward uniform so each zero lands at exactly `ymin`) or `"cap"` (clamp every probability up to `ymin` and renormalise), e.g. `list("floor", ymin = 0.001, rule = "cap")`.

`alphabet` Character vector. Optional. Override the data-derived alphabet (useful when the test set may include states unseen in training).

`weights` Numeric vector of per-sequence weights, length equal to the number of input rows / list elements. `NULL` (default) fits unweighted.

`group` Optional grouping for a **batch fit**: a vector with one entry per input sequence, a column name of a network object's `$metadata`, or — in long-format mode (`action` given) — a column name of `data` (collapsed to one value per sequence). When supplied (or when `data` is itself a grouped object such as a `netobject_group` or a `group_tna`), `context_tree()` fits one tree per group over a shared alphabet and returns a `transitiontrees_group` (a named list of `transitiontreess`). Default `NULL` (single tree).

`block` Optional block id for a stratified group comparison, in the same shapes as `group` (per-sequence vector, or a column name of `data` in long-format mode). Carried on the returned `transitiontrees_group` and used automatically by `compare_groups()` so the caller never re-aligns it. Only meaningful alongside `group`. Default `NULL`.

`actor`, `time`, `order`, `session`, `time_threshold`
Passed to `prepare_input()` when `action` is given: the unit each sequence belongs to (`actor`), the timestamp column (`time`), an explicit ordering (`order`) or session id (`session`), and the gap in seconds that starts a new session (`time_threshold`, default 900).

`action` Character. Naming a state/code column switches `data` to **long-format** mode: it is reshaped to a wide sequence frame with `prepare_input()` before fitting. Required to use any of `actor`/`time`/`order`/ `session`. Default `NULL` (data is already in sequence shape).

Details

Construction is bottom-up via k-gram counting. The root holds the marginal next-state distribution; depth-k nodes hold the next-state distribution conditional on the most recent k states. Nodes whose total count falls below `min_count` are not created. All nodes start "live" and unpruned; `prune_tree()` decides which to retain.

Value

For a single fit, a `transitiontrees` object (described below). For a grouped fit (`group` = supplied, or a grouped family object passed in) a `transitiontrees_group`: a named list of `transitiontreess`, one per group, in the group's key order, with its own `print` and `as.data.frame` methods. A single `transitiontrees` is a list with components

nodes Named list of node descriptors. Names are context strings (e.g. `"A -> B"`); the root is keyed by the literal sentinel `"<root>"`. Each entry has `depth` (integer), `counts` (numeric vector indexed by alphabet), `prob` (smoothed probability vector), and `n` (sum of counts).

edges data.frame with columns `parent`, `child`, `symbol` for fast tree traversal.

alphabet character vector of states.

max_depth Integer. The fitted depth (may be less than the requested `max_depth` if data are short).

nmin the chosen min-count threshold.

smoothing resolved smoothing list (`method` + hyperparameters).

n_seq, n_obs number of sequences and observations.

pruned Logical. `TRUE` after `prune_tree()` has been applied.

pruning When `pruned` is `TRUE`, a list capturing the criterion / alpha / threshold used; otherwise `NULL`.

data The cleaned trajectories (a list of character vectors) retained for downstream bootstrap and permutation routines.

References

Ron, D., Singer, Y., Tishby, N. (1996). The power of amnesia: learning probabilistic automata with variable memory length. *Machine Learning*, 25, 117-149.

Examples

```
set.seed(1)
seqs <- replicate(50, sample(c("A","B","C"), 12, replace = TRUE),
                  simplify = FALSE)
tree <- context_tree(seqs, max_depth = 3)
tree
summary(tree)
```

`divergent_pathways` *Most Predictively Divergent Pathways*

Description

Returns the top `n` pathways by Kullback-Leibler divergence from their (k-1)-suffix. These are the pathways whose extended history adds the most predictive information over the shorter one. Pathways whose most likely next state actually flips between orders are marked in the `changes_prediction` column.

Usage

```
divergent_pathways(tree, top = 10L, min_count = 1L, flips_only = FALSE)
```

Arguments

tree	A transitiontrees.
top	Integer. Number of pathways to return. Default 10.
min_count	Integer. Drop pathways with fewer than this many occurrences. Default 1.
flips_only	Logical. If TRUE, return only pathways that flip the most likely next state between orders. Default FALSE.

Value

A data.frame, same columns as tree_pathways, sorted by divergence descending.

Examples

```
seqs <- replicate(50, sample(c("A","B","C"), 12, replace = TRUE),
                  simplify = FALSE)
tree <- context_tree(seqs, max_depth = 3)
divergent_pathways(tree, top = 6)
divergent_pathways(tree, flips_only = TRUE)
```

---

engagement *Student engagement state sequences (wide format)*

Description

A wide set of student engagement-state sequences, one learner per row and one time-step per column. Used to demonstrate fitting from wide sequence data and from the network objects (**tna** / **Nestimate**) built on top of it. Bundled example dataset.

Usage

```
engagement
```

Format

A data.frame with 1000 rows (learners) and 25 columns (time-steps). States "Active", "Average", "Disengaged"; NA marks missing / dropped-out positions.

Source

Bundled example dataset.

Examples

```
data(engagement)
context_tree(engagement, max_depth = 2L)
```

`generate_sequences` *Sample Sequences from a Fitted Context Tree*

Description

Sample Sequences from a Fitted Context Tree

Usage

```
generate_sequences(tree, n = 5L, length = 10L, start = NULL)
```

Arguments

`tree`	A `transitiontrees`.
`n`	Integer. Number of sequences to sample.
`length`	Integer. Length of each sampled sequence.
`start`	NULL or character vector. If NULL (default), each sequence starts from the root marginal; otherwise must have length `n` and supply the first state of each sequence.

Value

A character matrix of dimension `n` x `length`.

Examples

```
seqs <- replicate(50, sample(c("A","B","C"), 12, replace = TRUE),
                  simplify = FALSE)
tree <- context_tree(seqs, max_depth = 3)
generate_sequences(tree, n = 5, length = 10)
```

group_regulation_long *Collaborative-regulation events (long format)*

Description

A long, one-row-per-event log of collaborative regulation moves, with timestamps. Used to demonstrate long-format loading: reshape with `prepare_input()` (or name `action =` in `context_tree()`) to split each actor's events into time-gap sessions. Bundled example dataset.

Usage

```
group_regulation_long
```

Format

A `data.frame` with 27533 rows and 6 columns:

Actor integer; the learner.

Achiever character; an achievement-level covariate.

Group numeric; the collaboration group.

Course character; the course.

Time POSIXct; the event timestamp.

Action character; the regulation move (the state).

Source

Bundled example dataset.

Examples

```
data(group_regulation_long)
context_tree(group_regulation_long, actor = "Actor", time = "Time",
             action = "Action", max_depth = 3L)
```

impute_sequences *Impute Missing States in Sequences*

Description

Fill the gaps in incomplete sequences using a fitted context tree: each missing state is predicted from the longest matching context of the states that precede it. Filling proceeds left to right, so a just-imputed state becomes part of the context for later gaps.

Usage

```
impute_sequences(tree, newdata, method = c("modal", "prob"), seed = NULL)
```

Arguments

tree	A transitiontrees.
newdata	Sequences with gaps: a list of character vectors, a character matrix / data.frame (one row per sequence, NA or "" marking a gap), or a single character vector.
method	One of "modal" (default; fill with the most likely state) or "prob" (sample from the predicted distribution).
seed	Integer or NULL. Optional RNG seed, used only when method = "prob".

Details

Only **internal** gaps are imputed. A run of trailing NA / "" cells (end-of-sequence padding in a wide frame) is left untouched, since there is no observed state after it to mark the sequence as continuing. A sequence that is entirely missing is returned unchanged (there is nothing to condition on).

Value

The same container shape as newdata (list, matrix, data.frame, or character vector) with internal gaps filled.

Examples

```
seqs <- replicate(60, sample(c("A", "B", "C"), 8, replace = TRUE),
                  simplify = FALSE)
tree <- context_tree(seqs, max_depth = 2L)
gappy <- list(c("A", NA, "C"), c("B", "B", NA, "A"))
impute_sequences(tree, gappy)
```

---

logLik.transitiontrees

Log-Likelihood of a context tree

Description

Returns a logLik object compatible with stats::AIC(), stats::BIC(), and the rest of the model-comparison toolchain. If newdata is NULL, returns the in-sample log-likelihood computed from the fitted node counts. Otherwise returns the held-out log-likelihood scoring newdata under the fitted tree.

Usage

```
## S3 method for class 'transitiontrees'
logLik(object, newdata = NULL, ...)
```

Arguments

object	A transitiontrees.
newdata	Optional. Sequence data in any format accepted by context_tree(). NULL (default) returns in-sample log-likelihood.
...	Ignored.

Details

Out-of-vocabulary handling: when newdata contains a state not in the tree's alphabet, the transition *into* that state is omitted from scoring (it is not penalised), and the transition *out* of it is scored against the root context (the unseen state cannot extend a history). The reported nobs therefore counts only the positions actually scored. perplexity(), score_sequences(), and score_positions() inherit the same behaviour.

Value

A logLik object with attributes nobs and df (number of free parameters in the fitted tree).

Examples

```
tree <- context_tree(matrix(sample(c("A","B","C"), 200, TRUE), 20),
                     max_depth = 2, min_count = 2)
logLik(tree)
AIC(tree); BIC(tree)
```

---

mine_contexts *Mine Contexts by Next-State Probability*

Description

Scan every context in a fitted tree for a chosen next state and return those whose predicted probability for that state falls in a requested range. A tidy context-mining table: "in which histories is the next move state unusually likely or unlikely?"

Usage

```
mine_contexts(tree, state, min_prob = NULL, max_prob = NULL, min_count = 1L)
```

Arguments

tree	A transitiontrees.
state	Character. The next state to score, one of the tree's alphabet.
min_prob, max_prob	Numeric in $[0, 1]$ or NULL. Keep contexts whose P(state | context) is at least min_prob and/or at most max_prob. NULL (default) leaves that side unbounded.
min_count	Integer. Drop contexts with fewer than this many occurrences. Default 1.

Value

A data.frame with columns `pathway`, `depth`, `count`, `state`, `prob` (`P(state | context)`), and `is_modal` (whether `state` is the context's most likely next state; ties broken by alphabet order), sorted by `prob` descending. The empty case returns a 0-row data.frame with the same schema.

Examples

```
seqs <- replicate(60, sample(c("A", "B", "C"), 10, replace = TRUE),
                  simplify = FALSE)
tree <- context_tree(seqs, max_depth = 2L)
mine_contexts(tree, state = "A", min_prob = 0.4)
```

`mine_sequences` *Mine Sequences by Predictive Surprise*

Description

Rank held-out sequences by how well the fitted tree predicts them. A tidy pattern-mining table: surface the subsequences the model finds most *surprising* (poor fit, high perplexity) or most *expected* (good fit, low perplexity).

Usage

```
mine_sequences(tree, newdata, n = 10L, which = c("surprising", "expected"))
```

Arguments

`tree`	A `transitiontrees`.
`newdata`	Sequence data in any format accepted by `context_tree()`.
`n`	Integer. Number of sequences to return. Default 10.
`which`	One of `"surprising"` (default; highest perplexity first) or `"expected"` (lowest perplexity first).

Value

A data.frame with the `score_sequences` columns (`sequence_id`, `n_scored`, `log_lik`, `perplexity`), the top `n` by the chosen direction.

Examples

```
fit  <- replicate(60, sample(c("A", "B", "C"), 10, replace = TRUE),
                  simplify = FALSE)
tree <- context_tree(fit, max_depth = 2L)
new  <- replicate(20, sample(c("A", "B", "C"), 10, replace = TRUE),
                  simplify = FALSE)
mine_sequences(tree, new, n = 5, which = "surprising")
```

model_fit *Model-Fit Scalars in One Call*

Description

Bundles the standard goodness-of-fit scalars for a fitted tree into one tidy row: `logLik`, the parameter count `df`, the observation count `nobs`, `AIC`, `BIC`, and `perplexity`. A one-call replacement for `logLik()`; `nobs()`; `AIC()`; `BIC()`; `perplexity()`.

Usage

```
model_fit(tree, newdata = NULL)
```

Arguments

`tree`	A `transitiontrees` or `transitiontrees_group`.
`newdata`	Optional sequence data. If supplied, the scalars are evaluated on it (held-out); if `NULL` (default), in-sample.

Details

With `newdata`, every scalar is computed **out-of-sample** (`AIC`/`BIC` use the held-out deviance with the model's training `df`). A `transitiontrees_group` returns one row per group, tagged with a leading `group` column.

Value

A one-row `data.frame` (one row per group for a `transitiontrees_group`) with columns `logLik`, `df`, `nobs`, `AIC`, `BIC`, `perplexity`.

See Also

`perplexity`, `logLik.transitiontrees`.

Examples

```
seqs <- replicate(60, sample(c("A","B","C"), 12, replace = TRUE),
                  simplify = FALSE)
tree <- context_tree(seqs, max_depth = 2L, min_count = 3L)
model_fit(tree)
```

n_nodes *Number of Contexts (Nodes) in a Tree*

Description

The count of contexts the tree represents — an intuitive accessor for `length(tree$nodes)` (the number printed in the tree banner).

Usage

```
n_nodes(tree)
```

Arguments

`tree`	A `transitiontrees` or `transitiontrees_group`.

Value

An integer. For a `transitiontrees_group`, a named integer vector with one count per group.

Examples

```
seqs <- replicate(50, sample(c("A","B","C"), 12, replace = TRUE),
                  simplify = FALSE)
n_nodes(context_tree(seqs, max_depth = 3L))
```

nobs.transitiontrees *Number of Observations Used to Fit a context tree*

Description

Number of Observations Used to Fit a context tree

Usage

```
## S3 method for class 'transitiontrees'
nobs(object, ...)
```

Arguments

`object`	A `transitiontrees`.
`...`	Ignored.

Value

Integer. The number of state observations the tree was fitted on — the (weight-adjusted) token total, equal to the "nobs" attribute of the *in-sample* logLik.transitiontrees(). (A held-out logLik(newdata) reports its own "nobs" — the positions actually scored, e.g. excluding states outside the tree's alphabet — which can be smaller.)

---

pathway_exists *Test Whether a Pathway Exists in the Tree*

Description

Test Whether a Pathway Exists in the Tree

Usage

```
pathway_exists(tree, pathway)
```

Arguments

tree	A transitiontrees.
pathway	Character. Pathway as arrow-notation string or character vector.

Value

Logical scalar.

Examples

```
tr <- context_tree(matrix(sample(c("A","B"), 50, TRUE), 5),
                   max_depth = 2L, min_count = 1L)
pathway_exists(tr, "A")
```

---

perplexity *Perplexity of a context tree*

Description

exp(-mean log-likelihood per observation), the standard language-modelling evaluation metric. Lower is better. A perplexity of k on an alphabet of size $|S|$ means the model is as predictive as a uniform distribution over k symbols. $k = |S|$ is the uniform baseline; $k = 1$ is perfect deterministic prediction.

Usage

```
perplexity(tree, newdata = NULL)
```

Arguments

tree	A transitiontrees.
newdata	Sequence data; NULL (default) returns in-sample perplexity.

Value

Numeric scalar.

Examples

```
tree <- context_tree(matrix(sample(c("A","B","C"), 200, TRUE), 20),
                     max_depth = 2, min_count = 2)
perplexity(tree)
```

---

plot.transitiontrees *Plot a Context Tree*

Description

Renders a fitted transitiontrees in one of four styles:

- "horizontal" (default) — pure-ggplot2 left-to-right phylogram: root on the left, contexts fanned out vertically to the right, each labelled beneath with its full arrow-notation context and (by default) its modal prediction "(state pct%)" on a second line. Node fill is the *most recent* move (rightmost token), so each branch off a depth-1 hub shares a colour. Set show_prediction = FALSE for context-only labels.
- "dendrogram" — pure-ggplot2 radial tree: root at the centre, leaves on the outer ring.
- "icicle" — pure-ggplot2 circular partition / sunburst; inner ring carries full state names, outer rings carry 3-letter abbreviations.
- "interactive" — visNetwork htmlwidget (Suggests). A draggable, zoomable hierarchical tree; node size = context count and edge width = child's count ("flow"), the same encoding as the static styles. Hover for a tooltip with the full pathway, count, modal next state, and the complete next-state distribution.

Common encoding: node size = context count, edge thickness = child's count ("flow"). Node fill is the most recent move (rightmost token of the context) in the "horizontal" style; the "dendrogram", "icicle", and "interactive" styles colour by the branching (oldest) token.

Usage

```
## S3 method for class 'transitiontrees'
plot(
  x,
  style = c("horizontal", "dendrogram", "icicle", "interactive"),
  point_size_range = NULL,
  edge_size_range = NULL,
  ...
)
```

Arguments

x	A transitiontrees.
style	One of "horizontal" (default), "dendrogram", "icicle", or "interactive".
point_size_range	Numeric length-2 vector controlling the minimum and maximum node-point size. Default c(5, 16) for "dendrogram", c(4, 14) for "horizontal", c(10, 45) (pixels) for "interactive". Ignored by "icicle".
edge_size_range	Numeric length-2 vector for edge width. Default c(0.3, 2.5) for the static styles, c(1, 10) (pixels) for "interactive". Ignored by "icicle".
...	Passed to the chosen backend. For style = "horizontal", show_prediction (logical, default TRUE) toggles each node's modal prediction "(state pct%)" on a second label line; set FALSE for context-only labels. For style = "interactive", width / height size the htmlwidget.

Details

The three static styles ("horizontal", "dendrogram", "icicle") are drawn in pure **ggplot2**. "interactive" requires visNetwork; the dispatcher errors informatively if it is missing.

Value

A ggplot object for the three static styles; an htmlwidget for "interactive".

Examples

```
set.seed(1)
m <- matrix(sample(c("A","B","C"), 200, TRUE), 20)
tr <- context_tree(m, max_depth = 2L, min_count = 3L)
plot(tr)                          # left-to-right phylogram (default)
plot(tr, style = "dendrogram")    # radial dendrogram
plot(tr, style = "icicle")        # sunburst
if (requireNamespace("visNetwork", quietly = TRUE))
  plot(tr, style = "interactive")  # visNetwork htmlwidget (Suggests)
```

---

plot.transitiontrees_bootstrap

Plot a context tree Bootstrap

Description

Forest plot of per-pathway G^2 (likelihood-ratio against parent) with bootstrap 95% CI bars. Pathways are ordered with *stable & informative* ones first, then by stability rate. The chi-square critical value at alpha_g2 is shown as a dashed reference line: pathways whose CI lies entirely above it are reproducibly informative.

Usage

```
## S3 method for class 'transitiontrees_bootstrap'
plot(x, top = 25L, min_stability = NULL, ...)
```

Arguments

x	A transitiontrees_bootstrap object.
top	Integer. Maximum pathways to show. Default 25.
min_stability	Numeric. Minimum stability_rate to display. Default NULL (use x$stability_threshold).
...	Ignored.

Value

A ggplot object.

Examples

```
seqs <- replicate(40, sample(c("A", "B", "C"), 10, replace = TRUE),
                  simplify = FALSE)
boot <- bootstrap_pathways(context_tree(seqs, max_depth = 1L),
                           iter = 50L)
plot(boot)
```

---

plot.transitiontrees_comparison

Plot a context tree Comparison

Description

Visualises the permutation-test result. Histogram of the null distribution of transitiontrees distances under shuffled group labels; a vertical line marks the observed distance; the panel header carries the p-value.

Usage

```
## S3 method for class 'transitiontrees_comparison'
plot(x, bins = 30L, ...)
```

Arguments

x	A transitiontrees_comparison object.
bins	Integer. Histogram bins. Default 30.
...	Ignored.

Value

A ggplot object.

plot.transitiontrees_group *Plot Each Tree in a context tree Group*

Description

Draw every member of a `transitiontrees_group` in turn via `plot.transitiontrees`. Each member's plot is printed (so the call produces one figure per group, e.g. in an R Markdown chunk), captioned with its group name; the named list of plot objects is returned invisibly for further use.

Usage

```
## S3 method for class 'transitiontrees_group'
plot(x, ...)
```

Arguments

`x`	A `transitiontrees_group`.
`...`	Passed to `plot.transitiontrees` (e.g. `style`).

Value

Invisibly, a named list (one entry per group, in group order) of the per-member plot objects.

Examples

```
m   <- matrix(sample(c("A","B","C"), 200, replace = TRUE), 40, 5)
grp <- context_tree(m, group = rep(c("x","y"), each = 20),
                    max_depth = 2L)
plot(grp)                          # one tree per group
```

plot.transitiontrees_group_comparison *Plot a Group Comparison*

Description

Plot a Group Comparison

Usage

```
## S3 method for class 'transitiontrees_group_comparison'
plot(x, style = c("divergence", "matrix"), top = 15L, alpha = 0.05, ...)
```

Arguments

x	A transitiontrees_group_comparison.
style	One of "divergence" (default; top pathways ranked by behavioral JSD, significant ones highlighted) or "matrix" (heatmap of the between-group symmetric-KL distance matrix).
top	Integer. Pathways to show in "divergence". Default 15.
alpha	Numeric. FDR cutoff for highlighting. Default 0.05.
...	Ignored.

Value

A ggplot object.

---

plot.transitiontrees_tune *Plot a context tree CV Grid*

Description

Visualises the held-out perplexity surface returned by tune_tree(). Lines track perplexity vs. max_depth; facets split by smoothing scheme and prune; colour encodes nmin. The minimum-perplexity configuration is highlighted with a star.

Usage

```
## S3 method for class 'transitiontrees_tune'
plot(x, ...)
```

Arguments

x	A transitiontrees_tune object.
...	Ignored.

Value

A ggplot object.

plot_difference *Difference (Subtraction) Map Between Two Groups*

Description

A per-context map of how two groups differ in their next-state predictions: one row per shared context, one column per next state. By default each cell is the Pearson standardized residual of the first group against the no-difference null (red = more than expected, blue = less; `|r| > 2` notable), which is support-aware and decomposes the per-context G^2; `measure = "probability"` shows the raw `P(group1) - P(group2)` instead.

Usage

```
plot_difference(
  group,
  groups = NULL,
  depth = NULL,
  min_count = 1L,
  comparison = NULL,
  alpha = 0.05,
  annotate = TRUE,
  layout = c("tile", "tree"),
  measure = c("residual", "probability")
)
```

Arguments

group	A `transitiontrees_group`.
groups	Optional length-2 character vector naming the two groups to subtract (`group1 - group2`). Defaults to the first two; it is required when the object has more than two groups.
depth	Integer or `NULL`. Restrict to contexts of this depth (`depth = 1` gives the order-1 transition-matrix difference). `NULL` (default) uses all shared contexts.
min_count	Integer. Drop contexts whose count in *either* group is below this. Default 1.
comparison	Optional `transitiontrees_group_comparison` (from `compare_groups()`). When supplied, contexts whose behavioral difference is significant (`jsd_padj < alpha`) are starred, so the map shows which differences survived the permutation test.
alpha	Numeric. FDR threshold for the significance stars when `comparison` is given. Default 0.05.
annotate	Logical. Print the signed difference in each cell (`layout = "tile"` only). Default `TRUE`.
layout	One of `"tile"` (default; per-context heatmap over all shared contexts) or `"tree"` (the horizontal context-tree phylogram drawn on a pooled backbone, with each node and branch coloured by which group reaches that context more — red for the first group, blue for the second; node size = pooled count).

measure — For `layout = "tile"`, what each cell encodes: `"residual"` (default) is the Pearson standardized residual of the first group against the no-group null (observed vs expected from the context's margins) — support-aware and decomposing the per-context G^2; `"probability"` is the raw next-state probability difference `P(group1) - P(group2)`. Ignored by `"tree"`.

Value

A ggplot object.

See Also

`compare_groups` for the significance test.

Examples

```
gx <- replicate(60, sample(c("A","B","C"), 8, replace = TRUE,
                           prob = c(.2,.6,.2)), simplify = FALSE)
gy <- replicate(60, sample(c("A","B","C"), 8, replace = TRUE,
                           prob = c(.2,.2,.6)), simplify = FALSE)
grp <- context_tree(c(gx, gy), group = rep(c("x","y"), each = 60),
                    max_depth = 2L)
plot_difference(grp)                      # residual heatmap
plot_difference(grp, layout = "tree")     # difference on the tree map
```

---

plot_distributions *Per-Context Next-State Distributions*

Description

Small-multiples bar chart of the full next-state distribution `P(next | context)`, one panel per context. A per-context probability display: where `plot_pathways()` renders the same numbers as a heatmap, this shows each context's distribution as its own panel, with the modal bar highlighted.

Usage

```
plot_distributions(tree, contexts = NULL, top = 12L, min_count = 1L)
```

Arguments

tree — A `transitiontrees`.

contexts — Character vector of pathway strings to show, or `NULL` (default) to take the `top` most frequent contexts.

top — Integer. Number of contexts to show when `contexts` is `NULL`. Default 12.

min_count — Integer. Drop contexts below this count. Default 1.

Value

A ggplot object.

Examples

```
seqs <- replicate(60, sample(c("A", "B", "C"), 10, replace = TRUE),
                  simplify = FALSE)
tree <- context_tree(seqs, max_depth = 2L)
plot_distributions(tree, top = 9)
```

---

plot_divergence *Lollipop Chart of Pathway Divergence*

Description

Lollipop chart of per-pathway Kullback-Leibler divergence from the (k-1)-suffix. Point size is proportional to pathway count; orange points mark pathways whose modal next state flips between orders. Annotates each flip with the prediction change, e.g. "Disengaged -> Active".

Usage

```
plot_divergence(tree, top = 15L, min_count = 5L, title = NULL, ...)
```

Arguments

tree	A transitiontrees.
top	Integer. Number of pathways to show. Default 15.
min_count	Integer. Drop pathways below this count. Default 5.
title	Character. Plot title; if NULL (default) a default title is used.
...	Ignored.

Value

A ggplot object.

Examples

```
seqs <- replicate(50, sample(c("A", "B", "C"), 12, replace = TRUE),
                  simplify = FALSE)
tree <- context_tree(seqs, max_depth = 3L)
plot_divergence(tree)
```

plot_pathway_resamples *Plot Bootstrap Resample Distributions per Pathway*

Description

Faceted histogram of the bootstrap resample values for a chosen pathway statistic, one panel per pathway.

Usage

```
plot_pathway_resamples(
  x,
  pathways = NULL,
  stat = c("count", "next_probability", "divergence", "G2"),
  top = 6L,
  bins = 30L
)
```

Arguments

x	A transitiontrees_bootstrap object.
pathways	Character vector of pathway names. NULL (default) picks the top top pathways from the summary.
stat	Character. One of "count" (default), "next_probability", "divergence", "G2".
top	Integer. Default 6.
bins	Integer. Histogram bins. Default 30.

Value

A ggplot object.

Examples

```
seqs <- replicate(40, sample(c("A", "B", "C"), 10, replace = TRUE),
                  simplify = FALSE)
boot <- bootstrap_pathways(context_tree(seqs, max_depth = 1L),
                           iter = 50L)
plot_pathway_resamples(boot, stat = "count")
```

plot_pathways *Plot Pathways as a Probability Heatmap*

Description

Heatmap visualisation of the pathway table: rows are pathways (sorted), columns are next-state probabilities under the fitted tree. The modal next state of each row is annotated in bold; rows whose modal next state *flips* relative to their parent pathway are flagged in the row labels (with a leading caret >). A side strip on the left encodes the pathway count on a log scale.

This is the natural pathway-focused visualisation: one glance shows which pathways are common, which are sharp (high mass on a single next state), which are diffuse (mass spread evenly), and which carry trajectory-specific structure that order-1 misses.

Usage

```
plot_pathways(
  tree,
  top = 20L,
  sort_by = c("count", "divergence", "depth"),
  min_count = 5L,
  show_flips = TRUE,
  title = NULL,
  ...
)
```

Arguments

tree	A transitiontrees.
top	Integer. Maximum number of pathways to show. Default 20.
sort_by	Character. One of "count" (default), "divergence", or "depth".
min_count	Integer. Drop pathways with fewer than this many occurrences. Default 5.
show_flips	Logical. Mark modal-flip pathways with a leading caret in the label. Default TRUE.
title	Character. Plot title; if NULL (default) a title is derived from sort_by.
...	Ignored.

Value

A ggplot object.

Examples

```
seqs <- replicate(50, sample(c("A","B","C"), 12, replace = TRUE),
                  simplify = FALSE)
tree <- context_tree(seqs, max_depth = 3)
```

```
plot_pathways(tree)
plot_pathways(tree, sort_by = "divergence", top = 12)
```

`plot_predictive` *Predictive Diagnostics for Held-Out Scoring*

Description

Visual diagnostics for how a fitted tree scores held-out sequences, built on `score_positions()`:

`type = "position"` predicted probability of the observed next state against position in the sequence — where the model is confident vs. surprised as a sequence unfolds.

`type = "ecdf"` the empirical cumulative distribution of those predicted probabilities — a calibration-style view of how often the model assigns high vs. low probability to what actually happened.

`type = "logloss"` the per-position log-loss $-\log_2 P(\text{observed})$ against position — a per-position log-loss view. Lower is better (0 = certain and correct); the dashed line is the mean log-loss over all scored positions.

Usage

```
plot_predictive(tree, newdata, type = c("position", "ecdf", "logloss"))
```

Arguments

`tree`	A `transitiontrees`.
`newdata`	Held-out sequence data in any format accepted by `context_tree()`.
`type`	One of `"position"` (default), `"ecdf"`, or `"logloss"`.

Value

A ggplot object.

Examples

```
fit  <- replicate(60, sample(c("A", "B", "C"), 10, replace = TRUE),
                  simplify = FALSE)
tree <- context_tree(fit, max_depth = 2L)
new  <- replicate(15, sample(c("A", "B", "C"), 10, replace = TRUE),
                  simplify = FALSE)
plot_predictive(tree, new, type = "position")
plot_predictive(tree, new, type = "ecdf")
plot_predictive(tree, new, type = "logloss")
```

`plot_pruning` *Illustrate Pruning Along a Pathway's Suffix Chain*

Description

A suffix-chain pruning view: take one pathway and show, side by side, the next-state distribution at every context along its suffix chain — the full context, then the context with its oldest move dropped, and so on down to the root — marking which contexts `prune_tree` (criterion `"G2"`) keeps versus prunes. It answers "how much memory does this pathway actually need?": each context is drawn as its own panel (deepest memory on the left, root on the right) and classified into three states by opacity. **Solid** contexts are *informative* — their own G^2 clears the cutoff, so they add predictive information over their one-shorter parent. **Mid-opacity** contexts are *retained*: their own G^2 is below the cutoff, but a deeper context diverges, so `prune_tree()` keeps them only as a structural bridge — they do not themselves add memory. **Faded** contexts are *pruned* (the redundant tail). The panel title carries the full context, the decision, and the G^2. The requested pathway must be a fitted context; the function errors rather than silently plotting a shorter suffix.

Usage

```
plot_pruning(tree, pathway, alpha = 0.05)
```

Arguments

`tree`	A `transitiontrees` (typically *unpruned*, so the full chain is visible).
`pathway`	A single pathway string in arrow form (`"A -> B -> C"`, oldest on the left).
`alpha`	Significance level for the `G2` keep/prune decision. Default 0.05.

Details

The keep/prune decision is exactly `prune_tree`'s G2 rule ($2N \cdot \mathrm{KL} > \chi^2_{1-\alpha,k-1}$); the cumulative `pruned` flag follows the leaf-up amnesia rule. The distributions and counts shown are the same node values reported throughout the pathway API (see `PARITY.md`).

Value

A ggplot object.

See Also

`prune_tree`, `plot_distributions`.

Examples

```
seqs <- replicate(80, sample(c("A","B","C"), 12, replace = TRUE),
                  simplify = FALSE)
tree <- context_tree(seqs, max_depth = 3L, min_count = 3L)
plot_pruning(tree, "A -> B -> C")
```

`plot_trajectories` *Forward Trajectory Tree (Prefix Tree)*

Description

Where `plot.transitiontrees` draws the fitted context tree *backwards* (each node is a suffix, the most-recent move), this draws the same prompts *forwards*: a prefix tree that starts at a common root and follows each sequence move by move in time. The one tree can be coloured two ways:

`measure = "frequency"` node and edge colour and width encode how many sequences walk each path.

`measure = "predictability"` colour encodes $P(\text{move} \mid \text{history})$ from the model (`tree`); edge width still encodes flow.

Higher values are drawn darker.

Usage

```
plot_trajectories(
  tree,
  measure = c("frequency", "predictability"),
  min_count = 4L
)
```

Arguments

`tree`	A `transitiontrees` (pass a pruned tree to read predictability off the pruned model).
`measure`	One of `"frequency"` (default) or `"predictability"`.
`min_count`	Integer. Keep only prefixes occurring at least this many times. Default 4.

Details

The predictability of a node's last move is the model's conditional probability of that move given the preceding history, truncated to the tree's `max_depth` and read via `query_pathway()` (the empty history for a first move uses the root distribution). This is the forward-reading complement to the backward context tree. It is drawn entirely in **ggplot2**, with no additional plotting dependency.

Value

A ggplot object.

See Also

`plot.transitiontrees` for the backward context tree, `plot_pruning` for the suffix-chain view.

Examples

```
seqs <- replicate(120, sample(c("A", "B", "C"), 8, replace = TRUE),
                  simplify = FALSE)
tree   <- context_tree(seqs, max_depth = 3L, min_count = 3L)
pruned <- prune_tree(tree)
plot_trajectories(tree,   measure = "frequency")
plot_trajectories(pruned, measure = "predictability")
```

`predict.transitiontrees` *Predict Next-State Probabilities from a Context Tree*

Description

Predict Next-State Probabilities from a Context Tree

Usage

```
## S3 method for class 'transitiontrees'
predict(object, newdata, type = c("prob", "class"), ...)
```

Arguments

`object`	A `transitiontrees`.
`newdata`	Either (i) a list of character vectors (each is the "history" leading up to the prediction point), (ii) a wide data.frame / matrix whose rows are histories, or (iii) a single character vector treated as one history.
`type`	One of `"prob"` (default; named numeric matrix of next-state probabilities) or `"class"` (character vector of modal predictions).
`...`	Ignored.

Value

If `type = "prob"`: a matrix with one row per history and one column per state. A list/data.frame/matrix `newdata` always returns a matrix (1 x k for a single-history container); a bare character vector returns a named vector for interactive convenience. If `type = "class"`: a character vector of modal predictions.

Examples

```
seqs <- replicate(50, sample(c("A","B","C"), 12, replace = TRUE),
                  simplify = FALSE)
tree <- context_tree(seqs, max_depth = 3)
predict(tree, newdata = list(c("A","B"), c("C","C","B")))
predict(tree, newdata = list(c("A","B")), type = "class")
predict(tree, newdata = c("A","B"))   # bare vector → named vector
```

prepare_input *Reshape Long Event Data into a Wide Sequence Frame*

Description

Turns a long, one-row-per-event table into the wide, one-row-per- sequence character frame that `context_tree()` consumes. Events are grouped by `actor` and ordered by `time` (or `order`); when `time` is given, each actor's events are split into **sessions** whenever the gap to the previous event exceeds `time_threshold` seconds. This mirrors the standard timestamp / session-splitting rule, in pure base R.

Usage

```
prepare_input(
  data,
  actor = NULL,
  time = NULL,
  action = NULL,
  order = NULL,
  session = NULL,
  time_threshold = 900,
  format = NULL,
  is_unix_time = FALSE,
  unix_time_unit = c("seconds", "milliseconds", "microseconds"),
  meta = NULL
)
```

Arguments

`data`	A long-format `data.frame`, one row per event.
`actor`	Character. Column(s) naming the unit each sequence belongs to (e.g. a user id). Several columns are combined with `"-"`. `NULL` (default) treats the whole table as one actor.
`time`	Character. Column holding the event timestamp, used both to order events and to split sessions by `time_threshold`. Numeric values are read as Unix time. `NULL` (default) uses `order` (or row order) instead and does no session splitting.
`action`	Character. Column holding the event's state / code — the symbol that becomes a cell of the sequence. Required.
`order`	Character. Optional column giving an explicit within- actor ordering (used when `time` is absent, or to break time ties). Defaults to row order.
`session`	Character. Optional column giving an explicit session id within an actor. If supplied, sessions are taken from it directly and no time-gap splitting is done.
`time_threshold`	Numeric. Seconds; a gap larger than this starts a new session. Default `900` (15 minutes), the standard session-splitting threshold.
`format`	Character. Optional `strptime` format for a character `time` column.

`is_unix_time`	Logical. Force `time` to be read as Unix time. Default `FALSE`.
`unix_time_unit`	One of `"seconds"` (default), `"milliseconds"`, `"microseconds"`.
`meta`	Optional character vector of column names to carry through the reshape as per-sequence metadata (one value per session, the first event's). Returned as a `data.frame` on the `"meta"` attribute of the result, aligned to its rows. Used by `context_tree()` to align column-name `group`/`block` to the wide rows. Default `NULL`.

Value

A wide character `data.frame`, one row per sequence (session), columns `T1`, `T2`, `...` holding the ordered states and trailing `NA`s past the end of each sequence. Row names are the session ids. When `meta` is given, an aligned per-sequence metadata `data.frame` is attached as `attr(., "meta")`. Pass it straight to `context_tree()`.

See Also

`context_tree`

Examples

```
long <- data.frame(
  user  = c("a","a","a","a","b","b"),
  t     = as.POSIXct("2020-01-01 09:00:00", tz = "UTC") +
            c(0, 60, 3600, 3660, 0, 30),
  state = c("X","Y","X","Z","Y","X"),
  stringsAsFactors = FALSE)
## one-hour gap splits user a into two sessions
wide <- prepare_input(long, actor = "user", time = "t", action = "state")
wide
context_tree(wide, max_depth = 2L, min_count = 1L)
```

---

print.summary.transitiontrees

Print a Context-Tree Summary

Description

Print method for the object returned by `summary.transitiontrees`: a one-line banner followed by the leading rows of the canonical pathway table.

Usage

```
## S3 method for class 'summary.transitiontrees'
print(x, n = 10L, ...)
```

Arguments

x	A summary.transitiontrees object.
n	Integer. Number of pathway rows to print. Default 10.
...	Ignored.

Value

x invisibly.

---

print.transitiontrees *Print a Context Tree*

Description

Print a Context Tree

Usage

```
## S3 method for class 'transitiontrees'
print(x, max_lines = 25L, digits = 2L, ...)
```

Arguments

x	A transitiontrees.
max_lines	Integer. Maximum tree-rendering lines. Default 25.
digits	Integer. Probability digits. Default 2.
...	Ignored.

Value

x invisibly.

---

print.transitiontrees_bootstrap *Print a context tree Bootstrap*

Description

Print a context tree Bootstrap

Usage

```
## S3 method for class 'transitiontrees_bootstrap'
print(x, n = 10L, digits = 3L, ...)
```

Arguments

x	A transitiontrees_bootstrap object.
n	Integer. Number of top pathways to print. Default 10.
digits	Integer. Numeric digits for the printed table. Default 3.
...	Ignored.

Value

x invisibly.

---

print.transitiontrees_comparison

Print a context tree Comparison

Description

Print a context tree Comparison

Usage

```
## S3 method for class 'transitiontrees_comparison'
print(x, digits = 3L, n = 6L, ...)
```

Arguments

x	A transitiontrees_comparison object.
digits	Integer. Numeric digits for the printed table. Default 3.
n	Integer. Number of top divergent pathways to print. Default 6.
...	Ignored.

Value

x invisibly.

print.transitiontrees_group *Print a Group of Context Trees*

Description

Print a Group of Context Trees

Usage

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

Arguments

x	A transitiontrees_group (named list of transitiontreess).
...	Ignored.

Value

x invisibly.

print.transitiontrees_group_comparison *Print a Group Comparison*

Description

Print method for a transitiontrees_group_comparison: a header with the groups and permutation setup, the omnibus behavioral / usage statistics, and the top pathways ranked by behavioral divergence.

Usage

```
## S3 method for class 'transitiontrees_group_comparison'
print(x, n = 10L, digits = 3L, ...)
```

Arguments

x	A transitiontrees_group_comparison object.
n	Integer. Number of top pathways to print. Default 10.
digits	Integer. Numeric digits for the printed tables. Default 3.
...	Ignored.

Value

x invisibly.

print.transitiontrees_tune *Print a context tree Tuning Grid*

Description

Print a context tree Tuning Grid

Usage

```
## S3 method for class 'transitiontrees_tune'
print(x, n = 10L, ...)
```

Arguments

x	A transitiontrees_tune object.
n	Integer. Number of configurations to print. Default 10.
...	Ignored.

Value

x invisibly.

prune_tree *Prune a Context Tree*

Description

Removes nodes that do not earn their depth under the chosen criterion. Pruning is applied bottom-up: a node is dropped when extending its parent's prediction at this context produces less information / likelihood than the depth penalty allows.

Renamed from prune() to avoid collision with other prune() generics.

Usage

```
prune_tree(
  tree,
  criterion = c("G2", "KL", "AIC", "BIC"),
  alpha = 0.05,
  threshold = 0.005
)
```

Arguments

tree	A transitiontrees, or a transitiontrees_group, in which case each member is pruned and the group wrapper is preserved.
criterion	One of "G2" (likelihood-ratio test against parent; default), "KL" (per-context Kullback-Leibler against parent), "AIC" (Akaike penalty), "BIC" (Bayesian penalty). Case-sensitive.
alpha	Numeric in (0, 1). Significance level for "G2"; ignored otherwise. Default 0.05.
threshold	Numeric. Minimum information gain in nats for "KL"; ignored otherwise. Default 0.005.

Details

For each leaf, compute the criterion against its parent. If the criterion does not exceed its threshold, drop the leaf and revisit the parent. Repeat until stable. The root is never dropped. Surviving nodes keep their original smoothed prob vector (whatever smoothing scheme was applied at fit time).

Note on units: the "KL" threshold is in **nats** (natural log), whereas the divergence column reported by tree_pathways() / divergent_pathways() is in **bits** (log base 2). Multiply a nats threshold by 1 / log(2) (~1.4427) to read it on the pathway-table scale.

Value

A pruned transitiontrees with tree$pruned = TRUE and tree$pruning carrying the criterion + threshold settings.

References

Ron, D., Singer, Y., Tishby, N. (1996). The power of amnesia. *Machine Learning*, 25, 117-149.

Examples

```
seqs   <- replicate(50, sample(c("A","B","C"), 12, replace = TRUE),
                    simplify = FALSE)
tree   <- context_tree(seqs, max_depth = 4)
pruned <- prune_tree(tree, criterion = "G2", alpha = 0.05)
```

---

query_pathway *Query the Probability of a Specific Pathway -> Next State*

Description

Returns the probability the fitted tree assigns to a given pathway / next-state pair. Two lookup modes:

- exact = TRUE: the pathway must appear as a node; otherwise returns NA.
- exact = FALSE (default): if the pathway is missing, falls back to the longest matching suffix that *is* in the tree (mirrors predict.transitiontrees()).

Usage

```
query_pathway(tree, pathway, next_state = NULL, exact = FALSE)
```

Arguments

tree	A transitiontrees.
pathway	Character. The conditioning pathway, either as a single arrow-notation string ("A -> B -> C") or as a character vector of states (c("A","B","C")).
next_state	Character. Next-state symbol to query, or NULL (default) to return the full conditional distribution.
exact	Logical. Default FALSE — fall back to longest matching suffix.

Value

If next_state is supplied, a numeric scalar. Otherwise a named numeric vector indexed by alphabet.

Examples

```
set.seed(1)
m <- matrix(sample(c("A","B","C"), 200, TRUE), 20)
tr <- context_tree(m, max_depth = 2L, min_count = 3L)
query_pathway(tr, c("A","B"))
query_pathway(tr, "A -> B", next_state = "C")
```

---

score_positions *Per-Position Scoring*

Description

Returns one row per held-out (sequence, position) with the matched context, predicted probability of the observed next state, and log-likelihood contribution. Useful for diagnostic plots showing where the model is confident vs. surprised.

Usage

```
score_positions(tree, newdata, worst = NULL)
```

Arguments

tree	A transitiontrees.
newdata	Sequence data.
worst	Integer or NULL. If given, return only the worst positions — those with the lowest predicted_prob (the moves the model was most surprised by). Default NULL (all positions, in sequence order).

Value

A data.frame with columns `sequence_id`, `position`, `matched_context`, `observed`, `predicted_prob`, `log_lik`.

Examples

```
fit  <- replicate(40, sample(c("A", "B", "C"), 10, replace = TRUE),
                  simplify = FALSE)
tree <- context_tree(fit, max_depth = 1L)
new  <- replicate(5, sample(c("A", "B", "C"), 10, replace = TRUE),
                  simplify = FALSE)
score_positions(tree, new, worst = 5L)
```

---

score_sequences *Per-Sequence Scoring*

Description

Returns one row per held-out sequence with its log-likelihood, number of scored positions, and per-sequence perplexity (`exp(-log_lik / n_scored)`).

Usage

```
score_sequences(tree, newdata)
```

Arguments

`tree`	A `transitiontrees`.
`newdata`	Sequence data.

Value

A data.frame with columns `sequence_id`, `n_scored`, `log_lik`, `perplexity`.

Examples

```
fit  <- replicate(40, sample(c("A", "B", "C"), 10, replace = TRUE),
                  simplify = FALSE)
tree <- context_tree(fit, max_depth = 1L)
new  <- replicate(5, sample(c("A", "B", "C"), 10, replace = TRUE),
                  simplify = FALSE)
score_sequences(tree, new)
```

sharp_pathways *Sharpest Pathways*

Description

Returns the top n pathways by predictive sharpness – the probability mass on their modal next state. High values indicate strongly deterministic continuations; low values indicate ambiguous next-state distributions.

Usage

```
sharp_pathways(tree, top = 10L, min_count = 1L)
```

Arguments

tree	A transitiontrees.
top	Integer. Number of pathways to return. Default 10.
min_count	Integer. Drop pathways with fewer than this many occurrences. Default 1.

Value

A data.frame, same columns as tree_pathways, sorted by next_probability descending.

Examples

```
seqs <- replicate(50, sample(c("A","B","C"), 12, replace = TRUE),
                  simplify = FALSE)
tree <- context_tree(seqs, max_depth = 3)
sharp_pathways(tree, top = 5)
```

simulate.transitiontrees *Simulate Sequences from a Fitted context tree*

Description

S3 simulate() method for transitiontrees objects. Wraps generate_sequences with the standard nsim argument name and an optional seed (set via set.seed() when supplied).

Usage

```
## S3 method for class 'transitiontrees'
simulate(object, nsim = 5L, seed = NULL, length = 10L, start = NULL, ...)
```

Arguments

object	A transitiontrees.
nsim	Integer. Number of sequences to simulate. Default 5.
seed	Integer or NULL. Optional RNG seed.
length	Integer. Length of each simulated sequence.
start	NULL or character vector. Optional first state for each sequence; see generate_sequences.
...	Ignored.

Value

A character matrix of dimension nsim x length.

---

smooth_tree *Re-Smooth a Fitted context tree*

Description

Replaces every node's probability vector with a new smoothing scheme without refitting the tree. Walks nodes top-down by depth so each node's parent is re-smoothed before its children read it.

Usage

```
smooth_tree(tree, smoothing = "floor")
```

Arguments

tree	A transitiontrees.
smoothing	Smoothing specification: either a method name as a string (uses defaults for that method's hyperparameters) or a list of the form list(method, ...kwargs) for explicit hyperparameters. Available methods: "floor" (ymin = 0.001), "laplace" (alpha = 1), "kneser_ney" (discount = 0.75), "witten_bell", "jelinek_mercer" (lambda = 0.5).

Details

For "kneser_ney" the canonical continuation-distribution formulation requires per-state *type counts*. transitiontrees does not track these; the implementation uses the parent's smoothed probability as the back-off distribution, an approximation discussed in Begleiter, El-Yaniv & Yona (2004), *JAIR* 22, §3.

Value

A new transitiontrees with re-smoothed probabilities. Counts and topology are unchanged.

Examples

```
set.seed(1)
m  <- matrix(sample(c("A","B","C"), 200, TRUE), 20)
tr <- context_tree(m, max_depth = 2L, min_count = 3L)
smooth_tree(tr, "kneser_ney")
smooth_tree(tr, list("kneser_ney", discount = 0.5))
```

`subtree` *Extract the Subtree Rooted at a Pathway*

Description

Returns a new `transitiontrees` containing only the queried node and its descendants. Node names are kept absolute (so the original pathway is preserved as a key), but the returned object has a `local_root` attribute pointing at the queried pathway.

Usage

```
subtree(tree, pathway)
```

Arguments

`tree`	A `transitiontrees`.
`pathway`	Character. The root pathway (arrow-notation or character vector). Must exist in the tree.

Value

A new `transitiontrees` whose nodes and edges are restricted to descendants of `pathway`. The alphabet, smoothing, and other hyperparameters are copied unchanged. Printing the subtree reports the context it was cut at (also stored in `attr(., "local_root")` for programmatic use).

Examples

```
set.seed(1)
m <- matrix(sample(c("A","B","C"), 200, TRUE), 20)
tr <- context_tree(m, max_depth = 2L, min_count = 3L)
subtree(tr, "A")   # prints "subtree of: A" in the banner
```

summary.transitiontrees *Summary of a Context Tree*

Description

Summary of a Context Tree

Usage

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

Arguments

object	A transitiontrees.
...	Ignored.

Value

A summary.transitiontrees object. The $table slot is the canonical pathway data.frame from tree_pathways (columns pathway, depth, count, likely_next, next_probability, divergence, changes_prediction), re-sorted by (depth, -count) so the structural tree order is read top-to-bottom.

summary.transitiontrees_bootstrap *Summarise a context tree Bootstrap*

Description

Summarise a context tree Bootstrap

Usage

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

Arguments

object	A transitiontrees_bootstrap object.
...	Ignored.

Value

The per-pathway summary data.frame (object$summary); see bootstrap_pathways for the full column vocabulary.

summary.transitiontrees_group_comparison *Summarise a Group Comparison*

Description

Prints a compact verdict for a `transitiontrees_group_comparison`: the omnibus behavioral and usage permutation p-values, and how many pathways pass the FDR cutoff on each axis or flip their modal next state between groups. Returns the per-pathway table invisibly.

Usage

```
## S3 method for class 'transitiontrees_group_comparison'
summary(object, alpha = 0.05, ...)
```

Arguments

`object`	A `transitiontrees_group_comparison` object.
`alpha`	Numeric. FDR cutoff used when counting significant pathways. Default 0.05.
`...`	Ignored.

Value

Invisibly, the per-pathway data.frame (`object$pathways`); see `compare_groups` for the column vocabulary.

trajectories *Student engagement trajectories*

Description

An example set of categorical learning-engagement trajectories used in the **transitiontrees** examples and the "trajectories" vignette. Each row is one learner, each column a time-step, and each cell the engagement state at that step. Trailing `NA`s mark the end of a trajectory. This wide character matrix is exactly the shape `context_tree()` consumes.

Usage

```
trajectories
```

Format

A character matrix with 138 rows (learners) and 15 columns (time-steps). Three states: `"Active"`, `"Average"`, `"Disengaged"`.

Source

Bundled example dataset.

Examples

```
data(trajectories)
dim(trajectories)
tree <- context_tree(trajectories)
tree
```

`tree_dependence` *Per-Context Path Dependence of a Context Tree*

Description

For each non-root node in the tree, reports the Kullback-Leibler divergence of its conditional next-state distribution against its parent's. Large values flag contexts where extending memory by one more step changes the prediction; the `changes_prediction` column flags contexts where the most likely next state changes between the node and its parent.

Renamed from `path_dependence()` to avoid a naming collision with a sibling package.

Usage

```
tree_dependence(
  tree,
  base = 2,
  sort_by = c("divergence", "entropy_drop", "entropy", "count", "depth"),
  top = NULL
)
```

Arguments

`tree`	A `transitiontrees`.
`base`	Numeric. Logarithm base for the KL divergence. Default 2 (bits). Use `exp(1)` for nats or `10` for hartleys.
`sort_by`	Character. Column to sort by, descending. One of `"divergence"` (default), `"entropy_drop"`, `"entropy"`, `"count"`, `"depth"`.
`top`	Integer or `NULL`. If given, keep only the top `top` rows after sorting. Default `NULL` (all rows).

Details

This is the *diagnostic* that the tree's pruning rule (under `criterion = "KL"`) is comparing against its threshold. It answers the substantive question: for which contexts does this tree disagree with a memoryless / shorter-memory model, and where does that disagreement actually flip the prediction?

The mean of `n * KL` across rows recovers, up to constants, the chain-level mutual-information gain from the variable-depth model over the order-1 model.

Value

A data.frame with one row per non-root pathway, sorted by `divergence` descending. Columns: `pathway`, `depth`, `count`, `divergence` (Kullback-Leibler divergence from the parent's prediction), `entropy` (Shannon entropy of this pathway's next-state distribution), `entropy_before` (entropy of the parent's distribution), `entropy_drop` (`entropy_before - entropy`, the uncertainty this step of history removes), `likely_next` (this node's most likely next state), `likely_before` (the parent context's most likely next state), `changes_prediction` (`likely_next != likely_before`). The empty case returns a 0-row data.frame with the same schema.

References

Cover, T.M. & Thomas, J.A. (2006). *Elements of Information Theory*, 2nd ed. Wiley.

Examples

```
seqs   <- replicate(50, sample(c("A","B","C"), 12, replace = TRUE),
                    simplify = FALSE)
tree   <- context_tree(seqs, max_depth = 3)
pruned <- prune_tree(tree, criterion = "G2")
tree_dependence(pruned)
```

---

tree_distance *Symmetric KL Distance Between Two context trees*

Description

Bare-metal scalar: a count-weighted average of per-context symmetric Kullback-Leibler divergence over the union of pathways present in either tree. No null distribution.

Usage

```
tree_distance(tree_a, tree_b, symmetric = TRUE)
```

Arguments

`tree_a`, `tree_b`	context trees with matching alphabets.
`symmetric`	Logical. `TRUE` (default) returns $0.5(D_{KL}(A\|B) + D_{KL}(B\|A))$; `FALSE` returns $D_{KL}(A\|B)$ only.

Value

Numeric scalar.

Examples

```
set.seed(1)
m1 <- matrix(sample(c("A","B","C"), 200, TRUE), 20)
m2 <- matrix(sample(c("A","B","C"), 200, TRUE), 20)
tr1 <- context_tree(m1, max_depth = 2L, min_count = 3L)
tr2 <- context_tree(m2, max_depth = 2L, min_count = 3L)
tree_distance(tr1, tr2)
```

`tree_pathways` *Pathways from a Fitted Tree*

Description

Returns a tidy data.frame with one row per pathway (= context) in the tree. The pathway is the sequence of states the tree conditions on; each row reports the count, depth, modal next state, and how surprising the next-state distribution is relative to a shorter history. This is the substantive consumer-facing API of a fitted tree – a ranked list of trajectories that the data actually supports, with a consistent interpretive frame.

Renamed from `pathways()` to avoid a naming collision with a sibling package.

Usage

```
tree_pathways(
  tree,
  min_count = 1L,
  sort_by = c("count", "divergence", "depth"),
  decreasing = TRUE,
  ...
)
```

Arguments

`tree`	A `transitiontrees` object.
`min_count`	Integer. Drop pathways with fewer than this many occurrences. Default 1.
`sort_by`	Character. One of `"count"` (default), `"divergence"`, or `"depth"`. Sorts the returned data.frame.
`decreasing`	Logical. Default `TRUE`.
`...`	Ignored.

Details

Each row is a pathway – a (possibly empty) sequence of states ending at the point where a prediction is made. The `divergence` column quantifies how much more information the pathway carries than its (k-1)-suffix in bits. `changes_prediction = TRUE` marks pathways where the longer history changes which next state is most likely.

Value

A data.frame with columns `pathway` (arrow notation, e.g. `"A -> B -> C"`; the root is reported as `"(start)"`), `depth` (history length), `count`, `likely_next` (the most likely next state), `next_probability` (its probability), `divergence` (Kullback-Leibler divergence from the parent context's prediction, in bits; `NA` for the root and when the parent context is absent, and `Inf` when the pathway places probability on a state the parent predicts with probability 0), and `changes_prediction` (logical, did the most likely next state change vs the parent context?). The empty case returns a 0-row data.frame with the same schema.

Examples

```
seqs <- replicate(50, sample(c("A","B","C"), 12, replace = TRUE),
                  simplify = FALSE)
tree <- context_tree(seqs, max_depth = 3)
tree_pathways(tree)
```

---

tune_tree *Cross-Validated Hyperparameter Tuning for context trees*

Description

Runs k-fold cross-validation over a grid of fitting and pruning hyperparameters. Returns a data.frame ranked by held-out perplexity. The configuration with minimum perplexity is exposed via `attr(result, "best")`.

Folds are at the sequence level (each fold holds out whole sequences, not positions within sequences).

Usage

```
tune_tree(
  data,
  max_depth = 2L:5L,
  min_count = c(3L, 5L, 10L),
  smoothing = "floor",
  prune = c(FALSE, TRUE),
  alpha = 0.05,
  folds = 5L,
  seed = 1L,
  actor = NULL,
  time = NULL,
  action = NULL,
  order = NULL,
  session = NULL,
  time_threshold = 900
)
```

Arguments

data	Sequence data; format accepted by context_tree().
max_depth	Integer vector. Grid values for tree depth. Default 2:5.
min_count	Integer vector. Grid for minimum-count threshold. Default c(3L, 5L, 10L).
smoothing	Smoothing grid. A character vector of method names (e.g. c("floor", "kneser_ney")) — each method is tried with its default hyperparameters — or a list of explicit specs (e.g. list(list("floor", ymin = 0.001), list("floor", ymin = 0.005)) for a hyperparameter sweep within one method).
prune	Logical vector. Whether to apply G^2 pruning. Default c(FALSE, TRUE).
alpha	Numeric. Significance level for G^2 pruning when prune = TRUE. Default 0.05.
folds	Integer. Number of CV folds. Default 5.
seed	Integer. RNG seed for reproducible folds. Default 1.
actor, time, action, order, session, time_threshold	Long-format reshaping, forwarded to prepare_input() when action is named (exactly as in context_tree()), so a long event log is reshaped before tuning rather than read as one row per sequence. Default NULL/900 (data already in sequence shape).

Value

A transitiontrees_tune object: a data.frame with one row per grid point and columns max_depth, nmin, smoothing, prune, logLik, n_scored, perplexity, n_nodes_avg, and folds_failed (the number of CV folds that errored for that configuration), sorted by perplexity ascending. attr(result, "best") carries the minimum-perplexity row **among configurations whose every fold scored**; it is NULL if none did. A warning is issued when any configuration had a failed fold.

Examples

```
set.seed(1)
m <- matrix(sample(c("A","B","C"), 30 * 12, replace = TRUE), 30, 12)
tune_tree(m, max_depth = 1:3,
             smoothing = c("floor", "kneser_ney"),
             prune = FALSE, folds = 4)
```

Index

∗ **datasets**
ai_long, 3
engagement, 18
group_regulation_long, 20
trajectories, 54

ai_long, 3
as.data.frame.transitiontrees, 4
as.data.frame.transitiontrees_bootstrap, 5
as.data.frame.transitiontrees_comparison, 5
as.data.frame.transitiontrees_group, 6
as.data.frame.transitiontrees_group_comparison, 6

bootstrap_pathways, *5*, 7, *53*

common_pathways, 9
compare_groups, *7*, 10, *16*, *32*, *33*, *54*
compare_pruning, 11
compare_smoothing, 12
compare_trees, *11*, 14
context_tree, *12*, *13*, 15, *20*, *41*, *42*, *54*, *59*

divergent_pathways, 17, *47*

engagement, 18

generate_sequences, 19, *50*, *51*
group_regulation_long, 20

impute_sequences, 20

logLik.transitiontrees, 21, *24*, *26*

mine_contexts, 22
mine_sequences, 23
model_fit, 24

n_nodes, 25
nobs.transitiontrees, 25

pathway_exists, 26
perplexity, *24*, 26
plot.transitiontrees, 27, *30*, *39*
plot.transitiontrees_bootstrap, 28
plot.transitiontrees_comparison, 29
plot.transitiontrees_group, 30
plot.transitiontrees_group_comparison, 30
plot.transitiontrees_tune, 31
plot_difference, 32
plot_distributions, 33, *38*
plot_divergence, 34
plot_pathway_resamples, 35
plot_pathways, *33*, 36
plot_predictive, 37
plot_pruning, 38, *39*
plot_trajectories, 39
predict.transitiontrees, 40
prepare_input, *16*, *20*, 41, *59*
print.summary.transitiontrees, 42
print.transitiontrees, 43
print.transitiontrees_bootstrap, 43
print.transitiontrees_comparison, 44
print.transitiontrees_group, 45
print.transitiontrees_group_comparison, 45
print.transitiontrees_tune, 46
prune_tree, *11*, *12*, *15–17*, *38*, 46

query_pathway, *39*, 47

score_positions, *37*, 48
score_sequences, *23*, 49
sharp_pathways, 50
simulate.transitiontrees, 50
smooth_tree, *13*, 51
strptime, *41*
subtree, 52

summary.transitiontrees, *42*, 53
summary.transitiontrees_bootstrap, 53
summary.transitiontrees_group_comparison, 54

trajectories, 54
tree_dependence, 55
tree_distance, *11*, 56
tree_pathways, *4*, *6*, *9*, *18*, *47*, *50*, *53*, 57
tune_tree, *12, 13*, 58