Package 'Proximum'

September 17, 2026

Title Statistical Analysis of Ensemble Proximity Matrices

Version 1.1.0

Description Treats the proximity matrices produced by tree ensembles as first-class statistical objects rather than as model by-products. Provides a unified extractor across ensemble engines, in-bag and out-of-bag definitions, transformations to dissimilarities with metric diagnostics, corrections that make an indefinite proximity usable as a kernel, and permutation inference for comparing two proximity matrices or partitioning one across the terms of a design. Also provides a Nystrom approximation, a thresholded sparse representation and a streaming form that never allocates the matrix at all, for samples too large to hold it, measures how far the proximity moves between replicates of the ensemble, and draws each object through 'ggplot2': the seriated matrix, the configuration it implies, and the thresholded graph with its communities.

License MIT + file LICENSE

URL https://github.com/agostinognasso/Proximum

BugReports https://github.com/agostinognasso/Proximum/issues

Depends R (>= 3.5)

Imports ggplot2, Matrix, rlang, stats, utils

Suggests covr, e2tree, igraph, knitr, randomForest, ranger, rmarkdown, seriation, testthat (>= 3.2.0), vegan

VignetteBuilder knitr

LazyData true

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Agostino Gnasso [aut, cre, cph] (ORCID: <https://orcid.org/0000-0002-8046-3923>)

Maintainer Agostino Gnasso <agostino.gnasso@unina.it>

Repository CRAN

Date/Publication 2026-09-17 11:40:02 UTC

Contents

`as.dist.proximity` *Dissimilarity induced by a proximity matrix*

Description

Converts a `proximity` object into a `stats::dist()` object using the square-root transform $D_{ij} = \sqrt{1 - P_{ij}}$, which is the transform that makes the proximity behave like a kernel. Use `as_dissimilarity()` to choose a different transform.

Usage

```
## S3 method for class 'proximity'
as.dist(m, diag = FALSE, upper = FALSE)
```

Arguments

`m`	A `proximity` object.
`diag`, `upper`	Passed to `stats::as.dist()`.

Value

A `dist` object.

`as.matrix.proximity_nystrom` *Reconstruct a Nystrom approximation as a dense matrix*

Description

Forms $LL^\top$, which is the $n \times n$ object the approximation exists to avoid, so the call is guarded rather than free. The diagonal is left as the approximation produced it: forcing it to one would hide the departure that `summary.proximity_nystrom()` reports.

Usage

```
## S3 method for class 'proximity_nystrom'
as.matrix(x, max_size = 500, ...)
```

Arguments

`x`	A `proximity_nystrom` object.
`max_size`	Largest matrix to reconstruct, in megabytes. Raise it deliberately.
`...`	Unused.

Value

A dense symmetric numeric matrix.

`as.matrix.proximity_sparse` *Densify a sparse proximity matrix*

Description

Returns the dense matrix, with the entries that were below the threshold as exact zeros. The values above it are unchanged, so the round trip is lossless where it kept anything and lossy exactly where it said it would be.

Usage

```
## S3 method for class 'proximity_sparse'
as.matrix(x, ...)
```

Arguments

x	A proximity_sparse object.
...	Unused.

Details

This is the deliberate way to spend the memory the thresholding saved. At n = 800 it was 0.6 MB in and 5.0 MB out.

Value

A dense symmetric numeric matrix.

---

as_dissimilarity *Transform a proximity into a dissimilarity*

Description

Both transforms are monotone decreasing in the proximity, but only the square-root one is guaranteed to behave like a metric: $1 - P$ can violate the triangle inequality, while $\sqrt{1 - P}$ is Euclidean whenever P is positive semi-definite. Use is_euclidean() to check the result rather than assuming it.

Usage

```
as_dissimilarity(px, transform = c("sqrt", "linear"))
```

Arguments

px	A proximity object, or a symmetric numeric matrix.
transform	"sqrt" for $\sqrt{1 - P}$, "linear" for $1 - P$.

Value

A symmetric numeric matrix of dissimilarities with a zero diagonal.

`as_proximity.e2tree` *Proximity matrix of a tree ensemble*

Description

Extracts the proximity matrix implied by a fitted tree ensemble: the proportion of trees in which two observations fall in the same terminal node.

Usage

```
## S3 method for class 'e2tree'
as_proximity(object, ...)

as_proximity(object, ...)

## S3 method for class 'randomForest'
as_proximity(object, newdata = NULL, type = c("inbag", "oob"), ...)

## S3 method for class 'ranger'
as_proximity(object, newdata = NULL, type = c("inbag", "oob"), ...)

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

## S3 method for class 'proximity'
as.matrix(x, ...)
```

Arguments

`object`	A fitted tree ensemble.
`...`	Arguments passed to methods.
`newdata`	Data frame on which proximities are computed. Required unless `object` already carries a proximity matrix (see Details of `as_proximity.randomForest()`).
`type`	Either `"inbag"` (average over all trees) or `"oob"` (average over the trees for which both observations are out-of-bag).
`x`	A `proximity` object.

Details

For an ensemble of B trees, the proximity between observations i and j is

$$P_{ij} = B^{-1} \sum_{b=1}^{B} I[\ell_b(x_i) = \ell_b(x_j)],$$

where $\ell_b(x)$ denotes the leaf of tree b reached by x.

With `type = "oob"` the average runs only over the trees for which both observations are out-of-bag. This removes the optimistic bias of the in-bag definition, at the cost of leaving some pairs undefined when the forest is small: such pairs are returned as `NA`.

Value

An object of class `proximity`: a symmetric numeric matrix with a unit diagonal, carrying the attributes `engine`, `n_trees` and `prox_type`.

Methods (by class)

- `as_proximity(e2tree)`: Method for the explanation tree that `e2tree::e2tree()` fits.

  An `e2tree` is one tree, so its proximity is an indicator rather than a proportion: P_{ij} is one when the two observations reach the same leaf and zero otherwise. It is still a Gram matrix over the leaf indicators, so it is positive semi-definite and its rank is the number of leaves.

  Takes no `newdata` and no `type`. The fit carries the observations it was built on and the partition it put them in, and neither out-of-bag nor a proportion over trees means anything for a single tree fitted to everything.

- `as_proximity(randomForest)`: Method for forests fitted with `randomForest::randomForest()`.

  `randomForest` does not store its training data, so `newdata` must be supplied. The single exception is a forest fitted with `proximity = TRUE`, whose stored matrix is reused when it is of the requested `type`.

  Beware that the stored matrix is *not* the in-bag proximity by default: `randomForest()` sets `oob.prox = proximity`, so asking for `proximity = TRUE` and nothing else gives back the out-of-bag matrix. Since the fit does not record the flag, `Proximum` recovers it from `object$call` and refuses to guess when the call does not settle the question.

  Using `type = "oob"` requires `keep.inbag = TRUE` at fitting time.

- `as_proximity(ranger)`: Method for forests fitted with `ranger::ranger()`.

  Requires the forest to have been kept (`write.forest = TRUE`, the default), and `keep.inbag = TRUE` for `type = "oob"`. `ranger` stores the bootstrap counts as a list of one vector per tree rather than as a matrix.

  The proximity of a `ranger` forest and the proximity of a `randomForest` forest are the same statistic computed on two different ensembles, so they are directly comparable: whether two implementations of "the same" forest represent the data the same way is a question the inference layer of phase F2 can answer.

Closing the loop with `e2tree`

`e2tree()` takes a dissimilarity `D` as an argument, which is the object `as_dissimilarity()` produces, and returns a single tree meant to explain the ensemble that dissimilarity came from. This method reads that tree back as a proximity, so the explanation can be compared with what it explains on the same footing:

```
px    <- as_proximity(rf, newdata = data)      # what the forest represents
tree  <- e2tree(y ~ ., data, D = as_dissimilarity(px), ensemble = rf)
mantel_test(px, as_proximity(tree))            # how much of it survives
```

The Mantel correlation between the two is a measure of how much of the ensemble's geometry the single tree reproduces, which is the question `e2tree` exists to answer and which it cannot ask of itself.

Why the name is `as_proximity()`

`e2tree`, which builds an explainable tree on the same similarity structure and is the package most likely to be attached alongside this one, exports a `proximity()` of its own. Two generics of the same name mask each other in the order the packages were attached, and the one the user gets is then a property of their `library()` calls rather than of what they asked for. The `as_` prefix also says what the function does: it coerces a fitted ensemble into an object of class `proximity`, the way `as.dist()` coerces into a `dist`.

See Also

`as.dist.proximity()` to obtain the induced dissimilarity, `summary.proximity()` for diagnostics, `as_proximity.e2tree()` for the method that reads an `e2tree` fit.

`autoplot.proximity` *Visualise a proximity object*

Description

Each class gets the views its object can support:

Usage

```
## S3 method for class 'proximity'
autoplot(
  object,
  type = c("heatmap", "mds", "network"),
  colour = NULL,
  threshold = 0.05,
  ...
)

## S3 method for class 'proximity_sparse'
autoplot(object, ...)

## S3 method for class 'proximity_nystrom'
autoplot(object, colour = NULL, ...)

## S3 method for class 'proximity_stability'
autoplot(object, ...)

## S3 method for class 'proximity_trees'
autoplot(object, ...)
```

Arguments

`object`	A `proximity` object, or one of the four other classes listed above.
`type`	Which view to draw. `proximity` objects only.
`colour`	Optional vector of length `n`, one value per observation, to colour the points of the `"mds"` view by. Anything the caller has: the response, a cluster label, the out-of-bag error they computed.
`threshold`	Proximities at or below this value carry no edge in the `"network"` view.
`...`	Unused. Named arguments here are refused rather than swallowed.

Details

class	views
`proximity`	`"heatmap"`, `"mds"`, `"network"`
`proximity_sparse`	the network, at the threshold it was built with
`proximity_nystrom`	the configuration, from the stored factor
`proximity_stability`	the pairwise agreements it holds
`proximity_trees`	the search path it recorded

Value

A `ggplot` object.

What the heatmap orders by

Rows and columns are ordered by an optimal leaf ordering of the complete-linkage hierarchical clustering of $\sqrt{1-P}$, so that the blocks the ensemble learned line up along the diagonal. The axis labels are suppressed along with the original ordering: after the seriation an index is a position, not an observation, and printing it would invite it to be read as one.

What the network thresholds

An edge is drawn for every pair whose proximity is above `threshold`, and the communities are those of `igraph::cluster_louvain()` on the weighted graph. The default of 0.05 matches `sparsify()`, and at small `n` it keeps a great many edges: a third of the pairs at n = 200, against a tenth at n = 1600. Raise it when the picture is a hairball.

What the views draw from the random stream

Both of them draw from it. The layout and the community search of the network are randomised outright. The seriation of the heatmap is randomised by the matrix it is given, which is less obvious and was measured the wrong way round first: a proximity is k/B, so it takes at most $B+1$ distinct values however many pairs it has. Over eighteen cells at two sample sizes and three ensemble sizes, the dissimilarity took 43 distinct values across 19,900 pairs at 50 trees and 370 across 319,600 at 500. The ordering breaks the rest at random: it drew from the stream in seventeen of the eighteen

and came back different under a second seed in seven. With the ties separated by a jitter below $1/B$ it came back the same in all eighteen, though it still drew from the stream.

Both views put back the stream they found, which is what keeps drawing a plot from moving the permutation tests around it. What neither can do is make the picture independent of the state it was called in, so seed before the call when the figure has to come back the same.

What the heatmap costs

It forms a data frame of n^2 rows, the one thing in this package that is quadratic in the sample size on purpose. Median of three draws, forests of 200 trees:

	n = 200	**400**	**800**	**1600**	**3200**
the seriation, seconds	0.06	0.07	0.08	0.14	0.46
`autoplot()` in all, seconds	0.08	0.07	0.10	0.21	0.78
drawing it, seconds	0.02	0.06	0.25	0.92	3.03
the object, MB	1.1	2.9	10.2	39.5	156.7

The seriation is not what costs, which is worth knowing because it is the part that looks expensive. The drawing is, and it grows as n^2 with the matrix. At n = 3200 the whole call is under a second and the object is 157 MB, so the view outlasts the point at which the matrix itself becomes the problem, and what runs out first is the page: ten million cells show a block structure and nothing finer.

The view that could not be drawn

This page used to promise a fourth view of a `proximity` object, the agreement between replicates against the number of trees, with a bootstrap band. A `proximity` object carries one number of trees and no replicates, so it holds no agreement to plot: that view is `autoplot()` of a `stability()` or an `n_trees_required()` result, which are the objects that have the numbers. The band went for the reason it went from `stability()`: the pairwise agreements are dependent and their quantiles are not a sampling distribution.

The same page promised an MDS "coloured by class and by out-of-bag error", and the object receives neither. `colour` is the caller's to fill.

Suggested packages

The heatmap needs `seriation` and the network needs `igraph`. Both are suggested rather than required, and a view whose package is missing fails naming it. The other three views need neither.

See Also

`embedding()` for the configuration the `"mds"` view draws, `sparsify()` for the threshold the network shares.

Examples

```
set.seed(1)
rf <- randomForest::randomForest(Species ~ ., data = iris, ntree = 200)
px <- as_proximity(rf, newdata = iris)
autoplot(px, type = "mds", colour = iris$Species)
```

cka *Centered kernel alignment between two proximity matrices*

Description

Treats each proximity matrix as a kernel and measures their alignment after centring. `cka()` is invariant to isotropic scaling and to orthogonal transformations of the implied feature spaces, which is what makes it the standard tool for comparing learned representations in the deep learning literature; `rv_coefficient()` is its classical multivariate ancestor.

Usage

```
cka(px1, px2, block_size = NULL)

rv_coefficient(px1, px2, block_size = NULL)
```

Arguments

`px1`, `px2`	`proximity` objects, or in-bag `proximity_stream()` objects, on the same `n` observations.
`block_size`	Rows of the proximity to manufacture at a time, when the arguments are streams. Ignored otherwise. The default divides a 64 MB budget by the sample size.

Details

Both are the normalised Frobenius inner product of the two matrices,

$$\frac{\langle A, B\rangle_F}{\|A\|_F \, \|B\|_F},$$

and they differ only in what they are computed on: `cka()` double-centres each matrix first with `double_centre()`, `rv_coefficient()` takes them as they are. Centring is what buys the invariance, since it removes the mean similarity that any two kernels on the same observations share whether or not they have learned the same structure.

Neither is a test. There is no null distribution and no p-value: they say how aligned two representations are, not whether the alignment is more than chance would give. `mantel_test()` answers that question.

Value

A single numeric value in $[0, 1]$.

Why an out-of-bag matrix is refused

An alignment between kernels needs two kernels. An in-bag proximity is one, being $ZZ^\top/B$ for the leaf-indicator matrix Z. An out-of-bag proximity is not, and growing the forest does not repair it: each entry is a ratio whose denominator counts only the trees where the pair was jointly out

of bag, and those denominators differ across pairs. Measured on `iris` at $n = 80$, the smallest eigenvalue was -1.23 with 50 trees, -0.53 with 200, -0.23 with 1000 and -0.11 with 5000.

What that costs is a biased number rather than an impossible one. Over 600 out-of-bag comparisons in `inst/simulations/inference-calibration.R`, the uncorrected alignment stayed inside $[0, 1]$ every time, and it was **below** the corrected one every time, by 0.089 on average and by as much as 0.154. The negative eigenvalues subtract from the numerator, so an uncorrected alignment understates how alike two representations are, and it does so quietly.

These functions therefore refuse the input and name the repair rather than returning a number that looks reasonable. Pass the matrix through `make_psd()`, and the decision about which correction to apply, and what it costs, stays with you.

Streaming

Given two in-bag `proximity_stream()` objects, both are computed without allocating either matrix. An alignment needs no permutations, so this is a single traversal and the streamed answer costs about what the dense one costs while holding `block_size` rows instead of n. The centring is never applied: it is folded into the accumulation through $\langle HAH, HBH\rangle_F = \langle A, B\rangle_F - \frac{2}{n}(A1)^\top(B1) + n^{-2}(1^\top A1)(1^\top B1)$, which matters because a doubly centred proximity has no zero left in it and would be dense at full size.

See Also

`mantel_test()` for the same comparison with a p-value, `make_psd()` for the correction these functions require.

Examples

```
set.seed(1)
rows <- sample(nrow(iris), 60)
shallow <- randomForest::randomForest(Species ~ ., data = iris[rows, ],
                                      ntree = 100, maxnodes = 4)
deep <- randomForest::randomForest(Species ~ ., data = iris[rows, ],
                                   ntree = 100)
cka(as_proximity(shallow, newdata = iris[rows, ]),
    as_proximity(deep, newdata = iris[rows, ]))
```

---

double_centre *Double centring of a squared dissimilarity matrix*

Description

Computes $G = -\frac{1}{2} J D^2 J$ with $J = I - n^{-1}\mathbf{1}\mathbf{1}^\top$.

Usage

```
double_centre(d)
```

Arguments

d	A dist object or a symmetric numeric matrix.

Details

Multiplying by J on both sides subtracts the row means, the column means, and adds back the grand mean; doing it that way costs $O(n^2)$ and one matrix, whereas forming J and multiplying costs $O(n^3)$ and three. Unlike the matrix-product form, it also keeps the dimnames of d.

Value

The Gower centred matrix, with the dimnames of as.matrix(d).

---

embedding *The configuration a proximity matrix implies*

Description

Reduces a proximity to a cloud of n points in k dimensions, by classical multidimensional scaling of the induced dissimilarity. This is what protest() superimposes and what a two-dimensional plot of a forest shows.

Usage

```
embedding(x, k = 2L, ...)

## Default S3 method:
embedding(x, k = 2L, transform = c("sqrt", "linear"), arg = "x", ...)

## S3 method for class 'proximity_nystrom'
embedding(x, k = 2L, transform = c("sqrt", "linear"), arg = "x", ...)
```

Arguments

x	A proximity object, a symmetric numeric matrix, or a proximity_nystrom object.
k	Number of dimensions to retain.
...	Passed to methods.
transform	Dissimilarity to scale, passed to as_dissimilarity().
arg	The name to report for x in error messages. Callers that hold the object under a different name pass their own; there is no reason to set it interactively.

Value

An n by k matrix of coordinates, centred, with the dimensions in decreasing order of the variance they carry.

Methods (by class)

- `embedding(default)`: Classical scaling of a dense proximity matrix.
- `embedding(proximity_nystrom)`: The same configuration, read off the stored factor of a Nystrom approximation without reconstructing the matrix.

  Only `transform = "sqrt"` is available. The squared dissimilarity is then $\mathbf{1}\mathbf{1}^\top - \tilde{P}$, whose centred form is $\frac{1}{2}(JL)(JL)^\top$, and the configuration is $JLV/\sqrt{2}$ with V the eigenvectors of $(JL)^\top(JL)$. The linear transform squares to $(1-P)^2$, which does not factor through L and would need the dense matrix.

  The diagonal of $\tilde{P}$ is not one, so this configuration and the one obtained by reconstructing the matrix and scaling it disagree by exactly the diagonal error that `summary.proximity_nystrom()` reports. They coincide when the landmarks span the sample.

The same thing twice

Classical scaling of $\sqrt{1-P}$ and the principal components of P read as a kernel are the same computation. With $D^2 = \mathbf{1}\mathbf{1}^\top - P$ and J the centring operator, $J\mathbf{1} = 0$ kills the first term and

$$G = -\tfrac{1}{2}JD^2J = \tfrac{1}{2}JPJ,$$

so the scaling never sees the dissimilarity at all. It holds exactly when the diagonal of P is one, which is why `nystrom()` objects, whose diagonal is not one, are handled by their own method rather than by reconstructing the matrix and pretending.

What it costs

On a dense matrix, the eigendecomposition of the centred matrix, so $O(n^3)$. On a `nystrom()` object the same configuration comes out of an $r \times r$ decomposition of the stored factor, at $O(nr^2)$, and the $n \times n$ matrix is never formed.

See Also

`protest()`, which superimposes two of these, and `nystrom()`.

Examples

```
set.seed(1)
rf <- randomForest::randomForest(Species ~ ., data = iris, ntree = 200)
px <- as_proximity(rf, newdata = iris)
head(embedding(px, k = 2))
```

is_euclidean *Is a dissimilarity matrix Euclidean?*

Description

A dissimilarity is Euclidean when the doubly centred matrix $G = -\frac{1}{2} J D^2 J$, with $J = I - n^{-1}\mathbf{1}\mathbf{1}^\top$, is positive semi-definite. This is the condition under which classical multidimensional scaling of `d` has no negative eigenvalues and the configuration it returns is exact.

Usage

```
is_euclidean(d, tol = 1e-08)
```

Arguments

`d`	A `dist` object or a symmetric numeric matrix of dissimilarities.
`tol`	Relative tolerance on the smallest eigenvalue, expressed as a fraction of the largest eigenvalue in absolute value.

Value

A single logical.

Examples

```
d <- stats::dist(matrix(rnorm(40), ncol = 2))
is_euclidean(d)
```

loans *A synthetic consumer loan portfolio*

Description

Two thousand personal loans over four origination years, with the outcome, the predictors a lender would model on, and one attribute a lender is not allowed to model on. Generated rather than collected, because the public credit panels that can be redistributed have had exactly that last column removed, and the fairness question this dataset exists for cannot be asked without it.

Usage

```
loans
```

Format

A data frame with 2,000 rows and 13 columns:

default Did the loan default? A factor, `"no"` or `"yes"`. 16.4 per cent `"yes"`, rising from 13.8 per cent in the 2019 vintage to 19.9 per cent in 2022.

amount Amount borrowed, in whole units of 25.

income Annual income as stated at application.

dti Debt-to-income ratio at origination.

term Repayment term, `"36 months"` or `"60 months"`.

employment Years in the current job, censored at 40.

score Credit score at application, 300 to 850.

utilisation Revolving credit utilisation, 0 to 1.

delinquencies Delinquencies in the previous two years.

home Housing status: `"rent"`, `"mortgage"` or `"own"`.

purpose Stated purpose of the loan, five levels.

vintage Origination year, 2019 to 2022.

applicant_group A protected attribute, `"A"` or `"B"`. It enters the default probability nowhere: given the predictors it carries no information about the outcome at all.

What was built into it, and why

A synthetic portfolio is only worth shipping if the questions asked of it have answers that are in the data rather than in the telling. Three things were put there on purpose.

Default depends on the score, the utilisation and the debt-to-income ratio, and on an interaction between a thin score and a high utilisation, so the proximity has blocks in it rather than one gradient.

The utilisation carries three times the weight in the 2022 vintage that it carries in 2019. Whether the forest reorganises its view of the portfolio between origination years is then a question about the data and not about sampling noise.

`applicant_group` shifts the median income by 8,902 and the median score by 29 points, and affects the default probability not at all. A forest fitted without it will still separate the groups, because the predictors carry them. How much of the proximity that accounts for is what `permanova()` measures, and it is a fairness diagnostic that does not require the model to have used the attribute.

Source

Generated by `inst/data-raw/loans.R`, which is shipped with the package and records the specification above as code. Re-runnable, seeded, and not a stand-in for a real portfolio: the numbers are realistic in shape and invented in fact.

See Also

`vignette("credit-scoring-case")`, which is the case study this data was generated for.

Examples

```
summary(loans$default)
round(tapply(loans$default == "yes", loans$vintage, mean), 3)
```

`make_psd` *Repair an indefinite proximity matrix*

Description

The out-of-bag proximity is not positive semi-definite (see `summary.proximity()`), so it is not a kernel, and every method that assumes one (centered kernel alignment, the RV coefficient, kernel PCA, classical multidimensional scaling with an exact embedding) is applied to it at the user's peril. `make_psd()` projects it onto the cone of positive semi-definite matrices first, and records how.

Usage

```
make_psd(px, method = c("clip", "flip", "shift"), rescale = TRUE, tol = 1e-08)
```

Arguments

`px`	A `proximity` object, or a symmetric numeric matrix. It must not contain `NA`: an out-of-bag proximity computed from too few trees has undefined pairs, and no eigendecomposition can be had of a matrix with holes in it. Grow more trees.
`method`	One of `"clip"`, `"flip"`, `"shift"`.
`rescale`	Return the corrected matrix with a unit diagonal.
`tol`	Eigenvalues below `-tol * max(abs(lambda))` count as negative.

Value

A `proximity` object with the same attributes as `px`, plus `psd_correction` recording the method used and the smallest eigenvalue that was repaired. When `px` was already positive semi-definite the matrix is returned unchanged and `psd_correction` is `"none"`.

Which correction

All three act on the eigenvalues λ_i of the symmetrised matrix and leave the eigenvectors alone. They differ in what they assume the negative eigenvalues *mean*.

- `"clip"` sets $\lambda_i \leftarrow \max(\lambda_i, 0)$. The negative directions are noise and are discarded. This is the nearest positive semi-definite matrix in Frobenius norm, and the default.
- `"flip"` sets $\lambda_i \leftarrow |\lambda_i|$. The negative directions carry signal whose sign is an artefact. Keeps the rank.
- `"shift"` sets $\lambda_i \leftarrow \lambda_i - \lambda_{\min}$, that is, adds $-\lambda_{\min}$ to the diagonal. It preserves every off-diagonal proximity exactly, at the price of declaring every observation more similar to itself than the data said.

There is no correction that is right in general. `"clip"` is the safe default because it changes the matrix least; `"shift"` is the one to use when the off-diagonal proximities must not move, for instance when the corrected matrix has to stay comparable with an uncorrected one.

On rescaling

None of the three preserves a unit diagonal, and a proximity with $P_{ii} \neq 1$ is hard to read. With `rescale = TRUE` the result is converted back to unit diagonal by the congruence $\tilde{P} = D^{-1/2} P D^{-1/2}$, with D the diagonal of P. A congruence by a positive diagonal matrix preserves positive semi-definiteness, so the repair survives the rescaling.

See Also

`summary.proximity()` for the diagnostic that tells you whether you need this.

Examples

```
if (requireNamespace("randomForest", quietly = TRUE)) {
  set.seed(1)
  rf <- randomForest::randomForest(
    Species ~ ., data = iris, ntree = 200, keep.inbag = TRUE
  )
  px <- as_proximity(rf, newdata = iris, type = "oob")
  summary(px)$euclidean          # FALSE: not a kernel
  summary(make_psd(px))$euclidean # TRUE
}
```

`mantel_test` *Mantel test between two proximity matrices*

Description

Permutation test for the correlation between the off-diagonal entries of two proximity matrices computed on the same observations. The partial variant conditions on a third matrix.

Usage

```
mantel_test(
  px1,
  px2,
  pxz = NULL,
  n_perm = 999,
  method = c("pearson", "spearman"),
  block_size = NULL
)
```

Arguments

`px1`, `px2`	`proximity` objects, symmetric numeric matrices, or `proximity_stream()` objects, on the same `n` observations.
`pxz`	Optional third matrix to condition on. When supplied, the partial Mantel statistic is computed.

n_perm	Number of permutations of the rows and columns.
method	Correlation coefficient, "pearson" or "spearman".
block_size	Rows of the proximity to manufacture at a time, when px1 and px2 are proximity_stream() objects. Ignored otherwise, since a matrix that has already been allocated has nothing to gain from being read in pieces. The default divides a 64 MB budget by the sample size.

Details

The statistic is the correlation between the strict lower triangles of the two matrices. Its size is not the point; the null distribution is. The entries of a proximity matrix are not independent of one another, since each one shares an observation with $2(n-2)$ others, so the sampling distribution of a correlation between two of them is nothing like the one a correlation between $n(n-1)/2$ independent pairs would have. Testing it against the usual table would reject almost always.

The null is therefore built by relabelling the observations: the rows and the columns of the second matrix are permuted **together**, which is the only rearrangement that leaves it a proximity matrix on the same observations. Permuting the entries instead would break the dependence the structure carries and give a null far too narrow.

Value

An object of class htest. The number of usable pairs and the number of permutations are in parameter; the permuted statistics are kept in null_distribution.

Undefined pairs

An out-of-bag proximity is NA for a pair that was never jointly out of bag, which is evidence the forest did not produce rather than a value to impute. The statistic uses the pairs that are defined in every matrix supplied, and the count is reported in the result so that a surprising p-value can be checked against how much of the matrix it rested on. A comparison with very few usable pairs is a reason to grow more trees.

The partial variant

With pxz supplied, the statistic is the correlation between the residuals of px1 and of px2 after each has been regressed on pxz. It answers a different question from the plain test: whether the two matrices still agree once whatever they both share with the third is taken out. The permutation is unchanged.

Streaming

Given two proximity_stream() objects the test runs without allocating either matrix, manufacturing block_size rows at a time and accumulating the six sums a Pearson correlation is a function of. The result is the same to floating point. Two things are not available on that path: method = "spearman", because a rank is a statement about every other pair and cannot be accumulated from blocks that have been discarded, and the partial variant, which needs two traversals. Both refuse rather than approximate. Note that the permutations dominate the cost and are unaffected by streaming: the null is n_perm further traversals of the same matrix.

See Also

cka() for a coefficient that needs no permutation, make_psd() when the matrix is to be used as a kernel.

Examples

```
set.seed(1)
rows <- sample(nrow(iris), 60)
shallow <- randomForest::randomForest(Species ~ ., data = iris[rows, ],
                                      ntree = 100, maxnodes = 4)
deep <- randomForest::randomForest(Species ~ ., data = iris[rows, ],
                                   ntree = 100)
mantel_test(as_proximity(shallow, newdata = iris[rows, ]),
            as_proximity(deep, newdata = iris[rows, ]), n_perm = 99)
```

---

nystrom *Nystrom approximation of a proximity matrix*

Description

The proximity matrix is $n \times n$, so it stops fitting in memory well before the ensemble stops fitting the data: 31 MB at n = 2,000 and about 760 MB at n = 10,000. The Nystrom approximation

$$\tilde{P} = P_{n,m} P_{m,m}^{-1} P_{m,n}$$

reconstructs it from m landmark observations, with $m \ll n$, and this function never forms either side of that identity.

Usage

```
nystrom(fit, data, landmarks = 500L, strata = NULL, tol = 1e-08)

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

Arguments

fit	A fitted tree ensemble.
data	The data on which proximities are computed.
landmarks	Number of landmark observations, or an integer vector of row indices to use as landmarks. A single number is a count; two or more are indices. A count of n or more takes every row, which makes the result exact.
strata	Optional factor of length n for stratified sampling of the landmarks. Allocation is proportional, by largest remainder, with at least one landmark per level. Use it when a class is rare enough that a simple sample would miss it.

`tol`	Relative tolerance on the eigenvalues of the landmark block. Directions below `tol` times the largest are discarded.
`x`	A `proximity_nystrom` object.
`...`	Unused.

Value

An object of class `proximity_nystrom`.

What is stored

Writing $C = P_{n,m}$ and $W = P_{m,m} = U\Lambda U^\top$, the approximation factors as $\tilde{P} = LL^\top$ with $L = CU\Lambda^{-1/2}$, an $n \times r$ matrix. That is what the object holds. The cost is $O(nm)$ in memory rather than $O(n^2)$, and every quantity the object reports is read off L without reconstructing anything: at n = 10,000 with 500 landmarks, 40 MB instead of 760.

W is singular as soon as two landmarks fall in the same leaves in every tree, so Λ^{-1} is a pseudo-inverse with the eigenvalues below `tol` discarded, not `solve()`. The number of directions dropped is reported by `summary()`.

Two invariants that do not survive

`?as_proximity` promises a symmetric matrix with a unit diagonal, and this is not one, which is why a `proximity_nystrom` is its own class rather than a `proximity`. $\tilde{P}_{ii} = \sum_k L_{ik}^2$, which equals one only when observation i is a landmark. The mean departure is reported by `summary()` and is the cheapest single measure of what the approximation cost.

The approximation is also exact only in the span of the landmarks. With `landmarks` at least `n` the sampling is the whole sample, $C = W = P$, and $LL^\top$ returns P to machine precision.

What it gives up, and where

The entries move much more than the geometry does, and the geometry is what the object is for. Relative error against the exact matrix on data with two informative predictors out of six, forests of 300 trees:

	m = 25	**50**	**100**	**200**
entries, n = 400	0.558	0.433	0.320	0.207
configuration, n = 400	0.437	0.270	0.140	0.049
entries, n = 800	0.660	0.532	0.419	0.315
configuration, n = 800	0.529	0.343	0.196	0.102

At n = 400 with half the rows as landmarks the entries are 21 per cent out and the configuration `embedding()` recovers is 5 per cent out, a factor of four. Read the object through `embedding()` and `protest()`, which is what the class is shaped for, and treat `as.matrix()` as a diagnostic rather than a result.

On data where the response is independent of every predictor the same table runs from 0.87 to 0.46: there is no low-rank structure to find, and no number of landmarks invents one.

What stratifying the landmarks is worth

Less than it sounds, and only where it was designed to help. On a response whose minority class is 2.4 per cent of the sample, against a simple sample of the same size:

	m = 20	40	80
draws in which a simple sample drew no minority landmark	0.625	0.250	0.125
error on the minority rows, simple	0.384	0.344	0.304
error on the minority rows, stratified	0.367	0.342	0.302
draws in which stratified was the better	0.800	0.575	0.500

At twenty landmarks a simple sample misses the class outright in five draws out of eight and the rows of that class are reconstructed 4 per cent worse; by eighty landmarks it draws some anyway and the two are indistinguishable. The error over the whole matrix barely moves in either case, because the minority is a fortieth of the rows and contributes a fortieth of the norm. So `strata` is worth setting when the landmark budget is small relative to how rare the class is, and is not worth reaching for otherwise.

In-bag only, and why

There is no `type` argument. The out-of-bag proximity is `NA` on pairs never jointly out of bag and is indefinite where it is defined, so W has holes in it and negative eigenvalues, and $\Lambda^{-1/2}$ does not exist. Repairing the landmark block with `make_psd()` would make the arithmetic run, at the price of an approximation to a matrix that is no longer the one the user asked about. The restriction is deliberate.

See Also

`embedding()` for the configuration the object exists to produce, `as.matrix.proximity_nystrom()` to reconstruct the matrix it avoided.

Examples

```
set.seed(1)
rf <- randomForest::randomForest(Species ~ ., data = iris, ntree = 200)
nystrom(rf, iris, landmarks = 30, strata = iris$Species)
```

`n_trees_required` *How many trees does a stable proximity matrix need?*

Description

Grows the ensemble in blocks and stops when the entries of the proximity matrix stop moving between them.

Usage

```
n_trees_required(fit, data, eps = 0.15, max_trees = 2000L)

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

Arguments

`fit`	A fitted tree ensemble.
`data`	The data on which proximities are computed.
`eps`	Target coefficient of variation. See the section below: the default is measured rather than chosen, and the value that suits a given dataset may be some way from it.
`max_trees`	Upper bound on the block size to try. The real ceiling is half the trees `fit` carries, since a block size needs two blocks.
`x`	A `proximity_trees` object.
`...`	Unused.

Value

The smallest `B` on the grid meeting the criterion, as an integer, or `NA_integer_` when none does. The attribute `path` holds the grid, the number of replicates at each point and the coefficient of variation there; `projected` holds the extrapolated requirement.

The integer carries the class `proximity_trees`, which buys it a `print()` and an `autoplot()` and costs it nothing: arithmetic on it returns the plain number, so `ntree = n_trees_required(...) + 100` is the integer it looks like.

The criterion

With R replicates of an ensemble of B trees, the coefficient of variation reported is

$$CV(B) = \frac{\overline{sd_r(P_{ij}^{(r)})}}{\overline{P_{ij}}},$$

the mean over pairs of the standard deviation between replicates, divided by the mean of all the off-diagonal proximities. Both averages run over pairs only: the diagonal is one in every replicate and would deflate the numerator and inflate the denominator.

The alternative, a coefficient of variation formed pair by pair and then averaged, is not used. Around eight per cent of pairs have a proximity of exactly zero in every replicate, at every number of trees measured, so their own coefficient of variation is zero over zero; the figure would then depend on which pairs were discarded. The aggregate form has one denominator and no such pairs.

What to expect of `eps`

$CV(B)$ falls as a power of the block size. Over four generating processes and both engines, a straight line through the measured points on the log scale had a slope between -0.55 and -0.60,

mean -0.56, with R^2 of at least 0.98 in every cell. It is a clean power law and its exponent is not one half, which is why the projection below fits the slope rather than assuming it.

The level, unlike the exponent, depends on the data far more than on the ensemble. Measured at n = 300 on forests of 2,000 trees:

data	B = 25	100	400	1000
one clean split	0.149	0.074	0.037	0.017
two noisy predictors	0.549	0.271	0.130	0.070
an unbalanced response	0.345	0.171	0.083	0.044
a response independent of everything	0.893	0.443	0.211	0.113

A factor of six between the easiest row and the hardest, at every block size, and almost nothing between `randomForest` and `ranger` on the same row. So `eps` is a target for the data in hand rather than a universal constant. The default of 0.15 is the tightest of the values tried that was reached in all eight cells, at a median of 300 trees; 0.10 was reached in six of them, failing on the process where the response is independent of every predictor and the proximity is sampling noise all the way down.

When no `B` on the grid reaches `eps` the result is `NA` and the `projected` attribute says how many trees the fitted line extrapolates to, which is the number worth acting on.

In-bag only

The proximity of each block is the in-bag one. Out-of-bag would leave a different set of pairs undefined in every block, so the standard deviation between replicates would be taken over a set that changes with the replicate, and the criterion would measure that as well as the instability.

See Also

`stability()` for the agreement between replicates that are already in hand.

Examples

```
set.seed(1)
rf <- randomForest::randomForest(Species ~ ., data = iris, ntree = 400)
n_trees_required(rf, iris, eps = 0.2)
```

`permanova` *PERMANOVA on the dissimilarity induced by a proximity matrix*

Description

Partitions the variation in the induced dissimilarity across the terms of `formula`, with a permutation test on each term. Answers the question of how much of the proximity structure learned by the ensemble is explained by the response, and how much by covariates the model was never given.

Usage

```
permanova(px, formula, data, n_perm = 999, transform = c("sqrt", "linear"))
```

Arguments

px	A proximity object, or a symmetric numeric matrix.
formula	A one-sided formula whose terms are looked up in data.
data	A data frame with nrow(px) rows.
n_perm	Number of permutations.
transform	Dissimilarity to partition, passed to as_dissimilarity(). "sqrt" is Euclidean whenever the proximity is positive semi-definite and "linear" need not be, which is why it is the default.

Details

The partition is the one of McArdle and Anderson: the dissimilarity is squared and doubly centred into the Gower matrix $G = -\frac{1}{2}JD^2J$, and the sum of squares attributed to a set of terms is $\mathrm{tr}(HGH)$ with H the hat matrix of that set. No coordinates are ever computed. That matters here, because the out-of-bag dissimilarity has no exact Euclidean representation, and a method that ordinated first would be partitioning an approximation of the matrix rather than the matrix.

Terms enter sequentially, so the sum of squares of a term is what it adds to the terms before it and the order of formula is part of the question. Every term is tested against the residual of the full model, which is what vegan::adonis2(by = "terms") does and what the two were checked to agree on, statistic by statistic and p-value by p-value.

Value

An object of class anova: one row per term, then Residual and Total, with columns Df, SumOfSqs, R2, F and Pr(>F).

Where a term sits changes its level

Put the terms you already believe in first and the term you are testing last. This is the one place in the package where a measurement belongs in the manual rather than only in NEWS.md, because it is a choice the caller makes at the moment of writing the formula.

A permutation destroys the whole matrix, including whatever the other terms explain. So the observed pseudo-F of an early term is divided by a residual that a strong later term has already shrunk, while its permuted values are divided by residuals that nothing has shrunk, and the ratio comes out too large. The same argument run the other way makes a term placed after a strong one conservative.

Measured over 600 replicates, on a term that explains nothing by construction, at a nominal level of 0.05 and with a Monte Carlo standard error near 0.009:

The model	Rejection rate
the null term alone	0.047
beside another null term	0.048
before a term taking a seventh of the variation	0.105
after that same term	0.020

`vegan::adonis2()` was measured on the same replicates and gave the same rejection rates, so this is a property of sequential permutation testing rather than of this implementation. There is nothing to work around: it is what the order of the formula means.

What it was measured to do

Against the response the forest was trained on, the test rejected in every one of 600 replicates, in-bag and out-of-bag alike, so the power is not the scarce thing here.

The out-of-bag Gower matrix is indefinite, which allows a term's sum of squares to come out negative and its R-squared to leave `[0, 1]`. Over 1800 out-of-bag values neither happened, and the same holds in-bag. The risk is real in principle and did not appear in practice, which is the same shape as the finding behind `cka()`'s refusal.

The null

One permutation relabels the observations, moving the rows and the columns of the Gower matrix together, and the whole sequential decomposition is recomputed on the result. That gives a null pseudo-F for every term from a single permutation, and it is the only rearrangement that leaves the matrix a dissimilarity on the same observations. The p-value counts the observed statistic among the draws, so it is never zero.

Undefined pairs

An out-of-bag proximity is `NA` for a pair that was never jointly out of bag. `mantel_test()` can drop such a pair, because a correlation is a sum over pairs. A sum of squares is a quadratic form over the whole matrix, and one undefined entry leaves the trace undefined with no honest way to recover it, so the input is refused rather than repaired. Growing more trees removes the undefined pairs; the in-bag matrix never has any.

See Also

`protest()` for a comparison of two matrices rather than a partition of one, `as_dissimilarity()` for the transform.

Examples

```
set.seed(1)
rows <- sample(nrow(iris), 60)
rf <- randomForest::randomForest(Species ~ ., data = iris[rows, ],
                                 ntree = 100, keep.inbag = TRUE)
permanova(as_proximity(rf, newdata = iris[rows, ]), ~ Species,
          data = iris[rows, ], n_perm = 99)
```

protest *Procrustes comparison of two proximity matrices*

Description

Superimposes the classical multidimensional scaling configurations of two proximity matrices and reports the residual sum of squares m^2, together with the PROTEST permutation test of its significance.

Usage

```
protest(px1, px2, k = 2L, n_perm = 999, transform = c("sqrt", "linear"))
```

Arguments

px1, px2	proximity objects, or symmetric numeric matrices, on the same n observations.
k	Number of MDS dimensions to retain.
n_perm	Number of permutations for the PROTEST test.
transform	Dissimilarity to scale, passed to as_dissimilarity().

Details

Where mantel_test() correlates the pairwise values and cka() aligns the matrices as kernels, this asks a geometric question: after each matrix has been reduced to a configuration of k dimensions, can one be laid on top of the other? The superimposition is free to translate, rotate, reflect and rescale, since none of those change the dissimilarities the configuration encodes, and what is left over after the best such fit is m^2.

Both configurations are centred and scaled to unit sum of squares before fitting, which makes the comparison symmetric: protest(a, b) and protest(b, a) report the same number. The reported statistic is $r = \sqrt{1 - m^2}$, which is 1 for a perfect fit and 0 for none, so that a larger value means more agreement.

Value

An object of class htest. The number of dimensions and the number of permutations are in parameter; the residual m^2 is in m2 and the permuted statistics in null_distribution.

The null

The rows of the second configuration are permuted, which relabels its observations while leaving the first alone. That is the same null as permuting the rows and columns of the second proximity matrix and scaling it again, because classical scaling commutes with relabelling: the configuration of a permuted dissimilarity is the permuted configuration. Permuting the configuration is the cheap way to compute it, not a different test.

What k costs

The statistic depends on k, and it is not monotone in it. Both configurations are rescaled to unit sum of squares before the fit, so a further dimension changes what is being compared rather than adding to what was compared already, and the correlation can fall as easily as it can rise. Measured over 600 replicates: on forests fitted to unrelated data the mean correlation rose by half again between two dimensions and six, while on forests fitted to the same data it fell in most replicates. Both directions say the same thing. k is part of the question, not a knob to turn until the answer improves, so fix it before looking. Two dimensions is the default because it is what a reader will plot, not because it is enough.

The level holds across the range regardless. On two forests fitted to independent data the test rejected between 0.033 and 0.050 of the time at a nominal 0.05, at every k tried and on both definitions of the proximity, and on two forests fitted to the same data it rejected in every replicate.

A dissimilarity that is not Euclidean, and the out-of-bag one is not, has no exact configuration in any number of dimensions. The scaling discards the negative eigenvalues rather than representing them, which is a second reason the answer moves with k.

See Also

`mantel_test()` for the same comparison on the pairwise values, `permanova()` for a partition of one matrix rather than a comparison of two.

Examples

```
set.seed(1)
rows <- sample(nrow(iris), 60)
shallow <- randomForest::randomForest(Species ~ ., data = iris[rows, ],
                                      ntree = 100, maxnodes = 4)
deep <- randomForest::randomForest(Species ~ ., data = iris[rows, ],
                                   ntree = 100)
protest(as_proximity(shallow, newdata = iris[rows, ]),
        as_proximity(deep, newdata = iris[rows, ]), n_perm = 99)
```

---

proximity_stream *A proximity that is never allocated*

Description

Holds what the proximity matrix is made of rather than the matrix itself, so that a scalar statistic can be computed over it at a sample size where the matrix would not fit in memory.

Usage

```
proximity_stream(fit, data, type = c("inbag", "oob"))

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

```
## S3 method for class 'proximity_stream'
as.matrix(x, ...)
```

Arguments

fit	A fitted tree ensemble, from `randomForest::randomForest()` or `ranger::ranger()`.
data	The data frame to push through it.
type	Either `"inbag"` or `"oob"`, with the meanings they have in `as_proximity()`. `"oob"` requires `keep.inbag = TRUE` at fitting time.
x	A `proximity_stream` object.
...	Unused.

Details

The in-bag proximity is $P = B^{-1}ZZ^\top$ for Z the $n \times L$ leaf indicator (see `as_proximity()`). Z has exactly one entry per observation per tree, so it costs $O(nB)$ to store where P costs $O(n^2)$. A `proximity_stream` keeps Z, and `mantel_test()` and `cka()` rebuild P a block of rows at a time, accumulate what they need from the block and discard it.

Value

An object of class `proximity_stream`, carrying the same `engine`, `n_trees` and `prox_type` attributes a `proximity` carries.

When the saving is a saving

$O(nB)$ beats $O(n^2)$ only once n is past B, and the constants decide where. Measured over a grid of fifteen cells in `inst/simulations/streaming-cost.R`, the indicator costs 13.1 bytes per observation per tree against the matrix's $8n^2$, so the two cross at

$$n \approx 1.64B.$$

Below that a stream holds **more** than the matrix it stands for, and on that grid it did so in 8 cells of 15: at $n = 100$ with $B = 500$ it holds eight times as much. Above it the ratio grows linearly, reaching 9.7 at $n = 1600$ with $B = 100$. Carrying the measured constant out to $n = 100{,}000$ with $B = 500$, which is past anything the simulation could allocate to check against, puts the indicator at some 655 MB where the matrix would need 80 GB.

`print()` shows both numbers side by side, so whether this object is saving anything is a question you can answer by looking at it.

What this costs

Time, and more of it than "no faster" would suggest. The dense path builds the matrix once and then indexes it; the streaming path manufactures every entry each time it needs one, and a permutation test needs the whole matrix once per permutation. On the same grid the streamed alignment took up to 6 times the dense one, and the streamed Mantel test up to 12.8 times, at $n = 800$ with 200 trees: 0.089 seconds per permutation, against a dense path that pays for the matrix once and then permutes indices.

So this is not the fast path and should not be chosen as though it were. It is the path that returns an answer where the dense one returns an allocation error, and `n_perm` is the knob that decides whether the answer arrives.

Why there is no `streaming` argument

Earlier versions of this documentation promised one, on `mantel_test()`. There cannot be one. `mantel_test()` takes a `proximity`, which is a matrix that has already been built, and a matrix that has already been built cannot be traversed instead of built. The saving has to be made at the point where the object is constructed or it is not made at all, so it lives in the type of the input rather than in a flag on the function.

See Also

`as_proximity()` for the dense object, `nystrom()` when the geometry rather than a scalar is wanted at large `n`, and `vignette("large-n")` for how to choose between them.

Examples

```
set.seed(1)
shallow <- randomForest::randomForest(Species ~ ., data = iris,
                                      ntree = 100, maxnodes = 4)
deep <- randomForest::randomForest(Species ~ ., data = iris, ntree = 100)

s1 <- proximity_stream(shallow, iris)
s2 <- proximity_stream(deep, iris)
s1

# The same number the dense path gives, from an object that never held it.
cka(s1, s2)
cka(as_proximity(shallow, newdata = iris), as_proximity(deep, newdata = iris))
```

`sparsify` *Sparse representation of a proximity matrix*

Description

Thresholds the proximity and stores the result as a sparse matrix. This is a storage format, not a faster proximity: the object holds the same numbers in less memory and hands them back on request.

Usage

```
sparsify(px, threshold = 0.05)

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

Arguments

px	A proximity object, or a symmetric numeric matrix. It must not contain NA. A stored NA is not a structural zero, and dropping those pairs would assert that the two observations are maximally dissimilar, which is the opposite of what an undefined pair means.
threshold	Proximities at or below this value are dropped. Must be below one, since the diagonal is one and a proximity matrix without its diagonal is not one.
x	A proximity_sparse object.
...	Unused.

Value

An object of class proximity_sparse, carrying the engine, n_trees and prox_type attributes of px.

How much it saves, and for how long

Most pairs of observations share a leaf in at least one tree, so the raw matrix is not sparse; thresholding is what makes it so, and how well depends on the sample size. Proportions of the pairs, measured on forests of 500 trees:

	n = 200	400	800	1600
exact zeros, in-bag	0.143	0.283	0.438	0.585
kept at 0.05, in-bag	0.366	0.257	0.162	0.102
exact zeros, out-of-bag	0.316	0.472	0.622	0.741
kept at 0.05, out-of-bag	0.407	0.283	0.175	0.109

At n = 200 a seventh of the pairs are zero and the object is barely worth having; the gain grows with n, which is the direction that matters, and at n = 800 it was a factor of eight in memory: 5.0 MB dense against 0.6 MB sparse.

It does not survive being used. Every statistic in this package runs on the induced dissimilarity or on the doubly centred matrix, and both are dense whatever the proximity was: $1 - P$ turns every structural zero into a one, and the Gower centring leaves no zero at all. Across every cell of the table above, $\sqrt{1 - P}$ had between 99.5 and 99.9 per cent of its entries non-zero and double_centre() of it had every entry non-zero. That is why the result is not a proximity object and why the inference functions refuse it: they would densify it silently and the user would have paid the thresholding for nothing.

See Also

nystrom() for the approximation that does survive being used, as.matrix.proximity_sparse() to get the dense matrix back.

Examples

```
set.seed(1)
rf <- randomForest::randomForest(Species ~ ., data = iris, ntree = 200)
px <- as_proximity(rf, newdata = iris)
```

```
sparsify(px, threshold = 0.05)
```

`stability` *Agreement between replicate proximity matrices*

Description

Measures how much the proximity matrix moves when the ensemble is grown again: every pair of replicates is compared, and the spread of those comparisons is what the object reports.

Usage

```
stability(px_list, statistic = c("mantel", "cka"), level = 0.95)

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

Arguments

`px_list`	A list of two or more `proximity` objects computed on the same observations.
`statistic`	Agreement measure between replicates. `"mantel"` is the Pearson correlation between the pairwise values, computed on the pairs defined in both replicates; `"cka"` is the centred kernel alignment, which refuses input that is not positive semi-definite, so out-of-bag replicates have to go through `make_psd()` first. Read the section below before choosing `"cka"` here.
`level`	Width of the percentile interval.
`x`	A `proximity_stability` object.
`...`	Unused.

Value

An object of class `proximity_stability`.

What the interval is, and is not

`R` replicates give $R(R-1)/2$ pairwise agreements, and the reported interval is the empirical `level` percentile interval of those values. It describes how far apart two replicates of this ensemble fall. It is not a bootstrap interval and not a confidence interval for a parameter: the comparisons are not independent, since each replicate enters $R-1$ of them, so their quantiles carry no coverage guarantee. Read it as the spread of the agreements, which is the quantity the question is about.

Which statistic separates what

The two do not agree, and one of them barely moves. Median agreement over 20 replications, four replicates each, n = 300:

what the replicates were	mantel	cka
the same ensemble, refitted	0.978	0.991
the same data, four tree depths	0.838	0.749
different predictors, the same response	0.043	0.959
nothing in common	0.000	0.670

The Mantel correlation runs the whole range and reads zero when the forests share nothing. The centred kernel alignment has a floor near two thirds on forests with nothing in common at all, and cannot tell replicates of one ensemble from forests fitted to different predictors: 0.991 against 0.959. It is measuring the coarse structure both matrices have by construction. Use `"mantel"`, which is the default, unless the question really is about the kernels; `"cka"` is kept because the alignment is the right quantity when the matrices are being used as kernels, not because it is interchangeable here.

The replicates are the caller's to make. Fitting the ensemble again under a different seed is one way; taking disjoint blocks of trees out of a single larger ensemble is another, and `n_trees_required()` does it that way because the two were measured to agree. Whichever it is, this function is told nothing about it and assumes only that every element describes the same observations in the same order.

See Also

`n_trees_required()` for the number of trees that makes this agreement acceptable, `mantel_test()` for the same statistic tested rather than summarised.

Examples

```
set.seed(1)
replicates <- lapply(1:4, function(i) {
  rf <- randomForest::randomForest(Species ~ ., data = iris, ntree = 100)
  as_proximity(rf, newdata = iris)
})
stability(replicates)
```

`summary.proximity` *Diagnostics for a proximity matrix*

Description

Reports the distribution of the off-diagonal proximities, the sparsity of the matrix, and whether the induced dissimilarity is Euclidean.

Usage

```
## S3 method for class 'proximity'
summary(object, max_eigen = 500L, ...)

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

Arguments

object	A proximity object.
max_eigen	Largest n for which the Euclidean check is performed.
...	Unused.
x	A summary.proximity object.

Details

Expect euclidean to be TRUE for an in-bag proximity and FALSE for an out-of-bag one. The in-bag matrix is an average of the Gram matrices $Z_b Z_b^\top$ of the leaf indicators, hence positive semi-definite; the out-of-bag matrix divides each entry by the number of trees in which that pair was jointly out-of-bag, and a matrix of ratios with varying denominators is not a Gram matrix. Debiasing the estimate costs the geometry.

The Euclidean check is skipped for n > max_eigen, where the eigen decomposition of the doubly centred matrix becomes the dominant cost; the corresponding entry is then NA.

Value

An object of class summary.proximity.

---

summary.proximity_nystrom

Diagnostics for a Nystrom approximation

Description

Every figure is computed from the stored factor without reconstructing the matrix. The mean and standard deviation of the off-diagonal entries are exact, not sampled: with $\tilde{P} = LL^\top$, the sum of all entries is $\sum_k (\sum_i L_{ik})^2$ and the sum of their squares is $\|L^\top L\|_F^2$, both of which cost $O(nr^2)$ rather than $O(n^2)$.

Usage

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

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

Arguments

object	A proximity_nystrom object.
...	Unused.
x	A summary.proximity_nystrom object.

Details

diagonal_error is the mean absolute departure of $\tilde{P}_{ii}$ from one. It is zero for a landmark row and grows with how poorly the rest of the sample is spanned by the landmarks, so it is the cheapest single measure of what the approximation cost.

Value

An object of class summary.proximity_nystrom.

---

summary.proximity_sparse

Diagnostics for a sparse proximity matrix

Description

Reports what was kept and what the kept values look like. The Euclidean check that summary.proximity() performs is absent on purpose: it needs the eigendecomposition of the dense doubly centred matrix, and running it here would quietly undo the thresholding.

Usage

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

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

Arguments

object	A proximity_sparse object.
...	Unused.
x	A summary.proximity_sparse object.

Value

An object of class summary.proximity_sparse.

Index