Package ‘tdarec’

June 20, 2025

Title A 'recipes' Extension for Persistent Homology and Its Vectorizations

Version 0.2.0

Description Topological data analytic methods in machine learning rely on vectorizations of the persistence diagrams that encode persistent homology, as surveyed by Ali &al (2000) <doi:10.48550/arXiv.2212.09703>. Persistent homology can be computed using 'TDA' and 'ripserr' and vectorized using 'TDAvec'. The Tidymodels package collection modularizes machine learning in R for straightforward extensibility; see Kuhn & Silge (2022, ISBN:978-1-4920-9644-3). These 'recipe' steps and 'dials' tuners make efficient algorithms for computing and vectorizing persistence diagrams available for Tidymodels workflows.

License GPL (>= 3)

URL `https://github.com/tdaverse/tdarec`

BugReports `https://github.com/tdaverse/tdarec/issues`

Depends R (>= 3.5.0), recipes (>= 0.1.17), dials

Imports rlang (>= 1.1.0), vctrs (>= 0.5.0), scales, tibble, purrr (>= 1.0.0), tidyr, magrittr

Suggests ripserr (>= 0.1.1), TDA, TDAvec (>= 0.1.4), testthat (>= 3.0.0), modeldata, tdaunif, knitr (>= 1.20), rmarkdown (>= 1.10), tidymodels, ranger

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

NeedsCompilation no

Author Jason Cory Brunson [cre, aut]

Maintainer Jason Cory Brunson <cornelioid@gmail.com>

Repository CRAN

Date/Publication 2025-06-20 16:50:02 UTC

Contents

blur . . . 2
blur_sigmas . . . 3
get_blur_range . . . 4
mnist . . . 6
step_blur . . . 6
step_pd_degree . . . 8
step_pd_point_cloud . . . 10
step_pd_raster . . . 12
step_vpd_algebraic_functions . . . 14
step_vpd_betti_curve . . . 16
step_vpd_complex_polynomial . . . 19
step_vpd_descriptive_statistics . . . 21
step_vpd_euler_characteristic_curve . . . 23
step_vpd_normalized_life_curve . . . 26
step_vpd_persistence_block . . . 28
step_vpd_persistence_image . . . 31
step_vpd_persistence_landscape . . . 34
step_vpd_persistence_silhouette . . . 36
step_vpd_persistent_entropy_summary . . . 39
step_vpd_tent_template_functions . . . 41
step_vpd_tropical_coordinates . . . 44
vpd-dials . . . 46
vpd-finalizers . . . 49
vpd-summarizers . . . 50

Index **55**

`blur` *Gaussian blur of an array*

Description

This function takes a numeric array of any dimension as input and returns a blurred array of the same dimensions as output.

Usage

```
blur(
  x,
  xmin = 0,
  xmax = 2^ceiling(log(max(x + 1), 2)) - 1,
  sigma = max(dim(x))/2^(length(dim(x)) + 1)
)
```

Arguments

`x`	a numerical 'array' (including 'matrix')
`xmin`	the smallest possible value in `x`; defaults to 0
`xmax`	the largest possible value in `x`; defaults to the smallest integer $2^k - 1 \leq$ `max(x)`
`sigma`	the standard deviation of the gaussian distribution with which to convolve `x`; defaults to `max(dim(x))`$/2^{D+1}$, where D is the dimensionality of `x`

Details

This function is adapted from `spatstat.explore::blur()`, part of the spatstat package collection.

The procedure takes the following steps:

1. Rescale the value range from [`xmin`, `xmax`] to $[0, 1]$.
2. Convolve `x` with $N(0, \texttt{sigma}^2)$.
3. Rescale the result back to the original value range.

Value

An array of the same dimensions as `x`.

Examples

```
square <- matrix(byrow = TRUE, nrow = 6L, c(
  0, 0, 0, 0, 0, 0, 0, 0,
  0, 1, 1, 1, 1, 0, 0, 0,
  0, 1, 0, 0, 1, 1, 1, 0,
  0, 1, 0, 0, 1, 0, 1, 0,
  0, 1, 1, 1, 1, 1, 1, 0,
  0, 0, 0, 0, 0, 0, 0, 0
))
square_blur <- blur(square)
image(t(square))
image(t(square_blur))
```

`blur_sigmas` *Standard deviation of Gaussian blur*

Description

The standard deviation of the noise function convolved with array values to induce blur in raster data.

Usage

```
blur_sigmas(range = c(unknown(), unknown()), trans = transform_log1p())
```

Arguments

- `range` — A two-element vector holding the *defaults* for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the *transformed units*.
- `trans` — A `trans` object from the `scales` package, such as `scales::transform_log10()` or `scales::transform_reciprocal()`. If not provided, the default is used which matches the units used in `range`. If no transformation, `NULL`.

Details

The gaussian blur step deploys `blur()`. See there for definitions and references.

`get_blur_range()` varies the parameter logarithmically from 0 to an order of magnitude greater than the `blur()` default.

Value

A `param` object or list of `param` objects.

Examples

```
img_dat <- data.frame(img = I(list(volcano)))

(blur_man <- blur_sigmas(range = c(0, 3)))
grid_regular(blur_man)

(blur_fin <- blur_sigmas() %>% get_blur_range(x = img_dat))
grid_regular(blur_fin)
```

---

get_blur_range — *(Maximum) topological dimension or homological degree*

Description

The degree of the homology group to vectorize, or the degree at which to stop vectorizing.

Usage

```
get_blur_range(object, x, ...)

hom_degree(range = c(0L, unknown()), trans = NULL)

max_hom_degree(range = c(0L, unknown()), trans = NULL)

get_hom_range(object, x, max_dim = 2L, ...)
```

Arguments

`object`	A `param` object or a list of `param` objects.
`x`	The predictor data. In some cases (see below) this should only include numeric data.
`...`	Other arguments to pass to the underlying parameter finalizer functions. For example, for `get_rbf_range()`, the dots are passed along to `kernlab::sigest()`.
`range`	A two-element vector holding the *defaults* for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the *transformed units*.
`trans`	A `trans` object from the `scales` package, such as `scales::transform_log10()` or `scales::transform_reciprocal()`. If not provided, the default is used which matches the units used in `range`. If no transformation, `NULL`.
`max_dim`	Bound on the maximum dimension determined from the data.

Details

Topological features have whole number dimensions that determine the degrees of homology that encode them. Any finite point cloud will have finite topological dimension, but most practical applications exploit features of degree at most 3.

Steps may vectorize features of a single degree (`hom_degree()`) or of degrees zero through some maximum (`max_hom_degree()`).

In case the (maximum) degree is not provided, `get_hom_range()` queries each list-column for the maximum dimension of its point cloud and returns the smaller of this maximum and `max_dim` (which defaults to `2L`, the highest homological degree of interest in most practical applications).

Value

A `param` object or list of `param` objects.

Examples

```
# toy data set
klein_sampler <- function(n, prob = .5) {
  if (rbinom(1, 1, prob) == 0) {
    tdaunif::sample_klein_flat(n)
  } else {
    tdaunif::sample_klein_tube(n)
  }
}
sample_data <- data.frame(
  id = LETTERS[seq(4L)],
  sample = I(c(replicate(4L, klein_sampler(60), simplify = FALSE)))
)

# options to calibrate homological degree
hom_degree(range = c(2, 5))
hom_degree() %>% get_hom_range(x = sample_data[, 2, drop = FALSE])
hom_degree() %>% get_hom_range(x = sample_data[, 2, drop = FALSE], max_dim = 5)
```

```
# heterogeneous data types
hetero_data <- tibble(dataset = list(mtcars, nhtemp, eurodist, HairEyeColor))
hetero_data %>%
  mutate(class = vapply(dataset, function(x) class(x)[[1L]], ""))
get_hom_range(
  hom_degree(),
  hetero_data,
  max_dim = 60
)
```

---

mnist *MNIST handwritten digits*

Description

This is a 1% stratified random sample from the MNIST handwritten digit data set.

Usage

```
mnist_train; mnist_test
```

Format

Two data frames of 600 and 100 rows, respectively, and 2 variables:

`digit` list column of 28×28 numeric matrices

`label` integer digit

Source

http://yann.lecun.com/exdb/mnist/

---

step_blur *Blur raster data*

Description

The function `step_blur()` creates a *specification* of a recipe step that will induce Gaussian blur in numerical arrays. The input and output must be list-columns.

Usage

```
step_blur(
  recipe,
  ...,
  role = NA_character_,
  trained = FALSE,
  xmin = 0,
  xmax = 1,
  blur_sigmas = NULL,
  skip = FALSE,
  id = rand_id("blur")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose variables for this step. See selections() for more details.
role	For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as *predictors* in a model.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
xmin, xmax, blur_sigmas	Parameters passed to blur().
skip	A logical. Should the step be skipped when the recipe is baked by bake()? While all operations are baked when prep() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.
id	A character string that is unique to this step to identify it.

Details

The gaussian blur step deploys blur(). See there for definitions and references.

TODO: Explain the importance of blur for PH of image data.

Value

An updated version of recipe with the new step added to the sequence of any existing operations.

Tuning Parameters

This step has 1 tuning parameter(s):

- blur_sigmas: Gaussian Blur std. dev.s (type: double, default: NULL)

Examples

```
topos <- data.frame(pix = I(list(volcano)))

blur_rec <- recipe(~ ., data = topos) %>% step_blur(pix)
blur_prep <- prep(blur_rec, training = topos)
blur_res <- bake(blur_prep, topos)

tidy(blur_rec, number = 1)
tidy(blur_prep, number = 1)

with_sigmas <- recipe(~ ., data = topos) %>% step_blur(pix, blur_sigmas = 10)
with_sigmas <- bake(prep(with_sigmas, training = topos), topos)

ops <- par(mfrow = c(1, 3))
image(topos$pix[[1]])
image(blur_res$pix[[1]])
image(with_sigmas$pix[[1]])
par(ops)
```

---

step_pd_degree *Separate persistent pairs by homological degree*

Description

The function `step_pd_degree()` creates a *specification* of a recipe step that will separate data sets of persistent pairs by homological degree. The input and output must be list-columns.

Usage

```
step_pd_degree(
  recipe,
  ...,
  role = NA_character_,
  trained = FALSE,
  hom_degrees = NULL,
  columns = NULL,
  keep_original_cols = FALSE,
  skip = FALSE,
  id = rand_id("pd_degree")
)
```

Arguments

`recipe`	A recipe object. The step will be added to the sequence of operations for this recipe.
`...`	One or more selector functions to choose variables for this step. See `selections()` for more details.

role
: For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as *predictors* in a model.

trained
: A logical to indicate if the quantities for preprocessing have been estimated.

hom_degrees
: Integer vector of homological degrees.

columns
: A character string of the selected variable names. This field is a placeholder and will be populated once `prep()` is used.

keep_original_cols
: A logical to keep the original variables in the output. Defaults to `FALSE`.

skip
: A logical. Should the step be skipped when the recipe is baked by `bake()`? While all operations are baked when `prep()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

id
: A character string that is unique to this step to identify it.

Details

The `hom_degrees` argument sets the homological degrees for which to return new list-columns. If not `NULL` (the default), it is intersected with the degrees found in any specified columns of the training data; otherwise all found degrees are used. This parameter cannot be tuned.

Value

An updated version of `recipe` with the new step added to the sequence of any existing operations.

See Also

Other topological feature extraction via persistent homology: `step_pd_point_cloud()`, `step_pd_raster()`

Examples

```
dat <- data.frame(
  roads = I(list(eurodist, UScitiesD * 1.6)),
  topos = I(list(volcano, 255 - volcano))
)

ph_rec <- recipe(~ ., data = dat) %>%
  step_pd_point_cloud(roads) %>%
  step_pd_raster(topos) %>%
  step_pd_degree(roads, topos)
ph_prep <- prep(ph_rec, training = dat)
(ph_res <- bake(ph_prep, dat))

tidy(ph_rec, number = 3)
tidy(ph_prep, number = 3)

with_degs <- recipe(~ ., data = dat) %>%
  step_pd_point_cloud(roads) %>%
```

```
  step_pd_raster(topos) %>%
  step_pd_degree(roads, topos, hom_degrees = c(1, 2))
with_degs <- prep(with_degs, training = dat)
bake(with_degs, dat)
```

---

step_pd_point_cloud *Persistent homology of point clouds*

Description

The function step_pd_point_cloud() creates a *specification* of a recipe step that will convert compatible data formats (distance matrices, coordinate matrices, or time series) to 3-column matrix representations of persistence diagram data. The input and output must be list-columns.

Usage

```
step_pd_point_cloud(
  recipe,
  ...,
  role = NA_character_,
  trained = FALSE,
  filtration = "Rips",
  max_hom_degree = 1L,
  radius_max = NULL,
  diameter_max = NULL,
  field_order = 2L,
  engine = NULL,
  columns = NULL,
  skip = FALSE,
  id = rand_id("pd_point_cloud")
)
```

Arguments

recipe A recipe object. The step will be added to the sequence of operations for this recipe.

... One or more selector functions to choose variables for this step. See selections() for more details.

role For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as *predictors* in a model.

trained A logical to indicate if the quantities for preprocessing have been estimated.

filtration The type of filtration from which to compute persistent homology; one of "Rips", "Vietoris" (equivalent), or "alpha".

max_hom_degree, radius_max, diameter_max, field_order
Parameters passed to persistence engines.

`engine`	The computational engine to use (see 'Details'). Reasonable defaults are chosen based on `filtration`.
`columns`	A character string of the selected variable names. This field is a placeholder and will be populated once `prep()` is used.
`skip`	A logical. Should the step be skipped when the recipe is baked by `bake()`? While all operations are baked when `prep()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
`id`	A character string that is unique to this step to identify it.

Details

Persistent homology (PH) is a tool of algebraic topology to extract features from data whose *persistence* measures their robustness to scale. The computation relies on a sequence of maps between discrete topological spaces (usually a filtration comprising only inclusions) constructed from the data.

Value

An updated version of `recipe` with the new step added to the sequence of any existing operations.

PH of Point Clouds

The PH of a point cloud arises from a simplicial filtration (usually Vietoris–Rips, Čech, or alpha) along an increasing distance threshold.

Ripser is a highly efficient implementation of PH on a point cloud (a finite metric space) via the Vietoris–Rips filtration and is ported to R through **ripserr**. **TDA** calls the Dionysus, PHAT, and GUDHI libraries to compute PH via Vietoris–Rips and alpha filtrations. The `filtration` parameter controls the choice of filtration while the `engine` parameter allows the user to manually select an implementation.

Both engines accept data sets in distance matrix, coordinate matrix, data frame, and time series formats.

The `max_hom_degree` argument determines the highest-dimensional features to be calculated. Either `diameter_max` (preferred) or `radius_max` can be used to bound the distance threshold along which PH is computed. The `field_order` argument should be prime and will be the order of the field of coefficients used in the computation. In most applications, only `max_hom_degree` will be tuned, and to at most `3L`.

Tuning Parameters

This step has 1 tuning parameter(s):

- `max_hom_degree`: Maximum Homological Degree (type: integer, default: 1)

See Also

Other topological feature extraction via persistent homology: `step_pd_degree()`, `step_pd_raster()`

Examples

```
roads <- data.frame(dist = I(list(eurodist, UScitiesD * 1.6)))

ph_rec <- recipe(~ ., data = roads) %>%
  step_pd_point_cloud(dist, max_hom_degree = 1, filtration = "Rips")
ph_prep <- prep(ph_rec, training = roads)
ph_res <- bake(ph_prep, roads)

tidy(ph_rec, number = 1)
tidy(ph_prep, number = 1)

ops <- par(mfrow = c(1, 2), mar = c(2, 2, 0, 0) + 0.1)
for (i in seq(nrow(ph_res))) {
  with(ph_res$dist[[i]], plot(
    x = birth, y = death, pch = dimension + 1, col = dimension + 1,
    xlab = NA, ylab = "", asp = 1
  ))
}
par(ops)

with_max <- recipe(~ ., data = roads) %>%
  step_pd_point_cloud(dist, max_hom_degree = 1, diameter_max = 200)
with_max <- prep(with_max, training = roads)
bake(with_max, roads)
```

`step_pd_raster` *Persistent homology of raster data (images)*

Description

The function `step_pd_raster()` creates a *specification* of a recipe step that will convert compatible data formats (numerical arrays, including matrices, of 2, 3, or 4 dimensions) to 3-column matrix representations of persistence diagram data. The input and output must be list-columns.

Usage

```
step_pd_raster(
  recipe,
  ...,
  role = NA_character_,
  trained = FALSE,
  filtration = "cubical",
  value_max = 9999L,
  method = c("link_join", "compute_pairs"),
  engine = NULL,
  columns = NULL,
  skip = FALSE,
  id = rand_id("pd_raster")
)
```

Arguments

`recipe`	A recipe object. The step will be added to the sequence of operations for this recipe.
`...`	One or more selector functions to choose variables for this step. See `selections()` for more details.
`role`	For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as *predictors* in a model.
`trained`	A logical to indicate if the quantities for preprocessing have been estimated.
`filtration`	The type of filtration from which to compute persistent homology; currently only `"cubical"`.
`value_max`, `method`	Parameters passed to persistence engines.
`engine`	The computational engine to use (see 'Details'). Reasonable defaults are chosen based on `filtration`.
`columns`	A character string of the selected variable names. This field is a placeholder and will be populated once `prep()` is used.
`skip`	A logical. Should the step be skipped when the recipe is baked by `bake()`? While all operations are baked when `prep()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
`id`	A character string that is unique to this step to identify it.

Details

Persistent homology (PH) is a tool of algebraic topology to extract features from data whose *persistence* measures their robustness to scale. The computation relies on a sequence of maps between discrete topological spaces (usually a filtration comprising only inclusions) constructed from the data.

Value

An updated version of `recipe` with the new step added to the sequence of any existing operations.

PH of Rasters

The PH of numeric arrays such as (greyscale) digital images is computed from the cubical filtration of the pixel or voxel array, treated as a function from a cubical mesh to a finite value range.

Cubical Ripser is an efficient implementation of cubical PH and is ported to R through **ripserr**. It accepts numerical arrays.

The `value_max` argument bounds the value range along which PH is computed. Cubical Ripser is implemented using both of two methods, link-join and compute-pairs, controlled by the `method` parameter.

Tuning Parameters

This step has 1 tuning parameter(s):

- `max_hom_degree`: Maximum Homological Degree (type: integer, default: NULL)

See Also

Other topological feature extraction via persistent homology: `step_pd_degree()`, `step_pd_point_cloud()`

Examples

```
topos <- data.frame(pix = I(list(volcano)))

ph_rec <- recipe(~ ., data = topos) %>%
  step_pd_raster(pix)
ph_prep <- prep(ph_rec, training = topos)
ph_res <- bake(ph_prep, topos)

tidy(ph_rec, number = 1)
tidy(ph_prep, number = 1)

with(ph_res$pix[[1]], plot(
  x = birth, y = death, pch = dimension + 1, col = dimension + 1,
  xlab = NA, ylab = "", asp = 1
))

with_max <- recipe(~ ., data = topos) %>%
  step_pd_raster(pix, value_max = 150)
with_max <- prep(with_max, training = topos)
bake(with_max, topos)
```

---

step_vpd_algebraic_functions *Algebraic Functions Vectorization of Persistent Homology*

Description

The function `step_vpd_algebraic_functions()` creates a *specification* of a recipe step that will convert a list-column of 3-column matrices of persistence data to a list-column of 1-row matrices of vectorizations.

Usage

```
step_vpd_algebraic_functions(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  hom_degree = 0L,
```

```
  columns = NULL,
  keep_original_cols = TRUE,
  skip = FALSE,
  id = rand_id("vpd_algebraic_functions")
)
```

Arguments

- `recipe` A recipe object. The step will be added to the sequence of operations for this recipe.
- `...` One or more selector functions to choose variables for this step. See `selections()` for more details.
- `role` For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as *predictors* in a model.
- `trained` A logical to indicate if the quantities for preprocessing have been estimated.
- `hom_degree` The homological degree of the features to be transformed.
- `columns` A character string of the selected variable names. This field is a placeholder and will be populated once `prep()` is used.
- `keep_original_cols` A logical to keep the original variables in the output. Defaults to `FALSE`.
- `skip` A logical. Should the step be skipped when the recipe is baked by `bake()`? While all operations are baked when `prep()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
- `id` A character string that is unique to this step to identify it.

Details

Persistent homology is usually encoded as birth–death pairs (barcodes or diagrams), but the space of persistence data sets does not satisfy convenient statistical properties. Such applications as hypothesis testing and machine learning benefit from transformations of persistence data, often to Hilbert spaces (vector spaces with inner products and induced metrics).

Value

An updated version of `recipe` with the new step added to the sequence of any existing operations.

Engine

The algebraic functions vectorization deploys `TDAvec::computeAlgebraicFunctions()`. See there for definitions and references.

Tuning Parameters

This step has 1 tuning parameter:

- `hom_degree`: Homological degree (type: integer, default: `0L`)

Examples

```
library(recipes)

# inspect vectorized features
volc_dat <- data.frame(image = I(list(volcano / 10)))
recipe(~ image, data = volc_dat) %>%
  step_pd_raster(image, method = "link_join") %>%
  step_vpd_algebraic_functions(image, hom_degree = 1) %>%
  print() -> volc_rec
print(volc_rec)
volc_rec %>%
  prep(training = volc_dat) %>%
  bake(new_data = volc_dat)

# dimension-reduce using vectorized features
data(permeability_qsar, package = "modeldata")
permeability_qsar %>%
  transform(perm_cut = cut(permeability, breaks = seq(0, 60, 10))) %>%
  subset(select = -permeability) %>%
  tidyr::nest(chem_fp = -perm_cut) %>%
  print() -> perm_dat
recipe(perm_cut ~ chem_fp, data = perm_dat) %>%
  step_pd_point_cloud(chem_fp, max_hom_degree = 2) %>%
  step_vpd_algebraic_functions(chem_fp, hom_degree = 1) %>%
  step_pca(starts_with("chem_fp_"), num_comp = 2) %>%
  print() -> perm_rec
perm_est <- prep(perm_rec, training = perm_dat)
perm_res <- bake(perm_est, new_data = perm_dat)
# inspect results
tidy(perm_rec)
tidy(perm_rec, number = 2)
tidy(perm_est, number = 2)
# visualize results
with(perm_res, {
  plot(PC1, PC2, type = "n", asp = 1)
  text(PC1, PC2, labels = perm_cut)
})
```

---

step_vpd_betti_curve *Betti Curve Vectorization of Persistent Homology*

Description

The function `step_vpd_betti_curve()` creates a *specification* of a recipe step that will convert a list-column of 3-column matrices of persistence data to a list-column of 1-row matrices of vectorizations.

Usage

```
step_vpd_betti_curve(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  hom_degree = 0L,
  xseq = NULL,
  xmin = NULL,
  xmax = NULL,
  xlen = NULL,
  xby = NULL,
  evaluate = "intervals",
  columns = NULL,
  keep_original_cols = TRUE,
  skip = FALSE,
  id = rand_id("vpd_betti_curve")
)
```

Arguments

`recipe` A recipe object. The step will be added to the sequence of operations for this recipe.

`...` One or more selector functions to choose variables for this step. See `selections()` for more details.

`role` For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as *predictors* in a model.

`trained` A logical to indicate if the quantities for preprocessing have been estimated.

`hom_degree` The homological degree of the features to be transformed.

`xseq` A discretization grid, as an increasing numeric vector. `xseq` overrides the other `x*` parameters with a warning.

`xmin, xmax, xlen, xby`
Limits and resolution of a discretization grid; specify only one of `xlen` and `xby`.

`evaluate` The method by which to vectorize continuous functions over a grid, either 'intervals' or 'points'. Some functions only admit one method.

`columns` A character string of the selected variable names. This field is a placeholder and will be populated once `prep()` is used.

`keep_original_cols`
A logical to keep the original variables in the output. Defaults to `FALSE`.

`skip` A logical. Should the step be skipped when the recipe is baked by `bake()`? While all operations are baked when `prep()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

`id` A character string that is unique to this step to identify it.

Details

Persistent homology is usually encoded as birth–death pairs (barcodes or diagrams), but the space of persistence data sets does not satisfy convenient statistical properties. Such applications as hypothesis testing and machine learning benefit from transformations of persistence data, often to Hilbert spaces (vector spaces with inner products and induced metrics).

Value

An updated version of `recipe` with the new step added to the sequence of any existing operations.

Engine

The Betti curve vectorization deploys `TDAvec::computeBettiCurve()`. See there for definitions and references.

Tuning Parameters

This step has 1 tuning parameter:

- `hom_degree`: Homological degree (type: integer, default: `0L`)

Examples

```
library(recipes)

# inspect vectorized features
volc_dat <- data.frame(image = I(list(volcano / 10)))
recipe(~ image, data = volc_dat) %>%
  step_pd_raster(image, method = "link_join") %>%
  step_vpd_betti_curve(image, hom_degree = 1) %>%
  print() -> volc_rec
print(volc_rec)
volc_rec %>%
  prep(training = volc_dat) %>%
  bake(new_data = volc_dat)

# dimension-reduce using vectorized features
data(permeability_qsar, package = "modeldata")
permeability_qsar %>%
  transform(perm_cut = cut(permeability, breaks = seq(0, 60, 10))) %>%
  subset(select = -permeability) %>%
  tidyr::nest(chem_fp = -perm_cut) %>%
  print() -> perm_dat
recipe(perm_cut ~ chem_fp, data = perm_dat) %>%
  step_pd_point_cloud(chem_fp, max_hom_degree = 2) %>%
  step_vpd_betti_curve(chem_fp, hom_degree = 1) %>%
  step_pca(starts_with("chem_fp_"), num_comp = 2) %>%
  print() -> perm_rec
perm_est <- prep(perm_rec, training = perm_dat)
perm_res <- bake(perm_est, new_data = perm_dat)
# inspect results
tidy(perm_rec)
```

```
tidy(perm_rec, number = 2)
tidy(perm_est, number = 2)
# visualize results
with(perm_res, {
  plot(PC1, PC2, type = "n", asp = 1)
  text(PC1, PC2, labels = perm_cut)
})
```

---

step_vpd_complex_polynomial

Complex Polynomial Vectorization of Persistent Homology

Description

The function `step_vpd_complex_polynomial()` creates a *specification* of a recipe step that will convert a list-column of 3-column matrices of persistence data to a list-column of 1-row matrices of vectorizations.

Usage

```
step_vpd_complex_polynomial(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  hom_degree = 0L,
  num_coef = 1L,
  poly_type = "R",
  columns = NULL,
  keep_original_cols = TRUE,
  skip = FALSE,
  id = rand_id("vpd_complex_polynomial")
)
```

Arguments

`recipe`	A recipe object. The step will be added to the sequence of operations for this recipe.
`...`	One or more selector functions to choose variables for this step. See `selections()` for more details.
`role`	For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as *predictors* in a model.
`trained`	A logical to indicate if the quantities for preprocessing have been estimated.
`hom_degree`	The homological degree of the features to be transformed.

num_coef — The number of coefficients of a convex polynomial fitted to finite persistence pairs.

poly_type — The type of complex polynomial to fit ('R', 'S', or 'T').

columns — A character string of the selected variable names. This field is a placeholder and will be populated once prep() is used.

keep_original_cols — A logical to keep the original variables in the output. Defaults to FALSE.

skip — A logical. Should the step be skipped when the recipe is baked by bake()? While all operations are baked when prep() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.

id — A character string that is unique to this step to identify it.

Details

Persistent homology is usually encoded as birth–death pairs (barcodes or diagrams), but the space of persistence data sets does not satisfy convenient statistical properties. Such applications as hypothesis testing and machine learning benefit from transformations of persistence data, often to Hilbert spaces (vector spaces with inner products and induced metrics).

Value

An updated version of recipe with the new step added to the sequence of any existing operations.

Engine

The complex polynomial vectorization deploys TDAvec::computeComplexPolynomial(). See there for definitions and references.

Tuning Parameters

This step has 3 tuning parameters:

- hom_degree: Homological degree (type: integer, default: 0L)
- num_coef: # Polynomial coefficients (type: integer, default: 1L)
- poly_type: Type of polynomial (type: character, default: "R")

Examples

```
library(recipes)

# inspect vectorized features
volc_dat <- data.frame(image = I(list(volcano / 10)))
recipe(~ image, data = volc_dat) %>%
  step_pd_raster(image, method = "link_join") %>%
  step_vpd_complex_polynomial(image, hom_degree = 1) %>%
  print() -> volc_rec
print(volc_rec)
```

```
volc_rec %>%
  prep(training = volc_dat) %>%
  bake(new_data = volc_dat)

# dimension-reduce using vectorized features
data(permeability_qsar, package = "modeldata")
permeability_qsar %>%
  transform(perm_cut = cut(permeability, breaks = seq(0, 60, 10))) %>%
  subset(select = -permeability) %>%
  tidyr::nest(chem_fp = -perm_cut) %>%
  print() -> perm_dat
recipe(perm_cut ~ chem_fp, data = perm_dat) %>%
  step_pd_point_cloud(chem_fp, max_hom_degree = 2) %>%
  step_vpd_complex_polynomial(chem_fp, hom_degree = 1) %>%
  step_pca(starts_with("chem_fp_"), num_comp = 2) %>%
  print() -> perm_rec
perm_est <- prep(perm_rec, training = perm_dat)
perm_res <- bake(perm_est, new_data = perm_dat)
# inspect results
tidy(perm_rec)
tidy(perm_rec, number = 2)
tidy(perm_est, number = 2)
# visualize results
with(perm_res, {
  plot(PC1, PC2, type = "n", asp = 1)
  text(PC1, PC2, labels = perm_cut)
})
```

`step_vpd_descriptive_statistics`

Descriptive Statistics Vectorization of Persistent Homology

Description

The function `step_vpd_descriptive_statistics()` creates a *specification* of a recipe step that will convert a list-column of 3-column matrices of persistence data to a list-column of 1-row matrices of vectorizations.

Usage

```
step_vpd_descriptive_statistics(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  hom_degree = 0L,
  columns = NULL,
  keep_original_cols = TRUE,
  skip = FALSE,
```

```
  id = rand_id("vpd_descriptive_statistics")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose variables for this step. See selections() for more details.
role	For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as *predictors* in a model.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
hom_degree	The homological degree of the features to be transformed.
columns	A character string of the selected variable names. This field is a placeholder and will be populated once prep() is used.
keep_original_cols	A logical to keep the original variables in the output. Defaults to FALSE.
skip	A logical. Should the step be skipped when the recipe is baked by bake()? While all operations are baked when prep() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.
id	A character string that is unique to this step to identify it.

Details

Persistent homology is usually encoded as birth–death pairs (barcodes or diagrams), but the space of persistence data sets does not satisfy convenient statistical properties. Such applications as hypothesis testing and machine learning benefit from transformations of persistence data, often to Hilbert spaces (vector spaces with inner products and induced metrics).

Value

An updated version of recipe with the new step added to the sequence of any existing operations.

Engine

The descriptive statistics vectorization deploys TDAvec::computeStats(). See there for definitions and references.

Tuning Parameters

This step has 1 tuning parameter:

- hom_degree: Homological degree (type: integer, default: 0L)

Examples

```
library(recipes)

# inspect vectorized features
volc_dat <- data.frame(image = I(list(volcano / 10)))
recipe(~ image, data = volc_dat) %>%
  step_pd_raster(image, method = "link_join") %>%
  step_vpd_descriptive_statistics(image, hom_degree = 1) %>%
  print() -> volc_rec
print(volc_rec)
volc_rec %>%
  prep(training = volc_dat) %>%
  bake(new_data = volc_dat)

# dimension-reduce using vectorized features
data(permeability_qsar, package = "modeldata")
permeability_qsar %>%
  transform(perm_cut = cut(permeability, breaks = seq(0, 60, 10))) %>%
  subset(select = -permeability) %>%
  tidyr::nest(chem_fp = -perm_cut) %>%
  print() -> perm_dat
recipe(perm_cut ~ chem_fp, data = perm_dat) %>%
  step_pd_point_cloud(chem_fp, max_hom_degree = 2) %>%
  step_vpd_descriptive_statistics(chem_fp, hom_degree = 1) %>%
  step_pca(starts_with("chem_fp_"), num_comp = 2) %>%
  print() -> perm_rec
perm_est <- prep(perm_rec, training = perm_dat)
perm_res <- bake(perm_est, new_data = perm_dat)
# inspect results
tidy(perm_rec)
tidy(perm_rec, number = 2)
tidy(perm_est, number = 2)
# visualize results
with(perm_res, {
  plot(PC1, PC2, type = "n", asp = 1)
  text(PC1, PC2, labels = perm_cut)
})
```

---

step_vpd_euler_characteristic_curve

Euler Characteristic Curve Vectorization of Persistent Homology

---

Description

The function `step_vpd_euler_characteristic_curve()` creates a *specification* of a recipe step that will convert a list-column of 3-column matrices of persistence data to a list-column of 1-row matrices of vectorizations.

Usage

```
step_vpd_euler_characteristic_curve(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  xseq = NULL,
  xmin = NULL,
  xmax = NULL,
  xlen = NULL,
  xby = NULL,
  max_hom_degree = Inf,
  evaluate = "intervals",
  columns = NULL,
  keep_original_cols = TRUE,
  skip = FALSE,
  id = rand_id("vpd_euler_characteristic_curve")
)
```

Arguments

`recipe` A recipe object. The step will be added to the sequence of operations for this recipe.

`...` One or more selector functions to choose variables for this step. See `selections()` for more details.

`role` For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as *predictors* in a model.

`trained` A logical to indicate if the quantities for preprocessing have been estimated.

`xseq` A discretization grid, as an increasing numeric vector. `xseq` overrides the other `x*` parameters with a warning.

`xmin, xmax, xlen, xby` Limits and resolution of a discretization grid; specify only one of `xlen` and `xby`.

`max_hom_degree` The highest degree, starting from 0, of the features to be transformed.

`evaluate` The method by which to vectorize continuous functions over a grid, either 'intervals' or 'points'. Some functions only admit one method.

`columns` A character string of the selected variable names. This field is a placeholder and will be populated once `prep()` is used.

`keep_original_cols` A logical to keep the original variables in the output. Defaults to `FALSE`.

`skip` A logical. Should the step be skipped when the recipe is baked by `bake()`? While all operations are baked when `prep()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

`id` A character string that is unique to this step to identify it.

Details

Persistent homology is usually encoded as birth–death pairs (barcodes or diagrams), but the space of persistence data sets does not satisfy convenient statistical properties. Such applications as hypothesis testing and machine learning benefit from transformations of persistence data, often to Hilbert spaces (vector spaces with inner products and induced metrics).

Value

An updated version of `recipe` with the new step added to the sequence of any existing operations.

Engine

The Euler characteristic curve vectorization deploys `TDAvec::computeEulerCharacteristic()`. See there for definitions and references.

Tuning Parameters

This step has 1 tuning parameter:

- `max_hom_degree`: Highest homological degree (type: integer, default: `Inf`)

Examples

```
library(recipes)

# inspect vectorized features
volc_dat <- data.frame(image = I(list(volcano / 10)))
recipe(~ image, data = volc_dat) %>%
  step_pd_raster(image, method = "link_join") %>%
  step_vpd_euler_characteristic_curve(image, max_hom_degree = 2) %>%
  print() -> volc_rec
print(volc_rec)
volc_rec %>%
  prep(training = volc_dat) %>%
  bake(new_data = volc_dat)

# dimension-reduce using vectorized features
data(permeability_qsar, package = "modeldata")
permeability_qsar %>%
  transform(perm_cut = cut(permeability, breaks = seq(0, 60, 10))) %>%
  subset(select = -permeability) %>%
  tidyr::nest(chem_fp = -perm_cut) %>%
  print() -> perm_dat
recipe(perm_cut ~ chem_fp, data = perm_dat) %>%
  step_pd_point_cloud(chem_fp, max_hom_degree = 2) %>%
  step_vpd_euler_characteristic_curve(chem_fp, max_hom_degree = 2) %>%
  step_pca(starts_with("chem_fp_"), num_comp = 2) %>%
  print() -> perm_rec
perm_est <- prep(perm_rec, training = perm_dat)
perm_res <- bake(perm_est, new_data = perm_dat)
# inspect results
tidy(perm_rec)
```

```
tidy(perm_rec, number = 2)
tidy(perm_est, number = 2)
# visualize results
with(perm_res, {
  plot(PC1, PC2, type = "n", asp = 1)
  text(PC1, PC2, labels = perm_cut)
})
```

---

step_vpd_normalized_life_curve

Normalized Life Curve Vectorization of Persistent Homology

Description

The function `step_vpd_normalized_life_curve()` creates a *specification* of a recipe step that will convert a list-column of 3-column matrices of persistence data to a list-column of 1-row matrices of vectorizations.

Usage

```
step_vpd_normalized_life_curve(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  hom_degree = 0L,
  xseq = NULL,
  xmin = NULL,
  xmax = NULL,
  xlen = NULL,
  xby = NULL,
  evaluate = "intervals",
  columns = NULL,
  keep_original_cols = TRUE,
  skip = FALSE,
  id = rand_id("vpd_normalized_life_curve")
)
```

Arguments

`recipe`	A recipe object. The step will be added to the sequence of operations for this recipe.
`...`	One or more selector functions to choose variables for this step. See `selections()` for more details.
`role`	For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as *predictors* in a model.

`trained`	A logical to indicate if the quantities for preprocessing have been estimated.
`hom_degree`	The homological degree of the features to be transformed.
`xseq`	A discretization grid, as an increasing numeric vector. `xseq` overrides the other `x*` parameters with a warning.
`xmin, xmax, xlen, xby`	Limits and resolution of a discretization grid; specify only one of `xlen` and `xby`.
`evaluate`	The method by which to vectorize continuous functions over a grid, either 'intervals' or 'points'. Some functions only admit one method.
`columns`	A character string of the selected variable names. This field is a placeholder and will be populated once `prep()` is used.
`keep_original_cols`	A logical to keep the original variables in the output. Defaults to `FALSE`.
`skip`	A logical. Should the step be skipped when the recipe is baked by `bake()`? While all operations are baked when `prep()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
`id`	A character string that is unique to this step to identify it.

Details

Persistent homology is usually encoded as birth–death pairs (barcodes or diagrams), but the space of persistence data sets does not satisfy convenient statistical properties. Such applications as hypothesis testing and machine learning benefit from transformations of persistence data, often to Hilbert spaces (vector spaces with inner products and induced metrics).

Value

An updated version of `recipe` with the new step added to the sequence of any existing operations.

Engine

The normalized life curve vectorization deploys `TDAvec::computeNormalizedLife()`. See there for definitions and references.

Tuning Parameters

This step has 1 tuning parameter:

- `hom_degree`: Homological degree (type: integer, default: `0L`)

Examples

```
library(recipes)

# inspect vectorized features
volc_dat <- data.frame(image = I(list(volcano / 10)))
recipe(~ image, data = volc_dat) %>%
```

```
  step_pd_raster(image, method = "link_join") %>%
  step_vpd_normalized_life_curve(image, hom_degree = 1) %>%
  print() -> volc_rec
print(volc_rec)
volc_rec %>%
  prep(training = volc_dat) %>%
  bake(new_data = volc_dat)

# dimension-reduce using vectorized features
data(permeability_qsar, package = "modeldata")
permeability_qsar %>%
  transform(perm_cut = cut(permeability, breaks = seq(0, 60, 10))) %>%
  subset(select = -permeability) %>%
  tidyr::nest(chem_fp = -perm_cut) %>%
  print() -> perm_dat
recipe(perm_cut ~ chem_fp, data = perm_dat) %>%
  step_pd_point_cloud(chem_fp, max_hom_degree = 2) %>%
  step_vpd_normalized_life_curve(chem_fp, hom_degree = 1) %>%
  step_pca(starts_with("chem_fp_"), num_comp = 2) %>%
  print() -> perm_rec
perm_est <- prep(perm_rec, training = perm_dat)
perm_res <- bake(perm_est, new_data = perm_dat)
# inspect results
tidy(perm_rec)
tidy(perm_rec, number = 2)
tidy(perm_est, number = 2)
# visualize results
with(perm_res, {
  plot(PC1, PC2, type = "n", asp = 1)
  text(PC1, PC2, labels = perm_cut)
})
```

`step_vpd_persistence_block` *Persistence Block Vectorization of Persistent Homology*

Description

The function `step_vpd_persistence_block()` creates a *specification* of a recipe step that will convert a list-column of 3-column matrices of persistence data to a list-column of 1-row matrices of vectorizations.

Usage

```
step_vpd_persistence_block(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
```

```
  hom_degree = 0L,
  xseq = NULL,
  xmin = NULL,
  xmax = NULL,
  xlen = NULL,
  xby = NULL,
  yseq = NULL,
  ymin = NULL,
  ymax = NULL,
  ylen = NULL,
  yby = NULL,
  block_size = 0.3,
  columns = NULL,
  keep_original_cols = TRUE,
  skip = FALSE,
  id = rand_id("vpd_persistence_block")
)
```

Arguments

`recipe` A recipe object. The step will be added to the sequence of operations for this recipe.

`...` One or more selector functions to choose variables for this step. See `selections()` for more details.

`role` For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as *predictors* in a model.

`trained` A logical to indicate if the quantities for preprocessing have been estimated.

`hom_degree` The homological degree of the features to be transformed.

`xseq` A discretization grid, as an increasing numeric vector. `xseq` overrides the other `x*` parameters with a warning.

`xmin, xmax, xlen, xby`
Limits and resolution of a discretization grid; specify only one of `xlen` and `xby`.

`yseq` Combined with `xseq` to form a 2-dimensional discretization grid.

`ymin, ymax, ylen, yby`
Limits and resolution of a discretization grid; specify only one of `ylen` and `yby`.

`block_size` The scaling factor of the squares used to obtain persistence blocks. The side length of the square centered at a feature (b, p) is obtained by multiplying $2p$ by this factor.

`columns` A character string of the selected variable names. This field is a placeholder and will be populated once `prep()` is used.

`keep_original_cols`
A logical to keep the original variables in the output. Defaults to `FALSE`.

`skip` A logical. Should the step be skipped when the recipe is baked by `bake()`? While all operations are baked when `prep()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

`id` A character string that is unique to this step to identify it.

Details

Persistent homology is usually encoded as birth–death pairs (barcodes or diagrams), but the space of persistence data sets does not satisfy convenient statistical properties. Such applications as hypothesis testing and machine learning benefit from transformations of persistence data, often to Hilbert spaces (vector spaces with inner products and induced metrics).

Value

An updated version of `recipe` with the new step added to the sequence of any existing operations.

Engine

The persistence block vectorization deploys `TDAvec::computePersistenceBlock()`. See there for definitions and references.

Tuning Parameters

This step has 2 tuning parameters:

- `hom_degree`: Homological degree (type: integer, default: `0L`)
- `block_size`: Square side length scaling factor (type: double, default: `0.3`)

Examples

```
library(recipes)

# inspect vectorized features
volc_dat <- data.frame(image = I(list(volcano / 10)))
recipe(~ image, data = volc_dat) %>%
  step_pd_raster(image, method = "link_join") %>%
  step_vpd_persistence_block(image, hom_degree = 1, block_size = 1) %>%
  print() -> volc_rec
print(volc_rec)
volc_rec %>%
  prep(training = volc_dat) %>%
  bake(new_data = volc_dat)

# dimension-reduce using vectorized features
data(permeability_qsar, package = "modeldata")
permeability_qsar %>%
  transform(perm_cut = cut(permeability, breaks = seq(0, 60, 10))) %>%
  subset(select = -permeability) %>%
  tidyr::nest(chem_fp = -perm_cut) %>%
  print() -> perm_dat
recipe(perm_cut ~ chem_fp, data = perm_dat) %>%
```

```
  step_pd_point_cloud(chem_fp, max_hom_degree = 2) %>%
  step_vpd_persistence_block(chem_fp, hom_degree = 1, block_size = 1) %>%
  step_pca(starts_with("chem_fp_"), num_comp = 2) %>%
  print() -> perm_rec
perm_est <- prep(perm_rec, training = perm_dat)
perm_res <- bake(perm_est, new_data = perm_dat)
# inspect results
tidy(perm_rec)
tidy(perm_rec, number = 2)
tidy(perm_est, number = 2)
# visualize results
with(perm_res, {
  plot(PC1, PC2, type = "n", asp = 1)
  text(PC1, PC2, labels = perm_cut)
})
```

---

step_vpd_persistence_image

Persistence Image Vectorization of Persistent Homology

Description

The function `step_vpd_persistence_image()` creates a *specification* of a recipe step that will convert a list-column of 3-column matrices of persistence data to a list-column of 1-row matrices of vectorizations.

Usage

```
step_vpd_persistence_image(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  hom_degree = 0L,
  xseq = NULL,
  xmin = NULL,
  xmax = NULL,
  xlen = NULL,
  xby = NULL,
  yseq = NULL,
  ymin = NULL,
  ymax = NULL,
  ylen = NULL,
  yby = NULL,
  img_sigma = 1,
  columns = NULL,
  keep_original_cols = TRUE,
  skip = FALSE,
```

```
  id = rand_id("vpd_persistence_image")
)
```

Arguments

`recipe` A recipe object. The step will be added to the sequence of operations for this recipe.

`...` One or more selector functions to choose variables for this step. See `selections()` for more details.

`role` For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as *predictors* in a model.

`trained` A logical to indicate if the quantities for preprocessing have been estimated.

`hom_degree` The homological degree of the features to be transformed.

`xseq` A discretization grid, as an increasing numeric vector. `xseq` overrides the other `x*` parameters with a warning.

`xmin, xmax, xlen, xby` Limits and resolution of a discretization grid; specify only one of `xlen` and `xby`.

`yseq` Combined with `xseq` to form a 2-dimensional discretization grid.

`ymin, ymax, ylen, yby` Limits and resolution of a discretization grid; specify only one of `ylen` and `yby`.

`img_sigma` The standard deviation of the gaussian distribution convolved with persistence diagrams to obtain persistence images.

`columns` A character string of the selected variable names. This field is a placeholder and will be populated once `prep()` is used.

`keep_original_cols` A logical to keep the original variables in the output. Defaults to `FALSE`.

`skip` A logical. Should the step be skipped when the recipe is baked by `bake()`? While all operations are baked when `prep()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

`id` A character string that is unique to this step to identify it.

Details

Persistent homology is usually encoded as birth–death pairs (barcodes or diagrams), but the space of persistence data sets does not satisfy convenient statistical properties. Such applications as hypothesis testing and machine learning benefit from transformations of persistence data, often to Hilbert spaces (vector spaces with inner products and induced metrics).

Value

An updated version of `recipe` with the new step added to the sequence of any existing operations.

Engine

The persistence image vectorization deploys TDAvec::computePersistenceImage(). See there for definitions and references.

Tuning Parameters

This step has 2 tuning parameters:

- hom_degree: Homological degree (type: integer, default: 0L)
- img_sigma: Convolved Gaussian standard deviation (type: double, default: 1)

Examples

```
library(recipes)

# inspect vectorized features
volc_dat <- data.frame(image = I(list(volcano / 10)))
recipe(~ image, data = volc_dat) %>%
  step_pd_raster(image, method = "link_join") %>%
  step_vpd_persistence_image(image, hom_degree = 1, img_sigma = 1) %>%
  print() -> volc_rec
print(volc_rec)
volc_rec %>%
  prep(training = volc_dat) %>%
  bake(new_data = volc_dat)

# dimension-reduce using vectorized features
data(permeability_qsar, package = "modeldata")
permeability_qsar %>%
  transform(perm_cut = cut(permeability, breaks = seq(0, 60, 10))) %>%
  subset(select = -permeability) %>%
  tidyr::nest(chem_fp = -perm_cut) %>%
  print() -> perm_dat
recipe(perm_cut ~ chem_fp, data = perm_dat) %>%
  step_pd_point_cloud(chem_fp, max_hom_degree = 2) %>%
  step_vpd_persistence_image(chem_fp, hom_degree = 1, img_sigma = 1) %>%
  step_pca(starts_with("chem_fp_"), num_comp = 2) %>%
  print() -> perm_rec
perm_est <- prep(perm_rec, training = perm_dat)
perm_res <- bake(perm_est, new_data = perm_dat)
# inspect results
tidy(perm_rec)
tidy(perm_rec, number = 2)
tidy(perm_est, number = 2)
# visualize results
with(perm_res, {
  plot(PC1, PC2, type = "n", asp = 1)
  text(PC1, PC2, labels = perm_cut)
})
```

step_vpd_persistence_landscape

Persistence Landscape Vectorization of Persistent Homology

Description

The function step_vpd_persistence_landscape() creates a *specification* of a recipe step that will convert a list-column of 3-column matrices of persistence data to a list-column of 1-row matrices of vectorizations.

Usage

```
step_vpd_persistence_landscape(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  hom_degree = 0L,
  xseq = NULL,
  xmin = NULL,
  xmax = NULL,
  xlen = NULL,
  xby = NULL,
  num_levels = 6L,
  generalized = FALSE,
  weight_func_pl = "triangle",
  bandwidth = NULL,
  columns = NULL,
  keep_original_cols = TRUE,
  skip = FALSE,
  id = rand_id("vpd_persistence_landscape")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose variables for this step. See selections() for more details.
role	For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as *predictors* in a model.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
hom_degree	The homological degree of the features to be transformed.
xseq	A discretization grid, as an increasing numeric vector. xseq overrides the other x* parameters with a warning.

xmin, xmax, xlen, xby
: Limits and resolution of a discretization grid; specify only one of xlen and xby.

num_levels
: The number of levels of a persistence landscape to vectorize. If num_levels is greater than the length of a landscape, then additional levels of zeros will be included.

generalized
: Logical indicator to compute generalized functions.

weight_func_pl
: A *single* character for the type of kernel function used to compute generalized landscapes.

bandwidth
: The bandwidth of a kernel function.

columns
: A character string of the selected variable names. This field is a placeholder and will be populated once prep() is used.

keep_original_cols
: A logical to keep the original variables in the output. Defaults to FALSE.

skip
: A logical. Should the step be skipped when the recipe is baked by bake()? While all operations are baked when prep() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.

id
: A character string that is unique to this step to identify it.

Details

Persistent homology is usually encoded as birth–death pairs (barcodes or diagrams), but the space of persistence data sets does not satisfy convenient statistical properties. Such applications as hypothesis testing and machine learning benefit from transformations of persistence data, often to Hilbert spaces (vector spaces with inner products and induced metrics).

Value

An updated version of recipe with the new step added to the sequence of any existing operations.

Engine

The persistence landscape vectorization deploys TDAvec::computePersistenceLandscape(). See there for definitions and references.

Tuning Parameters

This step has 4 tuning parameters:

- hom_degree: Homological degree (type: integer, default: 0L)
- num_levels: # Levels or envelopes (type: integer, default: 6L)
- weight_func_pl: Kernel distance weight function (type: character, default: "triangle")
- bandwidth: Kernel bandwidth (type: double, default: NULL)

Examples

```
library(recipes)

# inspect vectorized features
volc_dat <- data.frame(image = I(list(volcano / 10)))
recipe(~ image, data = volc_dat) %>%
  step_pd_raster(image, method = "link_join") %>%
  step_vpd_persistence_landscape(image, hom_degree = 1, num_levels = 3) %>%
  print() -> volc_rec
print(volc_rec)
volc_rec %>%
  prep(training = volc_dat) %>%
  bake(new_data = volc_dat)

# dimension-reduce using vectorized features
data(permeability_qsar, package = "modeldata")
permeability_qsar %>%
  transform(perm_cut = cut(permeability, breaks = seq(0, 60, 10))) %>%
  subset(select = -permeability) %>%
  tidyr::nest(chem_fp = -perm_cut) %>%
  print() -> perm_dat
recipe(perm_cut ~ chem_fp, data = perm_dat) %>%
  step_pd_point_cloud(chem_fp, max_hom_degree = 2) %>%
  step_vpd_persistence_landscape(chem_fp, hom_degree = 1, num_levels = 3) %>%
  step_pca(starts_with("chem_fp_"), num_comp = 2) %>%
  print() -> perm_rec
perm_est <- prep(perm_rec, training = perm_dat)
perm_res <- bake(perm_est, new_data = perm_dat)
# inspect results
tidy(perm_rec)
tidy(perm_rec, number = 2)
tidy(perm_est, number = 2)
# visualize results
with(perm_res, {
  plot(PC1, PC2, type = "n", asp = 1)
  text(PC1, PC2, labels = perm_cut)
})
```

---

step_vpd_persistence_silhouette

Persistence Silhouette Vectorization of Persistent Homology

Description

The function `step_vpd_persistence_silhouette()` creates a *specification* of a recipe step that will convert a list-column of 3-column matrices of persistence data to a list-column of 1-row matrices of vectorizations.

Usage

```
step_vpd_persistence_silhouette(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  hom_degree = 0L,
  xseq = NULL,
  xmin = NULL,
  xmax = NULL,
  xlen = NULL,
  xby = NULL,
  weight_power = 1,
  evaluate = "intervals",
  columns = NULL,
  keep_original_cols = TRUE,
  skip = FALSE,
  id = rand_id("vpd_persistence_silhouette")
)
```

Arguments

- `recipe` — A recipe object. The step will be added to the sequence of operations for this recipe.
- `...` — One or more selector functions to choose variables for this step. See `selections()` for more details.
- `role` — For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as *predictors* in a model.
- `trained` — A logical to indicate if the quantities for preprocessing have been estimated.
- `hom_degree` — The homological degree of the features to be transformed.
- `xseq` — A discretization grid, as an increasing numeric vector. `xseq` overrides the other `x*` parameters with a warning.
- `xmin, xmax, xlen, xby` — Limits and resolution of a discretization grid; specify only one of `xlen` and `xby`.
- `weight_power` — The power of weights in a persistence silhouette function.
- `evaluate` — The method by which to vectorize continuous functions over a grid, either 'intervals' or 'points'. Some functions only admit one method.
- `columns` — A character string of the selected variable names. This field is a placeholder and will be populated once `prep()` is used.
- `keep_original_cols` — A logical to keep the original variables in the output. Defaults to `FALSE`.
- `skip` — A logical. Should the step be skipped when the recipe is baked by `bake()`? While all operations are baked when `prep()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)).

Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

`id` A character string that is unique to this step to identify it.

Details

Persistent homology is usually encoded as birth–death pairs (barcodes or diagrams), but the space of persistence data sets does not satisfy convenient statistical properties. Such applications as hypothesis testing and machine learning benefit from transformations of persistence data, often to Hilbert spaces (vector spaces with inner products and induced metrics).

Value

An updated version of `recipe` with the new step added to the sequence of any existing operations.

Engine

The persistence silhouette vectorization deploys `TDAvec::computePersistenceSilhouette()`. See there for definitions and references.

Tuning Parameters

This step has 2 tuning parameters:

- `hom_degree`: Homological degree (type: integer, default: `0L`)
- `weight_power`: Exponent weight (type: double, default: `1`)

Examples

```
library(recipes)

# inspect vectorized features
volc_dat <- data.frame(image = I(list(volcano / 10)))
recipe(~ image, data = volc_dat) %>%
  step_pd_raster(image, method = "link_join") %>%
  step_vpd_persistence_silhouette(image, hom_degree = 1) %>%
  print() -> volc_rec
print(volc_rec)
volc_rec %>%
  prep(training = volc_dat) %>%
  bake(new_data = volc_dat)

# dimension-reduce using vectorized features
data(permeability_qsar, package = "modeldata")
permeability_qsar %>%
  transform(perm_cut = cut(permeability, breaks = seq(0, 60, 10))) %>%
  subset(select = -permeability) %>%
  tidyr::nest(chem_fp = -perm_cut) %>%
  print() -> perm_dat
recipe(perm_cut ~ chem_fp, data = perm_dat) %>%
  step_pd_point_cloud(chem_fp, max_hom_degree = 2) %>%
```

```
  step_vpd_persistence_silhouette(chem_fp, hom_degree = 1) %>%
  step_pca(starts_with("chem_fp_"), num_comp = 2) %>%
  print() -> perm_rec
perm_est <- prep(perm_rec, training = perm_dat)
perm_res <- bake(perm_est, new_data = perm_dat)
# inspect results
tidy(perm_rec)
tidy(perm_rec, number = 2)
tidy(perm_est, number = 2)
# visualize results
with(perm_res, {
  plot(PC1, PC2, type = "n", asp = 1)
  text(PC1, PC2, labels = perm_cut)
})
```

---

step_vpd_persistent_entropy_summary

Persistent Entropy Summary Vectorization of Persistent Homology

---

Description

The function `step_vpd_persistent_entropy_summary()` creates a *specification* of a recipe step that will convert a list-column of 3-column matrices of persistence data to a list-column of 1-row matrices of vectorizations.

Usage

```
step_vpd_persistent_entropy_summary(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  hom_degree = 0L,
  xseq = NULL,
  xmin = NULL,
  xmax = NULL,
  xlen = NULL,
  xby = NULL,
  evaluate = "intervals",
  columns = NULL,
  keep_original_cols = TRUE,
  skip = FALSE,
  id = rand_id("vpd_persistent_entropy_summary")
)
```

Arguments

`recipe`	A recipe object. The step will be added to the sequence of operations for this recipe.

...	One or more selector functions to choose variables for this step. See selections() for more details.
role	For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as *predictors* in a model.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
hom_degree	The homological degree of the features to be transformed.
xseq	A discretization grid, as an increasing numeric vector. xseq overrides the other x* parameters with a warning.
xmin, xmax, xlen, xby	Limits and resolution of a discretization grid; specify only one of xlen and xby.
evaluate	The method by which to vectorize continuous functions over a grid, either 'intervals' or 'points'. Some functions only admit one method.
columns	A character string of the selected variable names. This field is a placeholder and will be populated once prep() is used.
keep_original_cols	A logical to keep the original variables in the output. Defaults to FALSE.
skip	A logical. Should the step be skipped when the recipe is baked by bake()? While all operations are baked when prep() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations.
id	A character string that is unique to this step to identify it.

Details

Persistent homology is usually encoded as birth–death pairs (barcodes or diagrams), but the space of persistence data sets does not satisfy convenient statistical properties. Such applications as hypothesis testing and machine learning benefit from transformations of persistence data, often to Hilbert spaces (vector spaces with inner products and induced metrics).

Value

An updated version of recipe with the new step added to the sequence of any existing operations.

Engine

The persistent entropy summary vectorization deploys TDAvec::computePersistentEntropy(). See there for definitions and references.

Tuning Parameters

This step has 1 tuning parameter:

- hom_degree: Homological degree (type: integer, default: 0L)

Examples

```
library(recipes)

# inspect vectorized features
volc_dat <- data.frame(image = I(list(volcano / 10)))
recipe(~ image, data = volc_dat) %>%
  step_pd_raster(image, method = "link_join") %>%
  step_vpd_persistent_entropy_summary(image, hom_degree = 1) %>%
  print() -> volc_rec
print(volc_rec)
volc_rec %>%
  prep(training = volc_dat) %>%
  bake(new_data = volc_dat)

# dimension-reduce using vectorized features
data(permeability_qsar, package = "modeldata")
permeability_qsar %>%
  transform(perm_cut = cut(permeability, breaks = seq(0, 60, 10))) %>%
  subset(select = -permeability) %>%
  tidyr::nest(chem_fp = -perm_cut) %>%
  print() -> perm_dat
recipe(perm_cut ~ chem_fp, data = perm_dat) %>%
  step_pd_point_cloud(chem_fp, max_hom_degree = 2) %>%
  step_vpd_persistent_entropy_summary(chem_fp, hom_degree = 1) %>%
  step_pca(starts_with("chem_fp_"), num_comp = 2) %>%
  print() -> perm_rec
perm_est <- prep(perm_rec, training = perm_dat)
perm_res <- bake(perm_est, new_data = perm_dat)
# inspect results
tidy(perm_rec)
tidy(perm_rec, number = 2)
tidy(perm_est, number = 2)
# visualize results
with(perm_res, {
  plot(PC1, PC2, type = "n", asp = 1)
  text(PC1, PC2, labels = perm_cut)
})
```

---

step_vpd_tent_template_functions

Tent Template Functions Vectorization of Persistent Homology

Description

The function `step_vpd_tent_template_functions()` creates a *specification* of a recipe step that will convert a list-column of 3-column matrices of persistence data to a list-column of 1-row matrices of vectorizations.

Usage

```
step_vpd_tent_template_functions(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  hom_degree = 0L,
  tent_size = NULL,
  num_bins = 10L,
  tent_shift = NULL,
  columns = NULL,
  keep_original_cols = TRUE,
  skip = FALSE,
  id = rand_id("vpd_tent_template_functions")
)
```

Arguments

`recipe`	A recipe object. The step will be added to the sequence of operations for this recipe.
`...`	One or more selector functions to choose variables for this step. See `selections()` for more details.
`role`	For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as *predictors* in a model.
`trained`	A logical to indicate if the quantities for preprocessing have been estimated.
`hom_degree`	The homological degree of the features to be transformed.
`tent_size`	The length of the increment used to discretize tent template functions.
`num_bins`	The number of bins along each axis in the discretization grid.
`tent_shift`	The vertical shift applied to the discretization grid.
`columns`	A character string of the selected variable names. This field is a placeholder and will be populated once `prep()` is used.
`keep_original_cols`	A logical to keep the original variables in the output. Defaults to `FALSE`.
`skip`	A logical. Should the step be skipped when the recipe is baked by `bake()`? While all operations are baked when `prep()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.
`id`	A character string that is unique to this step to identify it.

Details

Persistent homology is usually encoded as birth–death pairs (barcodes or diagrams), but the space of persistence data sets does not satisfy convenient statistical properties. Such applications as hypothesis testing and machine learning benefit from transformations of persistence data, often to Hilbert spaces (vector spaces with inner products and induced metrics).

Value

An updated version of `recipe` with the new step added to the sequence of any existing operations.

Engine

The tent template functions vectorization deploys `TDAvec::computeTemplateFunction()`. See there for definitions and references.

Tuning Parameters

This step has 4 tuning parameters:

- `hom_degree`: Homological degree (type: integer, default: `0L`)
- `tent_size`: Discretization grid increment (type: double, default: `NULL`)
- `num_bins`: Discretization grid bins (type: integer, default: `10L`)
- `tent_shift`: Discretization grid shift (type: double, default: `NULL`)

Examples

```
library(recipes)

# inspect vectorized features
volc_dat <- data.frame(image = I(list(volcano / 10)))
recipe(~ image, data = volc_dat) %>%
  step_pd_raster(image, method = "link_join") %>%
  step_vpd_tent_template_functions(image, hom_degree = 1) %>%
  print() -> volc_rec
print(volc_rec)
volc_rec %>%
  prep(training = volc_dat) %>%
  bake(new_data = volc_dat)

# dimension-reduce using vectorized features
data(permeability_qsar, package = "modeldata")
permeability_qsar %>%
  transform(perm_cut = cut(permeability, breaks = seq(0, 60, 10))) %>%
  subset(select = -permeability) %>%
  tidyr::nest(chem_fp = -perm_cut) %>%
  print() -> perm_dat
recipe(perm_cut ~ chem_fp, data = perm_dat) %>%
  step_pd_point_cloud(chem_fp, max_hom_degree = 2) %>%
  step_vpd_tent_template_functions(chem_fp, hom_degree = 1) %>%
  step_pca(starts_with("chem_fp_"), num_comp = 2) %>%
  print() -> perm_rec
perm_est <- prep(perm_rec, training = perm_dat)
perm_res <- bake(perm_est, new_data = perm_dat)
# inspect results
tidy(perm_rec)
tidy(perm_rec, number = 2)
tidy(perm_est, number = 2)
# visualize results
```

```
with(perm_res, {
  plot(PC1, PC2, type = "n", asp = 1)
  text(PC1, PC2, labels = perm_cut)
})
```

---

step_vpd_tropical_coordinates
Tropical Coordinates Vectorization of Persistent Homology

Description

The function `step_vpd_tropical_coordinates()` creates a *specification* of a recipe step that will convert a list-column of 3-column matrices of persistence data to a list-column of 1-row matrices of vectorizations.

Usage

```
step_vpd_tropical_coordinates(
  recipe,
  ...,
  role = "predictor",
  trained = FALSE,
  hom_degree = 0L,
  num_bars = 1L,
  columns = NULL,
  keep_original_cols = TRUE,
  skip = FALSE,
  id = rand_id("vpd_tropical_coordinates")
)
```

Arguments

- `recipe`: A recipe object. The step will be added to the sequence of operations for this recipe.
- `...`: One or more selector functions to choose variables for this step. See `selections()` for more details.
- `role`: For model terms created by this step, what analysis role should they be assigned? By default, the new columns created by this step from the original variables will be used as *predictors* in a model.
- `trained`: A logical to indicate if the quantities for preprocessing have been estimated.
- `hom_degree`: The homological degree of the features to be transformed.
- `num_bars`: Number of bars (persistent pairs) over which to maximize....
- `columns`: A character string of the selected variable names. This field is a placeholder and will be populated once `prep()` is used.
- `keep_original_cols`: A logical to keep the original variables in the output. Defaults to `FALSE`.

skip
: A logical. Should the step be skipped when the recipe is baked by `bake()`? While all operations are baked when `prep()` is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using `skip = TRUE` as it may affect the computations for subsequent operations.

id
: A character string that is unique to this step to identify it.

Details

Persistent homology is usually encoded as birth–death pairs (barcodes or diagrams), but the space of persistence data sets does not satisfy convenient statistical properties. Such applications as hypothesis testing and machine learning benefit from transformations of persistence data, often to Hilbert spaces (vector spaces with inner products and induced metrics).

Value

An updated version of `recipe` with the new step added to the sequence of any existing operations.

Engine

The tropical coordinates vectorization deploys `TDAvec::computeTropicalCoordinates()`. See there for definitions and references.

Tuning Parameters

This step has 2 tuning parameters:

- `hom_degree`: Homological degree (type: integer, default: `0L`)
- `num_bars`: # Bars (persistence pairs) (type: integer, default: `1L`)

Examples

```
library(recipes)

# inspect vectorized features
volc_dat <- data.frame(image = I(list(volcano / 10)))
recipe(~ image, data = volc_dat) %>%
  step_pd_raster(image, method = "link_join") %>%
  step_vpd_tropical_coordinates(image, hom_degree = 1) %>%
  print() -> volc_rec
print(volc_rec)
volc_rec %>%
  prep(training = volc_dat) %>%
  bake(new_data = volc_dat)

# dimension-reduce using vectorized features
data(permeability_qsar, package = "modeldata")
permeability_qsar %>%
  transform(perm_cut = cut(permeability, breaks = seq(0, 60, 10))) %>%
  subset(select = -permeability) %>%
  tidyr::nest(chem_fp = -perm_cut) %>%
```

```
  print() -> perm_dat
recipe(perm_cut ~ chem_fp, data = perm_dat) %>%
  step_pd_point_cloud(chem_fp, max_hom_degree = 2) %>%
  step_vpd_tropical_coordinates(chem_fp, hom_degree = 1) %>%
  step_pca(starts_with("chem_fp_"), num_comp = 2) %>%
  print() -> perm_rec
perm_est <- prep(perm_rec, training = perm_dat)
perm_res <- bake(perm_est, new_data = perm_dat)
# inspect results
tidy(perm_rec)
tidy(perm_rec, number = 2)
tidy(perm_est, number = 2)
# visualize results
with(perm_res, {
  plot(PC1, PC2, type = "n", asp = 1)
  text(PC1, PC2, labels = perm_cut)
})
```

---

vpd-dials *Tune Vectorizations of Persistent Homology*

Description

These tuning functions govern the parameters of vectorizations implemented in **TDAvec**.

Usage

```
num_coef(range = c(1L, unknown()), trans = NULL)

poly_type(values = c("R", "S", "T"), trans = NULL)

img_sigma(range = c(unknown(), unknown()), trans = transform_log10())

num_levels(range = c(1L, unknown()), trans = NULL)

weight_func_pl(
  values = c("triangle", "epanechnikov", "tricubic"),
  trans = NULL
)

bandwidth(range = c(unknown(), unknown()), trans = transform_log10())

weight_power(range = c(1, 2), trans = NULL)

num_bars(range = c(1L, unknown()), trans = NULL)

num_bins(range = c(2L, 20L), trans = NULL)

tent_shift(range = c(unknown(), unknown()), trans = transform_log10())
```

Arguments

range
: A two-element vector holding the *defaults* for the smallest and largest possible values, respectively. If a transformation is specified, these values should be in the *transformed units*.

trans
: A trans object from the scales package, such as scales::transform_log10() or scales::transform_reciprocal(). If not provided, the default is used which matches the units used in range. If no transformation, NULL.

values
: A character string of possible values.

Details

The parameter num_coef is passed to m in TDAvec::computeComplexPolynomial().

The parameter poly_type is passed to polyType in TDAvec::computeComplexPolynomial().

The parameter img_sigma is passed to sigma in TDAvec::computePersistenceImage().

The parameter num_levels is passed to k in TDAvec::computePersistenceLandscape().

The parameter weight_func_pl is passed to kernel in TDAvec::computePersistenceLandscape().

The parameter bandwidth is passed to h in TDAvec::computePersistenceLandscape().

The parameter weight_power is passed to p in TDAvec::computePersistenceSilhouette().

The parameter num_bars is passed to r in TDAvec::computeTropicalCoordinates().

The parameter num_bins is passed to d in TDAvec::computeTemplateFunction().

The parameter tent_shift is passed to epsilon in TDAvec::computeTemplateFunction().

Value

A param object or list of param objects.

Examples

```
data.frame(dist = I(list(eurodist, UScitiesD * 1.6))) %>%
  transform(pd = I(lapply(dist, ripserr::vietoris_rips))) %>%
  subset(select = c(pd)) %>%
  print() -> pd_data

# `num_coef` for `step_vpn_complex_polynomial()`

(nc_man <- num_coef(range = c(1L, 3L)))
grid_regular(nc_man)

# `poly_type` for `step_vpn_complex_polynomial()`

(pt_man <- poly_type(values = c("R", "S")))
grid_regular(pt_man)

# `img_sigma` for `step_vpn_persistence_image()`

(is_man <- img_sigma(range = c(100, 400), trans = NULL))
grid_regular(is_man)
```

```
(is_dat <- img_sigma() %>% get_pers_max_frac(x = pd_data))
grid_regular(is_dat)

(is_hom <- img_sigma() %>% get_pers_max_frac(x = pd_data, hom_degrees = seq(2L)))
grid_regular(is_hom)

# `num_levels` for `step_vpn_persistence_landscape()`

(nl_man <- num_levels(range = c(1L, 6L)))
grid_regular(nl_man)

# `weight_func_pl` for `step_vpn_persistence_landscape()`

(wfp_man <- weight_func_pl(values = c("triangle", "tricubic")))
grid_regular(wfp_man)

# `bandwidth` for `step_vpn_persistence_landscape()`

(b_man <- bandwidth(range = c(500, 1500), trans = NULL))
grid_regular(b_man)

(b_dat <- bandwidth() %>% get_pers_max_frac(x = pd_data))
grid_regular(b_dat)

(b_hom <- bandwidth() %>% get_pers_max_frac(x = pd_data, hom_degrees = seq(2L)))
grid_regular(b_hom)

# `weight_power` for `step_vpn_persistence_silhouette()`

(wp_man <- weight_power(range = c(1, 3)))
grid_regular(wp_man)

# `num_bars` for `step_vpn_tropical_coordinates()`

(nb_man <- num_bars(range = c(1L, 3L)))
grid_regular(nb_man)

# `num_bins` for `step_vpn_tent_template_functions()`

(nb_man <- num_bins(range = c(5L, 10L)))
grid_regular(nb_man)

# `tent_shift` for `step_vpn_tent_template_functions()`

(ts_man <- tent_shift(range = c(100, 200), trans = NULL))
grid_regular(ts_man)

(ts_dat <- tent_shift() %>% get_pers_min_mult(x = pd_data))
grid_regular(ts_dat)

(ts_hom <- tent_shift() %>% get_pers_min_mult(x = pd_data, hom_degrees = seq(2L)))
grid_regular(ts_hom)
```

`vpd-finalizers` *Finalizers for persistent homology vectorizations*

Description

These functions take a persistent homology vectorization parameter object and modify the `dials::unknown()` parts of `ranges` based on a data set and heuristics used in inaugural studies.

Usage

```
get_pairs_max(object, x, hom_degrees = NULL, ...)

get_pers_max_frac(
  object,
  x,
  hom_degree = NULL,
  log_vals = TRUE,
  frac = 1/100,
  ...
)

get_pers_min_mult(
  object,
  x,
  hom_degree = NULL,
  log_vals = TRUE,
  mult = 100,
  ...
)
```

Arguments

`object`	A `param` object or a list of `param` objects.
`x`	Persistence data in a recognizable format.
`...`	Other arguments to pass to the underlying parameter finalizer functions.
`hom_degree`, `hom_degrees`	Integer (vector) of homological degree(s).
`log_vals`	A logical: should the ranges be set on the log10 scale?
`frac`	A double for the fraction of the data to be used for the upper bound. For `get_n_frac_range()` and `get_batch_sizes()`, a vector of two fractional values are required.
`mult`	A double for the multiple of the data to be used for the lower bound.

Details

`get_pairs_max()` sets the upper bound to the maximum number of persistent pairs.

`get_pers_max_frac()` sets both bounds to fractions of the maximum finite persistence (lifespan). A single number is used as the lower bound fraction and takes the upper bound fraction to be 1.

`get_pers_min_mult()` sets both bounds to multiples of the minimum positive persistence (lifespan). A single number is used as the upper bound multiple and takes the lower bound multiple to be 1.

Value

An updated `param` object or a list of updated `param` objects depending on what is provided in `object`.

---

vpd-summarizers *Summarize topological data*

Description

These miscellaneous functions are used by various `get_*_range()` functions to finalize hyperparameter ranges.

Usage

```
ph_dim(x)

## Default S3 method:
ph_dim(x)

## S3 method for class 'matrix'
ph_dim(x)

## S3 method for class 'array'
ph_dim(x)

## S3 method for class 'data.frame'
ph_dim(x)

## S3 method for class 'dist'
ph_dim(x)

## S3 method for class 'ts'
ph_dim(x)

pairs_min(x, hom_degrees)

## Default S3 method:
```

```
pairs_min(x, hom_degrees)

## S3 method for class 'matrix'
pairs_min(x, hom_degrees)

## S3 method for class 'data.frame'
pairs_min(x, hom_degrees)

## S3 method for class 'diagram'
pairs_min(x, hom_degrees)

## S3 method for class 'PHom'
pairs_min(x, hom_degrees)

## S3 method for class 'persistence'
pairs_min(x, hom_degrees)

pairs_max(x, hom_degrees)

## Default S3 method:
pairs_max(x, hom_degrees)

## S3 method for class 'matrix'
pairs_max(x, hom_degrees)

## S3 method for class 'data.frame'
pairs_max(x, hom_degrees)

## S3 method for class 'diagram'
pairs_max(x, hom_degrees)

## S3 method for class 'PHom'
pairs_max(x, hom_degrees)

## S3 method for class 'persistence'
pairs_max(x, hom_degrees)

birth_range(x, hom_degree)

## Default S3 method:
birth_range(x, hom_degree)

## S3 method for class 'matrix'
birth_range(x, hom_degree)

## S3 method for class 'data.frame'
birth_range(x, hom_degree)
```

```
## S3 method for class 'diagram'
birth_range(x, hom_degree)

## S3 method for class 'PHom'
birth_range(x, hom_degree)

## S3 method for class 'persistence'
birth_range(x, hom_degree)

pers_max(x, hom_degree)

## Default S3 method:
pers_max(x, hom_degree)

## S3 method for class 'matrix'
pers_max(x, hom_degree)

## S3 method for class 'data.frame'
pers_max(x, hom_degree)

## S3 method for class 'diagram'
pers_max(x, hom_degree)

## S3 method for class 'PHom'
pers_max(x, hom_degree)

## S3 method for class 'persistence'
pers_max(x, hom_degree)

pers_min(x, hom_degree)

## Default S3 method:
pers_min(x, hom_degree)

## S3 method for class 'matrix'
pers_min(x, hom_degree)

## S3 method for class 'data.frame'
pers_min(x, hom_degree)

## S3 method for class 'diagram'
pers_min(x, hom_degree)

## S3 method for class 'PHom'
pers_min(x, hom_degree)

## S3 method for class 'persistence'
pers_min(x, hom_degree)
```

```
pers_range(x, hom_degree)

## Default S3 method:
pers_range(x, hom_degree)

## S3 method for class 'matrix'
pers_range(x, hom_degree)

## S3 method for class 'data.frame'
pers_range(x, hom_degree)

## S3 method for class 'diagram'
pers_range(x, hom_degree)

## S3 method for class 'PHom'
pers_range(x, hom_degree)

## S3 method for class 'persistence'
pers_range(x, hom_degree)

life_support(x, hom_degree)

## Default S3 method:
life_support(x, hom_degree)

## S3 method for class 'matrix'
life_support(x, hom_degree)

## S3 method for class 'data.frame'
life_support(x, hom_degree)

## S3 method for class 'diagram'
life_support(x, hom_degree)

## S3 method for class 'PHom'
life_support(x, hom_degree)

## S3 method for class 'persistence'
life_support(x, hom_degree)
```

Arguments

`x` Persistence data in a recognizable format.

`hom_degree`, `hom_degrees`
Integer (vector) of homological degree(s).

Details

The functions compute the following summaries:

- `ph_dim()`: Dimension of a data set for purposes of PH
- `pairs_min()`: Minimum number of persistent pairs of any degree
- `pairs_max()`: Maximum number of persistent pairs of any degree
- `birth_range()`: Range of finite birth values for a given degree
- `pers_max()`: Maximum positive finite persistence for a given degree
- `pers_min()`: Minimum positive finite persistence for a given degree
- `pers_range()`: Range of positive finite persistence for a given degree
- `life_support()`: Range of union of birth–death ranges for a given degree

Value

A vector of one or two numeric values.

Index

∗ **datasets**
mnist, 6
∗ **topological feature extraction via persistent homology**
step_pd_degree, 8
step_pd_point_cloud, 10
step_pd_raster, 12

bake(), *7*, *9*, *11*, *13*, *15*, *17*, *20*, *22*, *24*, *27*, *29*, *32*, *35*, *37*, *40*, *42*, *45*
bandwidth *(*vpd-dials*)*, 46
birth_range *(*vpd-summarizers*)*, 50
blur, 2
blur(), *4*, *7*
blur_sigmas, 3

dials::unknown(), *49*

finalize, *50*

get_blur_range, 4
get_hom_range *(*get_blur_range*)*, 4
get_pairs_max *(*vpd-finalizers*)*, 49
get_pers_max_frac *(*vpd-finalizers*)*, 49
get_pers_min_mult *(*vpd-finalizers*)*, 49

hom_degree *(*get_blur_range*)*, 4

img_sigma *(*vpd-dials*)*, 46

kernlab::sigest(), *5*

life_support *(*vpd-summarizers*)*, 50

max_hom_degree *(*get_blur_range*)*, 4
mnist, 6
mnist_test *(*mnist*)*, 6
mnist_train *(*mnist*)*, 6

num_bars *(*vpd-dials*)*, 46
num_bins *(*vpd-dials*)*, 46
num_coef *(*vpd-dials*)*, 46
num_levels *(*vpd-dials*)*, 46

pairs_max *(*vpd-summarizers*)*, 50
pairs_min *(*vpd-summarizers*)*, 50
pers_max *(*vpd-summarizers*)*, 50
pers_min *(*vpd-summarizers*)*, 50
pers_range *(*vpd-summarizers*)*, 50
ph_dim *(*vpd-summarizers*)*, 50
poly_type *(*vpd-dials*)*, 46
prep(), *7*, *9*, *11*, *13*, *15*, *17*, *20*, *22*, *24*, *27*, *29*, *32*, *35*, *37*, *40*, *42*, *44*, *45*

ripserr, *11*, *13*

selections(), *7*, *8*, *10*, *13*, *15*, *17*, *19*, *22*, *24*, *26*, *29*, *32*, *34*, *37*, *40*, *42*, *44*
step_blur, 6
step_pd_degree, 8, *11*, *14*
step_pd_point_cloud, *9*, 10, *14*
step_pd_raster, *9*, *11*, 12
step_vpd_algebraic_functions, 14
step_vpd_betti_curve, 16
step_vpd_complex_polynomial, 19
step_vpd_descriptive_statistics, 21
step_vpd_euler_characteristic_curve, 23
step_vpd_normalized_life_curve, 26
step_vpd_persistence_block, 28
step_vpd_persistence_image, 31
step_vpd_persistence_landscape, 34
step_vpd_persistence_silhouette, 36
step_vpd_persistent_entropy_summary, 39
step_vpd_tent_template_functions, 41
step_vpd_tropical_coordinates, 44

TDA, *11*
TDAvec::computeAlgebraicFunctions(), *15*

TDAvec::computeBettiCurve(), *18*
TDAvec::computeComplexPolynomial(), *20*, *47*
TDAvec::computeEulerCharacteristic(), *25*
TDAvec::computeNormalizedLife(), *27*
TDAvec::computePersistenceBlock(), *30*
TDAvec::computePersistenceImage(), *33*, *47*
TDAvec::computePersistenceLandscape(), *35*, *47*
TDAvec::computePersistenceSilhouette(), *38*, *47*
TDAvec::computePersistentEntropy(), *40*
TDAvec::computeStats(), *22*
TDAvec::computeTemplateFunction(), *43*, *47*
TDAvec::computeTropicalCoordinates(), *45*, *47*
tent_shift *(*vpd-dials*)*, 46
tidy.step_blur *(*step_blur*)*, 6
tidy.step_pd_degree *(*step_pd_degree*)*, 8
tidy.step_pd_point_cloud *(*step_pd_point_cloud*)*, 10
tidy.step_pd_raster *(*step_pd_raster*)*, 12
tidy.step_vpd_algebraic_functions *(*step_vpd_algebraic_functions*)*, 14
tidy.step_vpd_betti_curve *(*step_vpd_betti_curve*)*, 16
tidy.step_vpd_complex_polynomial *(*step_vpd_complex_polynomial*)*, 19
tidy.step_vpd_descriptive_statistics *(*step_vpd_descriptive_statistics*)*, 21
tidy.step_vpd_euler_characteristic_curve *(*step_vpd_euler_characteristic_curve*)*, 23
tidy.step_vpd_normalized_life_curve *(*step_vpd_normalized_life_curve*)*, 26
tidy.step_vpd_persistence_block *(*step_vpd_persistence_block*)*, 28
tidy.step_vpd_persistence_image *(*step_vpd_persistence_image*)*, 31
tidy.step_vpd_persistence_landscape *(*step_vpd_persistence_landscape*)*, 34
tidy.step_vpd_persistence_silhouette *(*step_vpd_persistence_silhouette*)*, 36
tidy.step_vpd_persistent_entropy_summary *(*step_vpd_persistent_entropy_summary*)*, 39
tidy.step_vpd_tent_template_functions *(*step_vpd_tent_template_functions*)*, 41
tidy.step_vpd_tropical_coordinates *(*step_vpd_tropical_coordinates*)*, 44

vpd-dials, 46
vpd-finalizers, 49
vpd-summarizers, 50

weight_func_pl *(*vpd-dials*)*, 46
weight_power *(*vpd-dials*)*, 46