Package: hypervolume (via r-universe)

October 17, 2026

Type Package

Title High Dimensional Geometry, Set Operations, Projection, and Inference Using Kernel Density Estimation, Support Vector Machines, and Convex Hulls

Version 3.1.7

Date 2026-09-17

Description Estimates the shape and volume of high-dimensional datasets and performs set operations: intersection / overlap, union, unique components, inclusion test, and hole detection. Uses stochastic geometry approach to high-dimensional kernel density estimation, support vector machine delineation, and convex hull generation. Applications include modeling trait and niche hypervolumes and species distribution modeling.

License GPL-3

Depends Rcpp, methods, R (>= 3.5.0)

LinkingTo Rcpp, RcppArmadillo, progress

Imports raster, maps, MASS, geometry, ks, hitandrun, pdist, fastcluster, compiler, e1071, progress, mvtnorm, data.table, terra, sp, foreach, doParallel, parallel, ggplot2, pbapply, palmerpenguins, purrr, dplyr, caret

Suggests rgl, magick, alphahull, knitr, rmarkdown, gridExtra

NeedsCompilation yes

RoxygenNote 7.1.1

URL https://github.com/bblonder/hypervolume

BugReports https://github.com/bblonder/hypervolume/issues

VignetteBuilder knitr

Author Benjamin Blonder [aut, cre], Cecina Babich Morrow [aut], Stuart Brown [aut], Gregoire Butruille [aut], Daniel Chen [aut], Alex Laini [aut], David J. Harris [aut], Clement Violet [aut]

Maintainer Benjamin Blonder <benjamin.blonder@berkeley.edu>

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

Date/Publication 2026-09-17 17:50:09 UTC

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

RemoteRef HEAD

RemoteSha bf3b1ac68e80ea781cb8b82d3f7d8bedfb9bcaf6

Contents

hypervolume-package . . . 3
acacia_pinus . . . 4
circles . . . 5
copy_param_hypervolume . . . 5
estimate_bandwidth . . . 6
expectation_ball . . . 8
expectation_box . . . 9
expectation_convex . . . 9
expectation_maximal . . . 11
find_optimal_occupancy_thin . . . 11
get_centroid . . . 13
get_centroid_weighted . . . 14
get_occupancy_intersection_bootstrap . . . 15
get_occupancy_stats . . . 17
get_occupancy_unshared_bootstrap . . . 19
get_occupancy_volume_bootstrap . . . 21
get_relative_volume . . . 23
get_volume . . . 25
hypervolume . . . 25
Hypervolume-class . . . 26
hypervolume_box . . . 27
hypervolume_distance . . . 28
hypervolume_distance_point . . . 29
hypervolume_estimate_probability . . . 31
hypervolume_funnel . . . 32
hypervolume_gaussian . . . 33
hypervolume_general_model . . . 35
hypervolume_holes . . . 36
hypervolume_inclusion_test . . . 38
hypervolume_join . . . 39
hypervolume_n_occupancy . . . 40
hypervolume_n_occupancy_permute . . . 43
hypervolume_n_occupancy_test . . . 45
hypervolume_n_resample . . . 47
hypervolume_overlap_confidence . . . 49
hypervolume_overlap_statistics . . . 50
hypervolume_overlap_test . . . 51
hypervolume_permute . . . 54

hypervolume_project . . . 55
hypervolume_prune . . . 57
hypervolume_redundancy . . . 58
hypervolume_resample . . . 59
hypervolume_save_animated_gif . . . 61
hypervolume_segment . . . 62
hypervolume_set . . . 63
hypervolume_set_n_intersection . . . 65
hypervolume_set_n_union . . . 67
hypervolume_svm . . . 69
hypervolume_thin . . . 70
hypervolume_threshold . . . 71
hypervolume_to_data_frame . . . 72
hypervolume_variable_importance . . . 74
HypervolumeList-class . . . 75
morphSnodgrassHeller . . . 75
occupancy_bootstrap_gof . . . 77
occupancy_filter . . . 78
occupancy_to_intersection . . . 79
occupancy_to_union . . . 81
occupancy_to_unshared . . . 83
padded_range . . . 85
plot.HypervolumeList . . . 86
print.Hypervolume . . . 89
quercus . . . 89
summary.Hypervolume . . . 90
to_hv_list . . . 91
weight_data . . . 92

Index **94**

`hypervolume-package` *High Dimensional Geometry, Set Operations, Projection, and Inference Using Kernel Density Estimation, Support Vector Machines, and Convex Hulls*

Description

Estimates the shape and volume of high-dimensional datasets and performs set operations: intersection / overlap, union, unique components, inclusion test, and hole detection. Uses stochastic geometry approach to high-dimensional kernel density estimation, support vector machine delineation, and convex hull generation. Applications include modeling trait and niche hypervolumes and species distribution modeling.

Details

A frequently asked questions document (FAQ) can be found at http://www.benjaminblonder.org/hypervolume_faq.html. More details are also available in a user guide within our 2018 paper (see reference below).

Author(s)

Benjamin Blonder [aut, cre], Cecina Babich Morrow [aut], Stuart Brown [aut], Gregoire Butruille [aut], Daniel Chen [aut], Alex Laini [aut], David J. Harris [aut], Clement Violet [aut]

Maintainer: Benjamin Blonder <benjamin.blonder@berkeley.edu>

References

Blonder, B., Lamanna, C., Violle, C. and Enquist, B. J. (2014), The n-dimensional hypervolume. Global Ecology and Biogeography, 23: 595-609. doi: 10.1111/geb.12146

Blonder, B. Do Hypervolumes Have Holes?, The American Naturalist, 187(4) E93-E105. doi: 10.1086/685444

Blonder, B., Morrow, C.B., Maitner, B., et al. New approaches for delineating n-dimensional hypervolumes. Methods Ecol Evol. 2018;9:305-319. doi: 10.1111/2041-210X.12865

---

acacia_pinus *Data for Acacia and Pinus tree distributions*

Description

Data for occurrences of Acacia and Pinus species based on geographic observations.

Usage

```
data(acacia_pinus)
```

Format

A `data.frame` with 37845 observations on the following 3 variables.

`Species` a `character` containing 139 unique values

`Latitude` a numeric vector

`Longitude` a numeric vector

Source

Occurrence data come from the BIEN database (`https://biendata.org/`).

References

Blonder, B., Lamanna, C., Violle, C., Enquist, B. The n-dimensional hypervolume. Global Ecology and Biogeography (2014).

circles *Circles simulated dataset*

Description

Data generated by picking 100 points randomly within a circle of radius 1. See the vignette on occupancy for information about its usage.

Usage

```
data(circles)
```

Format

A `list` with 20 objects of class `matrix`. Each `matrix` contains 100 rows and 2 columns.

`[,1]` x coordinate

`[,2]` y coordinate

copy_param_hypervolume *Generate hypervolumes using pre-existing parameters*

Description

`copy_param_hypervolume` takes in a hypervolume and data. After detecting the method used to generate the input hypervolume, the function returns a new hypervolume generated from the data using the same method and parameters as the input hypervolume.

Usage

```
copy_param_hypervolume(hv, data, name = NULL)
```

Arguments

`hv`	hypervolume object
`data`	A m x n matrix or data frame, where m is the number of observations and n is the dimensionality.
`name`	String name of hypervolume

Details

`copy_param_hypervolume` only works if the input hypervolume was generated using `method = "box"`, `method = "gaussian"`, or `method = "svm"`. Calling this function on hypervolumes generated from `hypervolume_set` will result in an error. Note that `kde.bandwidth` is affected by size of the data and will be re-estimated using whichever method was used to generate the original bandwidth if `method = "gaussian"` or `method = "box"`. Use `hv@Parameters` to see what parameters are copied from the input hypervolume.

Value

hypervolume object

Examples

```
## Not run:
library(palmerpenguins)
data("penguins")
bill_data = na.omit(penguins[,3:4])
hv = hypervolume(data = bill_data,
                 method = "gaussian",
                 quantile.requested = .9,
                 quantile.requested.type = "volume")

# Generates a new hypervolume using the same hypervolume and data
hv_copy = copy_param_hypervolume(hv, hv@Data)
# Check to see that the information of the two hypervolumes is the same
print(hv)
print(hv_copy)

## End(Not run)
```

---

estimate_bandwidth *Kernel bandwidth estimators for hypervolumes*

Description

Estimates bandwidth vector from data using multiple approaches.

Usage

```
estimate_bandwidth(data,method="silverman",value=NULL)
```

Arguments

data	m x n matrix or data frame, where m is the number of observations and n the number of dimensions.
method	One of "fixed", "silverman", "silverman-1d", "plug-in", or "cross-validation" - see 'details' section.
value	If method="fixed", a scalar or vector value to be used. Otherwise ignored.

Details

The fixed ("fixed") is a constant value (scalar or vector of length equal to the dimensionality of the data). The value can be set via the value argument. If the input has length 1, the value will be repeated for all dimensions.

The Silverman ("silverman") estimator is defined as (4/(n+2))^(1/(n+4)) * m^(-1/(n+4))*sd(X) where m is the number of observations, n is the dimensionality, and X is the data vector in each

dimension. This corresponds to the Silverman rule of thumb for multivariate data and is chosen as the default for computational speed, though other more advanced algorithms may perform better.

The Silverman (`"silverman-1d"`) estimator is defined as 1.06 * sd(X) * m^(-1/5) where m is the number of observations and X is the data vector in each dimension. Minimizes mean integrated square error under the assumption the data are univariate normal. This was the default behavior in versions 1.x and 2.x of the package.

The plug-in (`"plug-in"`) estimator is defined using a diagonal plug-in estimator with a 2-stage pilot estimation and a pre-scaling transformation (in `ks::Hpi.diag`). The resulting diagonal variances are then transformed to standard deviations and multiplied by two to be consistent for the box kernels used here. Available only in n<7 dimensions. Minimizes sum of asymptotic mean squared error.

The cross-validation (`"cross-validation"`) estimator is defined using a diagonal smoothed cross validation estimator with a 2-stage pilot estimation and a pre-scaling transformation (in `ks::Hscv.diag`). The resulting diagonal variances are then transformed to standard deviations and multiplied by two to be consistent for the box kernels used here. Available only in n<7 dimensions. Minimizes sum of asymptotic mean squared error.

Note that all estimators are optimal only for normal kernels, whereas the hypervolume algorithms use box kernels - as the number of data points increases, this difference will become increasingly less important.

Computational run-times for the plug-in and cross-validation estimators may become infeasibly large in n>=4 dimensions.

Value

Vector of length n with each entry corresponding to the estimated bandwidth along each axis. An attribute `method` is also set indicating the algorithm used.

References

Duong, T. (2007) ks: Kernel Density Estimation and Kernel Discriminant Analysis for Multivariate Data in R. Journal of Statistical Software 21, (7)

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                    c("bill_length_mm","bill_depth_mm","flipper_length_mm")]

estimate_bandwidth(penguins_adelie,method="fixed",value=c(2,1,2))
estimate_bandwidth(penguins_adelie,method="silverman")
estimate_bandwidth(penguins_adelie,method="plug-in") # may be quite slow to run
estimate_bandwidth(penguins_adelie,method="cross-validation") # may be quite slow to run

## End(Not run)
```

`expectation_ball` *Hypersphere expectation*

Description

Generates expectation hypervolume corresponding to a hypersphere that minimally encloses the data.

Usage

```
expectation_ball(input, point.density = NULL, num.samples = NULL,
                 use.random = FALSE)
```

Arguments

`input`	A m x n matrix or data frame, where m is the number of observations and n is the dimensionality.
`point.density`	The point density of the output expectation. If `NULL`, defaults to `v / num.points` where d is the dimensionality of the input and v is the volume of the hypersphere.
`num.samples`	The number of points in the output expectation. If `NULL`, defaults to `10^(3+sqrt(ncol(d)))` where d is the dimensionality of the input. `num.points` has priority over `point.density`; both cannot be specified.
`use.random`	If `TRUE` and the `input` is of class `Hypervolume`, sets boundaries based on the `@RandomPoints` slot; otherwise uses `@Data`.

Value

A `Hypervolume-class` object corresponding to the expectation.

Examples

```
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                    c("bill_length_mm","bill_depth_mm","flipper_length_mm")]
e_ball <- expectation_ball(penguins_adelie)
```

expectation_box *Hyperbox expectation*

Description

Generates expectation hypervolume corresponding to an axis-aligned hyperbox that minimally encloses the data.

Usage

```
expectation_box(input, point.density = NULL, num.samples = NULL, use.random = FALSE)
```

Arguments

input	A m x n matrix or data frame, where m is the number of observations and n is the dimensionality.
point.density	The point density of the output expectation. If NULL, defaults to v / num.points where d is the dimensionality of the input and v is the volume of the hypersphere.
num.samples	The number of points in the output expectation. If NULL, defaults to 10^(3+sqrt(ncol(d))) where d is the dimensionality of the input. num.points has priority over point.density; both cannot be specified.
use.random	If TRUE and the input is of class Hypervolume, sets boundaries based on the @RandomPoints slot; otherwise uses @Data.

Value

A Hypervolume-class object corresponding to the expectation.

Examples

```
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                    c("bill_length_mm","bill_depth_mm","flipper_length_mm")]
e_box <- expectation_box(penguins_adelie)
```

expectation_convex *Convex expectation*

Description

Generates expectation hypervolume corresponding to a convex hull (polytope) that minimally encloses the data.

Usage

```
expectation_convex(input, point.density = NULL, num.samples = NULL,
                   num.points.on.hull = NULL, check.memory = TRUE,
                   verbose = TRUE, use.random = FALSE, method =
                   "hitandrun", chunksize = 1000)
```

Arguments

input	A m x n matrix or data frame, where m is the number of observations and n is the dimensionality.
point.density	The point density of the output expectation. If NULL, defaults to v / num.points where d is the dimensionality of the input and v is the volume of the hypersphere.
num.samples	The number of points in the output expectation. If NULL, defaults to 10^(3+sqrt(ncol(d))) where d is the dimensionality of the input. num.points has priority over point.density; both cannot be specified.
num.points.on.hull	Number of points of the input used to calculate the convex hull. Larger values are more accurate but may lead to slower runtimes. If NULL, defaults to using all of the data (most accurate).
check.memory	If TRUE, reports expected number of convex hull simplices required for calculation and stops further memory allocation. Also warns if dimensionality is high.
verbose	If TRUE, prints diagnostic progress messages.
use.random	If TRUE and the input is of class Hypervolume, sets boundaries based on the @RandomPoints slot; otherwise uses @Data.
method	One of "rejection" (rejection sampling) or "hitandrun" (adaptive hit and run Monte Carlo sampling)
chunksize	Number of random points to process per internal step. Larger values may have better performance on machines with large amounts of free memory. Changing this parameter does not change the output of the function; only how this output is internally assembled.

Details

The rejection sampling algorithm generates random points within a hyperbox enclosing the points, then sequentially tests whether each is in or out of the convex polytope based on a dot product test. It becomes exponentially inefficient in high dimensionalities. The hit-and-run sampling algorithm generates a Markov chain of samples that eventually converges to the true distribution of points within the convex polytope. It performs better in high dimensionalities but may not converge quickly. It will also be slow if the number of simplices on the convex polytope is large.

Both algorithms may become impracticably slow in >= 6 or 7 dimensions.

Value

A Hypervolume-class object corresponding to the expectation hypervolume.

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                    c("bill_length_mm","bill_depth_mm","flipper_length_mm")]
e_convex <- expectation_convex(penguins_adelie, check.memory=FALSE)

## End(Not run)
```

`expectation_maximal` *Maximal expectation*

Description

Creates a hypervolume from a set of points reflecting the maximal expectation.

Usage

```
expectation_maximal(input, ...)
```

Arguments

`input`	A dataset to be used as input to the `hypervolume` function
`...`	Arguments to the `hypervolume` function

Details

This function is effectively an alias for the `hypervolume` function. You must decide what the maximal expectation is yourself!

Value

A Hypervolume object.

`find_optimal_occupancy_thin`
Find optimal parameters to calculate occupancy

Description

The `find_optimal_occupancy_thin()` function is used to find the optimal parameters for `hypervolume_n_occupancy()`.

Usage

```
find_optimal_occupancy_thin(...,
                            verbose = TRUE,
                            sequence = seq(0, 1, 0.1),
                            n = 10,
                            res_type = "raw")
```

Arguments

...	Parameters to be used to run hypervolume_n_occupancy().
verbose	Logical value; print diagnostic output if TRUE.
sequence	Quantiles to be tested.
n	Number of seeds to be tested.
res_type	If raw print all the seeds and quantiles tested together with the resulting root mean square error (RMSE). If summary print RMSE mean and standard deviation for each quantile.

Details

The find_optimal_occupancy_thin() function searches for the optimal parameters for running hypervolume_n_occupancy(). It works by testing different quantiles and n seeds for random number generation (the same set of n seeds is tested for each quantile). RMSE is returned as the measure of the goodness of fit and results are ordered by increasing RMSE when res_type = "raw". Quantile equal to 0 correspond to no thin. The obtained parameters can be used to feed arguments quant.thin and seed within the function hypervolume_n_occupancy().

Value

A data.frame.

See Also

hypervolume_n_occupancy

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))

# split the dataset on species and sex
penguins_no_na_split = split(penguins_no_na,
                             paste(penguins_no_na$species, penguins_no_na$sex, sep = "_"))

# calculate the hypervolume for each element of the splitted dataset
hv_list = mapply(function(x, y)
  hypervolume_gaussian(x[, c("bill_length_mm","bill_depth_mm","flipper_length_mm")],
                       samples.per.point=100, name = y),
  x = penguins_no_na_split,
```

```
  y = names(penguins_no_na_split))


# transform the list into an HypervolumeList
hv_list = hypervolume_join(hv_list)

# find optimal parameters
opt_par = find_optimal_occupancy_thin(hv_list,
                                      classification = rep(c("female", "male"), 3),
                                      n = 20)

head(opt_par)

unoptimized_hv_occ = hypervolume_n_occupancy(hv_list,
                        classification = rep(c("female", "male"), 3))

optimized_hv_occ = hypervolume_n_occupancy(hv_list,
                        classification = rep(c("female", "male"), 3),
                        quant.thin = opt_par[1, 2], seed = opt_par[1, 1])


## End(Not run)
```

`get_centroid` *Get centroid of hypervolume or hypervolume list*

Description

Returns the column mean of the random points in each hypervolume.

Usage

```
get_centroid(hv)
```

Arguments

hv	A `Hypervolume` or `HypervolumeList` object.

Value

Either a vector or a matrix of column of centroid values along each axis.

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                    c("bill_length_mm","bill_depth_mm","flipper_length_mm")]
hv = hypervolume_gaussian(penguins_adelie)
```

```
get_centroid(hv)

## End(Not run)
```

`get_centroid_weighted` *Get weighted centroid of hypervolume or hypervolume list*

Description

Returns the column weighted mean of the random points in each hypervolume. Useful for hypervolumes generated with `hypervolume_n_occupancy()` or `hypervolume_n_occupancy_test()`.

Usage

```
get_centroid_weighted(hv)
```

Arguments

hv	A `Hypervolume` or `HypervolumeList` object.

Details

The function `get_centroid_weighted()` differs from `get_centroid()` because it uses occupancy values to weight random points for evaluating centroids position.

Value

Either a vector or a matrix of column of centroid values along each axis.

See Also

`hypervolume_n_occupancy`, `hypervolume_n_occupancy_test`

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))


penguins_no_na_split = split(penguins_no_na,
paste(penguins_no_na$species, penguins_no_na$sex, sep = "_"))


hv_list = lapply(penguins_no_na_split, function(x)
  hypervolume_gaussian(x[, c("bill_length_mm","bill_depth_mm","flipper_length_mm")],
  samples.per.point=100))

hv_list = hypervolume_join(hv_list)
```

```
hv_occupancy = hypervolume_n_occupancy(hv_list)

# unweighted centroids
get_centroid(hv_occupancy)

# weighted centroids
get_centroid_weighted(hv_occupancy)


## End(Not run)
```

`get_occupancy_intersection_bootstrap` *Volume of the intersection of a bootstrapped occupancy object*

Description

The `get_occupancy_intersection_bootstrap()` function is used to get the volume of the intersection of objects generated with `hypervolume_n_occupancy_bootstrap()`. It provides raw values or summary statistics for all the hypervolumes or their n_wise combinations.

Usage

```
get_occupancy_intersection_bootstrap(path,
                                     method = "n_wise",
                                     res_type = "summary",
                                     m = 2,
                                     relative = FALSE,
                                     tol = 1e-10)
```

Arguments

`path`	A path to a directory of bootstrapped occupancy objects obtained with `hypervolume_n_occupancy_bootstrap()`.
`method`	If `all` compute the volume of the intersection among all the hypervolumes for each bootstrapped occupancy object found in `path`. If `n_wise` compute the volume of the intersection for each n_wise combination of hypervolumes within the bootstrapped occupancy objects found in `path`.
`res_type`	It can be `raw` or `pairwise`. See details.
`m`	Number of elements to choose. Default to 2 (pairwise comparisons). This argument is ignored when `method` is set to `all`.
`relative`	If `TRUE` it computes relative instead of absolute volumes.
`tol`	Set the tolerance for reconstructing whole volume. See details.

Details

The function `get_occupancy_intersection_bootstrap()` returns the volume of the intersection for each bootstrapped occupancy object if `res_type = "raw"` and `method = "all"`. When `res_type = "summary"` and `method = "all"` this function returns the mean volume as well as the standard deviation, median, minimum, maximum, 2.5% and 97.5% quantiles, skewness and kurtosis of the intersection. The same summary statistics are calculated for each n_wise combination of hypervolumes when `res_type = "summary"` and `method = "n_wise"`. The number of elements of n_wise combinations is set with the argument `m`. The intersection is calculated by finding the set of random points shared by all or n_wise combinations of hypervolumes in each of the bootstrapped occupancy objects. More details on how the intersection is computed in `occupancy_to_intersection()`.

The `get_occupancy_intersection_bootstrap()` function attempts to reconstruct the volume of the intersection from each bootstrapped occupancy object. At first, the volume of the union of hypervolumes is calculated for each hypervolume of the jth bootstrapped occupancy object as the ratio between the total number of random points and the number of random points of the ith hypervolume of the jth bootstrapped occupancy object, multiplied by the volume of the ith hypervolume of the jth bootstrapped occupancy object. This step results in a number of reconstructed volumes equal to the number of hypervolumes in the jth bootstrapped occupancy object. Reconstructed volumes are then compared among each other to ensure the consistency of the reconstruction. To do this, the distance among reconstructed volumes is calculated using the `dist()` function of the `stats` package. If at least one of the distances is greater than `tol` the computation is stopped and some suggestions are returned. The volume of the intersection is then calculated as the ratio between the number of random points of the intersection and the total number of random points, multiplied by the volume of the union of hypervolumes.

When `relative = TRUE` relative instead of absolute volumes are returned. The relative volume is calculated as the ratio between the volume of the intersection and the volume of the union of all the hypervolumes (or combination of hypervolumes when `method = "n_wise"`). The same approach described above is used to reconstruct the volume of the union of hypervolumes.

Value

A `data.frame` with bootstrapped volumes or summary statistics of the intersection.

See Also

`hypervolume_n_occupancy`, `hypervolume_n_occupancy_bootstrap`, `occupancy_to_intersection`

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))

# split the dataset on species and sex
penguins_no_na_split = split(penguins_no_na,
paste(penguins_no_na$species, penguins_no_na$sex, sep = "_"))

# calculate the hypervolume for each element of the splitted dataset
hv_list = mapply(function(x, y)
```

```
  hypervolume_gaussian(x[, c("bill_length_mm","bill_depth_mm","flipper_length_mm")],
                       samples.per.point=100, name = y),
                       x = penguins_no_na_split,
                       y = names(penguins_no_na_split))


# transform the list into an HypervolumeList
hv_list = hypervolume_join(hv_list)

# bootstrap the hypervolumes
hv_list_boot = hypervolume_n_resample(name = "example", hv_list)

# calculate occupancy on bootstrapped hypervolumes
hv_occupancy_boot_sex = hypervolume_n_occupancy_bootstrap(path = hv_list_boot,
                                    name = "example_occ",
                                    classification = rep(c("female", "male"), 3))

# get the intersection
get_occupancy_intersection_bootstrap(hv_occupancy_boot_sex)


## End(Not run)
```

`get_occupancy_stats` *Stats from occupancy objects*

Description

Functions `get_occupancy_stats()` and `get_occupancy_stats_bootstrap()` return the results of a function applied to hypervolumes generated with `hypervolume_n_occupancy()`, `hypervolume_n_occupancy_bootstr` `hypervolume_n_occupancy_permute()` or `hypervolume_n_occupancy_test()`.

Usage

```
get_occupancy_stats(hv, FUN, remove_zeroes = TRUE)

get_occupancy_stats_bootstrap(path,
                              FUN,
                              remove_zeroes = TRUE,
                              method = "pairwise",
                              res_type = "summary",
                              verbose = TRUE,
                              cores = 1)
```

Arguments

hv	A `Hypervolume` or `HypervolumeList` object generated with `hypervolume_n_occupancy()`, `hypervolume_n_occupancy_bootstrap()`, `hypervolume_n_occupancy_permute()` or `hypervolume_n_occupancy_test()`.

FUN	The function to be applied.
remove_zeroes	Remove zeroes before the calculation. See Details.
path	A path to a directory of bootstrapped hypervolumes obtained with hypervolume_n_occupancy_bootstra
method	If all returns the results for each hypervolume. If pairwise returns the results for all the pairwise comparisons of individual hypervolumes.
res_type	It can be raw or pairwise. See details.
verbose	Logical value; print diagnostic output if TRUE.
cores	Number of logical cores to use while generating permuted hypervolumes. If parallel backend already registered to doParallel, function will use that backend and ignore the argument in cores.

Details

The get_occupancy_stats() and get_occupancy_stats_bootstrap() functions take ValueAtRandomPoints of each hypervolume as input to FUN (e.g. mean, median).
The get_occupancy_stats_bootstrap() function applies the function to bootstrapped occupancy objects generated with hypervolume_n_occupancy_bootstrap(). If res_type = "raw" raw values of the applied functions are returned for each occupancy object in path, only when method = "all". If res_type = "summary" the mean value as well as the standard deviation, median, minimum, maximum, 2.5% and 97.5% quantiles, skewness and kurtosis are returned either for individual hypervolumes (method = "all") or pairwise comparisons (method = "pairwise").
The get_occupancy_stats() and get_occupancy_stats_bootstrap() functions remove occupancy values equal to 0 by default. These values are generated during the occupancy routine when a random point is included in some groups of hypervolumes but not in others. A tipical usage of get_occupancy_stats() or get_occupancy_stats_bootstrap() should remove zeroes before applying a function (the default).

Value

Either a vector, a matrix or a data.frame with the results of the applied function.

See Also

hypervolume_n_occupancy, hypervolume_n_occupancy_bootstrap, hypervolume_n_occupancy_permute, hypervolume_n_occupancy_test

Examples

```
## Not run:

##### single occupancy object #####
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))

# split the dataset on species and sex
penguins_no_na_split = split(penguins_no_na,
paste(penguins_no_na$species, penguins_no_na$sex, sep = "_"))
```

```
# calculate the hypervolume for each element of the splitted dataset
hv_list = mapply(function(x, y)
  hypervolume_gaussian(x[, c("bill_length_mm","bill_depth_mm","flipper_length_mm")],
                       samples.per.point=100, name = y),
                       x = penguins_no_na_split,
                       y = names(penguins_no_na_split))


# transform the list into an HypervolumeList
hv_list = hypervolume_join(hv_list)

# calculate occupancy based on sex
hv_occupancy_list_sex = hypervolume_n_occupancy(hv_list,
                          classification = rep(c("female", "male"), 3))


# calculate the mean occupancy value
get_occupancy_stats(hv_occupancy_list_sex, mean)

##### bootstrapped occupancy objects #####

# bootstrap input hypervolumes
hv_boot = hypervolume_n_resample(name = "example", hv_list = hv_list,  n = 9)

# calculate occupancy on bootstrapped hypervolumes
hv_boot_occ = hypervolume_n_occupancy_bootstrap(hv_boot, name = "example_occ",
                                  classification = rep(c("female", "male"), 3))

# calculate summary statistics for pairwise comparisons
get_occupancy_stats_bootstrap(hv_boot_occ, FUN = mean)


## End(Not run)
```

`get_occupancy_unshared_bootstrap` *Volume of the unshared fraction of a bootstrapped occupancy object*

Description

The `get_occupancy_unshared_bootstrap()` function is used to get the volume of the unshared fraction of an object generated with `hypervolume_n_occupancy_bootstrap()`. It provides raw values or summary statistics for both individual hypervolumes or their pairwise comparisons.

Usage

```
get_occupancy_unshared_bootstrap(path,
                                 method = "pairwise",
                                 res_type = "summary",
                                 relative = FALSE,
                                 tol = 1e-10)
```

Arguments

path	A path to a directory of bootstrapped occupancy objects obtained with hypervolume_n_occupancy_bootstrap().
method	If all compute the volume of the unique fraction of each hypervolume compared to all the hypervolumes for each occupancy object in path. If pairwise compute the difference of the volume of the unshared fraction for each pairwise combination of hypervolumes within the bootstrapped occupancy objects found in path.
res_type	It can be raw or pairwise. See details.
relative	If TRUE it computes relative instead of absolute volumes.
tol	Set the tolerance for reconstructing whole volume. See details.

Details

The function get_occupancy_unshared_bootstrap() returns the volume of the unshared fraction for each hypervolume in the bootstrapped occupancy object if res_type = "raw" and method = "all". When res_type = "summary" and method = "all" this function returns the mean volume as well as the standard deviation, median, minimum, maximum, 2.5% and 97.5% quantiles, skewness and kurtosis of the unshared fraction for each hypervolume. The same summary statistics are calculated for the difference of volume of the unshared fraction for each pairwise combination of hypervolumes when res_type = "summary" and method = "pairwise". The unshared fraction is calculated by finding the set of random points that are not shared with other hypervolumes or pairwise combinations of hypervolumes in each bootstrapped occupancy object. More details on how the unshared fraction is computed in occupancy_to_unshared().

The get_occupancy_unshared_bootstrap() function attempts to reconstruct the volume of the unshared fraction from each bootstrapped occupancy object. At first, the volume of the union of hypervolumes is calculated for each hypervolume of the jth bootstrapped occupancy object as the ratio between the total number of random points and the number of random points of the ith hypervolume of the jth bootstrapped occupancy object, multiplied by the volume of the ith hypervolume of the jth bootstrapped occupancy object. This step results in a number of reconstructed volumes equal to the number of hypervolumes in the jth bootstrapped occupancy object. Reconstructed volumes are then compared among each other to ensure the consistency of the reconstruction. To do this, the distance among reconstructed volumes is calculated with the dist() function of the stats package. If at least one of the distances is greater than tol the computation is stopped and some suggestions are returned. The volume of the unshared fraction is then calculated as the ratio between the number of random points of the unshared fraction and the total number of random points, multiplied by the volume of the union of hypervolumes.

When relative = TRUE relative instead of absolute volumes are returned. The relative volume is calculated as the ratio between the volume of the unshared fraction and the volume of the union of all the hypervolumes (or combination of hypervolumes when method = "pairwise"). The same approach described above is used to reconstruct the volume of the union of hypervolumes.

Value

A data.frame with bootstrapped volumes or summary statistics of the unshared fraction.

See Also

hypervolume_n_occupancy, hypervolume_n_occupancy_bootstrap occupancy_to_unshared

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))

# split the dataset on species and sex
penguins_no_na_split = split(penguins_no_na,
paste(penguins_no_na$species, penguins_no_na$sex, sep = "_"))

# calculate the hypervolume for each element of the splitted dataset
hv_list = mapply(function(x, y)
  hypervolume_gaussian(x[, c("bill_length_mm","bill_depth_mm","flipper_length_mm")],
                       samples.per.point=100, name = y),
                       x = penguins_no_na_split,
                       y = names(penguins_no_na_split))


# transform the list into an HypervolumeList
hv_list = hypervolume_join(hv_list)

# bootstrap the hypervolumes
hv_list_boot = hypervolume_n_resample(name = "example", hv_list)

# calculate occupancy on bootstrapped hypervolumes
hv_occupancy_boot_sex = hypervolume_n_occupancy_bootstrap(path = hv_list_boot,
                                    name = "example_occ",
                                    classification = rep(c("female", "male"), 3))

# get the unshared fraction
get_occupancy_unshared_bootstrap(hv_occupancy_boot_sex)


## End(Not run)
```

---

get_occupancy_volume_bootstrap

Extract the volume from occupancy bootstrap objects

Description

The function `get_occupancy_volume_bootstrap()` extract the volume from objects generated with `hypervolume_n_occupancy_bootstrap()`. It provides raw values or summary statistics for both single hypervolumes or their pairwise comparisons.

Usage

```
get_occupancy_volume_bootstrap(path,
                               method = "all",
                               res_type = "raw",
                               relative = FALSE,
                               tol = 1e-10)
```

Arguments

path	A path to a directory containing bootstrapped occupancy objects generated with hypervolume_n_occupancy_bootstrap().
method	If all the function returns the volume of each bootstrapped hypervolume for each bootstrapped occupancy object in path. If pairwise returns the volume difference for each pairwise combination of hypervolumes within the bootstrapped occupancy objects found in path.
res_type	It can be raw or pairwise. See details.
relative	If TRUE it computes relative instead of absolute volumes.
tol	Set the tolerance for reconstructing whole volume. See details.

Details

The function get_occupancy_volume_bootstrap() returns the volume for each bootstrapped hypervolume if res_type = "raw" and method = "all". When res_type = "summary" and method = "all" this function returns the mean volume as well as the standard deviation, median, minimum, maximum, 2.5% and 97.5% quantiles, skewness and kurtosis for each of hypervolume. The same summary statistics are calculated for the difference of volume for each pairwise combination of hypervolumes when res_type = "summary" and method = "pairwise".
When relative = TRUE relative instead of absolute volumes are returned. The relative volume is calculated as the ratio between the volume of an hypervolume and the volume of the union of all the hypervolumes. The get_occupancy_volume_bootstrap() function attempts to reconstruct the volume of the union of all the hypervolumes from each bootstrapped hypervolume. At first, the volume of the union of hypervolumes is calculated for each hypervolume of the jth bootstrapped occupancy_object as the the ratio between the total number of random points and the number of random points of the ith hypervolume of the jth bootstrapped occupancy_object, multiplied by the volume of the ith hypervolume of the jth bootstrapped occupancy_object. This step results in a number of reconstructed volumes equal to the number of hypervolumes in the jth bootstrapped occupancy_object. Reconstructed volumes are then compared among each other to ensure the consistency of the reconstruction. To do this, the distance among reconstructed volumes is calculated with the dist() function of the stats package. If at least one of the distances is greater than tol the computation is stopped and some suggestions are returned.

Value

A data.frame with bootstrapped volumes or summary statistics for single hypervolumes or their pairwise comparisons.

See Also

hypervolume_n_occupancy, hypervolume_n_occupancy_bootstrap

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))

# split the dataset on species and sex
penguins_no_na_split = split(penguins_no_na,
paste(penguins_no_na$species, penguins_no_na$sex, sep = "_"))

# calculate the hypervolume for each element of the splitted dataset
hv_list = mapply(function(x, y)
  hypervolume_gaussian(x[, c("bill_length_mm","bill_depth_mm","flipper_length_mm")],
                       samples.per.point=100, name = y),
                       x = penguins_no_na_split,
                       y = names(penguins_no_na_split))


# transform the list into an HypervolumeList
hv_list = hypervolume_join(hv_list)

# bootstrap the hypervolumes
hv_list_boot = hypervolume_n_resample(name = "example", hv_list)

# calculate occupancy on bootstrapped hypervolumes
hv_occupancy_boot_sex = hypervolume_n_occupancy_bootstrap(path = hv_list_boot,
                                    name = "example_occ",
                                    classification = rep(c("female", "male"), 3))

# get the volume of the bootstrapped hypervolumes
get_occupancy_volume_bootstrap(hv_occupancy_boot_sex)

## End(Not run)
```

---

get_relative_volume *Extract the relative volume*

Description

The function get_relative_volume() computes the relative volume from objects generated with the occupancy routine.

Usage

```
get_relative_volume(hv_list, tol = 1e-10)
```

Arguments

hv_list	A Hypervolume or HypervolumeList object generated with hypervolume_n_occupancy(), hypervolume_n_occupancy_permute(), hypervolume_n_occupancy_test(), occupancy_to_union(), occupancy_to_intersection(), occupancy_to_unshared(), or occupancy_filter().
tol	Set the tolerance for reconstructing whole volume. See details.

Details

The relative volume is calculated as the ratio between hypervolumes of an HypervolumeList and the volume resulting from the union of hypervolumes in the same HypervolumeList. Relative volumes can be calculated only for HypervolumeList generated with functions hypervolume_n_occupancy(), hypervolume_n_occupancy_test(), hypervolume_n_occupancy_permute(), occupancy_to_union(), occupancy_to_ushared(), occupancy_to_intersection() or occupancy_filter().
The get_relative_volume() function attempts to reconstruct the volume of the union of hypervolumes from hv_list. At first, the volume of the union of hypervolumes is calculated for each hypervolume of hv_list as the the ratio between the total number of random points and the number of random points of the ith hypervolume of hv_list, multiplied by the volume of the ith hypervolume hv_list. This step results in a number of reconstructed volumes equal to the number of hypervolumes in hv_list. Reconstructed volumes are then compared to ensure the consistency of the reconstruction. To do this, the distance among reconstructed volumes is calculated with the dist() function of the stats package. If at least one of the distances is greater than tol the computation is stopped and some suggestions are returned.

Value

A named numeric vector with the relative volume of each input hypervolume

See Also

hypervolume_n_occupancy, hypervolume_n_occupancy_permute, hypervolume_n_occupancy_test, occupancy_to_union, occupancy_to_unshared, occupancy_to_intersection, occupancy_filter

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))

# split the dataset on species and sex
penguins_no_na_split = split(penguins_no_na,
paste(penguins_no_na$species, penguins_no_na$sex, sep = "_"))

# calculate the hypervolume for each element of the splitted dataset
hv_list = mapply(function(x, y)
  hypervolume_gaussian(x[, c("bill_length_mm","bill_depth_mm","flipper_length_mm")],
                       samples.per.point=100, name = y),
                       x = penguins_no_na_split,
                       y = names(penguins_no_na_split))
```

```
# transform the list into an HypervolumeList
hv_list = hypervolume_join(hv_list)

# calculate occupancy based on sex
hv_occupancy_list_sex = hypervolume_n_occupancy(hv_list,
                          classification = rep(c("female", "male"), 3))


# get the relative volume
get_relative_volume(hv_occupancy_list_sex)

## End(Not run)
```

`get_volume` *Extract volume*

Description

Extract volume from Hypervolume or HypervolumeList object

Usage

```
## S3 method for class 'Hypervolume'
get_volume(object)
## S3 method for class 'HypervolumeList'
get_volume(object)
```

Arguments

`object`	A `Hypervolume` or `HypervolumeList` object

Value

A named numeric vector with the volume of each input hypervolume

---

`hypervolume` *Hypervolume construction methods*

Description

Constructs hypervolumes using one of several possible methods after error-checking input data.

Usage

```
hypervolume(data, method = "gaussian", ...)
```

Arguments

data	A m x n matrix or data frame, where m is the number of observations and n is the dimensionality.
method	One of "box" (box kernel density estimation), "gaussian" (Gaussian kernel density estimation), or "svm" (one-class support vector machine). See respective functions for details.
...	Further arguments passed to hypervolume_box, hypervolume_gaussian, or hypervolume_svm.

Details

Checks for collinearity, missingness of input data, and appropriate random point coverage. Generates warning/errors as appropriate.

Value

A Hypervolume-class object corresponding to the inferred hypervolume.

See Also

weight_data, estimate_bandwidth, expectation_convex, expectation_ball, expectation_box, hypervolume_threshold

Examples

```
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                    c("bill_length_mm","bill_depth_mm","flipper_length_mm")]
hv = hypervolume(penguins_adelie,method='box')
```

Hypervolume-class *Class* "Hypervolume"

Description

Primary storage class for stochastic descriptions of hypervolumes

Objects from the Class

Objects can be created by calls of the form new("Hypervolume", ...).

Slots

`Name`: Object of class `"character"` ~~ the name of the hypervolume

`Method`: Object of class `"character"` ~~ the method used to construct this hypervolume

`Data`: Object of class `"matrix"` ~~ May be empty if the hypervolume is not associated with data (e.g. convex expectation, set operations)

`Dimensionality`: Object of class `"numeric"` ~~ Dimensionality of the hypervolume

`Volume`: Object of class `"numeric"` ~~ Volume of the hypervolume

`PointDensity`: Object of class `"numeric"` ~~ Number of random points per unit volume

`Parameters`: Object of class `"list"` ~~ List of parameters that will depend on the method used to construct the hypervolume

`RandomPoints`: Object of class `"matrix"` ~~ A matrix of uniformly random points distributed within the hypervolume

`ValueAtRandomPoints`: Object of class `"numeric"` ~~ A vector of positive numbers representing the probabilty density at each random point in `@RandomPoints`

---

hypervolume_box *Hypervolume construction via hyperbox kernel density estimation*

Description

Constructs a hypervolume from a set of observations via thresholding a kernel density estimate of the observations. Assumes an axis-aligned hyperbox kernel.

Usage

```
hypervolume_box(data, name = NULL, verbose = TRUE, samples.per.point =
                  ceiling((10^(3 + sqrt(ncol(data))))/nrow(data)),
                  kde.bandwidth = 2*estimate_bandwidth(data),
                  tree.chunksize = 10000)
```

Arguments

`data`	A m x n matrix or data frame, where m is the number of observations and n is the dimensionality.
`name`	A string to assign to the hypervolume for later output and plotting. Defaults to the name of the variable if NULL.
`verbose`	Logical value; print diagnostic output if `TRUE`.
`samples.per.point`	Number of random points to be evaluated per data point in `data`.
`kde.bandwidth`	A scalar or a n x 1 vector corresponding to the half-width of the box kernel in each dimension. If a scalar input, the single value is used for all dimensions. Several esimation methods are available in `estimate_bandwidth`.
`tree.chunksize`	Number of random points to process per internal step. Larger values may have better performance on machines with large amounts of free memory. Changing this parameter does not change the output of the function; only how this output is internally assembled.

Details

Constructs a kernel density estimate by overlaying hyperbox kernels on each datapoint, then sampling uniformly random points from each kernel. Kernel density at each point is then determined by a range query on a recursive partitioning tree and used to resample these random points to a uniform density and fixed number, from which a volume can be inferred.

Note that when comparing among hypervolumes constructed with fixed bandwidth, volume will be approximately a an approximately linear function of the number of input data points.

Note that this function returns an unthresholded hypervolume. To assign a quantile threshold, use `hypervolume_threshold`.

Value

A `Hypervolume-class` object corresponding to the inferred hypervolume.

See Also

`hypervolume_threshold`, `estimate_bandwidth`

Examples

```
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                    c("bill_length_mm","bill_depth_mm","flipper_length_mm")]
hv = hypervolume_box(penguins_adelie,name='Adelie')
summary(hv)
```

---

hypervolume_distance *Distance between two hypervolumes*

Description

Calculates the distance between two hypervolumes either defined as the Euclidean distance between centroids or as the minimum Euclidean distance between the random points comprising either hypervolume.

Usage

```
hypervolume_distance(hv1, hv2, type = "centroid",
  num.points.max = 1000, check.memory = TRUE)
```

Arguments

hv1	A Hypervolume object.
hv2	A Hypervolume object.
type	If 'centroid', the centroid distance; if 'minimum', the minimum distance.
num.points.max	The number of random points to subsample from each input hypervolume. Ignored if type='centroid'.
check.memory	If TRUE, prints expected memory usage and returns an error before allocating memory. Ignored if type='centroid'.

Details

Minimum distance calculations scale quadratically with npmax and may be computationally costly.

Value

The distance between the two hypervolumes.

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                    c("bill_length_mm","bill_depth_mm","flipper_length_mm")]
penguins_chinstrap = penguins_no_na[penguins_no_na$species=="Chinstrap",
                      c("bill_length_mm","bill_depth_mm","flipper_length_mm")]

hv1 = hypervolume_gaussian(penguins_adelie)
hv2 = hypervolume_gaussian(penguins_chinstrap)

# note that minimum distance is smaller than centroid distance as expected
hypervolume_distance(hv1, hv2, type='centroid')
hypervolume_distance(hv1, hv2, type='minimum', num.points.max=500, check.memory=FALSE)

## End(Not run)
```

---

hypervolume_distance_point

Estimate the distance between a focal point and a sampled hypervolume point.

Description

Estimate the Euclidean distance between a focal point and the nearest or farthest randomly sampled point within a hypervolume.

Code by Clement Violet.

Usage

```
hypervolume_distance_point(hv1, x, type = "minimum",
  num.points.max = 1000, check.memory = TRUE)
```

Arguments

hv1	A Hypervolume object.
x	An object coercible to a matrix object.
type	If 'minimum', compute the smallest distance to the hypervolume margin; if 'maximum' calculates the greatest distance to the margin of the hypervolume.
num.points.max	The number of random points to subsample from the input hypervolume.
check.memory	If TRUE, prints expected memory usage and returns an error before allocating memory. Ignored if type='centroid'.

Details

Accuracy of the estimation relies on the num.points.max. However, distance calculations scale quadratically with num.points.max and may be computationally costly.

Value

The Euclidean distance between the focal point and the nearest or farthest randomly sampled point within the hypervolume.

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                                 c("bill_length_mm","bill_depth_mm","flipper_length_mm")]

hv1 = hypervolume_gaussian(penguins_adelie)

point = penguins_no_na[penguins_no_na$species=="Chinstrap",
                       c("bill_length_mm","bill_depth_mm","flipper_length_mm")][42, ]

hypervolume_distance_point(hv1, point, type='minimum', num.points.max=500, check.memory=FALSE)
hypervolume_distance_point(hv1, point, type='maximum', num.points.max=500, check.memory=FALSE)

## End(Not run)
```

hypervolume_estimate_probability
Estimate probability a given location

Description

Estimates probability density at one or more of points within or outside a hypervolume. The estimation is carried out as the weighted sum of the probability density of all subsampled random points in the input hypervolume, where the weights are proportional to the distance from the test point raised to a certain power. The default power, -1, corresponds to inverse distance weighting.

Usage

```
hypervolume_estimate_probability(hv, points,
                reduction.factor = 1, weight.exponent = -1,
                set.edges.zero = TRUE, edges.zero.distance.factor = 1,
                parallel = FALSE, n.cores = 1,
                verbose = TRUE, ...)
```

Arguments

hv	An input hypervolume
points	A m x n matrix of m points of dimensionality n (same as the input hypervolume). These are the points at which the probability is to be estimated.
reduction.factor	A value between 0 and 1 corresponding to a thinning factor applied to random points of the input hypervolume. Smaller values result in faster runtimes but lower accuracy.
weight.exponent	The exponent of the distance weights. Should be negative and probably does not need to be changed.
set.edges.zero	If TRUE, any test points more than a critical distance (multiplied by edges.zero.distance.factor) away from a random point in the input hypervolume are assumed to have probability zero. Otherwise the weighted sum is used with no further modification.
edges.zero.distance.factor	Positive number used to multiply the critical distance for set.edges.zero. Larger values lead to more stringent criteria for test points being set to zero.
parallel	If TRUE, uses multiple cores.
n.cores	Number of cores to use in parallel operation.
verbose	If TRUE, prints diagnostic progress messages.
...	Other arguments to be passed to pbsapply for parallelization.

Details

Identifies the uniformly random points enclosed within a hypersphere centered on the point of interest, then averages the probability density at each of these points.

Value

A vector of probability densities of length corresponding to m, the number of input points.

See Also

hypervolume_inclusion_test, hypervolume_redundancy

Examples

```
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                    c("bill_length_mm","bill_depth_mm","flipper_length_mm")]
hv = hypervolume_box(penguins_adelie,name='Adelie')

new_points = data.frame(bill_length_mm=c(0,38), bill_depth_mm=c(0,18),flipper_length_mm=c(0,190))

probs <- hypervolume_estimate_probability(hv, points=new_points)
probs
# should give a zero value and a non-zero value

# example for parallel operation
# probs_new <- hypervolume_estimate_probability(hv, points=new_points, parallel=TRUE, n.cores=2)
```

---

hypervolume_funnel *Hypervolumes at different sample sizes*

Description

This function takes in hypervolumes bootstrapped at different sample sizes applies a function to each hypervolume. The output of the function can either be a plot of nonparametric confidence intervals or a table of the mean and quantiles.

Usage

```
hypervolume_funnel(input_path,
                   title = NULL,
                   func = get_volume,
                   CI = .95,
                   as_table = FALSE)
```

Arguments

input_path	output of resample with method = "bootstrap seq"; path to a sequence of different sample sized bootstraps
title	title of output plot, ignore if outputting as table
func	a function that takes a single parameter which is a hypervolume and returns a numerical value.

CI	Confidence interval is taken by using the the (1-CI)/2 and (1+CI)/2 quantile
as_table	If TRUE, returns a table with columns upper quantile, mean, lower quantile

Details

This function is used to evaluate the behavior of hypervolumes at different sample sizes and determine bias. Statistics such as volume are affected by sample size especially when the hypervolumes are constructed with method = "gaussian" since the bandwidth estimate is dependent on sample size.

Value

ggplot object, or dataframe object

Examples

```
## Not run:
# 3000 data point hypervolume
data(quercus)
hv_quercus = hypervolume(quercus[,c(2,3)])

# the seq argument is equivalent to a length 30 vector {10, 139, ... , 3649, 3779}
# 6hr sequential runtime
quercus_bootstrap_seq <- resample('quercus_bootstrap_seq',
                                  hv_quercus,
                                  method = 'bootstrap seq',
                                  points_per_resample = "sample_size",
                                  seq = floor(seq(10, 3779, length.out = 30)),
                                  cores = 20)

# Compatible with ggplot syntax when used with as_table = FALSE
hypervolume_funnel(quercus_bootstrap_seq,
                   title = 'Resampled volumes of Quercus',
                   func = get_volume) +
  geom_line(aes(y = get_volume(hv_quercus))) +
  ylab("Volume")

## End(Not run)
```

---

hypervolume_gaussian *Hypervolume construction via Gaussian kernel density estimation*

Description

Constructs a hypervolume by building a Gaussian kernel density estimate on an adaptive grid of random points wrapping around the original data points. The bandwidth vector reflects the axis-aligned standard deviations of a hyperelliptical kernel.

Because Gaussian kernel density estimates do not decay to zero in a finite distance, the algorithm evaluates the kernel density in hyperelliptical regions out to a distance set by sd.count.

After delineating the probability density, the function calls hypervolume_threshold to determine a boundary. The defaullt behavior ensures that 95 percent of the stimated probability density is enclosed by the chosen boundary. However note that theaccuracy of the total probability density depends on having set a large value of sd.count.

Most use cases should not require modification of any parameters except kde.bandwidth.

Optionally, weighting of the data (e.g. for abundance-weighting) is possible. By default, the function estimates the probability density of the observations via Gaussian kernel functions, assuming each data point contributes equally. By setting a weight parameter, the algorithm can instead take a weighted average the kernel functions centered on each observation. Code for weighting data written by Yuanzhi Li (Yuanzhi.Li@usherbrooke.ca).

Usage

```
hypervolume_gaussian(data, name = NULL,
                       weight = NULL,
                samples.per.point = ceiling((10^(3 + sqrt(ncol(data))))/nrow(data)),
                       kde.bandwidth = estimate_bandwidth(data),
                       sd.count = 3,
                       quantile.requested = 0.95,
                       quantile.requested.type = "probability",
                       chunk.size = 1000,
                       verbose = TRUE,
                       ...)
```

Arguments

data	A m x n matrix or data frame, where m is the number of observations and n is the dimensionality.
name	A string to assign to the hypervolume for later output and plotting. Defaults to the name of the variable if NULL.
weight	An optional vector of weights for the kernel density estimation. Defaults to even weighting (rep(1/nrow(data),nrow(data))) if NULL.
samples.per.point	Number of random points to be evaluated per data point in data.
kde.bandwidth	A bandwidth vector obtained by running estimate_bandwidth Note that previous package version (<3.0.0) allowed inputting a scalar/vector value here - this is now handled through the estimate_bandwidth interface.
sd.count	The number of standard deviations (converted to actual units by multiplying by kde.bandwidth) at which the 'edge' of the hypervolume should be evaluated. Larger values of threshold.sd.count will come closer to a true estimate of the Gaussian density over a larger region of hyperspace, but require rapidly increasing computational resources (see Details section). It is generally better to use a large/default value for this parameter. Warnings will be generated if chosen to take a value less than 3.
quantile.requested	The quantile value used to delineate the boundary of the kernel density estimate. See hypervolume_threshold.

quantile.requested.type
: The type of quantile (volume or probability) used for the boundary delineation. See hypervolume_threshold.

chunk.size
: Number of random points to process per internal step. Larger values may have better performance on machines with large amounts of free memory. Changing this parameter does not change the output of the function; only how this output is internally assembled.

verbose
: Logical value; print diagnostic output if TRUE.

...
: Other arguments to pass to hypervolume_threshold

Value

A Hypervolume-class object corresponding to the inferred hypervolume.

See Also

hypervolume_threshold

Examples

```
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                    c("bill_length_mm","bill_depth_mm","flipper_length_mm")]

# low samples per point for CRAN demo
hv = hypervolume_gaussian(penguins_adelie,name='Adelie',samples.per.point=100)
summary(hv)
```

---

hypervolume_general_model

Generates hypervolume by sampling from arbitrary model object.

Description

Uses rejection sampling to obtain predicted values of a model object at uniformly random points within a range box, then converts output to a hypervolume.

Usage

```
hypervolume_general_model(model, name = NULL, verbose = TRUE,
 data = NULL, range.box = NULL, num.samples = ceiling(10^(3 + sqrt(ncol(range.box)))),
  chunk.size = 10000, min.value = 0, ...)
```

Arguments

model	Any model object which can be used within a predict(model, newdata, ...) call.
name	Name of the output hypervolume
verbose	If TRUE, prints diagnostic output.
data	If not NULL, used to specify range.box=padded_range(data).
range.box	A 2 x n matrix, where n is the number of dimensions of the data, and the first row corresponds to a lower limit and the second row to an upper limit. Each column is thus the low and high limits of the range box along each axis. Can be generated via padded_range.
num.samples	Number of samples to draw from the range box.
chunk.size	Number of samples to process in each predict call. Changing this value may affect the speed of function return but not the returned values.
min.value	If TRUE, discards sampled values below this threshold. Effectively used to set hypervolume boundaries.
...	Other arguments to be passed to predict, e.g. type='response'.

Value

A Hypervolume-class object corresponding to retained values within the hyperbox of interest.

Examples

```
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_no_na$is_adelie = penguins_no_na$species=="Adelie"
penguins_no_na = penguins_no_na[,c("is_adelie","bill_length_mm","bill_depth_mm")]


m_glm = glm(is_adelie~.,data=penguins_no_na)

hv_general_glm = hypervolume_general_model(m_glm,
  range.box=padded_range(penguins_no_na[,2:3]),type='response')
plot(hv_general_glm)
```

---

hypervolume_holes *Hole detection*

Description

Detects the holes in an observed hypervolume relative to an expectation

Usage

```
hypervolume_holes(hv.obs, hv.exp, set.num.points.max = NULL, set.check.memory = TRUE)
```

Arguments

hv.obs	The observed hypervolume whose holes are to be detected
hv.exp	The expected hypervolume that provides a baseline expectation geometry
set.num.points.max	Maximum number of points to be used for set operations comparing hv_obs to hv_exp. Defaults to 10^(3+sqrt(n)), where n is the dimensionality of the input hypervolumes.
set.check.memory	If TRUE, estimates the memory usage required to perform set operations, then exits. If FALSE, prints resource usage and continues algorithm. It is useful for preventing crashes to check the estimated memory usage on large or high dimensional datasets before running the full algorithm.

Details

This algorithm has a good Type I error rate (rarely detects holes that do not actually exist). However it can have a high Type II error rate (failure to find holes when they do exist). To reduce this error rate, make sure to re-run the algorithm with input hypervolumes with higher values of @PointDensity, or increase set.num.points.max.

The algorithm performs the set difference between the observed and expected hypervolumes, then removes stray points in this hypervolume by deleting any random point whose distance from any other random point is greater than expected.

A 'rule of thumb' is that algorithm has acceptable statistical performance when log_e(m) > n, where m is the number of data points and n is the dimensionality.

Value

A Hypervolume object containing a uniformly random set of points describing the holes in hv_obs. Note that the point density of this object is likely to be much lower than that of the input hypervolumes due to the stochastic geometry algorithms used.

Examples

```
## Not run:
# generate annulus data
data_annulus <- data.frame(matrix(data=runif(4000),ncol=2))
names(data_annulus) <- c("x","y")
data_annulus  <- subset(data_annulus,
sqrt((x-0.5)^2+(y-0.5)^2) > 0.4 & sqrt((x-0.5)^2+(y-0.5)^2) < 0.5)

# MAKE HYPERVOLUME (low reps for fast execution)
hv_annulus <- hypervolume_gaussian(data_annulus,
                kde.bandwidth=0.05,name='annulus',samples.per.point=1)

# GET CONVEX EXPECTATION
hv_convex <- expectation_convex(hypervolume_thin(hv_annulus,num.samples=500),
                check.memory=FALSE,use.random=TRUE)

# DETECT HOLES (low npoints for fast execution)
```

```
features_annulus <- hypervolume_holes(
                      hv.obs=hv_annulus,
                      hv.exp=hv_convex,
                      set.check.memory=FALSE)

# CLEAN UP RESULTS
features_segmented <- hypervolume_segment(features_annulus,
                        check.memory=FALSE,distance.factor=2)
features_segmented_pruned <- hypervolume_prune(features_segmented,
                                volume.min=0.02)

# PLOT RETAINED HOLE(S)
plot(hypervolume_join(hv_annulus, features_segmented_pruned))

## End(Not run)
```

`hypervolume_inclusion_test` *Inclusion test*

Description

Determines if a set of points are within a hypervolume. Can operate using a 'fast' algorithm which determines whether at least one random point of the hypervolume is within a critical distance of the test point. This algorithm is very efficient but leads to noisy and error-prone results when the point density slow. A warning is generated if this algorithm is used.

The function can also operate using an 'accurate' algorithm which estimates the probability density at the test point, and rejects it if it is below the requested threshold value. This is very slow but guarantees good results.

Usage

```
hypervolume_inclusion_test(hv, points, reduction.factor = 1, fast.or.accurate =
                 "fast", fast.method.distance.factor = 1,
                 accurate.method.threshold =
                 quantile(hv@ValueAtRandomPoints,
                 0.5), verbose = TRUE, ...)
```

Arguments

`hv`	n-dimensional hypervolume to compare against
`points`	Candidate points. A m x n matrix or dataframe, where m is the number of candidate points and n is the number of dimensions.
`reduction.factor`	A number in (0,1] that represents the fraction of random points sampled from the hypervolume for the stochastic inclusion test. Larger values are more accurate but computationally slower.

fast.or.accurate
: If 'fast', uses the critical distance test. If 'accurate', uses a probability density estimate.

fast.method.distance.factor
: Numeric value; multiplicative factor applied to the critical distance for all inclusion tests (see below). Used only when fast.or.accurate='fast'.

accurate.method.threshold
: Numeric value; threshold probability value below which the point is determined to be out of the hypervolume. Used only when fast.or.accurate='accurate'.

verbose
: Logical value; print diagnostic output if true.

...
: Additional arguments to be passed to either hypervolume_estimate_probability or hypervolume_inclusion_test.

Value

A m x 1 logical vector indicating whether each candidate point is in the hypervolume.

Examples

```
## Not run:
# construct a hypervolume of points in the unit square [0,1] x [0,1]
data = data.frame(x=runif(100,min=0,max=1), y=runif(100,min=0,max=1))
hv = hypervolume_gaussian(data)

# test if (0.5,0.5) and (-1,1) are in - should return TRUE FALSE
hypervolume_inclusion_test(hv, points=data.frame(x=c(0.5,-1),y=c(0.5,-1)))

## End(Not run)
```

---

hypervolume_join *Concatenate hypervolumes*

Description

Combines multiple hypervolumes or hypervolume lists into a single HypervolumeList suitable for analysis or plotting.

Usage

```
hypervolume_join(...)
```

Arguments

...
: One or more objects of class Hypervolume or HypervolumeList, or a list() of Hypervolume objects.

Value

A HypervolumeList containing all hypervolumes in all arguments.

Examples

```
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                    c("bill_length_mm","bill_depth_mm","flipper_length_mm")]
penguins_chinstrap = penguins_no_na[penguins_no_na$species=="Chinstrap",
                      c("bill_length_mm","bill_depth_mm","flipper_length_mm")]

hv1 = hypervolume_box(penguins_adelie,name='Adelie')
hv2 = hypervolume_box(penguins_chinstrap,name='Chinstrap')

hvs_joined = hypervolume_join(hv1, hv2)
```

---

hypervolume_n_occupancy

Operations for groups of hypervolumes

Description

Computes the occupancy of hyperspace by one or more groups of hypervolumes.

Usage

```
hypervolume_n_occupancy(hv_list,
                        classification = NULL,
                        method = "subsample",
                        FUN = mean,
                        num.points.max = NULL,
                        verbose = TRUE,
                        distance.factor = 1,
                        check.hyperplane = FALSE,
                        box_density = 5000,
                        thin = FALSE,
                        quant.thin = 0.5,
                        seed = NULL,
                        print_log = FALSE)

hypervolume_n_occupancy_bootstrap(path,
                                  name = NULL,
                                  classification = NULL,
                                  method = "subsample",
                                  FUN = mean,
                                  num.points.max = NULL,
```

```
                                    verbose = TRUE,
                                    distance.factor = 1,
                                    check.hyperplane = FALSE,
                                    box_density = 5000,
                                    thin = FALSE,
                                    quant.thin = 0.5,
                                    seed = NULL)
```

Arguments

hv_list	An HypervolumeList.
classification	A vector assigning each hypervolume in the HypervolumeList to a group.
method	Can be subsample or box. See details.
FUN	A function to aggregate points within each group. Default to mean.
num.points.max	Maximum number of random points to use for set operations. If NULL defaults to 10^(3+sqrt(n)) where n is the dimensionality of the input hypervolumes. Note that this default parameter value has been increased by a factor of 10 since the 1.2 release of this package.
verbose	Logical value; print diagnostic output if TRUE.
distance.factor	Numeric value; multiplicative factor applied to the critical distance for all inclusion tests (see below). Recommended to not change this parameter.
check.hyperplane	Check if data is hyperplanar.
box_density	Density of random points to fill the hyperbox when method is equal to box.
thin	Take a subsample of random points to get a more uniform distribution of random points. Intended to be used with method = "subsample", but can be used with method = "box" too. Can be slow, especially in high dimensions. See details.
quant.thin	Set quantile for using when thin = TRUE. See details.
seed	Set seed for random number generation. Useful for having reproducible results and with the use of find_optimal_occupancy_thin()
print_log	Save a log file with the volume of each input hypervolume, recomputed volume and the ratio between the original and recomputed hypervolumes. It works for hypervolume_n_occupancy() only.
path	A path to a directory of bootstrapped hypervolumes obtained with hypervolume_n_resample().
name	File name; The function writes hypervolumes to file in "./Objects/<name>".

Details

Uses the inclusion test approach to count how many hypervolumes include each random point. Counts range from 0 (no hypervolumes contain a given random point), to the number of hypervolumes in a group (all the hypervolumes contain a given random point). A function FUN, usually mean or sum, is then applied. A hypervolume is then returned for each group and the occupancy stored in ValueAtRandomPoints. IMPORTANT: random points with ValueAtRandomPoints equal to 0 are

not removed to ease downstream calculation.
When method = "subsample" the computation is performed on a random sample from input hypervolumes, constraining each to have the same point density given by the minimum of the point density of each input hypervolume and the point density calculated using the volumes of each input hypervolume divided by num.points.max.
Because this algorithm is based on distances calculated between the distributions of random points, the critical distance (point density ^ (-1/n)) can be scaled by a user-specified factor to provide more or less liberal estimates (distance_factor greater than or less than 1).
Two methods can be used for calculating the occupancy. The method subsample is based on a random sample of points from input hypervolumes. Each point is selected with a probability set to the inverse of the number of neighbour points calculated according to the critical distance. This method performs accurately when input hypervolumes have a low degree of overlap. The method box create a bounding box around the union of input hypervolumes. The bounding box is filled with points following a uniform distribution and with a density set with the argument box_density. A greater density usually provides more accurate results. The method box performs better than the method subsample in low dimensions, while in higher dimensions the method box become computationally inefficient as nearly all of the hyperbox sampling space will end up being empty and most of the points will be rejected.
When verbose = TRUE the volume of each input hypervolume will be printed to screen togheter with the recomputed volume and the ratio between the original and recomputed hypervolumes. Mean absolute error (MAE) and root mean square error (RMSE) are also provided as overall measures of the goodness of fit. A log file will be saved in the working directory with the information about the volume of input hypervolumes, the recomputed volume and the ratio between the original and recomputed hypervolumes.
When thin = TRUE an algorithm is applied to try to make the distribution of random points more uniform. Moderate departures from uniform distribution can in fact result from applying hypervolume_n_occupancy() on hypervolumes with a high overlap degree. At first, the algorithm in thin calculates the minimum distance from the neighboor points within the critical distance for each random point. A quantile (set with quant.thin) of these distances is taken and set as the threshold distance. Random points are then subset so that the distance of a point to another is greater than the threshold distance.
The function hypervolume_n_occupancy_bootstrap() takes a path of bootstrapped hypervolumes generated with hypervolume_n_resample() as input. It creates a directory called Objects in the current working directory if a directory of that name doesn't already exist where storing occupancy objects. The function hypervolume_n_occupancy_bootstrap() returns the absolute path to the directory with bootstrapped hypervolumes. It automatically saves a log file with the volume of each input hypervolume, the recomputed volume and the ratio between the original and recomputed hypervolumes. The log file is used with occupancy_bootstrap_gof().

Value

hypervolume_n_occupancy() returns a Hypervolume or HypervolumeList whose number of hypervolumes equals the number of groups in classification. hypervolume_n_occupancy_bootstrap() returns a string containing an absolute path equivalent to ./Objects/<name>.

See Also

find_optimal_occupancy_thin, occupancy_bootstrap_gof

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))

# split the dataset on species and sex
penguins_no_na_split = split(penguins_no_na,
                        paste(penguins_no_na$species, penguins_no_na$sex, sep = "_"))


# calculate the hypervolume for each element of the splitted dataset
hv_list = mapply(function(x, y)
  hypervolume_gaussian(x[, c("bill_length_mm", "flipper_length_mm")],
                       samples.per.point=100, name = y),
  x = penguins_no_na_split,
  y = names(penguins_no_na_split))

hv_list <- hypervolume_join(hv_list)

# calculate occupancy without groups
hv_occupancy <- hypervolume_n_occupancy(hv_list)
plot(hv_occupancy, cex.random = 1)

# calculate occupancy with groups
hv_occupancy_list_sex <- hypervolume_n_occupancy(hv_list,
                          classification = rep(c("female", "male"), each = 3))

plot(hv_occupancy_list_sex, cex.random = 1, show.density = FALSE)


### hypervolume_n_occupancy_bootstrap  ###

# bootstrap the hypervolumes
hv_list_boot = hypervolume_n_resample(name = "example", hv_list)

# calculate occupancy on bootstrapped hypervolumes
hv_occupancy_boot_sex = hypervolume_n_occupancy_bootstrap(path = hv_list_boot,
                                    name = "example_occ",
                                    classification = rep(c("female", "male"), 3))


## End(Not run)
```

`hypervolume_n_occupancy_permute`

Hypervolumes through permuting labels of n pairwise groups of hypervolumes

---

Description

Permute labels of an `hypervolume_n_occupancy()` object and calculate `hypervolume_n_occupancy()` for the permuted objects. This function is meant for taking a sample of all permutations and does not guarantee that permutations are not repeated. Newly generated hypervolume objects are written to file. This function is to be used within the occupancy routine.

Usage

```
hypervolume_n_occupancy_permute(name,
                                hv_list1,
                                hv_list2,
                                verbose = TRUE,
                                n = 9,
                                cores = 1)
```

Arguments

`name`	File name; The function writes hypervolumes to file in ./Objects/<name>
`hv_list1`	An hypervolume list generated with `hypervolume_n_occupancy()`.
`hv_list2`	The hypervolume list used to generate `hv_list1`.
`verbose`	Logical value; outputs progress bar in console.
`n`	Number of permutations to take.
`cores`	Number of logical cores to use while generating permuted hypervolumes. If parallel backend already registered to `doParallel`, function will use that backend and ignore the argument in `cores`.

Details

`hypervolume_n_occupancy_permute()` creates a directory called Objects in the current working directory if a directory of that name doesn't already exist. Within this directory, it creates a directory for each pairwise combinations of elements within `hv_list1`. Group labels are permuted and a new `HypervolumeList` is saved as a rds file for each pairwise combination. IMPORTANT: only group labels are permuted, random points are kept fixed and will be the same across all the permuted hypervolumes.

`hypervolume_n_occupancy_test()` takes in a `hypervolume_n_occupuancy_permute()` filepath output.

It is also possible to access the hypervolumes by using readRDS to read the hypervolume objects in one by one.

Value

Returns a string containing an absolute path equivalent to ./Objects/<name>

Warning

`hypervolume_n_occupancy_permute()` requires a lot of disk space especially when building occupancy hypervolumes with `method = "box"`. Try with a small number of replications and check the folder Objects for memory usage before to proceed.

See Also

hypervolume_n_occupancy

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))


# split the dataset on species and sex
penguins_no_na_split = split(penguins_no_na,
                        paste(penguins_no_na$species, penguins_no_na$sex, sep = "_"))


# calculate the hypervolume for each element of the splitted dataset
hv_list = mapply(function(x, y)
  hypervolume_gaussian(x[, c("bill_length_mm", "flipper_length_mm")],
                       samples.per.point=100, name = y),
  x = penguins_no_na_split,
  y = names(penguins_no_na_split))


names(hv_list) <- names(penguins_no_na_split)
hv_list <- hypervolume_join(hv_list)


hv_occupancy_list_sex <- hypervolume_n_occupancy(hv_list,
                              classification = rep(c("female", "male"), each = 3))

# takes 9 permutations on 1 core
hypervolume_n_occupancy_permute("permute", hv_occupancy_list_sex,
                                hv_list , n = 9, cores = 1)




## End(Not run)
```

---

hypervolume_n_occupancy_test

Significance of random points occupancy

Description

The function hypervolume_n_occupancy_test() calculates the significance of the difference between occupancy values for each random point and for all the pairwise combinations of groups in objects generated with hypervolume_n_occupancy() and hypervolume_n_occupancy_permute().

Usage

```
hypervolume_n_occupancy_test(observed,
                             path,
                             alternative = "two_sided",
                             significance = 0.05,
                             cores = 1,
                             p_adjust = "none",
                             multi_comp_type = "pairwise")
```

Arguments

observed	An HypervolumeList generated from hypervolume_n_occupancy().
path	A path to a directory of permuted hypervolumes generated with hypervolume_n_occupancy_permute().
alternative	Alternative hypothesis, can be one of two_sided, more or less.
significance	Significance values lower than this threshold will be retained.
cores	Number of logical cores to use while generating permuted hypervolumes. If parallel backend already registered to doParallel, function will use that backend and ignore the argument in cores.
p_adjust	Method of correction for multiple comparisons, set to none by default. Otherwise look at p.adjust of the stats package for alternatives.
multi_comp_type	Type of multiple comparison. Can be pairwise, for which the number of comparisons is set to the length of ValueAtRandomPoints, all, for which the number of comparisons is set to the length of ValueAtRandomPoints times the number of groups, or none.

Details

The observed difference between ValueAtRandomPoints of two groups is compared against null expectations generated with hypervolume_n_occupancy_permute().

Value

An HypervolumeList of length equal to the number of pairwise combinations of the observed HypervolumeList elements. ValueAtRandomPoints are calculated as the difference between the ValueAtRandomPoints of the first and the second group for each pairwise combination. Only significant values are retained according to significance.

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))


# split the dataset on species and sex
penguins_no_na_split = split(penguins_no_na,
                        paste(penguins_no_na$species, penguins_no_na$sex, sep = "_"))
```

```
# calculate the hypervolume for each element of the splitted dataset
hv_list = mapply(function(x, y)
  hypervolume_gaussian(x[, c("bill_length_mm", "flipper_length_mm")],
                       samples.per.point=100, name = y),
  x = penguins_no_na_split,
  y = names(penguins_no_na_split))

names(hv_list) <- names(penguins_no_na_split)
hv_list <- hypervolume_join(hv_list)


hv_occupancy_list_sex <- hypervolume_n_occupancy(hv_list,
                               classification = rep(c("female", "male"), each = 3))

# takes 9 permutations on 1 core
hyper_permuted <- hypervolume_n_occupancy_permute("permute", hv_occupancy_list_sex,
                                 hv_list , n = 99, cores = 1)

hypervolume_test <- hypervolume_n_occupancy_test(hv_occupancy_list_sex, hyper_permuted,
                      alternative = "more", significance = 0.1)



## End(Not run)
```

`hypervolume_n_resample` *Bootstrap n hypervolumes*

Description

The function `hypervolume_n_resample()` generates n hypervolumes using data bootstrapped from original data of the input hypervolumes.

Usage

```
hypervolume_n_resample(name,
                       hv_list,
                       n = 10,
                       points_per_resample = 'sample_size',
                       cores = 1,
                       verbose = TRUE,
                       seed = NULL)
```

Arguments

`name`	File name; The function writes hypervolumes to file in "./Objects/<name>""

hv_list	A Hypervolume or HypervolumeList object.
n	Number of resamples to take. Used for every method.
points_per_resample	Number of points in each resample. If the input is sample_size, then the same number of points as the original sample is used.
cores	Number of logical cores to use while generating bootstraped hypervolumes. If parallel backend already registered to doParallel, function will use that backend and ignore the argument in cores.
verbose	Logical value; If function is being run sequentially, outputs progress bar in console.
seed	Set seed for random number generation.

Details

hypervolume_n_resample() creates a directory called Objects in the current working directory if a directory of that name doesn't already exist. A directory is then created for each hypervolume in hv_list. Returns an absolute path to directory with resampled hypervolumes.
It is possible to access the hypervolumes by using readRDS to read the hypervolume objects one by one.
The resampled hypervolumes are generated using the same parameters used to generate the input hypervolume. The only exception is that the bandwidth is re-estimated if method = "gaussian" or method = "box". See copy_param_hypervolume for more details.

Value

Returns a string containing an absolute path equivalent to ./Objects/<name>

See Also

hypervolume_n_occupancy_bootstrap

Examples

```
## Not run:

library(palmerpenguins)
data(penguins)
bill_data = na.omit(penguins[,3:4])
hv = hypervolume(bill_data)

# Example 1: get 50 resampled hypervolumes for each input hypervolume
# Use detectCores to see how many cores are availible in current environment
# Set cores = 1 to run sequentially (default)
# bootstrap the hypervolumes
hv_list_boot = hypervolume_n_resample(name = "example", hv_list, n = 50)


## End(Not run)
```

hypervolume_overlap_confidence *Confidence intervals for overlap statistics*

Description

Generates confidence intervals of four different overlap statistics. In order to find the confidence interval for the overlap statistics of two hypervolumes, use `hypervolume_resample` twice to generate bootstraps. The function takes in paths to two sets of bootstrapped hypervolumes and gets overlap statistics for each possible pair. Confidence interval is calculated by taking a quantile of generated overlap statistics.

Usage

```
hypervolume_overlap_confidence(path1, path2, CI = .95, cores = 1)
```

Arguments

`path1`	A path to a directory of bootstrapped hypervolumes
`path2`	A path to a directory of bootstrapped hypervolumes
`CI`	Desired confidence interval proportion
`cores`	Number of logical cores to use while generating overlap statistics. If parallel backend already registered to `doParallel`, function will use that backend and ignore the argument in cores.

Details

The four overlap statistics are Sorensen, Jaccard, frac_unique_1, frac_unique_2. See `hypervolume_overlap_statistics`

Each hypervolume from path1 is overlapped with each hypervolume from path2 using `hypervolume_set`. The four overlap statistics are calculated for each overlap.

Value

`jaccard`	Confidence interval for jaccard similarity score
`sorensen`	Confidence interval for sorensen similarity score
`frac_unique_1`	Confidence interval for fraction of first hypervolume that is unique
`frac_unique_2`	Confidence interval for fraction of second hypervolume that is unique
`distribution`	a matrix of overlap statistics used to generate the confidence intervals

See Also

`hypervolume_resample`

Examples

```
## Not run:
# Let us overlap two hypervolumes generated from multivariate nomral
# distributions with different means and same covariance matrices.
sample1 = rmvnorm(150, mean = c(0, 0))
sample2 = rmvnorm(150, mean = c(0.5, 0.5))

hv1 = hypervolume(sample1)
hv2 = hypervolume(sample2)

# generates confidence intervals from quantiles of 20*20 overlaps
path1 = hypervolume_resample("mean_0_0", hv1, n = 20)
path2 = hypervolume_resample("mean_0.5_0.5", hv2, n = 20)

result = hypervolume_overlap_confidence(path1, path2)
# confidence index of Sorensen coefficient
print(result["sorensen"])


## End(Not run)
```

---

hypervolume_overlap_statistics

Overlap statistics for set operations (Sorensen, Jaccard, etc.)

Description

Calculates overlap metrics for two hypervolumes

Usage

```
hypervolume_overlap_statistics(hvlist)
```

Arguments

hvlist	A set of hypervolumes calculated from hypervolume_set

Value

A set of multiple metrics

jaccard	Jaccard similarity (volume of intersection of 1 and 2 divided by volume of union of 1 and 2)
sorensen	Sorensen similarity (twice the volume of intersection of 1 and 2 divided by volume of 1 plus volume of 2)
frac_unique_1	Unique fraction 1 (volume of unique component of 1 divided by volume of 1))
frac_unique_2	Unique fraction 2 (volume of unique component of 2 divided by volume of 2))

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                    c("bill_length_mm","bill_depth_mm","flipper_length_mm")]
penguins_chinstrap = penguins_no_na[penguins_no_na$species=="Chinstrap",
                      c("bill_length_mm","bill_depth_mm","flipper_length_mm")]

hv1 = hypervolume_box(penguins_adelie,name='Adelie')
hv2 = hypervolume_box(penguins_chinstrap,name='Chinstrap')

hv_set <- hypervolume_set(hv1, hv2, check.memory=FALSE)

hypervolume_overlap_statistics(hv_set)

## End(Not run)
```

---

hypervolume_overlap_test

Null distribution for overlap statistics

Description

Generates null distribution of four different overlap statistics under the null hypothesis that two samples are drawn from the same population. Observed value of overlap statistic is calculated from inputed hypervolumes. calculates p value for observed value of each statistic with respect to the generated null distributions.

Usage

```
hypervolume_overlap_test(hv1, hv2, path, alternative = "one-sided", bins = 100, cores = 1)
```

Arguments

hv1	A hypervolume object
hv2	A hypervolume object
path	a path to a directory containing permuted hypervolumes, bootstrapped hypervolumes, or a vector of two paths to bootstrapped hypervolumes
alternative	"one-sided" or "two-sided"
bins	plotting parameter for histogram of overlap statistics
cores	Number of logical cores to use while generating overlap statistics. If parallel backend already registered to doParallel, function will use that backend and ignore the argument in cores.

Details

Generating overlap statistics can be parallelized using the `cores` argument.

hypervolume_overlap_test can generate a null distribution from the output of hypervolume_permute, hypervolume_resample with method = "bootstrap", or a vector of two bootstrap outputs. See examples for how to use each case.

path should point to hypervolumes generated from the two input hypervolumes. There are three valid choices:

path is generated from `hypervolume_permute(<name>, hv1, hv2, ...)`. In this case the null distribution is generated by taking the overlap statistics of every single pair of permutations and turning them into a histogram.

OR

path is generated by resampling the hypervolume generated by combining the data of hv1 and hv2 If the number of data points used to generate hv1 is the same as hv2 then the path is `hypervolume_resample(<name>, hv_combined, "bootstrap", points_per_resample = nrow(hv1@Data))`. In this case, the list bootstrapped hypervolumes is split in half and overlap statistics are taken for every possible pair of hypervolumes from the two halves. A histogram of these overlap statitics represent the null distribution.

If the number of data points is different between hv1 and hv2 path is a list of two paths generated from `hypervolume_resample(<name>, hv_combined, "bootstrap", points_per_resample = nrow(hv1@Data), ...)` and `hypervolume_resample(<name>, hv_combined, "bootstrap", points_per_resample = nrow(hv2@Data), ...)`. Overlap statistics are taken for every possible pair of hypervolumes from each bootstrap. A histogram of these overlap statitics represent the null distribution. See example for appropriate path inputs.

The four overlap statistics are Sorensen, Jaccard, frac_unique_1, frac_unique_2. See `hypervolume_overlap_statistics` for description of the statistics.

Value

`p_values`	a list of p_values indexed by the name of the relevant statistic
`plots`	a list of ggplot objects indexed by the name of the relevant statistic. The observed value of each statistic is represented as a vertical line on the x axis.
`distribution`	a matrix of overlap statistics used to generate the null distribution

See Also

`hypervolume_resample`, `hypervolume_permute`

Examples

```
## Not run:
# We will use the data in "quercus" as our population in this example
data("quercus")
# Consider taking two samples of size 150 from the population and you want to figure out whether
# the samples are similar by seeing if they occupy the same area in feature space.
qsample1 = quercus[sample(1:nrow(quercus), 150),]
qsample2 = quercus[sample(1:nrow(quercus), 150),]
```

```
# Construct two hypervolumes from the samples
hv1 = hypervolume(qsample1[,2:3])
hv2 = hypervolume(qsample2[,2:3])

# Approach 1
# Take 200 permutations of the 300 data points. Using more cores is faster.
perm_path = hypervolume_permute("Quercus_perm_150", hv1, hv2, n = 200, cores = 20)

# hypervolume_overlap_test takes perm_path as an input.
# Results include p values for the overlap statistics of hv1 and hv2 as well as
# the corresponding null distributions generated from perm_path.
results1 = hypervolume_overlap_test(hv1, hv2, perm_path, cores = 20)

# Approach 2
# Under our null hypothesis the samples come from the same population.
# Approximate the original population by combining the data
# then simulate drawing 150 data points 50 times.
hv_combine = hypervolume(rbind(qsample1[,2:3],qsample2[,2:3]))
bootstrap_path = hypervolume_resample("Quercus_boot_150",
                                      hv_combine,
                                      method = "bootstrap",
                                      n = 50,
                                      points_per_resample = 150,
                                      cores = 20)

# hypervolume_overlap_test splits the 50 resampled hypervolumes in half and gets
# overlap statistic for each of the 25*25 pairs to generate the null
# distribution. This method allows us to approximate the null distribution using
# 625 data points while only generating 50 hypervolumes as opposed to
# hypervolume_permute which uses 400 hypervolumes to generate 200 data points.
results2 = hypervolume_overlap_test(hv1, hv2, bootstrap_path)

# Approach 3
# Suppose we have a size 300 sample and a size 150 sample and we want to know
# whether they come from the same distribution.
qsample3 = quercus[sample(1:nrow(quercus), 300),]
hv3 = hypervolume(qsample3[,2:3])

# Permutation still works in this case, however we can also use bootstrap by
# combining the data and drawing size 150 then size 300 samples.
hv_combine = hypervolume(rbind(qsample1[,2:3],qsample3[,2:3]))
b150_path = resample("Quercus_150",
                     hv_combine,
                     method = "bootstrap",
                     n = 25,
                     points_per_resample = 150,
                     cores = 20)
b300_path = resample("Quercus_300",
                     hv_combine,
                     method = "bootstrap",
                     n = 25,
                     points_per_resample = 300,
```

```
                    cores = 20)

# hypervolume_overlap_test generates overlap statistics for each of the 25*25
# possible pairs of size 150 and size 300 hypervolumes.
results3 = hypervolume_overlap_test(hv1, hv2, c(b150_path, b300_path), cores = 1)

## End(Not run)
```

`hypervolume_permute` *Hypervolumes through permuting data of two hypervolumes*

Description

Takes two data of two hypervolume objects (with the same column labels) and generates pairs of hypervolumes with the original sizes of the input hypervolumes but with permuted data (the rows of the original hypervolumes' data are combined and redistributed to the two new hypervolumes). This function is meant for taking a sample of all permutations and does not guarantee that permutations are not repeated. Newly generated hypervolume objects are written to file.

Usage

```
hypervolume_permute(name,
                    hv1,
                    hv2,
                    n = 50,
                    cores = 1,
                    verbose = TRUE)
```

Arguments

`name`	File name; The function writes hypervolumes to file in ./Objects/<name>
`hv1`	A hypervolume object
`hv2`	A hypervolume object
`n`	number of permutations to take
`cores`	Number of logical cores to use while generating permuted hypervolumes. If parallel backend already registered to `doParallel`, function will use that backend and ignore the argument in cores.
`verbose`	Logical value; If function is being run sequentially, outputs progress bar in console.

Details

`hypervolume_permute` creates a directory called Objects in the current working directory if a directory of that name doesn't already exist. Returns an absolute path to directory with permuted hypervolumes. rds files are stored in separate subdirectories for each permutation. Use `hypervolume_permute` when generating null distribution of overlap statistics. `hypervolume_overlap_test` takes in a `hypervolume_permute` filepath output.

It is also possible to access the hypervolumes by using readRDS to read the hypervolume objects in one by one.

Value

returns a string containing an absolute path equivalent to ./Objects/<name>

See Also

hypervolume_overlap_test

Examples

```
## Not run:
data("quercus")
# For this example consider taking two samples of size 150 from the data.
qsample1 = quercus[sample(1:nrow(quercus), 150),]
qsample2 = quercus[sample(1:nrow(quercus), 150),]

# Construct two hypervolumes from the samples
hv1 = hypervolume(qsample1[,2:3])
hv2 = hypervolume(qsample2[,2:3])

# Take 200 permutations of the 300 data points. Using more cores is faster.
perm_path = hypervolume_permute("Quercus_perm_150", hv1, hv2, n = 200, cores = 20)

# hypervolume_overlap_test takes perm_path as an input.
# Results include p value for the overlap statistics of hv1 and hv2 as well as
# null distribution generated from perm_path. The null distribution assumes data
# for hv1 and hv2 are drawn from the same distribution and permuting data will
# not change the overlap statitsics.
results = hypervolume_overlap_test(hv1, hv2, perm_path)

## End(Not run)
```

---

hypervolume_project *Geographical projection of hypervolume for species distribution modeling, using the hypervolume as the environmental niche model.*

Description

Determines a suitability score by calculating the hypervolume value at each of a set of points in an input raster stack based on either a probability density estimation or inclusion test.

Note that projected values are not normalized and are not necessarily constrained to fall between 0 and 1.

Note also that additional arguments can be passed to this function to enable parallel operation (see ... below).

Usage

```
hypervolume_project(hv, rasters, type = "probability", verbose = TRUE,
                    ...)
```

Arguments

hv	An input hypervolume
rasters	A RasterStack with the same names as the dimension names of the hypervolume.
type	If 'probability', suitability scores correspond to probability density values estimated using hypervolume_estimate_probability; if 'inclusion', scores correspond to binary presence/absence values from calling hypervolume_inclusion_test.
...	Additional arguments to be passed to either hypervolume_estimate_probability or hypervolume_inclusion_test.
verbose	If TRUE, prints diagnostic and progress output.

Value

A raster object of same resolution and extent as the input layers corresponding to suitability values.

See Also

hypervolume_estimate_probability, hypervolume_inclusion_test

Examples

```
## Not run:
  # load in lat/lon data
  data('quercus')
  data_alba = subset(quercus, Species=="Quercus alba")[,c("Longitude","Latitude")]
  data_alba = data_alba[sample(1:nrow(data_alba),500),]

  # get worldclim data from internet
  require(maps)
  require(raster)
  climatelayers = getData('worldclim', var='bio', res=10, path=tempdir())

  # z-transform climate layers to make axes comparable
  climatelayers_ss = climatelayers[[c(1,12)]]
  for (i in 1:nlayers(climatelayers_ss))
  {
    climatelayers_ss[[i]] <-
     (climatelayers_ss[[i]] - cellStats(climatelayers_ss[[i]], 'mean')) /
     cellStats(climatelayers_ss[[i]], 'sd')
  }
  climatelayers_ss = crop(climatelayers_ss, extent(-150,-50,15,60))

  # extract transformed climate values
```

```
  climate_alba = extract(climatelayers_ss, data_alba[1:300,])

  # compute hypervolume
  hv_alba <- hypervolume_gaussian(climate_alba)

  # do geographical projection
  raster_alba_projected_accurate <- hypervolume_project(hv_alba,
                                    rasters=climatelayers_ss)
  raster_alba_projected_fast = hypervolume_project(hv_alba,
                                    rasters=climatelayers_ss,
                                    type='inclusion',
                                    fast.or.accurate='fast')

  # draw map of suitability scores
  plot(raster_alba_projected_accurate,xlim=c(-100,-60),ylim=c(25,55))
  map('usa',add=TRUE)

  plot(raster_alba_projected_fast,xlim=c(-100,-60),ylim=c(25,55))
  map('usa',add=TRUE)

## End(Not run)
```

`hypervolume_prune` *Removes small hypervolumes from a HypervolumeList*

Description

Identifies hypervolumes characterized either by a number of uniformly random points or a volume below a user-specified value and removes them from a `HypervolumeList`.

This function is useful for removing small features that can occur stochastically during segmentation after set operations or hole detection.

Usage

```
hypervolume_prune(hvlist, num.points.min = NULL, volume.min = NULL, return.ids=FALSE)
```

Arguments

`hvlist`	A `HypervolumeList` object.
`num.points.min`	The minimum number of points in each input hypervolume.
`volume.min`	The minimum volume in each input hypervolume
`return.ids`	If `TRUE`, returns indices of input list as well as a pruned hypervolume list

Details

Either `minnp` or `minvol` (but not both) must be specified.

Value

A HypervolumeList pruned to only those hypervolumes of sizes above the desired value. If returnids=TRUE, instead returns a list structure with first item being the HypervolumeList and the second item being the indices of the retained hypervolumes.

See Also

hypervolume_holes, hypervolume_segment

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                    c("bill_length_mm","bill_depth_mm","flipper_length_mm")]

hv = hypervolume_gaussian(penguins_adelie,name='Adelie')

hv_segmented <- hypervolume_segment(hv,
                          num.points.max=200, distance.factor=1,
                          check.memory=FALSE) # intentionally under-segment
hv_segmented_pruned <- hypervolume_prune(hv_segmented,
                          num.points.min=20)
plot(hv_segmented_pruned)

## End(Not run)
```

---

hypervolume_redundancy

Redundancy of a point in a hypervolume

Description

Estimates squared probability density at a given point. This metric is proportional to the number of data points multiplied by the probability density at a point.

Usage

```
hypervolume_redundancy(...)
```

Arguments

...	Arguments to be passed to hypervolume_estimate_probability

See Also

hypervolume_estimate_probability

hypervolume_resample *Hypervolume resampling methods*

Description

hypervolume_resample generates new hyperolumes based on the method input. Outputs written to file.

- "bootstrap": Generates n hypervolumes using data bootstrapped from original data

- "bootstrap seq": Generates n hypervolumes for each sample size in sequence specified by user

- "weighted bootstrap": Same procedure as bootstrap but with sampling weights applied to data (default weights are normal distributed)

Usage

```
hypervolume_resample(name,
                     hv,
                     method,
                     n = 10,
                     points_per_resample = "sample_size",
                     seq = 3:nrow(hv@Data),
                     cores = 1,
                     verbose = TRUE,
                     to_file = TRUE,
                     mu = NULL,
                     sigma = NULL,
                     cols_to_weigh = 1:ncol(hv@Data),
                     weights = NULL)
```

Arguments

name	File name; The function writes hypervolumes to file in ./Objects/<name>
hv	A hypervolume object
method	String input; options are "bootstrap", "bootstrap seq", and "weighted bootstrap".
n	Number of resamples to take. Used for every method.
points_per_resample	Number of points in each resample. If the input is "sample_size", then the same number of points as the original sample is used. Used for method = "bootstrap" and method = "weighted bootstrap".
seq	Sequence of sample sizes. If method = "bootstrap seq" then the function generates n bootstrapped hypervolumes for each sample size in seq. Used for method = "bootstrap seq".
cores	Number of logical cores to use while generating bootstraped hypervolumes. If parallel backend already registered to doParallel, function will use that backend and ignore the argument in cores.

verbose	Logical value; If function is being run sequentially, outputs progress bar in console.
to_file	Logical value; If TRUE, writes resampled hypervolumes to file, otherwise returns an object from HypervolumeList-class. Used for method = "bootstrap" and method = "weighted bootstrap"
mu	Array of values specifying the mean of multivariate normal weights. Used for method = "weighted bootstrap".
sigma	Array of values specifying the variance in each dimension. (higher variance corresponds to more even weights) Used for method = "weighted bootstrap".
cols_to_weigh	Array of column indices; must be same length as mu and sigma. Used for method = "weighted bootstrap".
weights	Custom weight assigned to each row of data when resampling. Used for method = "weighted bootstrap".

Details

hypervolume_resample creates a directory called Objects in the current working directory if a directory of that name doesn't already exist. Returns an absolute path to directory with resampled hypervolumes. rds files are stored in different file structures depending on which method is called.

Use to_hv_list to extract every hypervolume object in a directory into a HypervolumeList object. It is also possible to access the hypervolumes by using readRDS to read the hypervolume objects in one by one.

The resampled hypervolumes are generated using the same parameters used to generate the input hypervolume. The only exception is that the bandwidth is re-estimated if method = "gaussian" or method = "box". See copy_param_hypervolume for more details.

Value

returns a string containing an absolute path equivalent to ./Objects/<name>

See Also

to_hv_list, hypervolume_overlap_test, hypervolume_funnel, hypervolume_overlap_confidence

Examples

```
## Not run:
library(palmerpenguins)
data(penguins)
bill_data = na.omit(penguins[,3:4])
hv = hypervolume(bill_data)

# Example 1: Get 50 resampled hypervolumes
# Use detectCores to see how many cores are availible in current environment
# Set cores = 1 to run sequentially (default)
path = hypervolume_resample("example_bootstrap",
                            hv,
                            method = "bootstrap",
```

```
                                n = 50,
                                cores = 12)
hvs = to_hv_list(path)

# Example 2: Get resample with applied weights
# Get maximum bill length
max_bill = max(bill_data$bill_length_mm)
# Make data with larger bill length slightly more likley to be resampled
weighted_path = hypervolume_resample("weighted test",
                                     hv,
                                     method = "weighted bootstrap",
                                     n = 50,
                                     cores = 12,
                                     mu = max_bill,
                                     sigma = 90,
                                     cols_to_weigh = "bill_length_mm")
hvs_weighted = to_hv_list(weighted_path)

## End(Not run)
```

`hypervolume_save_animated_gif` *Saves animated GIF of three-dimensional hypervolume plot.*

Description

Rotates the plot around an axis at a given speed and saves results as a series of GIFs. Requires that the `rgl` library is installed. Assumes there is an open RGL plot (e.g. from calling `plot(hv, show.3d=TRUE)`). If the `magick` package is available, combines these GIFs into a single animation.

Usage

```
hypervolume_save_animated_gif(image.size = 400,
                              axis = c(0, 0, 1), rpm = 4, duration = 15, fps = 10,
                                file.name = "movie", directory.output = ".", ...)
```

Arguments

`image.size`	Number of pixels on each side of the animated image.
`axis`	A three-element vector describing the rotation axis.
`rpm`	Animation speed in rotations per minute.
`duration`	Animation duration in seconds.
`fps`	Animation speed in frames per second.
`file.name`	A base name (no extension) for the GIFs.
`directory.output`	The folder in which output should be located.
`...`	Other arguments to be passed to `rgl::movie3d`.

Value

None; used for the side-effect of producing files.

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                    c("bill_length_mm","bill_depth_mm","flipper_length_mm")]

hv = hypervolume_gaussian(penguins_adelie,name='Adelie')

if(interactive())
{
  plot(hv, show.3d=TRUE)
  hypervolume_save_animated_gif()
  rgl.close()
}

## End(Not run)
```

---

hypervolume_segment *Segments a hypervolume into multiple separate hypervolumes.*

Description

Performs hierarchical clustering (using the 'single' method described in `fastcluster::hclust`) on the input hypervolume to determine which sets of points are closest to others, then cuts the resulting tree at a height equal to the characteristic distance between points multiplied by a distance factor. Random points in the input hypervolume corresponding to each distinct cluster are assigned to distinct output hypervolumes.

Because clustering algorithms scale quadratically with the number of input points, this algorithm can run slowly. Therefore by default, the function can thin the input hypervolume to a reduced number of random points before analysis. This causes some loss of resolution but improves runtimes.

Usage

```
hypervolume_segment(hv, distance.factor = 1, num.points.max = NULL,
                    verbose = TRUE, check.memory = TRUE)
```

Arguments

`hv`	An input `Hypervolume` class object.
`distance.factor`	A numeric value characterizing the distance multiplication factor. Larger values result in fewer distinct output hypervolumes; smaller values result in more.

num.points.max	A numeric value describing the maximum number of random points to be retained in the input; passed to hypervolume_thin before analysis. Set to NULL to disable thinning.
verbose	Logical value; print diagnostic output if TRUE.
check.memory	Logical value; returns information about expected memory usage if true.

Value

A HypervolumeList object.

See Also

hypervolume_thin

Examples

```
# low sample sizes to meet CRAN time requirements
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                    c("bill_length_mm","bill_depth_mm","flipper_length_mm")]

# intentionally make a holey shape for segmentation example
hv = hypervolume_gaussian(penguins_adelie,name='Adelie',
  kde.bandwidth=estimate_bandwidth(penguins_adelie)/3)

hv_segmented <- hypervolume_segment(hv,
                          num.points.max=200, distance.factor=1.25,
                          check.memory=FALSE) # intentionally under-segment
plot(hv_segmented,show.contour=FALSE)
```

---

hypervolume_set *Set operations (intersection / union / unique components)*

Description

Computes the intersection, union, and unique components of two hypervolumes.

Usage

```
hypervolume_set(hv1, hv2, num.points.max = NULL,
  verbose = TRUE, check.memory = TRUE, distance.factor = 1)
```

Arguments

hv1	A n-dimensional hypervolume
hv2	A n-dimensional hypervolume
num.points.max	Maximum number of random points to use for set operations. If NULL defaults to 10^(3+sqrt(n)) where n is the dimensionality of the input hypervolumes. Note that this default parameter value has been increased by a factor of 10 since the 1.2 release of this package.
verbose	Logical value; print diagnostic output if true.
check.memory	Logical value; returns information about expected memory usage if true.
distance.factor	Numeric value; multiplicative factor applied to the critical distance for all inclusion tests (see below). Recommended to not change this parameter.

Details

Uses the inclusion test approach to identify points in the first hypervolume that are or are not within the second hypervolume and vice-versa, based on determining whether each random point in each hypervolume is within a critical distance of at least one random point in the other hypervolume.

The intersection is the points in both hypervolumes, the union those in either hypervolume, and the unique components the points in one hypervolume but not the other.

If you have more than two hypervolumes and wish to calculate only an intersection, consider instead using hypervolume_set_n_intersection rather than iteratively applying this function.

By default, the function uses check.memory=TRUE which will provide an estimate of the computational cost of the set operations. The function should then be re-run with check_memory=FALSE if the cost is acceptable. This algorithm's memory and time cost scale quadratically with the number of input points, so large datasets can have disproportionately high costs. This error-checking is intended to prevent the user from large accidental memory allocation.

The computation is actually performed on a random sample from both input hypervolumes, constraining each to have the same point density given by the minimum of the point density of each input hypervolume, and the point density calculated using the volumes of each input hypervolume divided by num.points.max.

Because this algorithm is based on distances calculated between the distributions of random points, the critical distance (point density ^ (-1/n)) can be scaled by a user-specified factor to provide more or less liberal estimates (distance_factor greater than or less than 1).

Value

If check_memory is false, returns a HypervolumeList object, with six items in its HVList slot:

HV1	The input hypervolume hv1
HV2	The input hypervolume hv2
Intersection	The intersection of hv1 and hv2
Union	The union of hv1 and hv2
Unique_1	The unique component of hv1 relative to hv2

Unique_2 The unique component of hv2 relative to hv1

Note that the output hypervolumes will have lower random point densities than the input hypervolumes.

You may find it useful to define a Jaccard-type fractional overlap between hv1 and hv2 as `hv_set@HVList$Intersection@Vo / hv_set@HVList$Union@Volume`.

If `check_memory=TRUE`, instead returns a scalar with the expected number of pairwise comparisons.

If one of the input hypervolumes has no random points, returns `NA` with a warning.

See Also

hypervolume_set_n_intersection

Examples

```
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                    c("bill_length_mm","bill_depth_mm","flipper_length_mm")]
penguins_chinstrap = penguins_no_na[penguins_no_na$species=="Chinstrap",
                      c("bill_length_mm","bill_depth_mm","flipper_length_mm")]

hv1 = hypervolume_box(penguins_adelie,name='Adelie')
hv2 = hypervolume_box(penguins_chinstrap,name='Chinstrap')

hv_set <- hypervolume_set(hv1, hv2, check.memory=FALSE)

hypervolume_overlap_statistics(hv_set)
# examine volumes of each set component
get_volume(hv_set)
```

---

hypervolume_set_n_intersection

Multi-way set intersection

Description

Intersection of n hypervolumes.

Usage

```
hypervolume_set_n_intersection(hv_list, num.points.max = NULL,
  verbose = TRUE, distance.factor = 1, check.hyperplane = FALSE)
```

Arguments

hv_list	A list of hypervolumes (HypervolumeList)
num.points.max	Maximum number of random points to use for the calculation of the intersection. If NULL defaults to 10^(3+sqrt(n)) where n is the dimensionality of the input hypervolumes. Note that this default parameter value has been increased by a factor of 10 since the 1.2 release of this package.
verbose	Logical value; print diagnostic output if true.
distance.factor	Numeric value; multiplicative factor applied to the critical distance for all inclusion tests (see below). Recommended to not change this parameter.
check.hyperplane	Checks whether data in the input hypervolumes forms a hyperplane (if so, the algorithm is not able to accurately calculate an intersection)

Details

Finds the intersection of multiple hypervolumes. Using this function is likely faster and more accurate than iteratively applying hypervolume_set to hypervolume pairs, as this function does not iteratively perform downsampling.

Stores all the points from the input hypervolumes in a single set. Then uses the inclusion test approach to identify and store points from this set that are within each individual resampled hypervolume, successively. All the points that are common to all the tests are grouped, resampled and used to generate the hypervolume corresponding to the intersection.

The computation is actually performed on a random sample from input hypervolumes, constraining each to have the same point density given by the minimum of the point density of each input hypervolume, and the point density calculated using the volumes of each input hypervolume divided by num.points.max. Because this algorithm is based on distances calculated between the distributions of random points, the critical distance (point density ^ (-1/n)) can be scaled by a user-specified factor to provide more or less liberal estimates (distance_factor greater than or less than 1).

Value

result	The intersection of the input hypervolumes, as a unique hypervolume

.

Note that the output hypervolumes will have lower random point densities than the input hypervolumes.

If one of the input hypervolumes has no random points, returns NA with a warning.

See Also

hypervolume_set

Examples

```
## Not run:
data(iris)
```

```
hv1 = hypervolume_gaussian(subset(iris, Species=="setosa")[,1:3],
name='setosa')
hv2 = hypervolume_gaussian(subset(iris, Species=="virginica")[,1:3],
name='virginica')
hv3 = hypervolume_gaussian(subset(iris, Species=="versicolor")[,1:3],
name='versicolor')

hv_list = hypervolume_join(hv1,hv2,hv3)
intersection = hv_set_n_intersection(hv_list)


## End(Not run)
```

`hypervolume_set_n_union` *Multi-way set union*

Description

Union of n hypervolumes.

Code by Clement Violet.

Usage

```
hypervolume_set_n_union(hv_list, num.points.max = NULL,
  verbose = TRUE, distance.factor = 1, check.hyperplane = FALSE)
```

Arguments

`hv_list`	A list of hypervolumes (HypervolumeList)
`num.points.max`	Maximum number of random points to use for the calculation of the union. If `NULL` defaults to 10^(3+sqrt(n)) where n is the dimensionality of the input hypervolumes. Note that this default parameter value has been increased by a factor of 10 since the 1.2 release of this package.
`verbose`	Logical value; print diagnostic output if true.
`distance.factor`	Numeric value; multiplicative factor applied to the critical distance for all inclusion tests (see below). Recommended to not change this parameter.
`check.hyperplane`	Checks whether data in the input hypervolumes forms a hyperplane (if so, the algorithm is not able to accurately calculate an union)

Details

Finds the union of multiple hypervolumes. Using this function is likely faster and more accurate than iteratively applying `hypervolume_set` to hypervolume pairs, as this function does not iteratively perform downsampling.

Stores all the points from the input hypervolumes in a single set. Then uses the inclusion test approach to identify and store points from this set that are within each individual resampled hypervolume, successively. All the points that are common to all the tests are grouped, resampled and used to generate the hypervolume corresponding to the union.

The computation is actually performed on a random sample from input hypervolumes, constraining each to have the same point density given by the minimum of the point density of each input hypervolume, and the point density calculated using the volumes of each input hypervolume divided by `num.points.max`. Because this algorithm is based on distances calculated between the distributions of random points, the critical distance (point density ^ (-1/n)) can be scaled by a user-specified factor to provide more or less liberal estimates (`distance_factor` greater than or less than 1).

Value

`result` The union of the input hypervolumes, as a unique hypervolume

.

Note that the output hypervolumes will have lower random point densities than the input hypervolumes.

If one of the input hypervolumes has no random points, returns `NA` with a warning.

See Also

`hypervolume_set`, `hypervolume_set_n_intersection`, `hypervolume_n_occupancy`

Examples

```
## Not run:
data(iris)
hv1 = hypervolume_gaussian(subset(iris, Species=="setosa")[,1:3],
name='setosa')
hv2 = hypervolume_gaussian(subset(iris, Species=="virginica")[,1:3],
name='virginica')
hv3 = hypervolume_gaussian(subset(iris, Species=="versicolor")[,1:3],
name='versicolor')

hv_list = hypervolume_join(hv1,hv2,hv3)
union = hv_set_n_union(hv_list)


## End(Not run)
```

hypervolume_svm *Hypervolume construction via one-class support vector machine (SVM) learning model*

Description

Constructs a hypervolume by building a one-class support vector machine that classifies data points as 'in' and other locations as 'out'. This is accomplished by 1) transforming the input data into a high-dimensional nonlinear space in which the data points can be optimally separated from background by a single hyperplane, 2) back-transforming the hyperplane into the original space, 3) delineating an adaptive grid of random points near the original data points, and 4) using the SVM to predict if each of these points is in or out.

Usage

```
hypervolume_svm(data, name = NULL,
            samples.per.point = ceiling((10^(3 + sqrt(ncol(data))))/nrow(data)),
                  svm.nu = 0.01, svm.gamma = 0.5,
                  scale.factor = 1,
                  chunk.size = 1000, verbose = TRUE)
```

Arguments

data	A m x n matrix or data frame, where m is the number of observations and n is the dimensionality.
name	A string to assign to the hypervolume for later output and plotting. Defaults to the name of the variable if NULL.
samples.per.point	Number of random points to be evaluated per data point in data.
svm.nu	A SVM parameter determining an upper bound on the fraction of training errors and a lower bound of the fraction of support vectors. Lower values result in tighter wrapping of the shape to the data (see section 2.2. of https://www.csie.ntu.edu.tw/~cjlin/papers/libs
svm.gamma	A SVM parameter defining the inverse radius of influence of a single point. Low values yield large influences (smooth less complex wraps around the data) and high values yield small influences (tighter but potentially noiser wraps around the data) (see http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html).
scale.factor	A multiplicative factor used to determine the boundaries of the hyperelliptical sampling region. Larger values yield larger boundaries and can prevent clipping. Should not need to be changed in almost any situation.
chunk.size	Number of random points to process per internal step. Larger values may have better performance on machines with large amounts of free memory. Changing this parameter does not change the output of the function; only how this output is internally assembled.
verbose	Logical value; print diagnostic output if TRUE.

Value

A Hypervolume-class object corresponding to the inferred hypervolume.

See Also

hypervolume_threshold

Examples

```
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                    c("bill_length_mm","bill_depth_mm","flipper_length_mm")]

hv = hypervolume_svm(penguins_adelie,name='Adelie')
summary(hv)
```

---

hypervolume_thin *Reduces the number of random points in a hypervolume*

Description

Many hypervolume algorithms have computational complexities that scale with the number of random points used to characterize a hypervolume (@RandomPoints). This value can be reduced to improve runtimes at the cost of lower resolution.

Usage

```
hypervolume_thin(hv, factor = NULL, num.points = NULL)
```

Arguments

hv	An object of class Hypervolume
factor	A number in (0,1) describing the fraction of random points to keep.
num.points	A number describing the number random points to keep.

Details

Either factor or npoints (but not both) must be specified.

Value

A Hypervolume object

Examples

```
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                    c("bill_length_mm","bill_depth_mm","flipper_length_mm")]

hv = hypervolume_box(penguins_adelie,name='Adelie')

# downsample to 1000 random points
hv_thinned = hypervolume_thin(hv, num.points=1000)
hv_thinned
```

---

hypervolume_threshold *Thresholds hypervolume and calculates volume quantile statistics (empirical cumulative distribution function)*

Description

Thresholds a hypervolume at a given value that can correspond to a quantile of the hypervolume. All random points below the threshold value are removed and the volume is adjusted accordingly. Provides threshold-quantile plots if multiple thresholds are specified (as by default).

Quantiles can be specified to be either of the total volume enclosed by the hypervolume p(proportional to `nrow(hv@RandomPoints)`), or of the total probability density (proportional to `sum(hv@ValueAtRandomPoints)`).

Usage

```
hypervolume_threshold(hv,
                      thresholds = NULL,
                      num.thresholds = 20,
                      quantile.requested = NULL,
                      quantile.requested.type = "volume",
                      uniform.density = TRUE,
                      plot = TRUE, verbose = TRUE)
```

Arguments

`hv`	An input hypervolume
`thresholds`	A sequence of probability threshold values. If `NULL`, defaults to a sequence of length `num.thresholds` spanning the minimum and maximum probability values in the hypervolume.
`num.thresholds`	The number of threshold values to use if `thresholds=NULL`. Otherwise ignored.
`quantile.requested`	If not `NULL`, selects a single hypervolume correspondong to the threshold value that comes closest to enclosing the requested quantile fraction of the type `quantile.requested.type`. Using high values of `num.thresholds` enables more accurate threshold and quantile selection.

quantile.requested.type
: Determines the quantile type: either "volume" or "probability".

uniform.density
: Logical value. If TRUE, sets all @ValueAtRandomPoints values to 1 in order to represent thresholded hypervolume as a solid geometrical shape.

plot
: Plots a threshold-quantile plot if TRUE. Quantiles are shown for both volume and probability density. This plot is similar to an empirical cumulative distribution function.

verbose
: If TRUE, prints diagnostic progress messages.

Details

Hypervolumes constructed using the hypervolume_box method may not always yield quantiles close to the requested value because of the flat shape of the kernel.

Value

A list containing two elements: a HypervolumeList or Hypervolume object corresponding to the hypervolumes at each threshold value, and a dataframe Statistics corresponding to the relevant quantiles and thresholds.

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                    c("bill_length_mm","bill_depth_mm","flipper_length_mm")]

hv = hypervolume_box(penguins_adelie,name='Adelie')

# get hypervolumes at multiple thresholds
hvlist = hypervolume_threshold(hv, plot=TRUE)
head(hvlist$Statistics)
plot(hvlist$HypervolumesThresholded[[c(1,5,10,15,20)]],
  show.random=TRUE, show.data=FALSE,show.centroid=FALSE)

# get hypervolume for a single low quantile value
plot(hypervolume_threshold(hv, plot=FALSE, verbose=FALSE,
  quantile.requested=0.1,quantile.requested.type="volume")[[1]])

## End(Not run)
```

---

hypervolume_to_data_frame

Convert hypervolumes to data.frame

Description

Convert objects of class Hypervolume or HypervolumeList to a data.frame.

Usage

```
hypervolume_to_data_frame(hv, remove_zeroes = TRUE)
```

Arguments

hv	A Hypervolume or HypervolumeList.
remove_zeroes	Remove zeroes from ValuesAtRandomPoints. See Details for further information. It works for hypervolume_n_occupancy(), hypervolume_n_occupancy_test(), occupancy_to_union(), occupancy_to_intersection() and occupancy_to_unshared(), otherwise ignored.

Details

Zero values are generated during the occupancy routine when a random point is included in some groups of hypervolumes but not in others. A tipical usage of hypervolume_to_data_frame() with objects generated with the occupancy routine should remove zeroes.

Value

A data.frame.

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))

# split the dataset on species and sex
penguins_no_na_split = split(penguins_no_na,
                        paste(penguins_no_na$species, penguins_no_na$sex, sep = "_"))


# calculate the hypervolume for each element of the splitted dataset
hv_list = mapply(function(x, y)
  hypervolume_gaussian(x[, c("bill_length_mm", "flipper_length_mm")],
                       samples.per.point=100, name = y),
  x = penguins_no_na_split,
  y = names(penguins_no_na_split))

hv_list <- hypervolume_join(hv_list)

# get the data.frame
hypervolume_to_data_frame(hv_list)

## End(Not run)
```

hypervolume_variable_importance
Hypervolume variable importance

Description

Assesses the contribution of each variable to the total hypervolume as a rough metric of variable importance.

Usage

```
hypervolume_variable_importance(hv, verbose = TRUE)
```

Arguments

hv	A hypervolume for which the importance of each variable should be calculated.
verbose	If TRUE, prints diagnostic progress messages.

Details

The algorithm proceeds by comparing the n-dimensional input hypervolume's volume to all possible n-1 dimensional hypervolumes where each variable of interest has been deleted. The importance score reported is the ratio of the n-dimensional hypervolume relative to each of the n-1 dimensional hypervolumes. Larger values indicate that a variable makes a proportionally higher contribution to the overall volume.

The algorithm can only be used on Hypervolumes that have a Data and Method value, because the variable deletion process is not well defined for objects that are not associated with a particular set of observations and construction method.

Value

A named vector with importance scores for each axis. Note that these scores are not dimensionless but rather have units corresponding to the original units of each variable.

Examples

```
# low parameter values for speed
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                    c("bill_length_mm","bill_depth_mm","flipper_length_mm")]

hv = hypervolume_box(penguins_adelie,name='Adelie')

varimp = hypervolume_variable_importance(hv,verbose=FALSE)
barplot(varimp,ylab='Importance',xlab='Variable')
```

HypervolumeList-class *Class* "HypervolumeList"

Description

A class used for storing more than one hypervolume.

Objects from the Class

Objects can be created by calls of the form `new("HypervolumeList", ...)`.

Slots

HVList: Object of class `"list"` containing multiple hypervolumes

morphSnodgrassHeller *Morphological data for Darwin's finches*

Description

Data for nine morphological traits for species of Darwin's finches occurring on the Galapagos Islands.

Note that the underlying morphological dataset has been augmented and improved since version 1.3.1 to include more species and islands. Results are not comparable to version 1.3.0 and below. To duplicate results in the Blonder et al. (2014) paper please install an older version of the package.

Usage

```
data("morphSnodgrassHeller")
```

Format

A data frame with 549 observations on the following 20 variables.

`Source` a factor with levels `Snodgrass` & `Heller (1904)`

`IslandID` a factor with levels `Balt_SS` `Drwn_Clp` `Esp_Hd` `Flor_Chrl` `Frn_Nrb` `Gnov_Twr` `Isa_Alb` `Mrch_Bndl` `Pnt_Abng` `Pnz_Dnc` `SCris_Chat` `SCru_Inde` `SFe_Brngt` `Snti_Jams` `Wlf_Wnm`

`TaxonOrig` a factor with levels `Certhidea cinerascens bifasciata` `Certhidea cinerascens cinerascens` `Certhidea olivacea becki` `Certhidea olivacea fusca` `Certhidea olivacea luteola` `Certhidea olivacea mentalis` `Certhidea olivacea olivacea` `Geospiza affinis` `Geospiza conirostris conirostris` `Geospiza conirostris propinqua` `Geospiza crassirostris` `Geospiza fortis dubia` `Geospiza fortis fortis` `Geospiza fortis fratercula` `Geospiza fortis platyrhyncha` `Geospiza fuliginosa acutirostris` `Geospiza fuliginosa difficilis` `Geospiza fuliginosa fuliginosa` `Geospiza fuliginosa minor` `Geospiza fuliginosa parvula` `Geospiza habeli` `Geospiza heliobates` `Geospiza paupera` `Geospiza prosthemelas prosthemelas`

Geospiza prosthemelas salvini Geospiza psittacula psittacula Geospiza scandens abingdoni Geospiza scandens fatigata Geospiza scandens rothschildi Geospiza scandens scandens Geospiza septentrionalis Geospiza strenua

GenusL69 a factor with levels Camarhynchus Certhidea Geospiza Platyspiza

SpeciesL69 a factor with levels conirostris crassirostris difficilis fortis fuliginosa heliobates magnirostris olivacea parvulus pauper psittacula scandens

SubspL69 a factor with levels abingdoni affinis becki bifasciatus cinerascens conirostris darwini fusca habeli intermedia luteola mentalis olivacea parvulus propinqua psittacula rothschildi salvini scandens septentrionalis strenua

SpeciesID a factor with levels Cam.hel Cam.par Cam.pau Cam.psi Cer.oli Geo.con Geo.dif Geo.for Geo.ful Geo.mag Geo.sca Pla.cra

SubspID a factor with levels Cam.hel Cam.par.par Cam.par.sal Cam.pau Cam.psi.aff Cam.psi.hab Cam.psi.psi Cer.oli.bec Cer.oli.bif Cer.oli.cin Cer.oli.fus Cer.oli.lut Cer.oli.men Cer.oli.oli Geo.con.con Geo.con.dar Geo.con.pro Geo.dif.sep Geo.for Geo.ful Geo.mag.str Geo.sca.abi Geo.sca.int Geo.sca.rot Geo.sca.sca Pla.cra

Sex a factor with levels F M

Plumage a logical vector

BodyL a numeric vector

WingL a numeric vector

TailL a numeric vector

BeakW a numeric vector

BeakH a numeric vector

LBeakL a numeric vector

UBeakL a numeric vector

N.UBkL a factor with levels 10 10.3 10.5 10.7 11 11.3 11.5 11.7 12 12.3 12.5 12.7 13 13.3 13.5 13.7 14 14.3 14.5 14.7 15 15.3 15.5 15.7 16 16.3 16.5 16.7 17 17.5 6.5 6.7 7 7.3 7.5 7.7 8 8..3 8.3 8.5 8.7 9 9.3 9.5 9.7

TarsusL a numeric vector

MToeL a logical vector

Source

Snodgrass RE and Heller E (1904) Papers from the Hopkins-Stanford Galapagos Expedition, 1898-99. XVI. Birds. Proceedings of the Washington Academy of Sciences 5: 231-372.

Downloaded from http://datadryad.org/resource/doi:10.5061/dryad.152

Examples

```
data(morphSnodgrassHeller)
finch_isabela <- morphSnodgrassHeller[morphSnodgrassHeller$IslandID=="Isa_Alb",]
```

occupancy_bootstrap_gof *Goodness of fit metrics for bootstrapped occupancy objects*

Description

The occupancy_bootstrap_gof() function calculates goodness of fit metrics for objects generated with hypervolume_n_occupancy_bootstrap().

Usage

```
occupancy_bootstrap_gof(path, FUN)
```

Arguments

path	A path to a directory of bootstrapped hypervolumes generated with hypervolume_n_occupancy_bootstr
FUN	Function to calculate the goodness of fit. It can be mae for the mean absolute error, rmse for the root mean square error or a function provided by the user.

Details

Goodness of fit metrics are calculated on the difference between input and recomputed volumes for each bootstrapped element (set with n in hypervolume_n_resample()). See hypervolume_n_occupancy() for details on the meaning of input and recomputed hypervolumes.

Value

A one row data.frame reporting mean, standard deviation, minimum, maximum, median, 2.5%, 25%, 75% ans 97.5% quantiles.

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))

# split the dataset on species and sex
penguins_no_na_split = split(penguins_no_na,
paste(penguins_no_na$species, penguins_no_na$sex, sep = "_"))

# calculate the hypervolume for each element of the splitted dataset
hv_list = mapply(function(x, y)
  hypervolume_gaussian(x[, c("bill_length_mm","bill_depth_mm","flipper_length_mm")],
                       samples.per.point=100, name = y),
                       x = penguins_no_na_split,
                       y = names(penguins_no_na_split))
```

```
# transform the list into an HypervolumeList
hv_list = hypervolume_join(hv_list)

# bootstrap hypervolumes based on sex
hv_resample = hypervolume_n_resample(hv_list, name = "boot_example")

# calculate occupancy for each bootstrap
hv_occupancy_bootstrap = hypervolume_n_occupancy_bootstrap(hv_resample,
                                    name = "occupancy_example",
                                    classification = rep(c("female", "male"), 3))


occupancy_bootstrap_gof(hv_occupancy_bootstrap, FUN = "rmse")


## End(Not run)
```

`occupancy_filter` *Subset occupancy hypervolumes*

Description

The `occupancy_filter()` function is used to subset an hypervolume generated with `hypervolume_n_occupancy()` or `hypervolume_n_occupancy_test()`.

Usage

```
occupancy_filter(hv, operator = NULL, filter = NULL, tol = 1e-10)
```

Arguments

`hv`	A `Hypervolume` or `HypervolumeList` object generated with `hypervolume_n_occupancy()` or `hypervolume_n_occupancy_test()`.
`operator`	Binary operator which allow the comparison.
`filter`	Threshold value to perform the operation.
`tol`	Set the tolerance for reconstructing whole volume. See details.

Details

The `occupancy_filter()` function set the occupancy values to 0 based on the user-provided operation. Volume of the hypervolumes are changed accordingly.
When `hv` is an `HypervolumeList`, the `occupancy_filter()` function attempts to reconstruct the volume of the union of hypervolumes from `hv_list`. At first, the volume of the union of hypervolumes is calculated for each element of `hv` as the the ratio between the total number of random points and the number of random points of the ith element of `hv`, multiplied by the volume of the ith element `hv`. This step results in a number of reconstructed volumes equal to the number of hypervolumes in the jth bootstrapped occupancy_object. Reconstructed volumes are then compared among each other to ensure the consistency of the reconstruction. To do this, the distance among reconstructed volumes is calculated with the `dist()` function of the `stats` package. If at least one of the distances is greater than `tol` the computation is stopped and some suggestions are returned.

Value

A Hypervolume-class or HypervolumeList-class object.

See Also

hypervolume_n_occupancy, hypervolume_n_occupancy_test

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))

# split the dataset on species and sex
penguins_no_na_split = split(penguins_no_na,
paste(penguins_no_na$species, penguins_no_na$sex, sep = "_"))

# calculate the hypervolume for each element of the splitted dataset
hv_list = mapply(function(x, y)
  hypervolume_gaussian(x[, c("bill_length_mm","bill_depth_mm","flipper_length_mm")],
                       samples.per.point=100, name = y),
                       x = penguins_no_na_split,
                       y = names(penguins_no_na_split))


# transform the list into an HypervolumeList
hv_list = hypervolume_join(hv_list)

# calculate occupancy based on sex
hv_occupancy_list_sex = hypervolume_n_occupancy(hv_list,
                          classification = rep(c("female", "male"), 3))


# set to 0 values lower than 0.35
occupancy_filter(hv_occupancy_list_sex, operator = "<", filter = "0.35")

## End(Not run)
```

---

occupancy_to_intersection

Get the intersection of an occupancy object

Description

The occupancy_to_intersection() function is used to get the intersection of hypervolumes of an object generated with the occupancy routine.

Usage

```
occupancy_to_intersection(hv_list, method = "all", m = 2, tol = 1e-10)
```

Arguments

hv_list	A HypervolumeList generated with hypervolume_n_occupancy(), hypervolume_n_occupancy_test(occupancy_to_union(), occupancy_to_unshared() or occupancy_filter().
method	If all compute the intersection among all the hypervolumes in hv_list. If n_wise compute the intersection for each n_wise combination of hypervolumes in hv_list.
m	Number of elements to choose. Default to 2 (pairwise comparisons). This argument is ignored when method is set to all.
tol	Set the tolerance for reconstructing whole volume. See details.

Details

The occupancy_to_intersection() function takes as input a HypervolumeList generated with an occupancy function (check See Also) and returns a Hypervolume or HypervolumeList depending on method. When method = "all" the occupancy_to_intersection() function returns a Hypervolume representing the intersection of all the hypervolumes in hv_list. When method = "n_wise" a HypervolumeList in which each hypervolume represent the intersection of a combination of the hypervolumes in hv_list is returned. The number of hypervolumes for each combination is set with the argument m. Argument m can not be higher than the number of hypervolumes in hv_list and lower than 2.
The occupancy_to_intersection() function attempts to reconstruct the volume of the intersection from the hv_list provided by the user. At first, the volume of the union of hypervolumes is calculated for each hypervolume in hv_list as the the ratio between the total number of random points and the number of random points of the ith hypervolume of hv_list, multiplied by the volume of the ith hypervolume of hv_list. This step results in a number of reconstructed volumes equal to the number of hypervolumes in the jth bootstrapped occupancy_object. Reconstructed volumes are then compared among each other to ensure the consistency of the reconstruction. To do this, the distance among reconstructed volumes is calculated with the dist() function of the stats package. If at least one of the distances is greater than tol the computation is stopped and some suggestions are returned. The volume of the intersection is then calculated as the ratio between the number of random points of the intersection and the total number of random points, multiplied by the volume of the union of hypervolumes.

Value

A Hypervolume-class or HypervolumeList-class object.

See Also

hypervolume_n_occupancy, hypervolume_n_occupancy_test, occupancy_to_union, occupancy_to_unshared, occupancy_filter

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
```

```
# split the dataset on species and sex
penguins_no_na_split = split(penguins_no_na,
                  paste(penguins_no_na$species, penguins_no_na$sex, sep = "_"))

# calculate the hypervolume for each element of the splitted dataset
hv_list = mapply(function(x, y)
  hypervolume_gaussian(x[, c("bill_length_mm", "flipper_length_mm")],
                       samples.per.point=100, name = y),
  x = penguins_no_na_split,
  y = names(penguins_no_na_split))


# transform the list into an HypervolumeList
hv_list = hypervolume_join(hv_list)

# calculate occupancy based on sex
hv_occupancy_list_sex = hypervolume_n_occupancy(hv_list,
                    classification = rep(c("Adelie", "Chinstrap", "Gentoo"), 2))

# get the hypervolume of intersection
hv_occupancy_sex_intersection <- occupancy_to_intersection(hv_occupancy_list_sex)
plot(hv_occupancy_sex_intersection)

# get hypervolumes with the intersection among 3 or 2 combinations of hypervolumes
hv_occ_sex_intersection_3 <- occupancy_to_intersection(hv_occupancy_list_sex,
                                                       method = "n_wise",
                                                       m = 3)
hv_occ_intersection_2 <- occupancy_to_intersection(hv_occupancy_list_sex,
                                                   method = "n_wise",
                                                   m = 2)


## End(Not run)
```

---

occupancy_to_union *Union of hypervolumes from an occupancy object*

Description

The occupancy_to_union() function is used to get the union of hypervolumes of an object generated with hypervolume_n_occupancy().

Usage

```
occupancy_to_union(hv_list, method = "all", m = 2, tol = 1e-10)
```

Arguments

hv_list	A HypervolumeList object generated with hypervolume_n_occupancy(), hypervolume_n_occupancy_ occupancy_to_intersection(), occupancy_to_unshared() or occupancy_filter().

method	If all compute the union of all the hypervolumes in hv_list. If n_wise compute of the union for each n_wise combination of hypervolumes in hv_list.
m	Number of elements to choose. Default to 2 (pairwise comparisons). This argument is ignored when method is set to all.
tol	Set the tolerance for reconstructing whole volume. See details.

Details

The occupancy_to_union() function takes as input a HypervolumeList generated with an occupancy function (check See Also) and returns a Hypervolume or HypervolumeList depending on method. When method = "all" the occupancy_to_union() function returns a Hypervolume representing the union of all the hypervolumes in hv_list. When method = "n_wise" a HypervolumeList in which each hypervolume represent the union of a combination of the hypervolumes in hv_list is returned. The number of hypervolumes for each combination is set with the argument m. Argument m can not be higher than the number of hypervolumes in hv_list and lower than 2.
The occupancy_to_union() function attempts to reconstruct the volume of the union from the hv_list provided by the user. For each hypervolume in hv_list, it calculates the volume of the union as the ratio between the total number of random points and the number of random points of the ith hypervolume of hv_list, multiplied by the volume of the ith hypervolume of hv_list. This step results in a number of reconstructed volumes equal to the number of hypervolumes in the jth bootstrapped occupancy_object. Reconstructed volumes are then compared among each other to ensure the consistency of the reconstruction. To do this, the distance among reconstructed volumes is calculated with the dist() function of the stats package. If at least one of the distances is greater than tol the computation is stopped and some suggestions are returned.

Value

A Hypervolume-class or HypervolumeList-class object.

See Also

hypervolume_n_occupancy, hypervolume_n_occupancy_test, occupancy_to_intersection, occupancy_to_unshared, occupancy_filter

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))

# split the dataset on species and sex
penguins_no_na_split = split(penguins_no_na,
paste(penguins_no_na$species, penguins_no_na$sex, sep = "_"))

# calculate the hypervolume for each element of the splitted dataset
hv_list = mapply(function(x, y)
  hypervolume_gaussian(x[, c("bill_length_mm", "flipper_length_mm")],
                       samples.per.point=100, name = y),
                       x = penguins_no_na_split,
                       y = names(penguins_no_na_split))
```

```
# transform the list into an HypervolumeList
hv_list = hypervolume_join(hv_list)

# calculate occupancy based on sex
hv_occupancy_list_sex = hypervolume_n_occupancy(hv_list,
                          classification = rep(c("female", "male"), 3))

# get the union of all the hypervolumes
hv_occupancy_sex_union <- occupancy_to_union(hv_occupancy_list_sex)
plot(hv_occupancy_sex_union)


## End(Not run)
```

---

occupancy_to_unshared *Unshared fraction from an occupancy object*

Description

The occupancy_to_unshared() function is used to get the unshared fraction of hypervolumes of an object generated with the occupancy routine.

Usage

```
occupancy_to_unshared(hv_list, method = "all", tol = 1e-10)
```

Arguments

hv_list	A HypervolumeList object generated with hypervolume_n_occupancy(), hypervolume_n_occupancy, occupancy_to_union(), occupancy_to_intersection() or occupancy_filter.
method	If all compute the unshared fraction of each hypervolume in hv_list. If pairwise compute the unshared fraction for each pairwise combination of hypervolumes in hv_list.
tol	Set the tolerance for reconstructing whole volume. See details.

Details

Unshared fraction is the fraction of the hypervolume not shared with other hypervolumes. It is calculated from occupancy objects only (check See Also). When method = "all" a HypervolumeList containing the unshared fraction of each hypervolume is returned. When method = "pairwise" an HypervolumeList containing the unshared fraction of the pairwise combination of hypervolumes is returned. Hypervolumes generated when method = "pairwise" include the unshared fraction of both hypervolumes under comparison. The first of the two hypervolumes is assigned with ValueAtRandomPoints equal to 1 while, the second is assigned with ValueAtRandomPoints equal to -1. This is useful when used in combination with occupancy_filter() or hypervolume_to_data_frame().

The occupancy_to_unshared() function attempts to reconstruct the volume of the unshared fraction from the hv_list provided by the user. At first, the volume of the union of hypervolumes is calculated for each hypervolume in hv_list as the the ratio between the total number of random points and the number of random points of the ith hypervolume of hv_list, multiplied by the volume of the ith hypervolume of hv_list. This step results in a number of reconstructed volumes equal to the number of hypervolumes in the jth bootstrapped occupancy_object. Reconstructed volumes are then compared among each other to ensure the consistency of the reconstruction. To do this, the distance among reconstructed volumes is calculated with the dist() function of the stats package. If at least one of the distances is greater than tol the computation is stopped and some suggestions are returned. The volume of the unshared fraction is then calculated as the ratio between the number of random points of the unshared fraction and the total number of random points, multiplied by the volume of the union of hypervolumes.

Value

A Hypervolume-class or HypervolumeList-class object.

See Also

hypervolume_n_occupancy, hypervolume_n_occupancy_test, occupancy_to_intersection, occupancy_to_union, occupancy_filter

Examples

```
## Not run:
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))

# split the dataset on species and sex
penguins_no_na_split = split(penguins_no_na,
paste(penguins_no_na$species, penguins_no_na$sex, sep = "_"))

# calculate the hypervolume for each element of the splitted dataset
hv_list = mapply(function(x, y)
  hypervolume_gaussian(x[, c("bill_length_mm", "flipper_length_mm")],
                       samples.per.point=100, name = y),
                       x = penguins_no_na_split,
                       y = names(penguins_no_na_split))


# transform the list into an HypervolumeList
hv_list = hypervolume_join(hv_list)

# calculate occupancy based on sex
hv_occupancy_list_sex = hypervolume_n_occupancy(hv_list,
                          classification = rep(c("female", "male"), 3))

# get hypervolumes with the unshared fraction
hv_occupancy_sex_unshared <- occupancy_to_unshared(hv_occupancy_list_sex)
plot(hv_occupancy_sex_unshared)
```

```
# get hypervolumes with the unshared fraction between each pairwise combination of hypervolumes
hv_occupancy_sex_unshared_pw <- occupancy_to_unshared(hv_occupancy_list_sex)

# plot the unshared fraction with ggplot2
require(ggplot2)

# extract data to plot
occupancy_sex_pw_df <- hypervolume_to_data_frame(hv_occupancy_sex_unshared_pw)

ggplot(occupancy_sex_pw_df, aes(bill_length_mm, flipper_length_mm, col = Name)) +
  geom_point() +
  theme_bw()

## End(Not run)
```

---

padded_range *Generates axis-wise range limits with padding*

Description

For each data axis, finds the minimum and maximum values. Then pads this range by a multiplicative factor of the range interval, and pads again by an additive amount.

Usage

```
padded_range(data, multiply.interval.amount = 0, add.amount = 0)
```

Arguments

data	A m x n matrix whose range limits should be found.
multiply.interval.amount	A non-negative factor used to multiply the range interval. Can have either dimensionality 1 or n.
add.amount	A non-negative factor used to add to the range limits. Can have either dimensionality 1 or n.

Value

A 2 x n matrix, whose first row is the low value along each axis and whose second row is the high value along each axis.

Examples

```
data(morphSnodgrassHeller)
finch_isabela <- na.omit(morphSnodgrassHeller[morphSnodgrassHeller$IslandID=="Isa_Alb",
      c("WingL","TailL","BeakW","BeakH")])

finch_isabela_rangebox_nopadding = padded_range(finch_isabela)
finch_isabela_rangebox_nopadding
```

```
finch_isabela_rangebox_padding = padded_range(finch_isabela,
  multiply.interval.amount=0.5, add.amount=0.1)
finch_isabela_rangebox_padding
```

`plot.HypervolumeList` *Plot a hypervolume or list of hypervolumes*

Description

Plots a single hypervolume or multiple hypervolumes as either a pairs plot (all axes) or a 3D plot (a subset of axes). The hypervolume is drawn as a uniformly random set of points guaranteed to be in the hypervolume.

Usage

```
## S3 method for class 'HypervolumeList'
plot(x,
   show.3d=FALSE,plot.3d.axes.id=NULL,
   show.axes=TRUE, show.frame=TRUE,
   show.random=TRUE, show.density=TRUE,show.data=TRUE,
   names=NULL, show.legend=TRUE, limits=NULL,
   show.contour=TRUE, contour.lwd=1.5,
    contour.type='kde',
    contour.alphahull.alpha=0.25,
    contour.ball.radius.factor=1,
    contour.kde.level=1e-04,
    contour.raster.resolution=20,
   show.centroid=TRUE, cex.centroid=2,
   colors=rainbow(floor(length(x@HVList)*1.5),alpha=0.8),
   point.alpha.min=0.2, point.dark.factor=0.5,
   cex.random=0.5,cex.data=0.75,cex.axis=0.75,cex.names=1.0,cex.legend=0.75,
   num.points.max.data = 1000, num.points.max.random = 2000, reshuffle=TRUE,
   plot.function.additional=NULL,
   verbose=FALSE,
   ...)
```

Arguments

`x`	A Hypervolume or HypervolumeList object. The objects to be plotted.
`show.3d`	If `TRUE`, makes a three-dimensional plot of a subset of axes determined by `plot.3d.axes.id`; otherwise, a pairs plot of all axes. Requires that the `rgl` library is installed.
`plot.3d.axes.id`	Numeric identities of axes to plot in three dimensions. Defaults to 1:3 if set to `NULL`.
`show.axes`	If `TRUE`, draws axes on the plot.

`show.frame` If `TRUE`, frames the plot with a box.

`show.random` If `TRUE`, shows random points from the hypervolume.

`show.density` If `TRUE`, draws random points with alpha level proportional to their unit-scaled probability density. Note that this has no effect when probability density is not relevant, i.e. for hypervolumes that are the output of set operations.

`show.data` If `TRUE`, draws data points from the hypervolume. Note that this has no effect if the hypervolume is not associated with data points, e.g. for those that are the output of set operations.

`names` A vector of strings in the same order as the input hypervolumes. Used to draw the axes labels.

`show.legend` If `TRUE`, draws a color legend.

`limits` A list of two-element vectors corresponding to the axes limits for each dimension. If a single two-element vector is provided it is re-used for all axes.

`show.contour` If `TRUE`, draws a boundary line saround each two-dimensional projection. Ignored if `show.3d=TRUE`.

`contour.lwd` Line width used for contour lines. Ignored if `show.contour=FALSE`.

`contour.type` Type of contour boundary: any of `"alphahull"` (alpha hull), `"ball"` (experimental ball covering), `"kde"` (2D KDE smoothing), or `"raster"` (grid-based rasterization).

`contour.alphahull.alpha`
Value of the alpha parameter for a `"alphahull"` contour. Can be increased to provide smoother contours.

`contour.ball.radius.factor`
Factor used to multiply radius of ball surrounding each random point for a `"ball"` contour.

`contour.kde.level`
Probability level used to delineate edges for a `"kde"` contour.

`contour.raster.resolution`
Grid resolution for a `"raster"` contour.

`show.centroid` If `TRUE`, draws a colored point indicating the centroid for each hypervolume.

`cex.centroid` Expansion factor for the centroid symbol.

`colors` A vector of colors to be used to plot each hypervolume, in the same order as the input hypervolumes.

`point.alpha.min`
Fractional value corresponding to the most transparent value for plotting random points. 0 corresponds to full transparency.

`point.dark.factor`
Fractional value corresponding to the darkening factor for plotting data points. 0 corresponds to fully black.

`cex.random` cex value for uniformly random points.

`cex.data` cex value for data points.

`cex.axis` cex value for axes, if pair=T.

`cex.names` cex value for variable names printed on the diagonal, if pair=T.

`cex.legend` cex value for the legend text

`num.points.max.data`
An integer indicating the maximum number of data points to be sampled from each hypervolume. Lower values result in faster plotting and smaller file sizes but less accuracy.

`num.points.max.random`
An integer indicating the maximum number of random points to be sampled from each hypervolume. Lower values result in faster plotting and smaller file sizes but less accuracy.

`reshuffle` A logical value relevant when pair=TRUE. If false, each hypervolume is drawn on top of the previous hypervolume; if true, all points of all hypervolumes are randomly shuffled so no hypervolume is given visual preference during plotting.

`plot.function.additional`
Any `function(i,j)` that will add additional plotting commands for column `i` and row `j` of the pairs plot. Should not create new plots or change `par()` settings. Has no effect if `show.3d=TRUE`.

`verbose` If `TRUE`, prints diagnostic information about the number of points being plotted

`...` Additional arguments to be passed to `rgl::plot3d`.

Value

None; used for the side-effect of producing a plot.

Note

Contour line plotting with `alphahull` requires the non-FOSS `alphahull` package to be installed. Please do so in order to use this functionality!

See Also

`hypervolume_save_animated_gif`

Examples

```
## Not run:
# low parameter values for speed
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                    c("bill_length_mm","bill_depth_mm","flipper_length_mm")]

hv = hypervolume_gaussian(penguins_adelie,name='Adelie')

# 2d plot
plot(hv, show.3d=FALSE)

# 3d plot
if(interactive())
{
```

```
  plot(hv, show.3d=TRUE)
}

## End(Not run)
```

`print.Hypervolume` *Print summary of hypervolume*

Description

Summarizes all slots of `Hypervolume-class` object.

Usage

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

Arguments

`x`	The hypervolume to summarize
`...`	Other arguments to be passed to base print function

Value

None; used for the side-effect of printing.

`quercus` *Data and demo for Quercus (oak) tree distributions*

Description

Data for occurrences of Quercus alba and Quercus rubra based on geographic observations. Demonstration analysis of how to use hypervolumes for species distribution modeling using WorldClim data.

Usage

```
data(quercus)
```

Format

A data frame with 3779 observations on the following 3 variables.

`Species` a factor with levels `Quercus alba` `Quercus rubra`

`Latitude` a numeric vector

`Longitude` a numeric vector

Source

Occurrence data come from the BIEN database (`https://biendata.org/`). Climate data are from WorldClim.

References

Blonder, B., Lamanna, C., Violle, C., Enquist, B. The n-dimensional hypervolume. Global Ecology and Biogeography (2014).

Examples

```
demo('quercus', package='hypervolume')
```

---

summary.Hypervolume *Summary of hypervolume*

Description

Prints basic information about Hypervolume or HypervolumeList structure.

Usage

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

Arguments

`object`	The hypervolume to summarize
`...`	Other arguments to be passed to base summary function

Value

None; used for the side-effect of printing.

`to_hv_list` *Read hypervolumes from directory*

Description

Takes a path to a directory containing only rds files and reads them into a HypervolumeList object.

Usage

```
to_hv_list(path)
```

Arguments

`path`	absolute or relative path to directory containing rds files

Details

Use to_hv_list on the output from `hypervolume_resample` when method = "bootstrap" to read bootstrapped hypervolumes into memory.

Value

`HypervolumeList` object

Examples

```
## Not run:
library(palmerpenguins)
data(penguins)
bill_data = na.omit(penguins[,3:4])
hv = hypervolume(bill_data)

# Use detectCores to see how many cores are availible in current environment
path = hypervolume_resample("example_bootstrap", hv, method = "bootstrap", n = 50, cores = 12)
hvs = to_hv_list(path)

## End(Not run)
```

`weight_data` *Abundance weighting and prior of data for hypervolume input*

Description

Resamples input data for hypervolume construction, so that some data points can be weighted more strongly than others in kernel density estimation. Also allows a multidimensional normal prior distribution to be placed on each data point to enable simulation of uncertainty or variation within each observed data point.

Note that this algorithm will change the number of data points and may thus lead to changes in the inferred hypervolume if the selected algorithm (e.g. for bandwidth selection) depends on sample size.

A direct weighting approach (which does not artificially change the sample size, and thus the kernel bandwidth estimate) is available for Gaussian hypervolumes within `hypervolume_gaussian`.

Usage

```
weight_data(data, weights, jitter.sd = matrix(0, nrow = nrow(data), ncol = ncol(data)))
```

Arguments

`data`	A data frame or matrix of unweighted data. Must only contain numeric values.
`weights`	A vector of weights with the same length as the number of rows in `data`. All values must take positive integer values.
`jitter.sd`	A matrix of the same size as `data` corresponding to the standard deviation of a normal distribution with mean equal to that of the observed data. If a vector of length equal to 1 or the number of columns of `data`, is repeated for all observations.

Details

Each data point is jittered a single time. To sample many points from a distribution around each observed data point, multiply all weights by a large number.

Value

A data frame with the rows of `data` repeated by `weights`, potentially with noise added. The output has the same columns as the input but `sum(weights)` total rows.

See Also

`hypervolume_gaussian`

Examples

```
data(penguins,package='palmerpenguins')
penguins_no_na = as.data.frame(na.omit(penguins))
penguins_adelie = penguins_no_na[penguins_no_na$species=="Adelie",
                    c("bill_length_mm","bill_depth_mm","flipper_length_mm")]

weighted_data <- weight_data(penguins_adelie,
  weights=1+rpois(n=nrow(penguins_adelie),lambda=3))
# color points by alpha to show overlaps
pairs(weighted_data,col=rgb(1,0,0,alpha=0.15))

weighted_noisy_data <- weight_data(penguins_adelie,
  weights=1+rpois(n=nrow(penguins_adelie),lambda=3),jitter.sd=0.5)
# color points by alpha to show overlaps
pairs(weighted_noisy_data,col=rgb(1,0,0,alpha=0.15))
```

Index

∗ **classes**
HypervolumeList-class, 75
∗ **datasets**
morphSnodgrassHeller, 75

acacia_pinus, 4

circles, 5
copy_param_hypervolume, 5, *48*, *60*

estimate_bandwidth, 6, *26–28*, *34*
expectation_ball, 8, *26*
expectation_box, 9, *26*
expectation_convex, 9, *26*
expectation_maximal, 11

find_optimal_occupancy_thin, 11, *42*
find_optimal_occupancy_thin(), *41*

get_centroid, 13
get_centroid_weighted, 14
get_occupancy_intersection_bootstrap, 15
get_occupancy_stats, 17
get_occupancy_stats_bootstrap *(*get_occupancy_stats*)*, 17
get_occupancy_unshared_bootstrap, 19
get_occupancy_volume_bootstrap, 21
get_relative_volume, 23
get_volume, 25
get_volume,Hypervolume-method *(*get_volume*)*, 25
get_volume,HypervolumeList-method *(*get_volume*)*, 25
get_volume.Hypervolume *(*get_volume*)*, 25
get_volume.HypervolumeList *(*get_volume*)*, 25

hypervolume, 25
Hypervolume-class, 26
hypervolume-package, 3
hypervolume_box, *26*, 27
hypervolume_distance, 28
hypervolume_distance_point, 29
hypervolume_estimate_probability, 31, *39*, *56*, *58*
hypervolume_funnel, 32, *60*
hypervolume_gaussian, *26*, 33, *92*
hypervolume_general_model, 35
hypervolume_holes, 36, *58*
hypervolume_inclusion_test, *32*, 38, *39*, *56*
hypervolume_join, 39
hypervolume_n_occupancy, *12*, *14*, *16*, *18*, *21*, *23*, *24*, 40, *45*, *68*, *79*, *80*, *82*, *84*
hypervolume_n_occupancy(), *12*, *46*, *77*
hypervolume_n_occupancy_bootstrap, *16*, *18*, *21*, *23*, *48*
hypervolume_n_occupancy_bootstrap *(*hypervolume_n_occupancy*)*, 40
hypervolume_n_occupancy_permute, *18*, *24*, 43
hypervolume_n_occupancy_permute(), *46*
hypervolume_n_occupancy_test, *14*, *18*, *24*, 45, *79*, *80*, *82*, *84*
hypervolume_n_resample, 47
hypervolume_overlap_confidence, 49, *60*
hypervolume_overlap_statistics, *49*, 50, *52*
hypervolume_overlap_test, 51, *54*, *60*
hypervolume_permute, *52*, 54
hypervolume_project, 55
hypervolume_prune, 57
hypervolume_redundancy, *32*, 58
hypervolume_resample, *49*, *52*, 59
hypervolume_save_animated_gif, 61, *88*
hypervolume_segment, *58*, 62
hypervolume_set, *50*, 63, *66*, *68*
hypervolume_set_n_intersection, *64*, 65, *65*, *68*

hypervolume_set_n_union, 67
hypervolume_svm, *26*, 69
hypervolume_thin, *63*, 70
hypervolume_threshold, *26*, *28*, *34, 35*, *70*, 71
hypervolume_to_data_frame, 72
hypervolume_variable_importance, 74
HypervolumeList-class, 75

morphSnodgrassHeller, 75

occupancy_bootstrap_gof, *42*, 77
occupancy_filter, *24*, 78, *80*, *82*, *84*
occupancy_to_intersection, *16*, *24*, 79, *82*, *84*
occupancy_to_intersection(), *16*
occupancy_to_union, *24*, *80*, 81, *84*
occupancy_to_unshared, *21*, *24*, *80*, *82*, 83
occupancy_to_unshared(), *20*

padded_range, *36*, 85
plot.Hypervolume *(*plot.HypervolumeList*)*, 86
plot.HypervolumeList, 86
print.Hypervolume, 89
print.HypervolumeList *(*print.Hypervolume*)*, 89

quercus, 89

show.Hypervolume *(*summary.Hypervolume*)*, 90
show.HypervolumeList *(*summary.Hypervolume*)*, 90
summary.Hypervolume, 90
summary.HypervolumeList *(*summary.Hypervolume*)*, 90

to_hv_list, *60*, 91

weight_data, *26*, 92