# Package 'hypervolume'

December 6, 2019

**Type** Package

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

**Version** 2.0.12

**Date** 2019-12-05

**Author** Benjamin Blonder, with contributions from David J. Harris

**Maintainer** Benjamin Blonder <bblonder@gmail.com>

**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, rgl, methods, R (>= 3.0.2)

**LinkingTo** Rcpp, RcppArmadillo, progress

**Imports** raster, maps, MASS, geometry, ks, pdist, fastcluster, compiler, e1071, hitandrun, progress, mvtnorm, data.table, rgeos, sp

**Suggests** magick, alphahull

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2019-12-06 06:10:12 UTC

## R topics documented:

---

`hypervolume-package` *High Dimensional Geometry and Set Operations 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, with contributions from David J. Harris

Maintainer: Benjamin Blonder <bblonder@gmail.com>

## 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

---

# estimate_bandwidth *Kernel bandwidth estimators for hypervolumes*

---

## Description

Estimates bandwidth vector from data using multiple approaches.

## Usage

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

## 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 `"silverman"`, `"plug-in"`, or `"cross-validation"` - see 'details' section. |

## Details

The Silverman (`"silverman"`) 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 along each axis independently. This is the default option due ONLY to computational simplicity.

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.

## References

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

## Examples

```
data(iris)
print(estimate_bandwidth(iris[,1:2],method="silverman"))
print(estimate_bandwidth(iris[,1:2],method="plug-in"))
print(estimate_bandwidth(iris[,1:2],method="cross-validation"))
```

---

# 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(iris)
e_ball <- expectation_ball(iris[,1:3])
```

---

## 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(iris)
e_box <- expectation_box(iris[,1:3])
```

---

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(iris)
e_convex <- expectation_convex(iris[,1:3], 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.

## 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(iris)
hv = hypervolume_gaussian(iris[,1:2])
get_centroid(hv)

## 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(iris)
hv = hypervolume(data=subset(iris, Species=="setosa")[,1:2],method='box')
summary(hv)
```

## 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`

## 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

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(iris)
hv = hypervolume_box(data=subset(iris, Species=="setosa")[,1:2],name='setosa')
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(iris)
hv1 = hypervolume_gaussian(subset(iris, Species=="setosa")[,1:3])
hv2 = hypervolume_gaussian(subset(iris, Species=="virginica")[,1:3])

# 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_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,
                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.

verbose
: If TRUE, prints diagnostic progress messages.

## 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(iris)
iris_ss = subset(iris, Species=="setosa")[,1:3]
hv = hypervolume_box(data=iris_ss,name='setosa')
probs <- hypervolume_estimate_probability(hv, points=iris_ss)
# first point should have non-zero density, second, zero
```

---

# 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 scalar or a n x 1 vector corresponding to the diagonal covariance matrix entries of the Gaussian kernel in each dimension. If a scalar input, the single value is used for all dimensions. Several esimation methods are available in `estimate_bandwidth`. |
| `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(iris)
hv = hypervolume_gaussian(data=subset(iris, Species=="setosa")[,1:2],name='setosa')
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(iris)
iris[,"Species"] <- iris[,"Species"] == "setosa"
m_glm = glm(Species~.,data=iris)

hv_general_glm = hypervolume_general_model(m_glm,
  range.box=padded_range(iris[,1:4]),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(iris)
# data_split = split(iris[,1:3],iris$Species)
# hvs_split = lapply(data_split, hypervolume);
# hvs_joined = hypervolume_join(hvs_split)
```

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(iris)
hv1 = hypervolume_gaussian(subset(iris, Species=="virginica")[,1:3])
hv2 = hypervolume_gaussian(subset(iris, Species=="versicolor")[,1:3])
hv_set <- hypervolume_set(hv1, hv2, check.memory=FALSE)

hypervolume_overlap_statistics(hv_set)

## 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.

## 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

```
# example does not run to meet CRAN runtime guidelines - set TRUE to run
hypervolume_project_demo = FALSE
if (hypervolume_project_demo==TRUE)
{
  # 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)
}
```

---

`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

```
# low sample sizes to meet CRAN time requirements
data(iris)
hv1 <- hypervolume_gaussian(iris[,1:3],kde.bandwidth=0.1)
hv1_segmented <- hypervolume_segment(hv1,
                          num.points.max=100, distance.factor=1,
                          check.memory=FALSE) # intentionally under-segment
hv1_segmented_pruned <- hypervolume_prune(hv1_segmented,
                          num.points.min=10)
plot(hv1_segmented_pruned)
```

---

## 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_save_animated_gif *Saves animated GIF of three-dimensional hypervolume plot.*

---

## Description

Assumes there is an open RGL plot (e.g. from calling `plot(hv,show.3d=TRUE)`). Rotates the plot around an axis at a given speed and saves results as a series of GIFs. 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 for speed - uncomment to try!
#data(iris)
# hv = hypervolume_gaussian(iris[,1:3])
# plot(hv, show.3d=TRUE)
# hypervolume_save_animated_gif()
```

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(iris)
hv1 <- hypervolume_gaussian(iris[,1:3],kde.bandwidth=0.1)
hv1_segmented <- hypervolume_segment(hv1, num.points.max=100,
                  distance.factor=2, check.memory=FALSE)
plot(hv1_segmented)
```

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.

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 is 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.

## 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_set12 = hypervolume_set(hv1, hv2, check.memory=FALSE)
hv_set23 = hypervolume_set(hv2, hv3, check.memory=FALSE)

# no overlap found between setosa and virginica
hypervolume_overlap_statistics(hv_set12)

# some overlap found between virginica and versicolor
hypervolume_overlap_statistics(hv_set23)
# examine volumes of each set component
get_volume(hv_set23)

## 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(iris)
hv = hypervolume_svm(data=subset(iris, Species=="setosa")[,1:2],name='setosa')
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(iris)
hv1 = hypervolume_gaussian(subset(iris, Species=="setosa")[,1:3])

# downsample to 1000 random points
hv1_thinned = hypervolume_thin(hv1, num.points=1000)
hv1_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(iris)
hv = hypervolume_gaussian(data=subset(iris, Species=="setosa")[,1:3],name='setosa')

# 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=FALSE, show.data=FALSE,show.centroid=FALSE)

# get hypervolume for a single low quantile value
plot(hypervolume_threshold(hv, plot=FALSE, verbose=FALSE,
  quantile.requested=0.2,quantile.requested.type="volume")[[1]])

## 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(iris)
hv = hypervolume_gaussian(subset(iris, Species=="versicolor")[,1:2],samples.per.point=10)
varimp = hypervolume_variable_importance(hv,verbose=FALSE)
barplot(varimp)
```

---

# 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",]
```

---

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(iris)
iris_rangebox_nopadding = padded_range(iris[,1:3])
iris_rangebox_padding = padded_range(iris[,1:3], multiply.interval.amount=0.5, add.amount=0.1)
```

`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=0.01,
    contour.raster.resolution=100,
   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. |
| `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:
data(iris)
hv = hypervolume_gaussian(iris[,1:3])

plot(hv, show.3d=TRUE)
plot(hv, show.3d=FALSE)

plot(hv,plot.function.additional=function(i,j){
  points( x=iris[iris$Species=="setosa",i],
          y=iris[iris$Species=="setosa",j],col='purple')
  })

## 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 |
| ... | |

### 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 BIEN2 database (http://bien.nceas.ucsb.edu/bien/). 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 |
| `...` | |

## Value

None; used for the side-effect of printing.

`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(iris)
weighted_data <- weight_data(iris[,1:3],weights=1+rpois(n=nrow(iris),lambda=3))
pairs(weighted_data)

weighted_noisy_data <- weight_data(iris[,1:3],weights=1+rpois(n=nrow(iris),lambda=3),jitter.sd=0.5)
pairs(weighted_noisy_data)
```

# Index