Package ‘hypervolume’

March 12, 2015

Type Package

Title High-Dimensional Kernel Density Estimation and Geometry Operations

Version 1.2.2

Date 2015-03-15

Author Benjamin Blonder

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 negative feature detection. Uses stochastic geometry approach to high-dimensional kernel density estimation. Builds n-dimensional convex hulls (polytopes). Can measure the n-dimensional ecological hypervolume and perform species distribution modeling.

License GPL-3

Depends Rcpp, rgl, methods, R (>= 3.0.0)

LinkingTo Rcpp

Imports MASS, geometry, pdist

NeedsCompilation yes

Repository CRAN

Date/Publication 2015-03-12 19:34:29

R topics documented:

hypervolume-package 2
estimate_bandwidth 2
expectation_box 3
expectation_convex 4
expectation_maximal 6
finch 7
get_volume 8
hypervolume 8
Hypervolume-class 9

HypervolumeList-class 10
hypervolume_distance 11
hypervolume_importance 12
hypervolume_inclusion_test 13
hypervolume_join 14
hypervolume_set 14
hypervolume_sorensen_overlap 16
negative_features 17
plot.HypervolumeList 18
quercus 20
summary.Hypervolume 21

Index **22**

`hypervolume-package` *N-dimensional hypervolume operations*

Description

Estimates the shape and volume of high-dimensional objects and performs set operations: intersection / overlap, union, unique components, and inclusion test. Can measure the n-dimensional ecological hypervolume and perform species distribution modeling.

Details

A FAQ can be found at http://www.benjaminblonder.org/hypervolume_faq.html.

Author(s)

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

`estimate_bandwidth` *Silverman bandwidth estimator for hypervolumes.*

Description

Estimates bandwidth vector from data using a Silverman bandwidth estimator applied independently to each axis of the data. This approach provides a heuristic when no other methods are available to choose kernel bandwidth.

Usage

```
estimate_bandwidth(data)
```

Arguments

`data`	m x n matrix or data frame, where m is the number of observations and n the number of dimensions.

Details

The 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. Note that this estimator is optimal only for univariate normal data and not for the box kernels implemented by the hypervolume algorithms.

Value

Vector of length n with each entry corresponding to the estimated bandwidth along each axis.

Examples

```
data(iris)
print(estimate_bandwidth(iris[,1:4]))
```

`expectation_box` *Hyperbox, hyperball expectation*

Description

Creates a hypervolume with geometry of the minimum hyperbox (hyperball) enclosing a set of points

Usage

```
expectation_box(input, npoints = NULL, userandom = FALSE)
expectation_ball(input, npoints = NULL, userandom = FALSE)
```

Arguments

`input`	A data frame or hypervolume object whose data are enclosed by the output hypervolume.
`npoints`	The number of random points in the output hypervolume. If `NULL` and `input` is of class `Hypervolume`, defaults to a value that matches the point density of the input hypervolume; if `NULL` and `input` is a data frame, defaults to 10*10^n, where n is the dimensionality.
`userandom`	If `input` is a hypervolume and `userandom` is `TRUE`, the output hypervolume will enclose the uniformly random points defining the hypervolume; otherwise, it will enclose the data points used to build the hypervolume. If `input` is a data frame, this argument is ignored.

Value

A Hypervolume object containing a uniformly random set of points sampled from the hyperbox (hyperball) enclosing the input data.

Author(s)

Benjamin Blonder

Examples

```
data(iris)
hv1 = hypervolume(subset(iris, Species=="setosa")[,1:4],bandwidth=0.2)

hvboxdata = expectation_box(hv1, userandom=FALSE)
hvboxrandom = expectation_box(hv1, userandom=TRUE)

# show the two expectations
plot(hypervolume_join(hv1, hvboxdata))

plot(hypervolume_join(hv1, hvboxrandom))


# ball expectation
hvballdata = expectation_ball(hv1@Data, userandom=FALSE, npoints=10000)
hvballrandom = expectation_ball(hv1, userandom=TRUE, npoints=10000)

# show the two expectations
plot(hypervolume_join(hv1, hvballdata))

plot(hypervolume_join(hv1, hvballrandom))
```

---

expectation_convex *Convex expectation*

Description

Creates a hypervolume with geometry of the convex polytope (hull) minimally enclosing a set of points

Usage

```
expectation_convex(input,
  npoints_inhull = NULL, npoints_onhull = NULL,
  check_memory = TRUE, userandom = FALSE,
  method = "rejection", burnin = NULL, delta = NULL)
```

Arguments

`input`	A data frame or hypervolume object whose data are enclosed by the output hypervolume.
`npoints_inhull`	Number of random points in the output hypervolume. If `NULL` and `input` is of class `Hypervolume`, defaults to a value that matches the point density of the input hypervolume; if `NULL` and `input` is a data frame, defaults to 10*10^n, where n is the dimensionality.
`npoints_onhull`	Number of data points to sample from the input to define the convex polytope. Samples are weighted by their Euclidean distance from the centroid of the input. If `NULL`, defaults to 10^sqrt(n).
`check_memory`	If `TRUE`, prints estimated memory usage and convex polytope size and exits. Otherwise contines the algorithm. Memory usage can be very high in high dimensions.
`userandom`	If `input` is a hypervolume and `userandom` is `TRUE`, the output hypervolume will enclose the uniformly random points defining the hypervolume; otherwise, it will enclose the data points used to build the hypervolume. If `input` is a data frame, this argument is ignored.
`method`	If `"rejection"`, constructs output hypervolume using hyperbox rejection sampling (recommended) from the convex polytope. If `"metropolis"`, uses Metropolis-Hastings algorithm, discarding the first `burnin` samples with a jump size of `delta` (experimental; not recommended)
`burnin`	Number of MCMC samples to discard if `method="metropolis"`. Defaults to 10^n (value not guaranteed to produce uniformly random results). Otherwise ignored.
`delta`	Jump size for MCMC sampling if `method="metropolis"`. Defaults to one percent of the maximum range of the data along any axis. Otherwise ignored.

Details

The computational complexity of convex polytope algorithms is very high. Running in n>5 dimensions may crash most desktop computers due to high memory requirements. Even if sufficient memory is available, rejection sampling becomes exponentially inefficient with dimensionality, resulting in long runtimes. The Metropolis-Hastings method may run faster in high dimensions but convergence to a uniformly random distribution may require very high burn-in values.

Value

A `Hypervolume` object containing a uniformly random set of points sampled from the convex polytope enclosing the input data.

Author(s)

Benjamin Blonder

Examples

```
# NOTE: uncomment next lines to try example (not run to pass CRAN runtime checks)

data(iris)

#hv1 = hypervolume(subset(iris, Species=="setosa")[,1:4],bandwidth=0.2)

#ec <- expectation_convex(hv1, userandom=F, check_memory=FALSE)
#plot(hypervolume_join(hv1, ec))
```

`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.

Author(s)

Benjamin Blonder

`finch` *Data and demo for Darwin's finches*

Description

Data for nine morphological traits for five species of Darwin's finches occurring on Isabela Island. Demonstration analysis of hypervolume overlaps between all pairs of species.

Usage

```
data(finch)
```

Format

A data frame with 146 observations on the following 10 variables.

`Species` a factor with levels `Geospiza fortis fortis` `Geospiza fortis platyrhyncha` `Geospiza fuliginosa parvula` `Geospiza heliobates` `Geospiza prosthemelas prosthemelas`

`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 numeric vector

`TarsusL` a numeric vector

Source

Data are from 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.

References

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

Examples

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

`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*

Description

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

Usage

```
hypervolume(data, repsperpoint=NULL, bandwidth,
  quantile = 0, name = NULL,
  verbose = T, warnings = T)
```

Arguments

`data`	A m x n matrix or data frame, where m is the number of observations and n is the dimensionality.
`repsperpoint`	The number of random points to generate in the kernel around each data point. Larger values are needed in higher dimensions, and generally produce more accurate results. If `NULL`, defaults to 100*10^sqrt(n) where n is the dimensionality of the input data.

`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. Bandwidth also can be estimated using `estimate_bandwidth` if necessary (not recommended).
`quantile`	A number in [0,1), corresponding to the fraction of probability density to exclude from the hypervolume. A value of 0 encloses all data, while a value closer to 1 excludes more data. Note that this is a requested value; due to the discrete nature of the estimation procedure the obtained quantile may differ. A value of 0 can always be obtained.
`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.
`warnings`	Logical value; checks for several potential issues in the input data if true. Checks for high variance in standard deviations between dimensions (indicating axis scale problems), highly correlated dimensions (indicating axis choice problems), and low number of observations (indicating algorithm applicability problems).

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.

Value

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

Examples

```
data(iris)
hv1 = hypervolume(subset(iris, Species=="setosa")[,1:4],bandwidth=0.2)
summary(hv1)
```

`Hypervolume-class` *Class hypervolume*

Description

A class used to store a stochastic description of a hypervolume.

Objects from the Class

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

Slots

`Name`: Object of class `"character"`. A string naming the hypervolume, used in plotting.

`Data`: Object of class `"matrix"`. If available, the raw data used to construct the hypervolume. Defaults to a one-row NaN vector for hypervolumes returned by set operations.

`Dimensionality`: Object of class `"numeric"`. The dimensionality of the hypervolume.

`Volume`: Object of class `"numeric"`. The volume of the hypervolume, in units of the product of all dimensions.

`PointDensity`: Object of class `"numeric"`. The number density of the uniformly sampled random points characterizing the hypervolume.

`Bandwidth`: Object of class `"numeric"`. If available, the bandwidth vector used to construct the hypervolume. Defaults to a one-row NaN vector for hypervolumes returned by set operations.

`DisjunctFactor`: Object of class `"numeric"`. The ratio of the inferred volume to the volume of a hypervolume constructed from the same data with disjunct data points (i.e. no kernels overlap). Varies from zero to one. High values suggest that bandwidth should be increased.

`RepsPerPoint`: Object of class `"numeric"`. If available, the number of random points used per observation to construct the hypervolume. Defaults to NaN for hypervolumes returned by set operations.

`QuantileThresholdDesired`: Object of class `"numeric"`. If available, the quantile requested by the user and used to construct the hypervolume. Defaults to NaN for hypervolumes returned by set operations.

`QuantileThresholdObtained`: Object of class `"numeric"`. If available, the quantile obtained by the hypervolume algorithm. Defaults to NaN for hypervolumes returned by set operations.

`RandomUniformPointsThresholded`: Object of class `"matrix"` A set of uniformly random points guaranteed to be in the hypervolume.

`ProbabilityDensityAtRandomUniformPoints`: Object of class `"numeric"` A vector of integers proportional to the probability density at each uniformly random point in the hypervolume. Defaults to a 1-valued vector for hypervolumes returned by set operations because set operations are well defined for volumes and not for probability density functions.

Methods

Summary and plot methods are available.

`HypervolumeList-class` *Class HypervolumeList*

Description

A convenience class used to streamline plotting and other hypervolume output.

Objects from the Class

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

Slots

`HVList`: Object of class `"list"`. A list of hypervolumes.

`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", npmax = 1000, check_memory = TRUE)
```

Arguments

`hv1`	A `Hypervolume` object.
`hv2`	A `Hypervolume` object.
`type`	If 'centroid', the centroid distance; if 'minimum', the minimum distance.
`npmax`	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

```
data(iris)
hv1 = hypervolume(subset(iris, Species=="setosa")[,1:4],reps=1000,bandwidth=0.2,warn=FALSE)
hv2 = hypervolume(subset(iris, Species=="virginica")[,1:4],reps=1000,bandwidth=0.2,warn=FALSE)

# note that minimum distance is smaller than centroid distance as expected
hypervolume_distance(hv1, hv2, type='centroid')
hypervolume_distance(hv1, hv2, type='minimum', npmax=500, check_memory=FALSE)
```

hypervolume_importance *Hypervolume variable importance*

Description

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

Usage

```
hypervolume_importance(hv)
```

Arguments

hv	A hypervolume for which the importance of each variable should be calculated.

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.

This function is currently experimental. Other definitions of variable importance may also be possible. The algorithm currently cannot be used on set operations hypervolumes because the variable deletion process is not well defined for objects that are not associated with a particular set of observations.

Value

A named vector with importance scores for each axis. Note that these scores are not dimensionless but rather have linear units.

Examples

```
data(iris)
hv1 = hypervolume(subset(iris, Species=="setosa")[,1:4],reps=1000,bandwidth=0.2)
imp = hypervolume_importance(hv1)

print(imp)
```

`hypervolume_inclusion_test` *Inclusion test*

Description

Determines if a set of points are within a hypervolume.

Usage

```
hypervolume_inclusion_test(hv, points, reduction_factor = 1, verbose = T)
```

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.
`verbose`	Logical value; print diagnostic output if true.

Details

Tests if a point is in a hypervolume by determining if it is within a characteristic distance of at least one point from a uniformly random sample of the hypervolume.

Value

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

Examples

```
# 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(data, reps=1000, bandwidth=0.1)

# 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)))
```

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`

Value

A `HypervolumeList` containing all hypervolumes in all arguments.

Author(s)

Benjamin Blonder

Examples

```
# UNCOMMENT TO RUN - disabled to meet CRAN runtime guidelines

#data(iris)
#hv1 = hypervolume(subset(iris, Species=="setosa")[,1:4],bandwidth=0.2,name='setosa')
#hv2 = hypervolume(subset(iris, Species=="versicolor")[,1:4],bandwidth=0.2,name='versicolor')

#hv_all = hypervolume_join(hv1, hv2)

#plot(hv_all)
```

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

Description

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

Usage

```
hypervolume_set(hv1, hv2, npoints_max = NULL, verbose = TRUE, check_memory = TRUE)
```

Arguments

`hv1`	A n-dimensional hypervolume
`hv2`	A n-dimensional hypervolume
`npoints_max`	Maximum number of random points to use for set operations. If `NULL` defaults to 100*10^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.

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. 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 `npoints_max`.

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$Intersection@Volu / hv_set@HVList$Union@Volume.

If check_memory is true, instead returns a scalar with the expected number of pairwise comparisons.

Examples

```
data(iris)
hv1 = hypervolume(subset(iris, Species=="setosa")[,1:4],reps=1000,bandwidth=0.2,
  warn=FALSE,name='setosa')
hv2 = hypervolume(subset(iris, Species=="virginica")[,1:4],reps=1000,bandwidth=0.2,
  warn=FALSE,name='virginica')
hv3 = hypervolume(subset(iris, Species=="versicolor")[,1:4],reps=1000,bandwidth=0.2,
  warn=FALSE,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_sorensen_overlap(hv_set12)

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

`hypervolume_sorensen_overlap`
Sorensen similarity index for hypervolume set operations

Description

Calculates the Sorensen index, i.e. for hypervolumes A and B, 2*|A int B| / (|A| + |B|). Note that this definition has been changed since version 1.2 of the package.

Usage

```
hypervolume_sorensen_overlap(hvlist)
```

Arguments

`hvlist`	A `HypervolumeList` object corresponding to the output of a `hypervolume_set` command.

Value

A number between 0 and 1 indicating how similar two input hypervolumes are.

Examples

```
data(iris)
hv2 = hypervolume(subset(iris, Species=="virginica")[,1:4],reps=1000,bandwidth=0.2,warn=FALSE)
hv3 = hypervolume(subset(iris, Species=="versicolor")[,1:4],reps=1000,bandwidth=0.2,warn=FALSE)
```

```
hv_set23 = hypervolume_set(hv2, hv3, check_memory=FALSE)

# no overlap found between setosa and virginica
hypervolume_sorensen_overlap(hv_set23)
```

---

negative_features *Negative feature detection*

Description

Detects the negative features of an observed hypervolume relative to an expectation

Usage

```
negative_features(hv_obs, hv_exp, set_npoints_max = NULL, set_check_memory = TRUE)
```

Arguments

`hv_obs`	The observed hypervolume whose negative features are to be detected
`hv_exp`	The expected hypervolume that provides a baseline expectation geometry
`set_npoints_max`	Maximum number of points to be used for set operations comparing `hv_obs` to `hv_exp`. Defaults to 100*10^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 negative features that do not actually exist). However it can have a high Type II error rate (failure to find negative features 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_npoints_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.

Value

A `Hypervolume` object containing a uniformly random set of points describing the negative features 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.

Author(s)

Benjamin Blonder

Examples

```
# 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(data_annulus,bandwidth=0.1,name='annulus',reps=500)

# GET CONVEX EXPECTATION
hv_convex <- expectation_convex(hv_annulus, check_memory=FALSE)

# DETECT FEATURES (low npoints for fast execution)
features_annulus <- negative_features(
                      hv_obs=hv_annulus,
                      hv_exp=hv_convex,
                      set_check_memory=FALSE)


# PLOT RESULTS
plot(hypervolume_join(hv_annulus, features_annulus))
```

`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,
npmax_data = 1000, npmax_random = 1000,
colors=rainbow(length(x@HVList),alpha=0.8), names=NULL,
reshuffle=TRUE, showrandom=TRUE, showdensity=TRUE,showdata=TRUE,darkfactor=0.5,
cex.random=0.5,cex.data=0.75,cex.axis=0.75,cex.names=1.0,cex.legend=0.75,
legend=TRUE, varlims=NULL,
showcontour=TRUE, contour.lwd=1, contour.filled=FALSE,contour.filled.alpha=0.5,
 showcentroid=TRUE, cex.centroid=3,
pairplot=TRUE,whichaxes=NULL,...)
```

Arguments

`x`	A Hypervolume or HypervolumeList object. The objects to be plotted.
`npmax_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.
`npmax_random`	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.
`colors`	A vector of colors to be used to plot each hypervolume, in the same order as the input hypervolumes.
`names`	A vector of strings in the same order as the input hypervolumes. Used to draw the axes labels.
`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.
`showrandom`	A logical value indicating whether the random points comprising the hypervolume should be drawn.
`showdensity`	A logical value indicating whether the probability density of each hypervolume should be drawn by modulating alpha values (more transparent for lower 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. Used only if `showrandom=TRUE`.
`showdata`	A logical value indicating whether the original data should be drawn on top of the uniformly random points. 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.
`darkfactor`	A value in [0,1] that modulates the color of data points, if shown. Values closer to 0 make data points more black, while values closer to 1 make data points closer to the input color.
`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
`legend`	Logical value indicating if a legend should be plotted, if pair=T.
`varlims`	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.
`showcontour`	A logical value indicating whether a boundary line should be drawn around each two-dimensional projection. Ignored if `pair=FALSE`.
`contour.lwd`	Line width used for contour lines. Ignored if `showcontour=FALSE`.
`contour.filled`	A logical value indicating whether contours should be shaded. Ignored if `showcontour=FALSE`.

`contour.filled.alpha`	An alpha value used to determine the transparency of shaded contours. Ignored if `contour.filled=FALSE`.
`showcentroid`	A logical value indicating whether a solid point indicating the centroid for each hypervolume should be drawn. Centroids are randomly jittered by 1 percent to improve visual clarity.
`cex.centroid`	Expansion factor for the centroid symbol.
`pairplot`	If true, a pair plot is produced. If false, a 3D plot is produced.
`whichaxes`	A length-three vector of integer IDs corresponding to the axes to be plotted when pair=F.
`...`	

Value

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

Note

Note that contour lines are drawn only for convenience and visual presentation. They are calculated using `MASS::kde2d` using projected data with a five percent quantile threshold. The true hypervolume boundary (on which volumes and set operations are calculated) is reflected by the less smooth uniformly random set of points drawn by `showrandom=TRUE`.

Examples

```
data(iris)
hv1 = hypervolume(subset(iris, Species=="setosa")[,1:4],reps=1000,bandwidth=0.2)

# choose fixed axes
plot(hv1, pair=TRUE, varlims=list(c(3,6),c(2,5),c(0,3),c(-1,1)))
```

`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 (in press)

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.

Index

∗Topic **\textasciitildekwd1**
  hypervolume_distance, 11
∗Topic **\textasciitildekwd2**
  hypervolume_distance, 11

estimate_bandwidth, 2, *9*
expectation_ball *(*expectation_box*)*, 3
expectation_box, 3
expectation_convex, 4
expectation_maximal, 6

finch, 7

get_volume, 8
get_volume,Hypervolume-method
  *(*get_volume*)*, 8
get_volume,HypervolumeList-method
  *(*get_volume*)*, 8
get_volume.Hypervolume *(*get_volume*)*, 8
get_volume.HypervolumeList
  *(*get_volume*)*, 8

hypervolume, 8
Hypervolume-class, 9
hypervolume-package, 2
hypervolume_distance, 11
hypervolume_importance, 12
hypervolume_inclusion_test, 13
hypervolume_join, 14
hypervolume_set, 14
hypervolume_sorensen_overlap, 16
HypervolumeList-class, 10

negative_features, 17

plot.Hypervolume
  *(*plot.HypervolumeList*)*, 18
plot.HypervolumeList, 18

quercus, 20

show.Hypervolume *(*summary.Hypervolume*)*,
  21
show.HypervolumeList
  *(*summary.Hypervolume*)*, 21
summary.Hypervolume, 21
summary.HypervolumeList
  *(*summary.Hypervolume*)*, 21