Package 'TDApplied'

October 29, 2024

Type Package

Title Machine Learning and Inference for Topological Data Analysis

Version 3.0.4

Author Shael Brown [aut, cre],
Dr. Reza Farivar [aut, fnd]

Maintainer Shael Brown <shaelebrown@gmail.com>

Description Topological data analysis is a powerful tool for finding non-linear global structure in whole datasets. The main tool of topological data analysis is persistent homology, which computes
a topological shape descriptor of a dataset called a persistence diagram. 'TDApplied' provides useful and efficient methods for analyzing groups of persistence diagrams with machine learning and statistical inference,
and these functions can also interface with other data science packages to form flexible and integrated
topological data analysis pipelines.

Depends R (>= 3.5.0)

Imports parallel, doParallel, foreach, clue, rdist, parallelly,
kernlab, iterators, methods, stats, utils, Rcpp (>= 0.11.0)

License GPL (>= 3)

URL https://github.com/shaelebrown/TDApplied

BugReports https://github.com/shaelebrown/TDApplied/issues

Encoding UTF-8

NeedsCompilation yes

RoxygenNote 7.3.2

Suggests rmarkdown, knitr, testthat (>= 3.0.0), TDAstats, reticulate,
TDA, igraph

LinkingTo Rcpp

VignetteBuilder knitr, rmarkdown

Config/testthat/edition 3

Repository CRAN

Date/Publication 2024-10-29 08:30:02 UTC

Contents

`analyze_representatives` *Analyze the data point memberships of multiple representative (co)cycles.*

Description

Multiple distance matrices with corresponding data points can contain the same topological features. Therefore we may wish to compare many representative (co)cycles across distance matrices to decide if their topological features are the same. The 'analyze_representatives' function returns a matrix of binary datapoint memberships in an input list of representatives across distance matrices. Optionally this matrix can be plotted as a heatmap with columns as data points and rows (i.e. representatives) reordered by similarity, and the contributions (i.e. percentage membership) of each point in the representatives can also be returned. The heatmap has dark red squares representing membership - location [i,j] is dark red if data point j is in representative i.

Usage

```
analyze_representatives(
  diagrams,
  dim,
  num_points,
  plot_heatmap = TRUE,
  return_contributions = FALSE,
  boxed_reps = NULL,
  d = NULL,
  lwd = NULL,
  title = NULL,
  return_clust = FALSE
)
```

Arguments

diagrams	a list of persistence diagrams, either the output of persistent homology calculations like ripsDiag/calculate_homology/PyH, diagram_to_df or bootstrap_persistence_threshold
dim	the integer homological dimension of representatives to consider.
num_points	the integer number of data points in all the original datasets (from which the diagrams were calculated).
plot_heatmap	a boolean representing if a heatmap of data point membership similarity of the representatives should be plotted, default 'TRUE'. A dendrogram of hierarchical clustering is plotted, and rows (representatives) are sorted according to this clustering.
return_contributions	a boolean indicating whether or not to return the membership contributions (i.e. percentages) of the data points (1:'num_points') across all the representatives, default 'FALSE'.
boxed_reps	a data frame specifying specific rows of the output heatmap which should have a box drawn around them (for highlighting), default NULL. See the details section for more information.
d	either NULL (default) or a "dist" object representing a distance matrix for the representatives, which must have the same number of rows and columns as cycles in the dimension 'dim'.
lwd	a positive number width for the lines of drawn boxes, if boxed_reps is not null.
title	a character string title for the plotted heatmap, default NULL.
return_clust	a boolean determining whether or not to return the result of the 'stats::hclust()' call when a heatmap is plotted, default 'FALSE'.

Details

The clustering dendrogram can be used to determine if there are any similar groups of representatives (i.e. shared topological features across datasets) and if so how many. The row labels of the heatmap are of the form 'DX[Y]', meaning the Yth representative of diagram X, and the column labels are the data point numbers. If diagrams are the output of the bootstrap_persistence_thresholds

function, then the subsetted_representatives (if present) will be analyzed. Therefore, a column label like 'DX[Y]' in the plotted heatmap would mean the Yth representative of diagram X. If certain representatives should be highlighted (by drawing a box around its row) in the heatmap, a dataframe 'boxed_reps' can be supplied with two integer columns - 'diagram' and 'rep'. For example, if we wish to draw a box for DX[Y] then we add the row (diagram = X,rep = Y) to 'boxed_reps'. If 'd' is supplied then it will be used to cluster the representatives, based on the distances in 'd'.

Value

either a matrix of data point contributions to the representatives, or a list with elements "memberships" (the matrix) and some combination of elements "contributions" (a vector of membership percentages for each data point across representatives) and "clust" (the results of 'stats::hclust()' on the membership matrix).

Author(s)

Shael Brown - <shaelebrown@gmail.com>

`bootstrap_persistence_thresholds` *Estimate persistence threshold(s) for topological features in a data set using bootstrapping.*

Description

Bootstrapping is used to find a conservative estimate of a 1-'alpha' percent "confidence interval" around each point in the persistence diagram of the data set, and points whose intervals do not touch the diagonal (birth == death) would be considered "significant" or "real". One threshold is computed for each dimension in the diagram.

Usage

```
bootstrap_persistence_thresholds(
  X,
  FUN_diag = "calculate_homology",
  FUN_boot = "calculate_homology",
  maxdim = 0,
  thresh,
  distance_mat = FALSE,
  ripser = NULL,
  ignore_infinite_cluster = TRUE,
  calculate_representatives = FALSE,
  num_samples = 30,
  alpha = 0.05,
  return_subsetted = FALSE,
  return_pvals = FALSE,
  return_diag = TRUE,
```

```
  num_workers = parallelly::availableCores(omit = 1),
  p_less_than_alpha = FALSE,
  ...
)
```

Arguments

X	the input dataset, must either be a matrix or data frame.
FUN_diag	a string representing the persistent homology function to use for calculating the full persistence diagram, either 'calculate_homology' (the default), 'PyH' or 'ripsDiag'.
FUN_boot	a string representing the persistent homology function to use for calculating the bootstrapped persistence diagrams, either 'calculate_homology' (the default), 'PyH' or 'ripsDiag'.
maxdim	the integer maximum homological dimension for persistent homology, default 0.
thresh	the positive numeric maximum radius of the Vietoris-Rips filtration.
distance_mat	a boolean representing if 'X' is a distance matrix (TRUE) or not (FALSE, default). dimensions together (TRUE, the default) or if one threshold should be calculated for each dimension separately (FALSE).
ripser	the imported ripser module when 'FUN_diag' or 'FUN_boot' is 'PyH'.
ignore_infinite_cluster	a boolean indicating whether or not to ignore the infinitely lived cluster when 'FUN_diag' or 'FUN_boot' is 'PyH'.
calculate_representatives	a boolean representing whether to calculate representative (co)cycles, default FALSE. Note that representatives cant be calculated when using the 'calculate_homology' function.
num_samples	the positive integer number of bootstrap samples, default 30.
alpha	the type-1 error threshold, default 0.05.
return_subsetted	a boolean representing whether or not to return the subsetted persistence diagram (with or without representatives), default FALSE.
return_pvals	a boolean representing whether or not to return p-values for features in the subsetted diagram, default FALSE.
return_diag	a boolean representing whether or not to return the calculated persistence diagram, default TRUE.
num_workers	the integer number of cores used for parallelizing (over bootstrap samples), default one less the maximum amount of cores on the machine.
p_less_than_alpha	a boolean representing whether or not subset further and return only feature whose p-values are strictly less than 'alpha', default 'FALSE'. Note that this is not part of the original bootstrap procedure.
...	additional parameters for internal methods.

Details

The thresholds are then determined by calculating the 1-‘alpha’‘ percentile of the bottleneck distance values between the real persistence diagram and other diagrams obtained by bootstrap resampling the data. Since ‘ripsDiag‘ is the slowest homology engine but is the only engine which calculates representative cycles (as opposed to co-cycles with ‘PyH‘), two homology engines are input to this function - one to calculate the actual persistence diagram, ‘FUN_diag‘ (possibly with representative (co)cycles) and one to calculate the bootstrap diagrams, ‘FUN_boot‘ (this should be a faster engine, like ‘calculate_homology‘ or ‘PyH‘). p-values can be calculated for any feature which survives the thresholding if both ‘return_subsetted‘ and ‘return_pvals‘ are ‘TRUE‘, however these values may be larger than the original ‘alpha‘ value in some cases. Note that this is not part of the original bootstrap procedure. If stricter thresholding is desired, or the p-values must be less than ‘alpha‘, set ‘p_less_than_alpha‘ to ‘TRUE‘. The minimum possible p-value is always 1/(‘num_samples‘ + 1). Note that since `calculate_homology` can ignore the longest-lived cluster, fewer "real" clusters may be found. To avoid this possibility try setting ‘FUN_diag‘ equal to ’ripsDiag’. Please note that due to the TDA package no longer being available on CRAN, if ‘FUN_diag‘ or ‘FUN_boot‘ are ’ripsDiag’ then ‘bootstrap_persistence_thresholds‘ will look for the ripsDiag function in the global environment, so the TDA package should be attached with ‘library("TDA")‘ prior to use.

Value

either a numeric vector of threshold values, with one for each dimension 0..‘maxdim‘ (in that order), or a list containing those thresholds and elements (if desired)

Author(s)

Shael Brown - <shaelebrown@gmail.com>

References

Chazal F et al (2017). "Robust Topological Inference: Distance to a Measure and Kernel Distance." https://www.jmlr.org/papers/volume18/15-484/15-484.pdf.

Examples

```
if(require("TDAstats"))
{
  # create a persistence diagram from a sample of the unit circle
  df <- TDAstats::circle2d[sample(1:100,size = 50),]

  # calculate persistence thresholds for alpha = 0.05
  # and return the calculated diagram as well as the subsetted diagram
  bootstrapped_diagram <- bootstrap_persistence_thresholds(X = df,
  maxdim = 1,thresh = 2,num_workers = 2)
}
```

check_PyH_setup *Make sure that python has been configured correctly for persistent homology calculations.*

Description

Ensures that the reticulate package has been installed, that python is available to be used by reticulate functions, and that the python module "ripser" has been installed.

Usage

```
check_PyH_setup()
```

Details

An error message will be thrown if any of the above conditions are not met.

Author(s)

Shael Brown - <shaelebrown@gmail.com>

check_ripser *Verify an imported ripser module.*

Description

Verify an imported ripser module.

Usage

```
check_ripser(ripser)
```

Arguments

ripser	the ripser module object.

Author(s)

Shael Brown - <shaelebrown@gmail.com>

`diagram_distance` *Calculate distance between a pair of persistence diagrams.*

Description

Calculates the distance between a pair of persistence diagrams, either the output from a `diagram_to_df` function call or from a persistent homology calculation like ripsDiag/`calculate_homology`/`PyH`, in a particular homological dimension.

Usage

```
diagram_distance(
  D1,
  D2,
  dim = 0,
  p = 2,
  distance = "wasserstein",
  sigma = NULL,
  rho = NULL
)
```

Arguments

`D1`	the first persistence diagram.
`D2`	the second persistence diagram.
`dim`	the non-negative integer homological dimension in which the distance is to be computed, default 0.
`p`	a number representing the wasserstein power parameter, at least 1 and default 2.
`distance`	a string which determines which type of distance calculation to carry out, either "wasserstein" (default) or "fisher".
`sigma`	either NULL (default) or a positive number representing the bandwidth for the Fisher information metric.
`rho`	either NULL (default) or a positive number. If NULL then the exact calculation of the Fisher information metric is returned and otherwise a fast approximation, see details.

Details

The most common distance calculations between persistence diagrams are the wasserstein and bottleneck distances, both of which "match" points between their two input diagrams and compute the "loss" of the optimal matching (see https://dl.acm.org/doi/10.1145/3064175 for details). Another method for computing distances, the Fisher information metric, converts the two diagrams into distributions defined on the plane, and calculates a distance between the resulting two distributions (https://proceedings.neurips.cc/paper/2018/file/959ab9a0695c467e7caf75431a872e5c-Paper.pdf). If the ‘distance‘ parameter is "fisher" then ‘sigma‘ must not be NULL. As noted in the Persistence Fisher paper, there is a fast speed-up approximation which has been implemented from

https://github.com/vmorariu/figtree and can be accessed by setting the 'rho' parameter. Smaller values of 'rho' will result in tighter approximations at the expense of longer runtime, and vice versa.

Value

the numeric value of the distance calculation.

Author(s)

Shael Brown - <shaelebrown@gmail.com>

References

Kerber M, Morozov D and Nigmetov A (2017). "Geometry Helps to Compare Persistence Diagrams." https://dl.acm.org/doi/10.1145/3064175.

Le T, Yamada M (2018). "Persistence fisher kernel: a riemannian manifold kernel for persistence diagrams." https://proceedings.neurips.cc/paper/2018/file/959ab9a0695c467e7caf75431a872e5c-Paper.pdf.

Vlad I. Morariu, Balaji Vasan Srinivasan, Vikas C. Raykar, Ramani Duraiswami, and Larry S. Davis. Automatic online tuning for fast Gaussian summation. Advances in Neural Information Processing Systems (NIPS), 2008.

See Also

distance_matrix for distance matrix calculations.

Examples

```
if(require("TDAstats"))
{
  # create two diagrams
  D1 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,size = 20),],
                     dim = 1,threshold = 2)
  D2 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,size = 20),],
                     dim = 1,threshold = 2)

  # calculate 2-wasserstein distance between D1 and D2 in dimension 1
  diagram_distance(D1,D2,dim = 1,p = 2,distance = "wasserstein")

  # calculate bottleneck distance between D1 and D2 in dimension 0
  diagram_distance(D1,D2,dim = 0,p = Inf,distance = "wasserstein")

  # Fisher information metric calculation between D1 and D2 for sigma = 1 in dimension 1
  diagram_distance(D1,D2,dim = 1,distance = "fisher",sigma = 1)

  # repeat but with fast approximation
  ## Not run:
  diagram_distance(D1,D2,dim = 1,distance = "fisher",sigma = 1,rho = 0.001)

## End(Not run)
```

```
}
```

`diagram_kernel` *Calculate persistence Fisher kernel value between a pair of persistence diagrams.*

Description

Returns the persistence Fisher kernel value between a pair of persistence diagrams in a particular homological dimension, each of which is either the output from a `diagram_to_df` function call or from a persistent homology calculation like ripsDiag/`calculate_homology`/`PyH`.

Usage

```
diagram_kernel(D1, D2, dim = 0, sigma = 1, t = 1, rho = NULL)
```

Arguments

`D1`	the first persistence diagram.
`D2`	the second persistence diagram.
`dim`	the non-negative integer homological dimension in which the distance is to be computed, default 0.
`sigma`	a positive number representing the bandwidth for the Fisher information metric, default 1.
`t`	a positive number representing the scale for the persistence Fisher kernel, default 1.
`rho`	an optional positive number representing the heuristic for Fisher information metric approximation, see `diagram_distance`. Default NULL.

Details

The persistence Fisher kernel is calculated from the Fisher information metric according to the formula $k_{PF}(D_1, D_2) = exp(-t * d_{FIM}(D_1, D_2))$, resembling a radial basis kernel for standard Euclidean spaces.

Value

the numeric kernel value.

Author(s)

Shael Brown - `<shaelebrown@gmail.com>`

References

Le T, Yamada M (2018). "Persistence fisher kernel: a riemannian manifold kernel for persistence diagrams." https://proceedings.neurips.cc/paper/2018/file/959ab9a0695c467e7caf75431a872e5c-Paper.pdf.

Murphy, K. "Machine learning: a probabilistic perspective", MIT press (2012).

See Also

gram_matrix for Gram (i.e. kernel) matrix calculations.

Examples

```
if(require("TDAstats"))
{
  # create two diagrams
  D1 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,20),],
                      dim = 1,threshold = 2)
  D2 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,20),],
                      dim = 1,threshold = 2)

  # calculate the kernel value between D1 and D2 with sigma = 2, t = 2 in dimension 1
  diagram_kernel(D1,D2,dim = 1,sigma = 2,t = 2)
  # calculate the kernel value between D1 and D2 with sigma = 2, t = 2 in dimension 0
  diagram_kernel(D1,D2,dim = 0,sigma = 2,t = 2)
}
```

---

diagram_kkmeans *Cluster a group of persistence diagrams using kernel k-means.*

Description

Finds latent cluster labels for a group of persistence diagrams, using a kernelized version of the popular k-means algorithm. An optimal number of clusters may be determined by analyzing the withinss field of the clustering object over several values of k.

Usage

```
diagram_kkmeans(
  diagrams,
  K = NULL,
  centers,
  dim = 0,
  t = 1,
  sigma = 1,
  rho = NULL,
  num_workers = parallelly::availableCores(omit = 1),
  ...
)
```

Arguments

diagrams	a list of n>=2 persistence diagrams which are either the output of a persistent homology calculation like ripsDiag/calculate_homology/PyH, or the diagram_to_df function.
K	an optional precomputed Gram matrix of persistence diagrams, default NULL.
centers	number of clusters to initialize, no more than the number of diagrams although smaller values are recommended.
dim	the non-negative integer homological dimension in which the distance is to be computed, default 0.
t	a positive number representing the scale for the persistence Fisher kernel, default 1.
sigma	a positive number representing the bandwidth for the Fisher information metric, default 1.
rho	an optional positive number representing the heuristic for Fisher information metric approximation, see diagram_distance. Default NULL. If supplied, Gram matrix calculation is sequential.
num_workers	the number of cores used for parallel computation, default is one less than the number of cores on the machine.
...	additional parameters for the kkmeans kernlab function.

Details

Returns the output of kkmeans on the desired Gram matrix of a group of persistence diagrams in a particular dimension. The additional list elements stored in the output are needed to estimate cluster labels for new persistence diagrams in the ‘predict_diagram_kkmeans‘ function.

Value

a list of class ’diagram_kkmeans’ containing the output of kkmeans on the Gram matrix, i.e. a list containing the elements

clustering an S4 object of class specc, the output of a kkmeans function call. The ‘.Data‘ slot of this object contains cluster memberships, ‘withinss‘ contains the within-cluster sum of squares for each cluster, etc.

diagrams the input ‘diagrams‘ argument.

dim the input ‘dim‘ argument.

t the input ‘t‘ argument.

sigma the input ‘sigma‘ argument.

Author(s)

Shael Brown - <shaelebrown@gmail.com>

References

Dhillon, I and Guan, Y and Kulis, B (2004). "A Unified View of Kernel k-means , Spectral Clustering and Graph Cuts." https://people.bu.edu/bkulis/pubs/spectral_techreport.pdf.

See Also

predict_diagram_kkmeans for predicting cluster labels of new diagrams.

Examples

```
if(require("TDAstats"))
{
  # create two diagrams
  D1 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,20),],
                     dim = 1,threshold = 2)
  D2 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,20),],
                     dim = 1,threshold = 2)
  g <- list(D1,D1,D2,D2)

  # calculate kmeans clusters with centers = 2, and sigma = t = 2 in dimension 0
 clust <- diagram_kkmeans(diagrams = g,centers = 2,dim = 0,t = 2,sigma = 2,num_workers = 2)

  # repeat with precomputed Gram matrix, gives the same result just much faster
  K <- gram_matrix(diagrams = g,num_workers = 2,t = 2,sigma = 2)
  cluster <- diagram_kkmeans(diagrams = g,K = K,centers = 2,dim = 0,sigma = 2,t = 2)

}
```

---

diagram_kpca *Calculate the kernel PCA embedding of a group of persistence diagrams.*

Description

Project a group of persistence diagrams into a low-dimensional embedding space using a kernelized version of the popular PCA algorithm.

Usage

```
diagram_kpca(
  diagrams,
  K = NULL,
  dim = 0,
  t = 1,
  sigma = 1,
  rho = NULL,
  features = 1,
  num_workers = parallelly::availableCores(omit = 1),
  th = 1e-04
)
```

Arguments

diagrams	a list of persistence diagrams which are either the output of a persistent homology calculation like ripsDiag/calculate_homology/PyH, or diagram_to_df.
K	an optional precomputed Gram matrix of the persistence diagrams in 'diagrams', default NULL.
dim	the non-negative integer homological dimension in which the distance is to be computed, default 0.
t	a positive number representing the scale for the persistence Fisher kernel, default 1.
sigma	a positive number representing the bandwidth for the Fisher information metric, default 1.
rho	an optional positive number representing the heuristic for Fisher information metric approximation, see diagram_distance. Default NULL. If supplied, Gram matrix calculation is sequential.
features	number of features (principal components) to return, default 1.
num_workers	the number of cores used for parallel computation, default is one less than the number of cores on the machine.
th	the threshold value under which principal components are ignored (default 0.0001).

Details

Returns the output of kernlab's kpca function on the desired Gram matrix of a group of persistence diagrams in a particular dimension. The prediction function predict_diagram_kpca can be used to project new persistence diagrams using an old embedding, and this could be one practical advantage of using diagram_kpca over diagram_mds. The embedding coordinates can also be used for further analysis, or simply as a data visualization tool for persistence diagrams.

Value

a list of class 'diagram_kpca' containing the elements

pca the output of kernlab's kpca function on the Gram matrix: an S4 object containing the slots 'pcv' (a matrix containing the principal component vectors (column wise)), 'eig' (the corresponding eigenvalues), 'rotated' (the original data projected (rotated) on the principal components) and 'xmatrix' (the original data matrix).

diagrams the input 'diagrams' argument.

t the input 't' argument.

sigma the input 'sigma' argument.

dim the input 'dim' argument.

Author(s)

Shael Brown - <shaelebrown@gmail.com>

References

Scholkopf, B and Smola, A and Muller, K (1998). "Nonlinear Component Analysis as a Kernel Eigenvalue Problem." https://www.mlpack.org/papers/kpca.pdf.

See Also

predict_diagram_kpca for predicting embedding coordinates of new diagrams.

Examples

```
if(require("TDAstats"))
{
  # create six diagrams
  D1 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,20),],
                      dim = 1,threshold = 2)
  D2 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,20),],
                      dim = 1,threshold = 2)
  D3 <- TDAstats::calculate_homology(TDAstats::sphere3d[sample(1:100,20),],
                      dim = 1,threshold = 2)
  D4 <- TDAstats::calculate_homology(TDAstats::sphere3d[sample(1:100,20),],
                      dim = 1,threshold = 2)
  D5 <- TDAstats::calculate_homology(TDAstats::sphere3d[sample(1:100,20),],
                      dim = 1,threshold = 2)
  D6 <- TDAstats::calculate_homology(TDAstats::sphere3d[sample(1:100,20),],
                      dim = 1,threshold = 2)
  g <- list(D1,D2,D3,D4,D5,D6)

  # calculate their 2D PCA embedding with sigma = t = 2 in dimension 1
 pca <- diagram_kpca(diagrams = g,dim = 1,t = 2,sigma = 2,features = 2,num_workers = 2,th = 1e-6)

  # repeat with precomputed Gram matrix, gives same result but much faster
  K <- gram_matrix(diagrams = g,dim = 1,t = 2,sigma = 2,num_workers = 2)
  pca <- diagram_kpca(diagrams = g,K = K,dim = 1,t = 2,sigma = 2,features = 2,th = 1e-6)

}
```

`diagram_ksvm` *Fit a support vector machine model where each training set instance is a persistence diagram.*

Description

Returns the output of kernlab's ksvm function on the Gram matrix of the list of persistence diagrams in a particular dimension.

Usage

```
diagram_ksvm(
  diagrams,
  cv = 1,
  dim,
  t = 1,
  sigma = 1,
  rho = NULL,
  y,
  type = NULL,
  distance_matrices = NULL,
  C = 1,
  nu = 0.2,
  epsilon = 0.1,
  prob.model = FALSE,
  class.weights = NULL,
  fit = TRUE,
  cache = 40,
  tol = 0.001,
  shrinking = TRUE,
  num_workers = parallelly::availableCores(omit = 1)
)
```

Arguments

diagrams	a list of persistence diagrams which are either the output of a persistent homology calculation like ripsDiag/calculate_homology/PyH, or diagram_to_df.
cv	a positive number at most the length of ‘diagrams‘ which determines the number of cross validation splits to be performed (default 1, aka no cross-validation). If ‘prob.model‘ is TRUE then cv is set to 1 since kernlab performs 3-fold CV internally in this case. When performing classification, classes are balanced within each cv fold.
dim	a non-negative integer vector of homological dimensions in which the model is to be fit.
t	either a vector of positive numbers representing the grid of values for the scale of the persistence Fisher kernel or NULL, default 1. If NULL then t is selected automatically, see details.
sigma	a vector of positive numbers representing the grid of values for the bandwidth of the Fisher information metric, default 1.
rho	an optional positive number representing the heuristic for Fisher information metric approximation, see diagram_distance. Default NULL. If supplied, distance matrix calculations are sequential.
y	a response vector with one label for each persistence diagram. Must be either numeric or factor, but doesn’t need to be supplied when ‘type‘ is "one-svc".
type	a string representing the type of task to be performed. Can be any one of "C-svc","nu-svc","one-svc","eps-svr","nu-svr" - default for regression is "eps-svr" and for classification is "C-svc". See ksvm for details.

`distance_matrices`
: an optional list of precomputed Fisher distance matrices, corresponding to the rows in 'expand.grid(dim = dim,sigma = sigma)', default NULL.

`C`
: a number representing the cost of constraints violation (default 1) this is the 'C'-constant of the regularization term in the Lagrange formulation.

`nu`
: numeric parameter needed for nu-svc, one-svc and nu-svr. The 'nu' parameter sets the upper bound on the training error and the lower bound on the fraction of data points to become Support Vector (default 0.2).

`epsilon`
: epsilon in the insensitive-loss function used for eps-svr, nu-svr and eps-bsvm (default 0.1).

`prob.model`
: if set to TRUE builds a model for calculating class probabilities or in case of regression, calculates the scaling parameter of the Laplacian distribution fitted on the residuals. Fitting is done on output data created by performing a 3-fold cross-validation on the training data. For details see references (default FALSE).

`class.weights`
: a named vector of weights for the different classes, used for asymmetric class sizes. Not all factor levels have to be supplied (default weight: 1). All components have to be named.

`fit`
: indicates whether the fitted values should be computed and included in the model or not (default TRUE).

`cache`
: cache memory in MB (default 40).

`tol`
: tolerance of termination criteria (default 0.001).

`shrinking`
: option whether to use the shrinking-heuristics (default TRUE).

`num_workers`
: the number of cores used for parallel computation, default is one less the number of cores on the machine.

Details

Cross validation is carried out in parallel, using a trick noted in doi: 10.1007/s4146801700087 - since the persistence Fisher kernel can be written as $d_{PF}(D_1, D_2) = exp(t * d_{FIM}(D_1, D_2)) = exp(d_{FIM}(D_1, D_2))^t$, we can store the Fisher information metric distance matrix for each sigma value in the parameter grid to avoid recomputing distances, and cross validation is therefore performed in parallel. Note that the response parameter 'y' must be a factor for classification - a character vector for instance will throw an error. If 't' is NULL then 1/'t' is selected as the 1,2,5,10,20,50 percentiles of the upper triangle of the distance matrix of its training sample (per fold in the case of cross-validation). This is the process suggested in the persistence Fisher kernel paper. If any of these values would divide by 0 (i.e. if the training set is small) then the minimum non-zero element is taken as the denominator (and hence the returned parameters may have duplicate rows except for differing error values). If cross-validation is performed then the mean error across folds is still recorded, but the best 't' parameter across all folds is recorded in the cv results table.

Value

a list of class 'diagram_ksvm' containing the elements

cv_results the cross-validation results - a matrix storing the parameters for each model in the tuning grid and its mean cross-validation error over all splits.

best_model a list containing the output of ksvm run on the whole dataset with the optimal model parameters found during cross-validation, as well as the optimal kernel parameters for the model.

diagrams the diagrams which were supplied in the function call.

Author(s)

Shael Brown - <shaelebrown@gmail.com>

References

Murphy, K. "Machine learning: a probabilistic perspective." MIT press (2012).

See Also

predict_diagram_ksvm for predicting labels of new diagrams.

Examples

```
if(require("TDAstats"))
{
  # create four diagrams
  D1 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,20),],
                      dim = 1,threshold = 2)
  D2 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,20),],
                      dim = 1,threshold = 2)
  D3 <- TDAstats::calculate_homology(TDAstats::sphere3d[sample(1:100,20),],
                      dim = 1,threshold = 2)
  D4 <- TDAstats::calculate_homology(TDAstats::sphere3d[sample(1:100,20),],
                      dim = 1,threshold = 2)
  g <- list(D1,D2,D3,D4)

  # create response vector
  y <- as.factor(c("circle","circle","sphere","sphere"))

  # fit model without cross validation
  model_svm <- diagram_ksvm(diagrams = g,cv = 1,dim = c(0),
                            y = y,sigma = c(1),t = c(1),
                            num_workers = 2)
}
```

---

diagram_mds *Dimension reduction of a group of persistence diagrams via metric multidimensional scaling.*

Description

Projects a group of persistence diagrams (or a precomputed distance matrix of diagrams) into a low-dimensional embedding space via metric multidimensional scaling. Such a projection can be used for visualization of data, or a static analysis of the embedding dimensions.

Usage

```
diagram_mds(
  diagrams,
  D = NULL,
  k = 2,
  distance = "wasserstein",
  dim = 0,
  p = 2,
  sigma = NULL,
  rho = NULL,
  eig = FALSE,
  add = FALSE,
  x.ret = FALSE,
  list. = eig || add || x.ret,
  num_workers = parallelly::availableCores(omit = 1)
)
```

Arguments

diagrams	a list of n>=2 persistence diagrams which are either the output of a persistent homology calculation like ripsDiag/calculate_homology/PyH, or diagram_to_df. Only one of 'diagrams' and 'D' need to be supplied.
D	an optional precomputed distance matrix of persistence diagrams, default NULL. If not NULL then 'diagrams' parameter does not need to be supplied.
k	the dimension of the space which the data are to be represented in; must be in {1,2,...,n-1}.
distance	a string representing the desired distance metric to be used, either 'wasserstein' (default) or 'fisher'.
dim	the non-negative integer homological dimension in which the distance is to be computed, default 0.
p	a positive number representing the wasserstein power, a number at least 1 (infinity for the bottleneck distance), default 2.
sigma	a positive number representing the bandwidth for the Fisher information metric, default NULL.
rho	an optional positive number representing the heuristic for Fisher information metric approximation, see diagram_distance. Default NULL. If supplied, distance matrix calculation is sequential.
eig	a boolean indicating whether the eigenvalues should be returned.
add	a boolean indicating if an additive constant c* should be computed, and added to the non-diagonal dissimilarities such that the modified dissimilarities are Euclidean.
x.ret	a boolean indicating whether the doubly centered symmetric distance matrix should be returned.
list.	a boolean indicating if a list should be returned or just the n*k matrix.
num_workers	the number of cores used for parallel computation, default is one less than the number of cores on the machine.

Details

Returns the output of cmdscale on the desired distance matrix of a group of persistence diagrams in a particular dimension. If ‘distance‘ is "fisher" then ‘sigma‘ must not be NULL.

Value

the output of cmdscale on the diagram distance matrix. If ‘list.‘ is false (as per default), a matrix with ‘k‘ columns whose rows give the coordinates of the points chosen to represent the dissimilarities.

Otherwise, a list containing the following components.

points a matrix with ‘k‘ columns whose rows give the coordinates of the points chosen to represent the dissimilarities.

eig the n eigenvalues computed during the scaling process if ‘eig‘ is true.

x the doubly centered distance matrix if ‘x.ret‘ is true.

ac the additive constant $c*$, 0 if ‘add‘ = FALSE.

GOF the numeric vector of length 2, representing the sum of all the eigenvalues divided by the sum of their absolute values (first vector element) or by the sum of the max of each eigenvalue and 0 (second vector element).

Author(s)

Shael Brown - <shaelebrown@gmail.com>

References

Cox M and Cox F (2008). "Multidimensional Scaling." doi: 10.1007/9783540330370_14.

Examples

```
if(require("TDAstats"))
{
  # create two diagrams
  D1 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,10),],
                     dim = 1,threshold = 2)
  D2 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,10),],
                     dim = 1,threshold = 2)
  g <- list(D1,D2)

  # calculate their 1D MDS embedding in dimension 0 with the bottleneck distance
  mds <- diagram_mds(diagrams = g,k = 1,dim = 0,p = Inf,num_workers = 2)

  # repeat but with a precomputed distance matrix, gives same result just much faster
  Dmat <- distance_matrix(diagrams = list(D1,D2),dim = 0,p = Inf,num_workers = 2)
  mds <- diagram_mds(D = Dmat,k = 1)

}
```

diagram_to_df *Convert a TDA/TDAstats persistence diagram to a data frame.*

Description

The output of homology calculations from the R packages TDA and TDAstats are not dataframes. This function converts these outputs into a data frame either for further usage in this package or for personalized analyses.

Usage

```
diagram_to_df(d)
```

Arguments

d	the output of a TDA/TDAstats homology calculation, like ripsDiag or calculate_homology.

Details

If a diagram is constructed using a TDA function like ripsDiag with the 'location' parameter set to true then the return value will ignore the location information.

Value

a 3-column data frame, with each row representing a topological feature. The first column is the feature dimension (a non-negative integer), the second column is the birth radius of the feature and the third column is the death radius.

Author(s)

Shael Brown - <shaelebrown@gmail.com>

Examples

```
if(require("TDAstats"))
{
  # create a persistence diagram from a 2D Gaussian
  df = data.frame(x = rnorm(n = 20,mean = 0,sd = 1),y = rnorm(n = 20,mean = 0,sd = 1))

  # compute persistence diagram with calculate_homology from package TDAstats
  phom_TDAstats = TDAstats::calculate_homology(mat = df,dim = 0,threshold = 1)

  # convert to data frame
  phom_TDAstats_df = diagram_to_df(d = phom_TDAstats)
}
```

`distance_matrix` *Compute a distance matrix from a list of persistence diagrams.*

Description

Calculate the distance matrix d for either a single list of persistence diagrams $(D_1, D_2, \ldots, D_n)$, i.e. $d[i,j] = d(D_i, D_j)$, or between two lists, $(D_1, D_2, \ldots, D_n)$ and $(D'_1, D'_2, \ldots, D'_n)$, $d[i,j] = d(D_i, D'_j)$, in parallel.

Usage

```
distance_matrix(
  diagrams,
  other_diagrams = NULL,
  dim = 0,
  distance = "wasserstein",
  p = 2,
  sigma = NULL,
  rho = NULL,
  num_workers = parallelly::availableCores(omit = 1)
)
```

Arguments

`diagrams`	a list of persistence diagrams, either the output of persistent homology calculations like ripsDiag/`calculate_homology`/`PyH`, or `diagram_to_df`.
`other_diagrams`	either NULL (default) or another list of persistence diagrams to compute a cross-distance matrix.
`dim`	the non-negative integer homological dimension in which the distance is to be computed, default 0.
`distance`	a character determining which metric to use, either "wasserstein" (default) or "fisher".
`p`	a number representing the wasserstein power parameter, at least 1 and default 2.
`sigma`	a positive number representing the bandwidth of the Fisher information metric, default NULL.
`rho`	an optional positive number representing the heuristic for Fisher information metric approximation, see `diagram_distance`. Default NULL. If not NULL then matrix is calculated sequentially, but functions in the "exec" directory of the package can be loaded to calculate distance matrices in parallel with approximation.
`num_workers`	the number of cores used for parallel computation, default is one less than the number of cores on the machine.

Details

Distance matrices of persistence diagrams are used in downstream analyses, like in the diagram_mds, permutation_test and diagram_ksvm functions. If ‘distance‘ is "fisher" then ‘sigma‘ must not be NULL. Since the matrix is computed sequentially when approximating the Fisher information metric this is only recommended when the persistence diagrams contain many points and when the number of available cores is small.

Value

the numeric distance matrix.

Author(s)

Shael Brown - <shaelebrown@gmail.com>

See Also

diagram_distance for individual distance calculations.

Examples

```
if(require("TDAstats"))
{
  # create two diagrams
  D1 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,10),],
                                     dim = 0,threshold = 2)
  D2 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,10),],
                                     dim = 0,threshold = 2)
  g <- list(D1,D2)

  # calculate their distance matrix in dimension 0 with the persistence Fisher metric
  # using 2 cores
  D <- distance_matrix(diagrams = g,dim = 0,distance = "fisher",sigma = 1,num_workers = 2)

  # calculate their distance matrix in dimension 0 with the 2-wasserstein metric
  # using 2 cores
  D <- distance_matrix(diagrams = g,dim = 0,distance = "wasserstein",p = 2,num_workers = 2)

  # now do the cross distance matrix, which is the same as the previous
  D_cross <- distance_matrix(diagrams = g,other_diagrams = g,
                             dim = 0,distance = "wasserstein",
                             p = 2,num_workers = 2)
}
```

enclosing_radius *Compute the enclosing radius for a dataset.*

Description

The enclosing radius is the minimum (Euclidean distance) radius beyond which no topological changes will occur.

Usage

```
enclosing_radius(X, distance_mat = FALSE)
```

Arguments

X	the input dataset, must either be a matrix or data frame.
distance_mat	whether or not 'X' is a distance matrix, default FALSE.

Value

the numeric enclosing radius.

Author(s)

Shael Brown - <shaelebrown@gmail.com>

Examples

```
# create a persistence diagram from a 2D Gaussian
df = data.frame(x = rnorm(n = 20,mean = 0,sd = 1),y = rnorm(n = 20,mean = 0,sd = 1))

# compute the enclosing radius from the point cloud
enc_rad <- enclosing_radius(df, distance_mat = FALSE)

# compute the distance matrix manually, stored as a matrix
dist_df <- as.matrix(dist(df))

# compute the enclosing radius from the distance matrix
enc_rad <- enclosing_radius(dist_df, distance_mat = TRUE)
```

gram_matrix *Compute the gram matrix for a group of persistence diagrams.*

Description

Calculate the Gram matrix K for either a single list of persistence diagrams $(D_1, D_2, \ldots, D_n)$, i.e. $K[i,j] = k_{PF}(D_i, D_j)$, or between two lists of persistence diagrams, $(D_1, D_2, \ldots, D_n)$ and $(D'_1, D'_2, \ldots, D'_n)$, $K[i,j] = k_{PF}(D_i, D'_j)$, in parallel.

Usage

```
gram_matrix(
  diagrams,
  other_diagrams = NULL,
  dim = 0,
  sigma = 1,
  t = 1,
  rho = NULL,
  num_workers = parallelly::availableCores(omit = 1)
)
```

Arguments

diagrams	a list of persistence diagrams, where each diagram is either the output of a persistent homology calculation like ripsDiag/calculate_homology/PyH, or diagram_to_df.
other_diagrams	either NULL (default) or another list of persistence diagrams to compute a cross-Gram matrix.
dim	the non-negative integer homological dimension in which the distance is to be computed, default 0.
sigma	a positive number representing the bandwidth for the Fisher information metric, default 1.
t	a positive number representing the scale for the kernel, default 1.
rho	an optional positive number representing the heuristic for Fisher information metric approximation, see diagram_distance. Default NULL. If supplied, code execution is sequential, but functions in the "exec" directory of the package can be loaded to calculate distance matrices in parallel with approximation.
num_workers	the number of cores used for parallel computation, default is one less than the number of cores on the machine.

Details

Gram matrices are used in downstream analyses, like in the ‘diagram_kkmeans‘, ‘diagram_nearest_cluster‘,‘diagram_kpca‘, ‘predict_diagram_kpca‘, ‘predict_diagram_ksvm‘ and ‘independence_test‘ functions.

Value

the numeric (cross) Gram matrix of class 'kernelMatrix'.

Author(s)

Shael Brown - <shaelebrown@gmail.com>

See Also

diagram_kernel for individual persistence Fisher kernel calculations.

Examples

```
if(require("TDAstats"))
{
  # create two diagrams
  D1 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,20),],
                      dim = 1,threshold = 2)
  D2 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,20),],
                      dim = 1,threshold = 2)
  g <- list(D1,D2)

  # calculate the Gram matrix in dimension 0 with sigma = 2, t = 2
  G <- gram_matrix(diagrams = g,dim = 0,sigma = 2,t = 2,num_workers = 2)

  # calculate cross-Gram matrix, which is the same as G
  G_cross <- gram_matrix(diagrams = g,other_diagrams = g,dim = 0,sigma = 2,
                         t = 2,num_workers = 2)
}
```

---

import_ripser *Import the python module ripser.*

Description

The ripser module is needed for fast persistent cohomology calculations with the PyH function.

Usage

```
import_ripser()
```

Details

Same as "reticulate::import("ripser")", just with additional checks.

Value

the python ripser module.

Author(s)

Shael Brown - <shaelebrown@gmail.com>

Examples

```
## Not run:
# import ripser
ripser <- import_ripser()

## End(Not run)
```

`independence_test` *Independence test for two groups of persistence diagrams.*

Description

Carries out inference to determine if two groups of persistence diagrams are independent or not based on kernel calculations (see (https://proceedings.neurips.cc/paper/2007/file/d5cfead94f5350c12c322b5b6
pdf) for details). A small p-value in a certain dimension suggests that the groups are not independent in that dimension.

Usage

```
independence_test(
  g1,
  g2,
  dims = c(0, 1),
  sigma = 1,
  rho = NULL,
  t = 1,
  num_workers = parallelly::availableCores(omit = 1),
  verbose = FALSE,
  Ks = NULL,
  Ls = NULL
)
```

Arguments

g1	the first group of persistence diagrams, where each diagram was either the output from a persistent homology calculation like ripsDiag/calculate_homology/PyH, or diagram_to_df.
g2	the second group of persistence diagrams, where each diagram was either the output from a persistent homology calculation like ripsDiag/calculate_homology/PyH, or diagram_to_df.
dims	a non-negative integer vector of the homological dimensions in which the test is to be carried out, default c(0,1).

sigma	a positive number representing the bandwidth for the Fisher information metric, default 1.
rho	an optional positive number representing the heuristic for Fisher information metric approximation, see diagram_distance. Default NULL. If supplied, calculation of Gram matrices is sequential.
t	a positive number representing the scale for the persistence Fisher kernel, default 1.
num_workers	the number of cores used for parallel computation, default is one less than the number of cores on the machine.
verbose	a boolean flag for if the time duration of the function call should be printed, default FALSE
Ks	an optional list of precomputed Gram matrices for the first group of diagrams, with one element for each dimension. If not NULL and 'Ls' is not NULL then 'g1' and 'g2' do not need to be supplied.
Ls	an optional list of precomputed Gram matrices for the second group of diagrams, with one element for each dimension. If not NULL and 'Ks' is not NULL then 'g1' and 'g2' do not need to be supplied.

Details

The test is carried out with a parametric null distribution, making it much faster than non-parametric approaches. If all of the diagrams in either g1 or g2 are the same in some dimension, then some p-values may be NaN.

Value

a list with the following elements:

dimensions the input 'dims' argument.

test_statisics a numeric vector of the test statistic value in each dimension.

p_values a numeric vector of the p-values in each dimension.

run_time the run time of the function call, containing time units.

Author(s)

Shael Brown - <shaelebrown@gmail.com>

References

Gretton A et al. (2007). "A Kernel Statistical Test of Independence." https://proceedings.neurips.cc/paper/2007/file/d5cfead94f5350c12c322b5b664544c1-Paper.pdf.

See Also

permutation_test for an inferential group difference test for groups of persistence diagrams.

Examples

```
if(require("TDAstats"))
{
  # create two independent groups of diagrams of length 6, which
  # is the minimum length
  D1 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,10),],
                                     dim = 0,threshold = 2)
  D2 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,10),],
                                     dim = 0,threshold = 2)
  g1 <- list(D1,D2,D2,D2,D2,D2)
  g2 <- list(D2,D1,D1,D1,D1,D1)

  # do independence test with sigma = t = 1 in dimension 0, using
  # precomputed Gram matrices
  K = gram_matrix(diagrams = g1,dim = 0,t = 1,sigma = 1,num_workers = 2)
  L = gram_matrix(diagrams = g2,dim = 0,t = 1,sigma = 1,num_workers = 2)
  indep_test <- independence_test(Ks = list(K),Ls = list(L),dims = c(0))

}
```

---

permutation_model_inference

Model inference with permutation test.

Description

An inference procedure to determine if two datasets were unlikely to be generated by the same process (i.e. if the persistence diagram of one dataset is a good model of the persistence diagram of the other dataset).

Usage

```
permutation_model_inference(
  D1,
  D2,
  iterations,
  num_samples,
  dims = c(0, 1),
  samp = NULL,
  paired = F,
  num_workers = parallelly::availableCores(omit = 1),
  verbose = F,
  FUN_boot = "calculate_homology",
  thresh,
  distance_mat = FALSE,
  ripser = NULL,
  return_diagrams = FALSE
)
```

Arguments

D1	the first dataset (a data frame).
D2	the second dataset (a data frame).
iterations	the number of iterations for permuting group labels, default 20.
num_samples	the number of bootstrap iterations, default 30.
dims	a non-negative integer vector of the homological dimensions in which the test is to be carried out, default c(0,1).
samp	an optional list of row-number samples of 'D1', default NULL. See details and examples for more information. Ignored when 'paired' is FALSE.
paired	a boolean flag for if there is a second-order pairing between diagrams at the same index in different groups, default FALSE.
num_workers	the number of cores used for parallel computation, default is one less than the number of cores on the machine.
verbose	a boolean flag for if the time duration of the function call should be printed, default FALSE
FUN_boot	a string representing the persistent homology function to use for calculating the bootstrapped persistence diagrams, either 'calculate_homology' (the default), 'PyH' or 'ripsDiag'.
thresh	the positive numeric maximum radius of the Vietoris-Rips filtration.
distance_mat	a boolean representing if 'X' is a distance matrix (TRUE) or not (FALSE, default). dimensions together (TRUE, the default) or if one threshold should be calculated for each dimension separately (FALSE).
ripser	the imported ripser module when 'FUN_boot' is 'PyH'.
return_diagrams	whether or not to return the two lists of bootstrapped persistence diagrams, default FALSE.

Details

Inference is carried out by generating bootstrap resampled persistence diagrams from the two datasets and carrying out a permutation test on the resulting two groups. A small p-value in a certain dimension suggests that the datasets are not good models of each other. 'samp' should only be provided when 'paired'is TRUE in order to generate the same row samplings of 'D1' and 'D2' for the bootstrapped persistence diagrams. This makes a paired permutation test more appropriate, which has higher statistical power for detecting topological differences. See the examples for how to properly supply 'samp'.

Value

a list which contains the output of the call to permutation_test and the two groups of bootstrapped persistence diagrams if desired, in entries called 'diagrams1' and 'diagrams2'.

Author(s)

Shael Brown - <shaelebrown@gmail.com>

References

Robinson T, Turner K (2017). "Hypothesis testing for topological data analysis." https://link.springer.com/article/10.1007/s41468-017-0008-7.

Chazal F et al (2017). "Robust Topological Inference: Distance to a Measure and Kernel Distance." https://www.jmlr.org/papers/volume18/15-484/15-484.pdf.

Abdallah H et al. (2021). "Statistical Inference for Persistent Homology applied to fMRI." https://github.com/hassan-abdallah/Statistical_Inference_PH_fMRI/blob/main/Abdallah_et_al_Statistical_Inference_PH_fMRI.pdf.

See Also

permutation_test for an inferential group difference test for groups of persistence diagrams and bootstrap_persistence_thresholds for computing confidence sets for persistence diagrams.

Examples

```
if(require("TDAstats"))
{
  # create two datasets
  D1 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,10),],
                                     dim = 0,threshold = 2)
  D2 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,10),],
                                     dim = 0,threshold = 2)

  # do model inference test with 1 iteration (for speed, more
  # iterations should be used in practice)
  model_test <- permutation_model_inference(D1, D2, iterations = 1,
                                            thresh = 1.75,num_samples = 3,
                                            num_workers = 2L)

  # with more iterations, p-values show a difference in the
  # clustering of points but not in the arrangement of loops
  model_test$p_values

  # to supply samp, when we believe there is a correspondence between
  # the rows in D1 and the rows in D2
  # note that the number of entries of samp (3 in this case) must
  # match the num_samples parameter to the function call
  samp <- lapply(X = 1:3,FUN = function(X){

           return(unique(sample(1:nrow(D1),size = nrow(D1),replace = TRUE)))

          })

  # model inference will theoretically have higher power now for a
  # paired test
  model_test2 <- permutation_model_inference(D1, D2, iterations = 1,
                                             thresh = 1.75,num_samples = 3,
                                             paired = TRUE,samp = samp,
                                             num_workers = 2L)
  model_test2$p_values
```

```
}
```

`permutation_test` *Permutation test for finding group differences between persistence diagrams.*

Description

A non-parametric ANOVA-like test for persistence diagrams (see https://link.springer.com/article/10.1007/s41468-017-0008-7 for details). In each desired dimension a test statistic (loss) is calculated, then the group labels are shuffled for some number of iterations and the loss is recomputed each time thereby generating a null distribution for the test statistic. This test generates a p-value in each desired dimension.

Usage

```
permutation_test(
  ...,
  iterations = 20,
  p = 2,
  q = 2,
  dims = c(0, 1),
  dist_mats = NULL,
  group_sizes = NULL,
  paired = FALSE,
  distance = "wasserstein",
  sigma = NULL,
  rho = NULL,
  num_workers = parallelly::availableCores(omit = 1),
  verbose = FALSE
)
```

Arguments

`...`	lists of persistence diagrams which are either the output of persistent homology calculations like ripsDiag/`calculate_homology`/`PyH`, or `diagram_to_df`. Each list must contain at least 2 diagrams.
`iterations`	the number of iterations for permuting group labels, default 20.
`p`	a positive number representing the wasserstein power parameter, a number at least 1 (and Inf if using the bottleneck distance) and default 2.
`q`	a finite number at least 1 for exponentiation in the Turner loss function, default 2.
`dims`	a non-negative integer vector of the homological dimensions in which the test is to be carried out, default c(0,1).

dist_mats	an optional list of precomputed distances matrices, one for each dimension, where the rows and columns would correspond to the unlisted groups of diagrams (in order), default NULL. If not NULL then no lists of diagrams need to be supplied.
group_sizes	a vector of group sizes, one for each group, when ‘dist_mats‘ is not NULL.
paired	a boolean flag for if there is a second-order pairing between diagrams at the same index in different groups, default FALSE
distance	a string which determines which type of distance calculation to carry out, either "wasserstein" (default) or "fisher".
sigma	the positive bandwidth for the Fisher information metric, default NULL.
rho	an optional positive number representing the heuristic for Fisher information metric approximation, see diagram_distance. Default NULL. If supplied, code execution is sequential.
num_workers	the number of cores used for parallel computation, default is one less than the number of cores on the machine.
verbose	a boolean flag for if the time duration of the function call should be printed, default FALSE

Details

The test is carried out in parallel and optimized in order to not recompute already-calculated distances. As such, memory issues may occur when the number of persistence diagrams is very large. Like in (https://github.com/hassan-abdallah/Statistical_Inference_PH_fMRI/blob/main/Abdallah_et_al_Statistical_Inference_PH_fMRI.pdf) an option is provided for pairing diagrams between groups to reduce variance (in order to boost statistical power), and like it was suggested in the original paper functionality is provided for an arbitrary number of groups (not just 2). A small p-value in a dimension suggests that the groups are different (separated) in that dimension. If ‘distance‘ is "fisher" then ‘sigma‘ must not be NULL. TDAstats also has a ‘permutation_test‘ function so care should be taken to use the desired function when using TDApplied with TDAstats. If ‘dist_mats‘ is supplied then the sum of the elements of ‘group_sizes‘ must equal the number of rows and columns of each of its elements.

Value

a list with the following elements:

dimensions the input ‘dims‘ argument.

permvals a numeric vector of length ‘iterations‘ with the permuted loss value for each iteration (permutation)

test_statisics a numeric vector of the test statistic value in each dimension.

p_values a numeric vector of the p-values in each dimension.

run_time the run time of the function call, containing time units.

Author(s)

Shael Brown - <shaelebrown@gmail.com>

References

Robinson T, Turner K (2017). "Hypothesis testing for topological data analysis." https://link.springer.com/article/10.1007/s41468-017-0008-7.

Abdallah H et al. (2021). "Statistical Inference for Persistent Homology applied to fMRI." https://github.com/hassan-abdallah/Statistical_Inference_PH_fMRI/blob/main/Abdallah_et_al_Statistical_Inference_PH_fMRI.pdf.

See Also

independence_test for an inferential test of independence for two groups of persistence diagrams.

Examples

```
if(require("TDAstats"))
{
  # create two groups of diagrams
  D1 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,10),],
                                     dim = 0,threshold = 2)
  D2 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,10),],
                                     dim = 0,threshold = 2)
  g1 <- list(D1,D2)
  g2 <- list(D1,D2)

  # run test in dimension 0 with 1 iteration, note that the TDA package function
  # "permutation_test" can mask TDApplied's function, so we will specify explicitly
  # which function we are using
  perm_test <- TDApplied::permutation_test(g1,g2,iterations = 1,
                                           num_workers = 2,
                                           dims = c(0))

  # repeat with precomputed distance matrix, gives similar results
  # (same but the randomness of the permutations can give small differences)
  # just much faster
  D <- distance_matrix(diagrams = list(D1,D2,D1,D2),dim = 0,
                       num_workers = 2)
  perm_test <- TDApplied::permutation_test(dist_mats = list(D),group_sizes = c(2,2),
                                           dims = c(0))
}
```

---

plot_diagram *Plot persistence diagrams*

Description

Plots a persistence diagram outputted from either a persistent homology calculation or from diagram_to_df, with maximum homological dimension no more than 12 (otherwise the legend doesn't fit in the plot). Each homological dimension has its own color (the rcartocolor color-blind safe color palette) and point type, and the main plot title can be altered via the 'title' parameter. Each feature is plotted with a black point at its center in order to distinguish between overlapping features and easily compare features to their persistence thresholds.

Usage

```
plot_diagram(
  D,
  title = NULL,
  max_radius = NULL,
  legend = TRUE,
  thresholds = NULL
)
```

Arguments

D	a persistence diagram, either outputted from either a persistent homology homology calculation like ripsDiag/calculate_homology/PyH or from diagram_to_df, with maximum dimension at most 12.
title	the character string plot title, default NULL.
max_radius	the x and y limits of the plot are defined as 'c(0,max_radius)', and the default value of 'max_radius' is the maximum death value in 'D'.
legend	a logical indicating whether to include a legend of feature dimensions, default TRUE.
thresholds	either a numeric vector with one persistence threshold for each dimension in 'D' or the output of a bootstrap_persistence_thresholds function call, default NULL.

Details

The 'thresholds' parameter, if not NULL, can either be a user-defined numeric vector, with one entry (persistence threshold) for each dimension in 'D', or the output of bootstrap_persistence_thresholds. Points whose persistence are greater than or equal to their dimension's threshold will be plotted in their dimension's color, and in gray otherwise.

Author(s)

Shael Brown - <shaelebrown@gmail.com>

Examples

```
if(require("TDAstats"))
{
  # create a sample diagram from the unit circle
  df <- TDAstats::circle2d[sample(1:100,50),]
  diag <- TDAstats::calculate_homology(df,threshold = 2)

  # plot without title
  plot_diagram(diag)

  # plot with title
  plot_diagram(diag,title = "Example diagram")

  # determine persistence thresholds
```

```
  thresholds <- bootstrap_persistence_thresholds(X = df,maxdim = 1,
  thresh = 2,num_samples = 3,
  num_workers = 2)

  # plot with bootstrap persistence thresholds
  plot_diagram(diag,title = "Example diagram with thresholds",thresholds = thresholds)

  #' # plot with personalized persistence thresholds
 plot_diagram(diag,title = "Example diagram with personalized thresholds",thresholds = c(0.5,1))
}
```

`plot_vr_graph` *Plot a VR graph using the igraph package.*

Description

This function will throw an error if the igraph package is not installed.

Usage

```
plot_vr_graph(
  graphs,
  eps,
  cols = NULL,
  layout = NULL,
  title = NULL,
  component_of = NULL,
  plot_isolated_vertices = FALSE,
  return_layout = FALSE,
  vertex_labels = TRUE
)
```

Arguments

`graphs`	the output of a ‘vr_graphs‘ function call.
`eps`	the numeric radius of the graph in ‘graphs‘ to plot.
`cols`	an optional character vector of vertex colors, default ‘NULL‘.
`layout`	an optional 2D matrix of vertex coordinates, default ‘NULL‘. If row names are supplied they can be used to subset a graph by those vertex names.
`title`	an optional str title for the plot, default ‘NULL‘.
`component_of`	a vertex name (integer or character), only the component of the graph containing that vertex will be plotted (useful for identifying representative (co)cycles in graphs). Default ‘NULL‘ (plot the whole graph).
`plot_isolated_vertices`	a boolean representing whether or not to plot isolated vertices, default ‘FALSE‘.
`return_layout`	a boolean representing whether or not to return the plotting layout (x-y coordinates of each vertex) and the vertex labels, default ‘FALSE‘.
`vertex_labels`	a boolean representing whether or not to plot vertex labels, default ‘TRUE‘.

Value

if ‘return_layout‘ is ‘TRUE‘ then a list with elements "layout" (the numeric matrix of vertex x-y coordinates) and "vertices" (character vertex labels), otherwise the function does not return anything.

Author(s)

Shael Brown - <shaelebrown@gmail.com>

See Also

vr_graphs for computing VR graphs.

Examples

```
if(require("TDAstats") & require("igraph"))
{
  # simulate data from the unit circle and calculate
  # its diagram
  df <- TDAstats::circle2d[sample(1:100,25),]
  diag <- TDAstats::calculate_homology(df,
                                       dim = 1,
                                       threshold = 2)

  # get minimum death radius of any data cluster
  min_death_H0 <-
  min(diag[which(diag[,1] == 0),3L])

  # get birth and death radius of the loop
  loop_birth <- as.numeric(diag[nrow(diag),2L])
  loop_death <- as.numeric(diag[nrow(diag),3L])

  # compute VR graphs at radii half of
  # min_death_H0 and the mean of loop_birth and
  # loop_death, returning clusters
  graphs <- vr_graphs(X = df,eps =
  c(0.5*min_death_H0,(loop_birth + loop_death)/2))

  # plot graph of smaller (first) radius
  plot_vr_graph(graphs = graphs,eps = 0.5*min_death_H0,
                plot_isolated_vertices = TRUE)

  # plot graph of larger (second) radius
  plot_vr_graph(graphs = graphs,eps = (loop_birth + loop_death)/2)

  # repeat but with rownames for df, each vertex
  # will be plotted with its rownames
  rownames(df) <- paste0("V",1:25)
  graphs <- vr_graphs(X = df,eps =
  c(0.5*min_death_H0,(loop_birth + loop_death)/2))
  plot_vr_graph(graphs = graphs,eps = 0.5*min_death_H0,
                plot_isolated_vertices = TRUE)
```

```
  # plot without vertex labels
  plot_vr_graph(graphs = graphs,eps = (loop_birth + loop_death)/2,
                vertex_labels = FALSE)

  # plot only the graph component containing vertex "1"
  plot_vr_graph(graphs = graphs,eps = 0.5*min_death_H0,
                component_of = "V1",plot_isolated_vertices = TRUE)

  # save the layout of the graph for adding features to
  # the same graph layout, like color
  layout <- plot_vr_graph(graphs = graphs,eps = (loop_birth + loop_death)/2,
                          return_layout = TRUE,vertex_labels = TRUE)
  cols <- rep("blue",25)
  cols[1:5] <- "red"
  plot_vr_graph(graphs = graphs,eps = (loop_birth + loop_death)/2,cols = cols,
                layout = layout)

}
```

`predict_diagram_kkmeans` *Predict the cluster labels for new persistence diagrams using a precomputed clustering.*

Description

Returns the nearest (highest kernel value) `kkmeans` cluster center label for new persistence diagrams. This allows for reusing old cluster models for new tasks, or to perform cross validation.

Usage

```
predict_diagram_kkmeans(
  new_diagrams,
  K = NULL,
  clustering,
  num_workers = parallelly::availableCores(omit = 1)
)
```

Arguments

`new_diagrams`	a list of persistence diagrams which are either the output of a persistent homology calculation like ripsDiag/`calculate_homology`/`PyH`, or `diagram_to_df`. Only one of 'new_diagrams' and 'K' need to be supplied.
`K`	an optional precomputed cross Gram matrix of the new diagrams and the diagrams used in 'clustering', default NULL. If not NULL then 'new_diagrams' does not need to be supplied.
`clustering`	the output of a `diagram_kkmeans` function call, of class 'diagram_kkmeans'.
`num_workers`	the number of cores used for parallel computation, default is one less than the number of cores on the machine.

Value

a vector of the predicted cluster labels for the new diagrams.

Author(s)

Shael Brown - <shaelebrown@gmail.com>

See Also

diagram_kkmeans for clustering persistence diagrams.

Examples

```
if(require("TDAstats"))
{
  # create two diagrams
  D1 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,20),],
                     dim = 1,threshold = 2)
  D2 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,20),],
                     dim = 1,threshold = 2)
  g <- list(D1,D1,D2,D2)

  # calculate kmeans clusters with centers = 2, and sigma = t = 2 in dimension 0
 clust <- diagram_kkmeans(diagrams = g,centers = 2,dim = 0,t = 2,sigma = 2,num_workers = 2)

  # create two new diagrams
  D3 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,20),],
                     dim = 1,threshold = 2)
  D4 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,20),],
                     dim = 1,threshold = 2)
  g_new <- list(D3,D4)

  # predict cluster labels
  predict_diagram_kkmeans(new_diagrams = g_new,clustering = clust,num_workers = 2)

  # predict cluster labels with precomputed Gram matrix, gives same result but
  # much faster
  K <- gram_matrix(diagrams = g_new,other_diagrams = clust$diagrams,
                   dim = clust$dim,t = clust$t,sigma = clust$sigma,
                   num_workers = 2)
  predict_diagram_kkmeans(K = K,clustering = clust)

}
```

---

predict_diagram_kpca *Project persistence diagrams into a low-dimensional space via a pre-computed kernel PCA embedding.*

Description

Compute the location in low-dimensional space of each element of a list of new persistence diagrams using a previously-computed kernel PCA embedding (from the diagram_kpca function).

Usage

```
predict_diagram_kpca(
  new_diagrams,
  K = NULL,
  embedding,
  num_workers = parallelly::availableCores(omit = 1)
)
```

Arguments

new_diagrams	a list of persistence diagrams which are either the output of a persistent homology calculation like ripsDiag/calculate_homology/PyH, or diagram_to_df. Only one of 'new_diagrams' and 'K' need to be supplied.
K	an optional precomputed cross-Gram matrix of the new diagrams and the ones used in 'embedding', default NULL. If not NULL then 'new_diagrams' does not need to be supplied.
embedding	the output of a diagram_kpca function call, of class 'diagram_kpca'.
num_workers	the number of cores used for parallel computation, default is one less than the number of cores on the machine.

Value

the data projection (rotation), stored as a numeric matrix. Each row corresponds to the same-index diagram in 'new_diagrams'.

Author(s)

Shael Brown - <shaelebrown@gmail.com>

See Also

diagram_kpca for embedding persistence diagrams into a low-dimensional space.

Examples

```
if(require("TDAstats"))
{
  # create six diagrams
  D1 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,20),],
                     dim = 1,threshold = 2)
  D2 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,20),],
                     dim = 1,threshold = 2)
  D3 <- TDAstats::calculate_homology(TDAstats::sphere3d[sample(1:100,20),],
                     dim = 1,threshold = 2)
```

```
  D4 <- TDAstats::calculate_homology(TDAstats::sphere3d[sample(1:100,20),],
                    dim = 1,threshold = 2)
  D5 <- TDAstats::calculate_homology(TDAstats::sphere3d[sample(1:100,20),],
                    dim = 1,threshold = 2)
  D6 <- TDAstats::calculate_homology(TDAstats::sphere3d[sample(1:100,20),],
                    dim = 1,threshold = 2)
  g <- list(D1,D2,D3,D4,D5,D6)

  # calculate their 2D PCA embedding with sigma = t = 2 in dimension 0
  pca <- diagram_kpca(diagrams = g,dim = 1,t = 2,sigma = 2,
                    features = 2,num_workers = 2,th = 1e-6)

  # project two new diagrams onto old model
  D7 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,50),],
                                     dim = 0,threshold = 2)
  D8 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,50),],
                                     dim = 0,threshold = 2)
  g_new <- list(D7,D8)

  # calculate new embedding coordinates
  new_pca <- predict_diagram_kpca(new_diagrams = g_new,embedding = pca,num_workers = 2)

  # repeat with precomputed Gram matrix, gives same result but much faster
  K <- gram_matrix(diagrams = g_new,other_diagrams = pca$diagrams,dim = pca$dim,
                   t = pca$t,sigma = pca$sigma,num_workers = 2)
  new_pca <- predict_diagram_kpca(K = K,embedding = pca,num_workers = 2)
}
```

`predict_diagram_ksvm` *Predict the outcome labels for a list of persistence diagrams using a pre-trained diagram ksvm model.*

Description

Returns the predicted response vector of the model on the new diagrams.

Usage

```
predict_diagram_ksvm(
  new_diagrams,
  model,
  K = NULL,
  num_workers = parallelly::availableCores(omit = 1)
)
```

Arguments

`new_diagrams`	a list of persistence diagrams which are either the output of a persistent homology calculation like ripsDiag/`calculate_homology`/`PyH`, or `diagram_to_df`. Only one of 'new_diagrams' and 'K' need to be supplied.

model	the output of a diagram_ksvm function call, of class 'diagram_ksvm'.
K	an optional cross-Gram matrix of the new diagrams and the diagrams in 'model', default NULL. If not NULL then 'new_diagrams' does not need to be supplied.
num_workers	the number of cores used for parallel computation, default is one less than the number of cores on the machine.

Details

This function is a wrapper of the kernlab predict function.

Value

a vector containing the output of predict.ksvm on the cross Gram matrix of the new diagrams and the support vector diagrams stored in the model.

Author(s)

Shael Brown - <shaelebrown@gmail.com>

See Also

diagram_ksvm for training a SVM model on a training set of persistence diagrams and labels.

Examples

```
if(require("TDAstats"))
{
  # create four diagrams
  D1 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,20),],
                     dim = 1,threshold = 2)
  D2 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,20),],
                     dim = 1,threshold = 2)
  D3 <- TDAstats::calculate_homology(TDAstats::sphere3d[sample(1:100,20),],
                     dim = 1,threshold = 2)
  D4 <- TDAstats::calculate_homology(TDAstats::sphere3d[sample(1:100,20),],
                     dim = 1,threshold = 2)
  g <- list(D1,D2,D3,D4)

  # create response vector
  y <- as.factor(c("circle","circle","sphere","sphere"))

  # fit model without cross validation
  model_svm <- diagram_ksvm(diagrams = g,cv = 1,dim = c(0),
                            y = y,sigma = c(1),t = c(1),
                            num_workers = 2)

  # create two new diagrams
  D5 <- TDAstats::calculate_homology(TDAstats::circle2d[sample(1:100,20),],
                     dim = 1,threshold = 2)
  D6 <- TDAstats::calculate_homology(TDAstats::sphere3d[sample(1:100,20),],
                     dim = 1,threshold = 2)
```

```
  g_new <- list(D5,D6)

  # predict with precomputed Gram matrix
  K <- gram_matrix(diagrams = g_new,other_diagrams = model_svm$diagrams,
                   dim = model_svm$best_model$dim,sigma = model_svm$best_model$sigma,
                   t = model_svm$best_model$t,num_workers = 2)
  predict_diagram_ksvm(K = K,model = model_svm,num_workers = 2)
}
```

PyH *Fast persistent homology calculations with python.*

Description

This function is a wrapper of the python wrapper of the ripser engine for persistent cohomology, but is still faster than using the R package TDAstats (see the TDApplied package vignette for details).

Usage

```
PyH(
  X,
  maxdim = 1,
  thresh,
  distance_mat = FALSE,
  ripser,
  ignore_infinite_cluster = TRUE,
  calculate_representatives = FALSE
)
```

Arguments

X	either a matrix or dataframe, representing either point cloud data or a distance matrix. In either case there must be at least two rows and 1 column.
maxdim	the non-negative integer maximum dimension for persistent homology, default 1.
thresh	the non-negative numeric radius threshold for the Vietoris-Rips filtration.
distance_mat	a boolean representing whether the input X is a distance matrix or not, default FALSE.
ripser	the ripser python module.
ignore_infinite_cluster	a boolean representing whether to remove clusters (0 dimensional cycles) which die at the threshold value. Default is TRUE as this is the default for TDAstats homology calculations, but can be set to FALSE which is the default for python ripser.
calculate_representatives	a boolean representing whether to return a list of representative cocycles for the topological features found in the persistence diagram, default FALSE.

Details

If ‘distance_mat‘ is ‘TRUE‘ then ‘X‘ must be a square matrix. The ‘ripser‘ parameter should be the result of an ‘import_ripser‘ function call, but since that function is slow the ripser object should be explicitly created before a PyH function call (see examples). Cohomology is computed over Z2, as is the case for the TDAstats function `calculate_homology` (this is also the default for ripser in c++). If representative cocycles are returned, then they are stored in a list with one element for each point in the persistence diagram, ignoring dimension 0 points. Each representative of a dimension d cocycle (1 for loops, 2 for voids, etc.) is a kxd dimension matrix/array containing the row number-labelled edges, triangles etc. in the cocycle.

Value

Either a dataframe containing the persistence diagram if ‘calculate_representatives‘ is ‘FALSE‘ (the default), otherwise a list with two elements: diagram of class diagram, containing the persistence diagram, and representatives, a list containing the edges, triangles etc. contained in each representative cocycle.

Author(s)

Shael Brown - <shaelebrown@gmail.com>

Examples

```
## Not run:
# create sample data
df <- data.frame(x = 1:10,y = 1:10)

# import the ripser module
ripser <- import_ripser()

# calculate persistence diagram up to dimension 1 with a maximum
# radius of 5
phom <- PyH(X = df,thresh = 5,ripser = ripser)

## End(Not run)
```

`universal_null` *Filtering topological features with the universal null distribution.*

Description

An inference procedure to determine which topological features (if any) of a datasets are likely signal (i.e. significant) vs noise (not).

Usage

```
universal_null(
  X,
  FUN_diag = "calculate_homology",
  maxdim = 1,
  thresh,
  distance_mat = FALSE,
  ripser = NULL,
  ignore_infinite_cluster = TRUE,
  calculate_representatives = FALSE,
  alpha = 0.05,
  return_pvals = FALSE,
  infinite_cycle_inference = FALSE
)
```

Arguments

X	the input dataset, must either be a matrix or data frame.
FUN_diag	a string representing the persistent homology function to use for calculating the full persistence diagram, either 'calculate_homology' (the default), 'PyH' or 'ripsDiag'.
maxdim	the integer maximum homological dimension for persistent homology, default 0.
thresh	the positive numeric maximum radius of the Vietoris-Rips filtration.
distance_mat	a boolean representing if 'X' is a distance matrix (TRUE) or not (FALSE, default). dimensions together (TRUE, the default) or if one threshold should be calculated for each dimension separately (FALSE).
ripser	the imported ripser module when 'FUN_diag' is 'PyH'.
ignore_infinite_cluster	a boolean indicating whether or not to ignore the infinitely lived cluster when 'FUN_diag' is 'PyH'. If infinite cycle inference is to be performed, this parameter should be set to FALSE.
calculate_representatives	a boolean representing whether to calculate representative (co)cycles, default FALSE. Note that representatives cant be calculated when using the 'calculate_homology' function. Note that representatives cannot be computed for (significant) infinite cycles.
alpha	the type-1 error threshold, default 0.05.
return_pvals	a boolean representing whether or not to return p-values for features in the subsetted diagram as well as a list of p-value thresholds, default FALSE. Infinite cycles that are significant (see below) will have p-value NA in this list, as the true value is unknown but less than its dimension's p-value threshold.
infinite_cycle_inference	a boolean representing whether or not to perform inference for features with infinite (i.e. 'thresh') death values, default FALSE. If 'FUN_diag' is 'calculate_homology' (the default) then no infinite cycles will be returned by the persistent homology calculation at all.

Details

For each feature in a diagram we compute its persistence ratio $\pi = death/birth$, and a test statistic $Aloglog\pi + B$ (where A and B are constants). This statistic is compared to a left-skewed Gumbel distribution to get a p-value. A Bonferroni correction is applied to all the p-values across all features, so when 'return_pvals' is TRUE a list of p-value thresholds is also returned, one for each dimension, which is 'alpha' divided by the number of features in that dimension. If desired, infinite cycles (i.e. cycles whose death value is equal to the maximum distance threshold parameter for the persistent homology calculation) can be anaylzed for significance by determining their minimum distance thresholds where they might be significant (using the Gumbel distribution again), calculating the persistence diagram up to those thresholds and seeing if they are still infinite (i.e. significant) or not. This function is significantly faster than the bootstrap_persistence_thresholds function. Note that the 'calculate_homology' function does not seem to store infinite cycles (i.e. cycles that have death value equal to 'thresh').

Value

a list containing the full persistence diagram, the subsetted diagram, representatives and/or subsetted representatives if desired, the p-values of subsetted features and the Bonferroni p-value thresholds in each dimension if desired.

Author(s)

Shael Brown - <shaelebrown@gmail.com>

References

Bobrowski O, Skraba P (2023). "A universal null-distribution for topological data analysis." https://www.nature.com/articles/s41598-023-37842-2.

Examples

```
if(require("TDA"))
{
  # create dataset
  theta <- runif(n = 100,min = 0,max = 2*pi)
  x <- cos(theta)
  y <- sin(theta)
  circ <- data.frame(x = x,y = y)

  # add noise
  x_noise <- -0.1 + 0.2*stats::runif(n = 100)
  y_noise <- -0.1 + 0.2*stats::runif(n = 100)
  circ$x <- circ$x + x_noise
  circ$y <- circ$y + y_noise

  # determine significant topological features
  library(TDA)
 res <- universal_null(circ, thresh = 2,alpha = 0.1,return_pvals = TRUE,FUN_diag = "ripsDiag")
  res$subsetted_diag
  res$pvals
```

```
  res$alpha_thresh

  # at a lower threshold we can check for
  # infinite cycles
  res2 <- universal_null(circ, thresh = 1.1,
                         infinite_cycle_inference = TRUE,
                         alpha = 0.1,
                         FUN_diag = "ripsDiag")
  res2$subsetted_diag
}
```

---

vr_graphs *Compute Vietoris-Rips graphs of a dataset at particular epsilon radius values.*

Description

Persistence diagrams computed from Rips-Vietoris filtrations contain information about distance radius scales at which topological features of a dataset exist, but the features can be challenging to visualize, analyze and interpret. In order to help solve this problem the ‘vr_graphs‘ function computes the 1-skeleton (i.e. graph) of Rips complexes at particular radii, called "Vietoris-Rips graphs" (VR graphs) in the literature.

Usage

```
vr_graphs(X, distance_mat = FALSE, eps, return_clusters = TRUE)
```

Arguments

X	either a point cloud data frame/matrix, or a distance matrix.
distance_mat	a boolean representing if the input ‘X‘ is a distance matrix, default value is ‘FALSE‘.
eps	a numeric vector of the positive scales at which to compute the Rips-Vietoris complexes, i.e. all edges at most the specified values.
return_clusters	a boolean determining if the connected components (i.e. data clusters) of the complex should be explicitly returned, default is ‘TRUE‘.

Details

This function may be used in conjunction with the igraph package to visualize the graphs (see plot_vr_graph).

Value

A list with a ‘vertices‘ field, containing the rownames of ‘X‘, and then a list ‘graphs‘ one (named) entry for each value in ‘eps‘. Each entry is a list with a ‘graph‘ field, storing the (undirected) edges in the Rips-Vietoris complex in matrix format, and a ‘clusters‘ field, containing vectors of the data indices (or row names) in each connected component of the Rips graph.

Author(s)

Shael Brown - <shaelebrown@gmail.com>

References

A Zomorodian, The tidy set: A minimal simplicial set for computing homology of clique complexes in Proceedings of the Twenty-Sixth Annual Symposium on Computational Geometry, SoCG '10. (Association for Computing Machinery, New York, NY, USA), p. 257–266 (2010).

See Also

plot_vr_graph for plotting VR graphs.

Examples

```
if(require("TDAstats") & require("igraph"))
{
  # simulate data from the unit circle and calculate
  # its diagram
  df <- TDAstats::circle2d[sample(1:100,25),]
  diag <- TDAstats::calculate_homology(df,
                                       dim = 1,
                                       threshold = 2)

  # get minimum death radius of any data cluster
  min_death_H0 <-
  min(diag[which(diag[,1] == 0),3L])

  # get birth and death radius of the loop
  loop_birth <- as.numeric(diag[nrow(diag),2L])
  loop_death <- as.numeric(diag[nrow(diag),3L])

  # compute VR graphs at radii half of
  # min_death_H0 and the mean of loop_birth and
  # loop_death, returning clusters
  graphs <- vr_graphs(X = df,eps =
  c(0.5*min_death_H0,(loop_birth + loop_death)/2))

  # verify that there are 25 clusters for the smaller radius
  length(graphs$graphs[[1]]$clusters)

}
```

Index