Package ‘DatabionicSwarm’

October 13, 2023

**Type** Package  
**License** GPL-3  
**Title** Swarm Intelligence for Self-Organized Clustering  
**Version** 1.2.1  
**Date** 2023-10-12  
**Maintainer** Michael Thrun <m.thrun@gmx.net>

**Description**

Algorithms implementing populations of agents that interact with one another and sense their environment may exhibit emergent behavior such as self-organization and swarm intelligence. Here, a swarm system called Databionic swarm (DBS) is introduced which was published in Thrun, M.C., Ultsch A.: ``Swarm Intelligence for Self-Organized Clustering'' (2020), Artificial Intelligence, <DOI:10.1016/j.artint.2020.103237>. DBS is able to adapt itself to structures of high-dimensional data such as natural clusters characterized by distance and/or density based structures in the data space. The first module is the parameter-free projection method called Pswarm (Pswarm()), which exploits the concepts of self-organization and emergence, game theory, swarm intelligence and symmetry considerations. The second module is the parameter-free high-dimensional data visualization technique, which generates projected points on the topographic map with hypsometric tints defined by the generalized U-matrix (GeneratePswarmVisualization()). The third module is the clustering method itself with non-critical parameters (DBSclustering()). Clustering can be verified by the visualization and vice versa. The term DBS refers to the method as a whole. It enables even a non-professional in the field of data mining to apply its algorithms for visualization and/or clustering to data sets with completely different structures drawn from diverse research fields. The comparison to common projection methods can be found in the book of Thrun, M.C.: "Projection Based Clustering through Self-Organization and Swarm Intelligence" (2018) <DOI:10.1007/978-3-658-20540-9>.

**Imports** Rcpp (>= 1.0.8), RcppParallel (>= 5.1.4), deldir, GeneralizedUmatrix

**Suggests** DataVisualizations, knitr (>= 1.12), rmarkdown (>= 0.9), plotrix, geometry, sp, spdep, parallel, rgl, png, ProjectionBasedClustering, parallelDist, pracma, dendextend

**LinkingTo** Rcpp, RcppArmadillo, RcppParallel

**Depends** R (>= 3.0)
R topics documented:

- DatabionicSwarm-package ........................................ 3
- DBSclustering .................................................. 6
- DefaultColorSequence ......................................... 9
- Delaunay4Points ................................................ 9
- Delta3DWeightsC ............................................... 10
- DijkstraSSSP .................................................. 11
- findPossiblePositionsCsingle ................................ 12
- GeneratePswarmVisualization .................................. 13
- getCartesianCoordinates ..................................... 15
- getUmatrix4Projection ....................................... 16
- Hepta ............................................................ 17
- Lsun3D ............................................................ 18
- plotSwarm ........................................................ 19
- ProjectedPoints2Grid ......................................... 19
- Pswarm ............................................................ 20
- PswarmCurrentRadiusC2botsPositive ......................... 22
- rDistanceToroidCsingle ..................................... 24
- RelativeDifference ........................................... 25
- sESOM4BMUs .................................................... 26
- setdiffMatrix ................................................... 27
- setGridSize ..................................................... 28
- setPolarGrid .................................................... 29
- setRmin .......................................................... 30
- ShortestGraphPathsC ......................................... 31
- trainstepC ....................................................... 32
- UniquePoints ................................................... 33

Index ................................................................. 35
Description

Algorithms implementing populations of agents that interact with one another and sense their environment may exhibit emergent behavior such as self-organization and swarm intelligence. Here, a swarm system called Databionic swarm (DBS) is introduced which was published in Thrun, M.C., Ultsch A.: "Swarm Intelligence for Self-Organized Clustering" (2020), Artificial Intelligence, <DOI:10.1016/j.artint.2020.103237>. DBS is able to adapt itself to structures of high-dimensional data such as natural clusters characterized by distance and/or density based structures in the data space. The first module is the parameter-free projection method called Pswarm (Pswarm()), which exploits the concepts of self-organization and emergence, game theory, swarm intelligence and symmetry considerations. The second module is the parameter-free high-dimensional data visualization technique, which generates projected points on the topographic map with hypsometric tints defined by the generalized U-matrix (GeneratePswarmVisualization()). The third module is the clustering method itself with non-critical parameters (DBSclustering()). Clustering can be verified by the visualization and vice versa. The term DBS refers to the method as a whole. It enables even a non-professional in the field of data mining to apply its algorithms for visualization and/or clustering to data sets with completely different structures drawn from diverse research fields. The comparison to common projection methods can be found in the book of Thrun, M.C.: "Projection Based Clustering through Self-Organization and Swarm Intelligence" (2018) <DOI:10.1007/978-3-658-20540-9>.

Details

For a brief introduction to DatabionicSwarm please see the vignette Short Intro to the Databionic Swarm (DBS). The license is CC BY-NC-SA 4.0.

Index of help topics:

- DBSclustering
- DatabionicSwarm-package
- DefaultColorSequence
- Delaunay4Points
- Delta3DWeightsC
- DijkstraSSSP
- GeneratePswarmVisualization
- Hepta
- Lsun3D
- ProjectedPoints2Grid
Pswarm A Swarm of Databots based on polar coordinates (Polar Swarm).
PswarmCurrentRadiusC2botsPositive intern function, do not use yourself
RelativeDifference Relative Difference
ShortestGraphPathsC Shortest GraphPaths = geodesic distances
UniquePoints Unique Points
findPossiblePositionsCsingle Intern function, do not use yourself
getCartesianCoordinates Intern function: Transformation of Databot indizes to coordinates
getUmatrix4Projection deprecated! see GeneralizedUmatrix()
plotSwarm Intern function for plotting during the Pswarm annealing process
rDistanceToroidCsingle Intern function for 'Pswarm'
sESOM4BMUs Intern function: Simplified Emergent Self-Organizing Map
setGridSize Sets the grid size for the Pswarm algorithm
setPolarGrid Intern function: Sets the polar grid
setRmin Intern function: Estimates the minimal radius for the Databot scent
setdiffMatrix setdiffMatrix shortens Matrix2Curt by those rows that are in both matrices.
trainstepC Internal function for sESOM

Note
For interactive Island Generation of a generalized Umatrix see interactiveGeneralizedUmatrixIsland function in the package ProjectionBasedClustering.
If you want to verify your clustering result externally, you can use Heatmap or SilhouettePlot of the CRAN package DataVisualizations.

Author(s)
Michal Thrun
Maintainer: Michael Thrun <m.thrun@gmx.net>

References


Successfully used in


Examples

data('LSun3D')

# 2d projection, without instant visualization of steps

# Alternative I:
# DistanceMatrix hast to be defined by the user.
InputDistances = as.matrix(dist(Lsun3D$Data))

projection = Pswarm(InputDistances)

# 2d projection, with instant visualization
## Not run:
# Alternative II: DataMatrix, Distance is Euclidean per default
projection=Pswarm(Lsun3D$Data, Cls=Lsun3D$Cls, PlotIt=T)

## End(Not run)
#
## Computation of Generalized Umatrix
# If Non Euclidean Distances are used, Please Use `code{MDS}`
# from the ProjectionBasedClustering package with the correct OutputDimension
# to generate a new DataMatrix from the distances (see SheppardDiagram
# or KruskalStress)
genUmatrixList=GeneratePswarmVisualization(Data = Lsun3D$Data,
projection$ProjectedPoints, projection$LC)

## Visualizuation of GeneralizedUmatrix,
## Estimation of the Number of Clusters=Number of valleys
library(GeneralizedUmatrix)#install if not installed
GeneralizedUmatrix::plotTopographicMap(genUmatrixList$Umatrix, genUmatrixList$Bestmatches)

## Automatic Clustering
# number of Cluster from dendrogram (PlotIt=TRUE) or visualization
Cls=DBSclustering(k=3, Lsun3D$Data,
genUmatrixList$Bestmatches, genUmatrixList$LC, PlotIt=FALSE)
# Verification, often its better to mark Outliers manually
GeneralizedUmatrix::plotTopographicMap(genUmatrixList$Umatrix, genUmatrixList$Bestmatches, Cls)

## Not run:
# To generate the 3D landscape in the shape of an island
# from the toroidal topograpic map visualization
# you may cut your island interactively around high mountain ranges
Imx = ProjectionBasedClustering::interactiveGeneralizedUmatrixIsland(genUmatrixList$Umatrix,
genUmatrixList$Bestmatches, Cls)

GeneralizedUmatrix::plotTopographicMap(genUmatrixList$Umatrix, 
genUmatrixList$Bestmatches, Cls=Cls, Imx = Imx)

## End(Not run)
## Not run:
library(ProjectionBasedClustering)#install if not installed
Cls2=ProjectionBasedClustering::interactiveClustering(genUmatrixList$Umatrix, 
genUmatrixList$Bestmatches, Cls)

## End(Not run)

**DBSclustering**

*Databonic swarm clustering (DBS)*
DBS is a flexible and robust clustering framework that consists of three independent modules. The first module is the parameter-free projection method Pswarm, which exploits the concepts of self-organization and emergence, game theory, swarm intelligence and symmetry considerations [Thrun/Ultsch, 2021]. The second module is a parameter-free high-dimensional data visualization technique, which generates projected points on a topographic map with hypsometric colors called the generalized U-matrix. The third module is a clustering method with no sensitive parameters (see [Thrun, 2018, p. 104 ff]). The clustering can be verified by the visualization and vice versa. The term DBS refers to the method as a whole.

The function applies the automated Clustering approach of the Databonic swarm using abstract U distances, which are the geodesic distances based on high-dimensional distances combined with low dimensional graph paths by using ShortestGraphPathsC.

### Usage

```r
dbsclustering(k, DataOrDistance, BestMatches, LC, StructureType = TRUE,
PlotIt = FALSE, ylab, main, method = "euclidean",...)
```

### Arguments

- **k**: number of clusters, how many to you see in the topographic map (3D landscape)?
- **DataOrDistance**: Either [1:n,1:d] Matrix of Data (n cases, d dimensions) that will be used. One DataPoint per row or symmetric Distance matrix [1:n,1:n]
- **BestMatches**: [1:n,1:2] Matrix with positions of Bestmatches or ProjectedPoints, one matrix line per data point
- **LC**: grid size c(Lines,Columns), please see details
- **StructureType**: Optional, bool; = TRUE: compact structure of clusters assumed, =FALSE: connected structure of clusters assumed. For the two options for Clusters, see [Thrun, 2018] or Handl et al. 2006
- **PlotIt**: Optional, bool, Plots Dendrogramm
- **ylab**: Optional, character vector, ylabel of dendrogramm
- **main**: Optional, character vector, title of dendrogramm
- **method**: Optional, one of 39 distance methods of `parDist` of package parallelDist, if Data matrix is chosen above
- **...**: Further arguments passed on to the `parDist` function, e.g. user-defined distance functions

### Details

The input of the LC parameter depends on the choice of Bestmatches input argument. Usually as the name of the argument states, the Bestmatches of the `GeneratePswarmVisualization` function are used which is define in the notation of self-organizing map. In this case please see example one.

However, as written above, clustering and visualization can be applied independently of each other. In this case the places of Lines L and Columns C are switched because Lines is a value slightly
above the maximum of the x-coordinates and Columns is a value slightly above the maximum of the y-coordinates of ProjectedPoint. Hence, one should give `DBSclustering` the argument LC[2,1] as shown in example 2.

Often it is better to mark the outliers manually after the process of clustering and sometimes a clustering can be improved through human interaction [Thrun/Ultsch,2017] <DOI:10.13140/RG.2.2.13124.53124>; use in this case the visualization `plotTopographicMap` of the package GeneralizedUmatrix. If you would like to mark the outliers interactively in the visualization use the `ProjectionBasedClustering` package with the function `interactiveClustering()`, or for full interactive clustering `IPBC()`. The package is available on CRAN. An example is shown in case of `interactiveClustering()` function in the third example.

**Value**

[1:n] numerical vector of numbers defining the classification as the main output of this cluster analysis for the n cases of data corresponding to the n bestmatches. It has k unique numbers representing the arbitrary labels of the clustering. You can use `plotTopographicMap(Umatrix,Bestmatches,Cls)` for verification.

**Note**

If you want to verify your clustering result externally, you can use `Heatmap` or `SilhouettePlot` of the package `DataVisualizations` available on CRAN.

**Author(s)**

Michael Thrun

**References**


**Examples**

data("Lsun3D")
Data=Lsun3D$Data
InputDistances=as.matrix(dist(Data))
projection=Pswarm(InputDistances)
## Example One
genUmatrixList=GeneratePswarmVisualization(Data,
projection$ProjectedPoints,projection$LC)
Cls=DBSclustering(k=3, Data,
genUmatrixList$Bestmatches, genUmatrixList$LC,PlotIt=TRUE)

## Example Two
#automatic Clustering without GeneralizedUmatrix visualization
Cls=DBSclustering(k=3, Data,
projection$ProjectedPoints, projection$LC[c(2,1)],PlotIt=TRUE)

## Not run:
## Example Three
## Sometimes an automatic Clustering can be improved
## through an interactive approach,
## e.g. if Outliers exist (see [Thrun/Ultsch, 2017])
library(ProjectionBasedClustering)
Cls2=ProjectionBasedClustering::interactiveClustering(genUmatrixList$Umatrix,
genUmatrixList$Bestmatches, Cls)

## End(Not run)

---

**DefaultColorSequence**

*Default color sequence for plots*

**Description**

Defines the default color sequence for plots made within the Projections package.

**Usage**

```r
data("DefaultColorSequence")
```

**Format**

A vector with 562 different strings describing colors for plots.

---

**Delaunay4Points**

*Adjacency matrix of the delaunay graph for BestMatches of Points*

**Description**

Calculates the adjacency matrix of the delaunay graph for BestMatches (BMs) in tiled form if BestMatches are located on a toroid grid.

**Usage**

```r
Delaunay4Points(Points, IsToroid = TRUE, LC, PlotIt = FALSE, Gabriel = FALSE)
```
## Arguments

**Points**

[1:n,1:3] matrix containing the BMKey, X and Y coordinates of the n, Best-Matches NEED NOT to be UNIQUE, however, there is an edge in the Deaunay between duplicate points!

**IsToroid**

Optional, logical, indicating if BM’s are on a toroid grid. Default is True

**LC**

Optional, A vector of length 2, containing the number of lines and columns of the Grid. Lines is a value slightly above the maximum of the x-coordinates and Columns is a value slightly above the maximum of the y-coordinates of Points.

**PlotIt**

Optional, bool, Plots the graph

**Gabriel**

Optional, bool, default: FALSE, If TRUE: calculates the gabriel graph instead of the delaunay graph

## Value

Delaunay[1:n,1:n] adjacency matrix of the Delaunay-Graph

## Author(s)

Michael Thrun

## References


---

**Delta3DWeightsC**

*Intern function*

## Description

Implementation of the main equation for SOM, ESOM or the sESOM algorithms

## Usage

Delta3DWeightsC(vx,Datasample)

## Arguments

**vx**

array of weights [1:Lines,1:Columns,1:Weights]

**Datasample**

NumericVector of one Datapoint[1:n]

## Details

Intern function in case of ComputeInR==FALSE in GeneratePswarmVisualization, see chapter 5.3 of [Thrun, 2018] for generalized Umatrix and especially the sESOM4BMUs algorithm.
DijkstraSSSP

Value
modified array of weights [1:Lines,1:Columns,1:]

Author(s)
Michael Thrun

References

DijkstraSSSP  
Internal function: Dijkstra SSSP

Description
Dijkstra’s SSSP (Single source shortest path) algorithm:
gets the shortest path (geodesic distance) from source vertice(point) to all other vertices(points)
defined by the edges of the adjacency matrix

Usage
DijkstraSSSP(Adj, Costs, source)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adj</td>
<td>[1:n,1:n] 0/1 adjacency matrix, e.g. from delaunay graph or gabriel graph</td>
</tr>
<tr>
<td>Costs</td>
<td>[1:n,1:n] matrix, distances between n points (normally euclidean)</td>
</tr>
<tr>
<td>source</td>
<td>integer vertice(point) from which to calculate the geodesic distance to all other points</td>
</tr>
</tbody>
</table>

Details
Preallocating space for DataStructures accordingly to the maximum possible number of vertices which is fixed set at the number 10001. This is an internal function of ShortestGraphPathsC, no errors or mis-usage is caught here.

Value
ShortestPaths[1:n] vector, shortest paths (geodesic) to all other vertices including the source vertice itself

Note
runs in O(E*Log(V))
findPossiblePositionsCsingle

Intern function, do not use yourself

Description
Finds all possible jumping position regarding a grid and a Radius for DataBots

Usage
findPossiblePositionsCsingle(RadiusPositionsschablone, jumplength, alpha, Lines)

Arguments
RadiusPositionsschablone NumericMatrix, see setPolarGrid
jumplength double radius of DataBots regarding neighborhood, they can jump to
alpha double, zu streichen
Lines double, jumpinglength has to smaller than Lines/2 and Lines/2 has to yield to an integer number.

Details
Algorithm is described in [Thrun, 2018, p. 95, Listing 8.1].

Value
OpenPositions NumericMatrix, indices of open positions

Author(s)
Michael Thrun

References
Generates the Umatrix for Pswarm algorithm

Description

DBS is a flexible and robust clustering framework that consists of three independent modules. The first module is the parameter-free projection method Pswarm, which exploits the concepts of self-organization and emergence, game theory, swarm intelligence and symmetry considerations. The second module is a parameter-free high-dimensional data visualization technique, which generates projected points on a topographic map with hypsometric colors, called the generalized U-matrix. The third module is a clustering method with no sensitive parameters. DBS clustering. The clustering can be verified by the visualization and vice versa. The term DBS refers to the method as a whole.

GeneratePswarmVisualization function generates the special case (please see [Thrun, 2018]) of the generalized U-matrix with the help of an unsupervised neural network (simplified emergent self-organizing map published in [Thrun/Ultsch, 2020]). From the generalized U-matrix a topographic map with hypsometric tints can be visualized. To see this visualization use plotTopographicMap of the package GeneralizedUmatrix.

Usage

GeneratePswarmVisualization(Data, ProjectedPoints, LC, PlotIt=FALSE, ComputeInR=FALSE, Parallel=TRUE)

Arguments

Data [1:n,1:d] array of data: n cases in rows, d variables in columns
ProjectedPoints matrix, ProjectedPoints[1:n,1:2] n by 2 matrix containing coordinates of the Projection: A matrix of the fitted configuration. See output of Pswarm for further details
LC size of the grid c(Lines,Columns), number of Lines and Columns automatic calculated by setGridSize in Pswarm Sometimes is better to choose a different grid size, e.g. to reduce computational effort contrary to SOM, here the grid size defined only the resolution of the visualizations. The real grid size is predefined by Pswarm, but you may choose a factor x*res$LC if you so desire. Therefore, The resulting grid size is given back in the Output.
PlotIt Optional, default(FALSE), If TRUE than uses plotTopographicMap of the package GeneralizedUmatrix is plotted as a topview in the tiled option, see details for explanation.
ComputeInR Optional, =TRUE: Rcode, =FALSE C++ implementation
Parallel Optional, =TRUE: Parallel C++ implementation, =FALSE C++ implementation
GeneratePswarmVisualization

Details

Tiled: The topographic map is visualized 4 times because the projection is toroidal. The reason is that there are no border in the visualizations and clusters (if they exist) are not disrupted by borders of the plot.

If you used Pswarm with distance matrix instead of a data matrix (in the sense that you do not have any data matrix available), you may transform your distances into data by using MDS of the ProjectionBasedClustering package in order to use the GeneratePswarmVisualization function. The correct dimension can be found through the Sheppard diagram or kruskals stress.

Value

list of

Bestmatches Numeric matrix [1:n,1:2], BestMatches of the Umatrix, contrary to ESOM they are always fixed, because predefined by GridPoints.

Umatrix Numeric matrix [1:Lines,1:Columns],

WeightsOfNeurons Numeric 3D array [1:Lines,1:Columns,1:d], d is the dimension of the weights, the same as in the ESOM algorithm

GridPoints Integer matrix [1:n,1:2], quantized projected points: projected points now lie on a predefined grid.

LC c(Lines,Columns), normally equal to grid size of Pswarm, sometimes it a better or a lower resolution for the visualization is better. Therefore here the grid size of the neurons is given back.

PlotlyHandle If PlotIt=FALSE: NULL, otherwise plotly object for ploting topview of topographic map.

Note

If you used pswarm with distance matrix instead of a data matrix you can mds transform your distances into data (see the MDS function of the ProjectionBasedClustering package.). The correct dimension can be found through the Sheppard diagram or kruskals stress.

Note

The extraction of an island out of the generalized Umatrix can be performed using the interactiveGeneralizedUmatrixIsland function in the package ProjectionBasedClustering.

The main code of both functions GeneralizedUmatrix and GeneratePswarmVisualization is the same C++ function sESOM4BMUs which is described in [Thrun/Ultsch, 2020].

Author(s)

Michael Thrun
getCartesianCoordinates

Intern function: Transformation of Databot indizes to coordinates

Description

Transforms Databot indizes to exact cartesian coordinates on an toroid two dimensional grid.

Arguments

- **DataBotsPos**: [1:N] complex vector Two Indizes per Databot describing its positions in an two dimensional grid
- **GridRadius**: [Lines,Columns] Radii Matrix of all possible Positions of DataBots in Grid, see also documentation of `setPolarGrid`
- **GridAngle**: [Lines,Columns] Angle Matrix of all possible Positions of DataBots in Grid, see also documentation of `setPolarGrid`
- **Lines**: Defines Size of planar toroid two dimensional grid

Examples

```r
data("Lsun3D")
Data=Lsun3D$Data
Cls=Lsun3D$Cls
InputDistances=as.matrix(dist(Data))

projList=Pswarm(InputDistances)
genUmatrixList=GeneratePswarmVisualization(Data,
  projList$ProjectedPoints,projList$LC,
  Parallel=FALSE)#CRAN guidelines do not allow =TRUE for testing
library(GeneralizedUmatrix)
plotTopographicMap(genUmatrixList$Umatrix,genUmatrixList$Bestmatches,Cls)
```
Columns  Defines Size of planar toroid two dimensional grid
QuadOrHexa Optional, FALSE=If DataPos on hexadiagonal grid, round to 2 decimals after value, Default=TRUE

Details
Transformation is described in [Thrun, 2018, p. 93].

Value
BestMatchingUnits[1:N,2] coordinates on an two dimensional grid for each databot excluding unique key, such that by using GeneratePswarmVisualization a visualization of the Pswarm projection is possible

Author(s)
Michael Thrun

References

getUmatrix4Projection  depricated! see GeneralizedUmatrix() Generalisierte U-Matrix fuer Projektionsverfahren

description

deprecated! see GeneralizedUmatrix()

Usage
getUmatrix4Projection(Data,ProjectedPoints,
PlotIt=TRUE,Cls=FALSE,toroid=T,Tiled=F,ComputeInR=F)

Arguments
Data [1:n,1:d] Numeric matrix: n cases in rows, d variables in columns
ProjectedPoints [1:n,2]n by 2 matrix containing coordinates of the Projection: A matrix of the fitted configuration.
PlotIt Optional, bool, default=FALSE, if =TRUE: U-Matrix of every current Position of Databots will be shown
Cls Optional, For plotting, see plotUmatrix in package Umatrix
toroid  Optional, Default=FALSE, ==FALSE planar computation ==TRUE: toroid borderless computation, set so only if projection method is also toroidal
Tiled  Optional, For plotting see plotUmatrix in package Umatrix
ComputeInR  Optional, =T: Rcode, =F Cpp Code

Value

List with

Umatrix  [1:Lines,1:Columns] (see ReadUMX in package DataIO)
EsomNeurons  [Lines,Columns,weights] 3-dimensional numeric array (wide format), not wts (long format)
Bestmatches  [1:n,OutputDimension] GridConverted Projected Points information converted by convertProjectionProjectedPoints() to predefined Grid by Lines and Columns
gplotres  Ausgabe von ggplot
unbesetztePositionen  Umatrix[unbesetztePositionen] = NA

Author(s)

Michael Thrun

References


Examples

data("Lsun3D")
Data=Lsun3D$Data
Cls=Lsun3D$Cls
InputDistances=as.matrix(dist(Data))
res=cmdscale(d=InputDistances, k = 2, eig = TRUE, add = FALSE, x.ret = FALSE)
ProjectedPoints=as.matrix(res$points)
# Stress = KruskalStress(InputDistances, as.matrix(dist(ProjectedPoints)))
# resUmatrix=GeneralizedUmatrix(Data,ProjectedPoints)
# plotTopographicMap(resUmatrix$Umatrix,resUmatrix$Bestmatches,Cls)

Hepta

Hepta is part of the Fundamental Clustering Problem Suit (FCPS) [Thrun/Ultsch, 2020].

Description

clearly defined clusters, different variances
Usage

data("Hepta")

Details

Size 212, Dimensions 3, stored in Hepta$Data
Classes 7, stored in Hepta$Cls

References


Examples

data(Hepta)
str(Hepta)

Lsun3D

Lsun3D is part of the Fundamental Clustering Problem Suit (FCPS)
[Thrun/Ultsch, 2020].

Description

clearly defined clusters, different variances

Usage

data("Lsun3D")

Details

Size 404, Dimensions 3
Dataset defined discontinuites, where the clusters have different variances. Three main Clusters, and four Outliers (in Cluster 4). See for a more detailed description in [Thrun, 2018].

References


Examples

data(Lsun3D)
str(Lsun3D)
Cls=Lsun3D$Cls
Data=Lsun3D$Data
plotSwarm

Intern function for plotting during the Pswarm annealing process

Description

Intern function, generates a scatter plot of the progress of the Pswarm algorithm after every nash equilibrium. Every point symbolizes a Databot. If a prior classification is given (Cls) then the Databots have the colors defined by the class labels.

Usage

plotSwarm(Points,Cls,xlab,ylab,main)

Arguments

Points: ProjectedPoints or DataBot positions in cartesian coordinates
Cls: optional, Classification as a numeric vector, if given
xlab = 'X', optional, string
ylab = 'Y', optional, string
main = "DataBots", optional, string

Author(s)

Michael Thrun

See Also

Pswarm with PlotIt=TRUE

ProjectedPoints2Grid

Transforms ProjectedPoints to a grid

Description

quantized xy cartesian coordinates of ProjectedPoints

Usage

ProjectedPoints2Grid(ProjectedPoints, Lines, Columns,PlotIt=FALSE, Cls)
Arguments

ProjectedPoints
[1:n,1:2] numeric matrix of cartesian xy coordinates

Lines
double, length of small side of the rectangular grid

Columns
double, length of big side of the rectangular grid

PlotIt
optional, bool, shows the result if TRUE

Cls
[1:n] numeric vector of classes for each projected point

Details

intern function, described in [Thrun, 2018, p.47]

Value

BestMatches[1:n,1:3] columns in order: Key,Lines,Columns

Author(s)

Michael Thrun

References


See Also

GeneratePswarmVisualization

Description

This projection method is a part of the databionic swarm which uses the nash equilibrium [Thrun/Ultsch, 2021]. Using polar coordinates for agents (here Databots) in two dimensions has many advantages, for further details see [Thrun, 2018] and [Thrun/Ultsch, 2021].

Usage

Pswarm(DataOrDistance, PlotIt=FALSE, Cls=NULL, Silent=TRUE, Debug=FALSE, LC=c(NULL, NULL), method= "euclidean", Parallel=FALSE,...)
**Arguments**

**DataOrDistance**
- Numeric matrix [1:n,1:n]: symmetric matrix of dissimilarities, if variable unsymmetric (Numeric matrix [1:d,1:n]) it is assumed as a dataset and the euclidean distances are calculated of d variables and n cases.

**PlotIt**
- Optional, bool, default=FALSE, If =TRUE, Plots the projection during the computation process after every Nash equilibrium.

**Cls**
- Optional, numeric vector [1:n], given Classification in numbers, only for plotting if PlotIt=TRUE, irrelevant for computations.

**Silent**
- Optional, bool, default=FALSE, If =TRUE results in various console messages

**Debug**
- Optional, Debug, default=FALSE, =TRUE results in various console messages, deprecated for CRAN, because cout is not allowed.

**LC**
- Optional, grid size c(Lines, Columns), sometimes it is better to call `setGridSize` separately.

**method**
- Optional, one of 39 distance methods of `parDist` of package parallelDist, if Data matrix is chosen above

**Parallel**
- Optional, =TRUE: Parallel C++ implementation, =FALSE C++ implementation

Further arguments passed on to the `parDist` function, e.g. user-defined distance functions

**Details**

DBS is a flexible and robust clustering framework that consists of three independent modules. The first module is the parameter-free projection method Pswarm, which exploits the concepts of self-organization and emergence, game theory, swarm intelligence and symmetry considerations.

The second module is a parameter-free high-dimensional data visualization technique, which generates projected points on a topographic map with hypsometric colors `GeneratePswarmVisualization`, called the generalized U-matrix. The third module is a clustering method with no sensitive parameters `DBSclustering`. The clustering can be verified by the visualization and vice versa. The term DBS refers to the method as a whole.

**Value**

List with

**ProjectedPoints**
- [1:n,1:2] xy cartesian coordinates of projection

**LC**
- number of Lines and Columns in c(Lines,Columns). Lines is a value slightly above the maximum of the x-coordinates and Columns is a value slightly above the maximum of the y-coordinates of ProjectedPoints

**Control**
- List, only for intern debugging

**Note**

LC is now automatically estimated; LC is the size of the grid c(Lines,Columns), number of Lines and Columns, default c(NULL,NULL) and automatic calculation by `setGridSize`
Author(s)

Michael Thrun

References


Examples

data("Lsun3D")
Data=Lsun3D$Data
Cls=Lsun3D$Cls
InputDistances=as.matrix(dist(Data))
#If not called separately setGridSize() is called in Pswarm
LC=setGridSize(InputDistances)
res=Pswarm(InputDistances,LC=LC,Cls=Cls,PlotIt=TRUE)

PswarmCurrentRadiusC2botsPositive

*intern function, do not use yourself*

Description

Finds the weak Nash equilibrium for DataBots in one epoch(Radius), requires the setting of constants, grid, and so on in Pswarm

Usage

PswarmCurrentRadiusC2botsPositive( AllDataBotsPosOld, Radius, DataDists, IndPossibleDBPosR, RadiusPositionsschablone, pp, Nullpunkt, Lines, Columns, nBots, limit, steigungsverlaufind, StressConstAditiv, debug)

Arguments

AllDataBotsPosOld
  ComplexVector [1:n,1], DataBots position in the last Nash-Equilibrium
Radius
double, Radius of payoff function, neighborhood, where other DataBots can be smelled
DataDists
  NumericMatrix, Inputdistances[1:n,1:n]
IndPossibleDBPosR
  ComplexVector, see output of findPossiblePositionsCsingle
RadiusPositionsschablone
   NumericMatrix, see AllallowedDBPosR0 in setPolarGrid

pp
   NumericVector, number of jumping simultaneously DataBots of one epoch (per
   nash-equilibirum), this vector is linearly monotonically decreasing

Nullpunkt
   NumericVector, equals which(AllallowedDBPosR0==0,arr.ind=T), see see
   AllallowedDBPosR0 in setPolarGrid

Lines
   double, small edge length of rectangulare grid

Columns
   double, big edge length of rectangulare grid

nBots
   double, intern constant, equals round(pp[Radius]*DBAnzahl)

limit
   int, intern constant, equals ceiling(1/pp[Radius])

steigungsverlaufind
   int, intern constant

StressConstAdditiv
   double, intern constant, sum of payoff of all databots in random condition before
   the algorithm starts

debug
   optional, bool: If TRUE prints status every 100 iterations

Details

   Algorithm is described in [Thrun, 2018, p. 95, Listing 8.1].

Value

   list of AllDataBotsPos ComplexVector, indizes of DataBot Positions after a weak Nash equilibrium is
   found

stressverlauf
   NumericVector, intern result, for debugging only

fokussiertlaufind
   NumericVector, intern result, for debugging only

Author(s)

   Michael Thrun

References

   [Thrun, 2018] Thrun, M. C.: Projection Based Clustering through Self-Organization and Swarm In-
   9783658205409, 2018.
rDistanceToroidCsingle

*Internal function for Pswarm*

**Description**

toroid distance calculation

**Usage**

```r
rDistanceToroidCsingle(AllDataBotsPosX, AllDataBotsPosY, AllallowedDBPosR0, Lines, Columns, Nullpunkt)
```

**Arguments**

- `AllDataBotsPosX`: NumericVector [1:n,1], positions of on grid
- `AllDataBotsPosY`: NumericVector [1:n,1], positions of on grid
- `AllallowedDBPosR0`: NumericMatrix
- `Lines`: double
- `Columns`: double
- `Nullpunkt`: NumericVector

**Details**

Part of the algorithm described in [Thrun, 2018, p. 95, Listing 8.1].

**Value**

numeric matrix of toroid Distances[1:n,1:n]

**Note**

do not use yourself

**Author(s)**

Michael Thrun

**References**

RelativeDifference

See Also

Pswarm

RelativeDifference (Relative Difference)

Description

Calculates the difference between positive x and y values

Usage

RelativeDifference(X, Y, epsilon = 10^-10, na.rm=FALSE)

Arguments

X either a value or numerical vector of [1:n]
Y either a value or numerical vector of [1:n]
epsilon Optional, If both x and y are approximatly zero the output is also zero
na.rm Optional, function does not work with non finite values. If these cases should be automatically removed, set parameter TRUE

Details

Contrary to other approaches in this cases the range of values lies between [-2,2]. The approach is only valid for positive values of X and Y. The relative difference \( R \) is defined with

\[
R = \frac{Y - X}{0.5 \times (X + Y)}
\]

Negative value indicate that \( X \) is higher than \( Y \) and positive values that \( X \) is lower than \( Y \).

Value

R

Note

It can be combined with the GabrielClassificationError if a clear baseline is defined.

Author(s)

Michael Thrun

References

**sESOM4BMUs**

**Intern function: Simplified Emergent Self-Organizing Map**

**Description**

Intern function for the simplified ESOM (sESOM) algorithm for fixed BestMatchingUnits.

**Usage**

`sESOM4BMUs(BMUs, Data, esom, toroid, CurrentRadius, ComputeInR=FALSE, Parallel=TRUE)`

**Arguments**

- **BMUs** [1:Lines,1:Columns], BestMAtchingUnits generated by ProjectedPoints2Grid()
- **Data** [1:n,1:d] array of data: n cases in rows, d variables in columns
- **esom** [1:Lines,1:Columns,1:weights] array of NeuronWeights, see ListAsEsomNeurons()
- **toroid** TRUE/FALSE - topology of points
- **CurrentRadius** number between 1 to x
- **ComputeInR** =TRUE: Rcode, =FALSE: Cpp
- **Parallel** Optional, =TRUE: Parallel C++ implementation, =FALSE C++ implementation

**Details**

Algorithm is described in [Thrun, 2018, p. 48, Listing 5.1].

**Value**

- **esom** numeric array [1:Lines,1:Columns,1:d], d is the dimension of the weights, the same as in the ESOM algorithm. modified esomneurons regarding a predefined neighborhood defined by a radius

**Note**

Usually not for separated usage!
setdiffMatrix

**Author(s)**
Michael Thrun

**References**


**See Also**

GeneratePswarmVisualization

---

**setdiffMatrix**

`setdiffMatrix` shortens `Matrix2Curt` by those rows that are in both matrices.

**Description**

`setdiffMatrix` shortens `Matrix2Curt` by those rows that are in both matrices.

**Arguments**

- `Matrix2Curt` [n,k] matrix, which will be shortened by x rows
- `Matrix2compare` [m,k] matrix whose rows will be compared to those of `Matrix2Curt` x rows in `Matrix2compare` equal rows of `Matrix2Curt` (order of rows is irrelevant). Has the same number of columns as `Matrix2Curt`.

**Value**

`V$CurtedMatrix[n-x,k]` Shortened `Matrix2Curt`

**Author(s)**

CL,MT 12/2014
setGridSize

Sets the grid size for the Pswarm algorithm

Description

Automatically sets the size of the grid, formula see [Thrun, 2018, p. 93-94].

Usage

setGridSize(InputDistances,minp=0.01,maxp=0.99,alpha=4)

Arguments

InputDistances  [1:n,1:n] symmetric matrix of input distances

minp  default value: 0.01, see quantile, first value in the vector of probs estimates robust minimum of distances

maxp  default value: 0.99, see quantile, last value of the vector of probs estimates robust maximum of distances

alpha  Do not change! Intern parameter, Only if Java Version of Pswarm instead of C++ version is used.

Details

grid is set such that minimum and maximum distances can be shown on the grid

Value

LC=c(Lines,Columns) size of the grid for Pswarm

Author(s)

Michael Thrun, Florian Lerch

References


See Also

automatic choice of LC for Pswarm
setPolarGrid

Examples

data("Lsun3D")
Data=Lsun3D$Data
Cls=Lsun3D$Cls
InputDistances=as.matrix(dist(Data))
#If not called separately setGridSize() is called in Pswarm
LC=setGridSize(InputDistances)

setPolarGrid

Intern function: Sets the polar grid

Description

Sets a polar grid for a swarm in an rectangular shape

Usage

setPolarGrid(Lines,Columns,QuadOrHexa,PlotIt,global)

Arguments

- Lines: Integer, hast to be able to be divided by 2
- Columns: Integer, with Columns>=Lines
- QuadOrHexa: bool, default(TRUE) If False Hexagonal grid, default quad grid
- PlotIt: bool, default(FALSE)
- global: bool, default(TRUE), intern parameter, how shall the radii be calculated?

Details

Part of the Algorithm described in [Thrun, 2018, p. 95, Listing 8.1].

Value

- list of
  - GridRadii: matrix [1:Lines,1:Columns], Radii Matrix of all possible Positions of DataBots in Grid
  - GridAngle: matrix [1:Lines,1:Columns], Angle Matrix of all possible Positions of DataBots in Grid
  - AllallowedDBPosR0: matrix [1:Lines+1,1:Columns+1], Matrix of radii in polar coordinates respecting origin (0,0) of all allowed DataBots Positions in one jump
  - AllallowedDBPosPhi0: matrix [1:Lines+1,1:Columns+1], # V$AllallowedDBPosPhi0[Lines+1,Lines+1] Matrix of angle in polar coordinates respecting origin (0,0) of all allowed DataBots Positions in one jump
setRmin

Intern function: Estimates the minimal radius for the Databot scent

Description
estimates the minimal radius on apolar grid in the automated annealing process of Pswarm, details of how can be read in [Thrun, 2018, p. 97]

Arguments

| Lines     | x-value determining the size of the map, i.e. how many open places for DataBots will be available on the 2-dimensional grid BEW ARE: has to be able to be divided by 2 |
| Columns   | y-value determining the size of the map, i.e. how many open places for DataBots will be available on the 2-dimensional grid Columns>Lines |
| AllallowedDBPosR0 | [1:Lines+1,1:Lines+1]Matrix of radii in polar coordinates respecting origin (0,0) of all allowed DataBots Positions in one jump |
| p         | percent of gitterpositions, which should be considered |

Value
Rmin Minimum Radius

Author(s)
Michael Thrun

References
ShortestGraphPathsC

Shortest GraphPaths = geodesic distances

Description

Dijkstra's SSSP (Single source shortest path) algorithm, from all points to all points

Usage

ShortestGraphPathsC(Adj, Cost)

Arguments

Adj [1:n,1:n] 0/1 adjascency matrix, e.g. from delaunay graph or gabriel graph.
Cost [1:n,1:n] matrix, distances between n points (normally euclidean)

Details

Vertices are the points, edges have the costs defined by weights (normally a distance). The algorithm runs in O(n*E*Log(V)), see also [Jungnickel, 2013, p. 87]. Further details can be found in [Jungnickel, 2013, p. 83-87] and [Thrun, 2018, p. 12].

Value

ShortestPaths[1:n,1:n] vector, shortest paths (geodesic) to all other vertices including the source vertice itself from al vertices to all vertices, stored as a matrix

Note

require C++11 standard (set flag in Compiler, if not set automatically)

Author(s)

Michael Thrun

References


trainstepC

*Internal function for sESOM*

**Description**

Does the training for fixed bestmatches in one epoch of the sESOM algorithm (see [Thrun, 2018] for details).

**Usage**

\[ \text{trainstepC}(vx, vy, \text{DataSampled}, \text{BMUsampled}, \text{Lines}, \text{Columns}, \text{Radius}, \text{toroid}) \]

**Arguments**

- **vx**: array [1:Lines,1:Columns,1:Weights], WeightVectors that will be trained, internally transformed von NumericVector to cube
- **vy**: array [1:Lines,1:Columns,1:2], meshgrid for output distance computation
- **DataSampled**: NumericMatrix, n cases shuffled Dataset[1:n,1:d] by sample
- **BMUsampled**: NumericMatrix, n cases shuffled BestMatches[1:n,1:2] by sample in the same way as DataSampled
- **Lines**: double, Height of the grid
- **Columns**: double, Width of the grid
- **Radius**: double, The current Radius that should be used to define neighbours to the bm
- **toroid**: bool, Should the grid be considered with cyclically connected borders?

**Details**

Algorithm is described in [Thrun, 2018, p. 48, Listing 5.1].

**Value**

WeightVectors, array[1:Lines,1:Columns,1:weights] with the adjusted Weights

**Note**

Usually not for seperated usage!

**Author(s)**

Michael Thrun
Unique Points

Reference


Description

return only the unique points in Datapoints

Usage

UniquePoints(Datapoints, Cls, Eps=1e-10)

Arguments

Datapoints [1:n,1:d] numeric matrix of Datapoints points of dimension d, the points are in the rows
Cls [1:n] numeric vector of classes for each datapoint.
Eps Optional, scalar above zero that defines minimum non-identical euclidean distance between two points

Details

Euclidean distance is computed and used within. Setting Eps to a very small number results in the identification of unique data points. Setting epsilon to a higher number results in the definition of mesh points within an d-dimensional R-ball graph.

Value

List with
Unique [1:k,1:d] Datapoints points without duplicate points
UniqueInd [1:k] index vector such that Unique == Datapoints[UniqueInd,], it has k non-consecutive numbers or labels, each label defines a row number within Datapoints[1:n,1:d] of a unique data point
Uniq2DatapointsInd [1:n] index vector. It has k unique index numbers representing the arbitrary labels. Each labels is mapped uniquely to a point in Unique. Logically in a way such that Datapoints == Unique[Uniq2DatapointsInd,] (will not work directly in R this way)
NewUniqueInd [1:k] index vector stating the index of the newly defined datastructure Unique.
NewUniq2DataIdx

[1:k] index vector such that Unique[NewUniq2DataIdx,] == Datapoints[Uniq2DatapointsInd,], it has n non-consecutive numbers or labels, each label defines a row number within Unique[1:k,1:d] of a unique data point

IsDuplicate

[1:n,1:n] matrix, for i!=j IsDuplicate[i,j]== 1 if Datapoints[i,] == Datapoints[j,]
IsDuplicate[i,i]==0

Eps

Numeric stating the neighborhood radius around unique points.

Author(s)

Michael Thrun

Examples

Datapoints = rbind(c(0,0), c(1,1), c(2,2))
Datapoints2 = rbind(Datapoints, Datapoints+0.001)
Datapoints3 = rbind(Datapoints2, c(1,1)-0.001)

Datapoints = rbind(c(0,0), c(0,0.015), c(0,0.01), c(0,0.015))

V1 = UniquePoints(Datapoints = Datapoints, Eps = 0.01)
V2 = UniquePoints(Datapoints = Datapoints2, Eps = 0.01)
V3 = UniquePoints(Datapoints = Datapoints3, Eps = 0.01)
Index

* DBS
  DatabionicSwarm-package, 3
  DBSclustering, 6
  Pswarm, 20
* DR
  DatabionicSwarm-package, 3
  GeneratePswarmVisualization, 13
  Pswarm, 20
* DataBionic
  DatabionicSwarm-package, 3
* Databonic swarm
  GeneratePswarmVisualization, 13
  Pswarm, 20
* Databionic
  DatabionicSwarm-package, 3
* Databonic swarm
  GeneratePswarmVisualization, 13
  Pswarm, 20
* Delaunay Graph
  Delaunay4Points, 9
* Delaunay
  Delaunay4Points, 9
* Dijkstra’s SSSP
  DijkstraSSSP, 11
* Dijkstra
  DijkstraSSSP, 11
* Dimensionality Reduction
  DatabionicSwarm-package, 3
* ESOM
  GeneratePswarmVisualization, 13
  sESOM4BMUs, 26
* FCPS
  Hepta, 17
  Lsun3D, 18
* GeneratePswarmVisualization
  ProjectedPoints2Grid, 19
  sESOM4BMUs, 26
* Hepta
  Hepta, 17
* Lsun3D
  Lsun3D, 18
* PSwarm
  Pswarm, 20
* Points
  Delaunay4Points, 9
* Polar Swarm
  Pswarm, 20
* Pswarm
  DatabionicSwarm-package, 3
  plotSwarm, 19
* RelativeDifference
  RelativeDifference, 25
* SOM
  GeneratePswarmVisualization, 13
* SSSP
  DijkstraSSSP, 11
* ShortestGraphPaths
  ShortestGraphPathsC, 31
* ShortestPaths
  ShortestGraphPathsC, 31
* Single source shortest path
  DijkstraSSSP, 11
* U-matrix
  GeneratePswarmVisualization, 13
* Umatrix
  GeneratePswarmVisualization, 13
* Voronoi
  Delaunay4Points, 9
* cluster analysis
  DatabionicSwarm-package, 3
  DBSclustering, 6
  GeneratePswarmVisualization, 13
  Pswarm, 20
* clustering
  DatabionicSwarm-package, 3
  DBSclustering, 6
* cluster
  DBSclustering, 6
* data points
UniquePoints, 33
* datasets
  Lsun3D, 18
* dataset
  Hepta, 17
* difference
  RelativeDifference, 25
* dimensionality reduction
  Pswarm, 20
* distances
  rDistanceToroidCsingle, 24
* emergence
  DatabionicSwarm-package, 3
* equilibrium
  DatabionicSwarm-package, 3
* game theory
  DatabionicSwarm-package, 3
* pswarm
  Pswarm, 20
* package
  DatabionicSwarm-package, 3
* points
  UniquePoints, 33
* polar
  PswarmCurrentRadiusC2botsPositive, 22
* positions
  findPossiblePositionsCsingle, 12
* projection method
  DatabionicSwarm-package, 3
  Pswarm, 20
* projection
  DatabionicSwarm-package, 3
* sESOM
  GeneratePswarmVisualization, 13
* self-organization
  DatabionicSwarm-package, 3
* self-organizing-map
  GeneratePswarmVisualization, 13
* swarm intelligence
  DatabionicSwarm-package, 3
  Pswarm, 20
* swarms
  findPossiblePositionsCsingle, 12
  Pswarm, 20
* swarm
  DatabionicSwarm-package, 3
  DBScustering, 6
  GeneratePswarmVisualization, 13
  plotSwarm, 19
  Pswarm, 20
  PswarmCurrentRadiusC2botsPositive, 22
  setPolarGrid, 29
* toroid
  rDistanceToroidCsingle, 24
* unique
  UniquePoints, 33
* visualization
  DatabionicSwarm-package, 3
  GeneratePswarmVisualization, 13
  Pswarm, 20

DatabionicSwarm
  (DatabionicSwarm-package), 3
DatabionicSwarm-package, 3
DBScustering, 6, 7, 8, 13, 21
DefaultColorSequence, 9
Delaunay4Points, 9
Delta3DWeightsC, 10
DijkstraSSSP, 11, 32
findPossiblePositionsCsingle, 12, 22
GabrielClassificationError, 26
GeneralizedUmatrix, 15
INDEX

GeneratePswarmVisualization, 7, 10, 13, 13, 14, 16, 20, 21, 27
getCartesianCoordinates, 15
getUmatrix4Projection, 16

Hepta, 17

Lsun3D, 18

plotSwarm, 19
plotTopographicMap, 8, 13, 15
ProjectedPoints2Grid, 19
Pswarm, 7, 13–15, 19, 20, 21, 22, 24, 25, 28, 30
pswarm (Pswarm), 20
pswarmCpp (Pswarm), 20
PswarmCurrentRadiusC2botsPositive, 22

quantile, 28

rDistanceToroidCsingle, 24
RelativeDifference, 25

sESOM4BMUs, 26
setdiffMatrix, 27
setGridSize, 13, 21, 28
setPolarGrid, 12, 13, 15, 23, 29
setRmin, 30
ShortestGraphPathsC, 11, 31

trainstepC, 32

UniquePoints, 33