Package ‘DatabionicSwarm’

January 12, 2021

Type Package

Title Swarm Intelligence for Self-Organized Clustering

Version 1.1.5

Date 2021-01-12

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>. A comparison to 26 common clustering algorithms on 15 datasets is presented on the website.

License GPL-3

Imports Rcpp, deldir, GeneralizedUmatrix

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

LinkingTo Rcpp, RcppArmadillo
**R topics documented:**

- DatabionicSwarm-package
- ClusteringAccuracy
- DBSclustering
- DefaultColorSequence
- Delaunay4Points
- DelaunayClassificationError
- Delta3DWeightsC
- DijkstraSSSP
- findPossiblePositionsCsingle
- GeneratePswarmVisualization
- getCartesianCoordinates
- getUmatrix4Projection
- Hepta
- Lsun3D
- plotSwarm
- ProjectedPoints2Grid
- Pswarm
- PswarmCurrentRadiusC2botsPositive
- rDistanceToroidCsingle
- RelativeDifference
- RobustNormalization
- RobustNorm_BackTrafo
- sESOM4BMUs
- setdiffMatrix
- setGridSize
- setPolarGrid
- setRmin
- ShortestGraphPathsC
- trainstepC
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>. A comparison to 26 common clustering algorithms on 15 datasets is presented on the website.

Details

For a brief introduction to DatabionicSwarm please see the vignette Short Intro to the Databionic Swarm (DBS).

<table>
<thead>
<tr>
<th>Package</th>
<th>Databionic swarm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Package</td>
</tr>
<tr>
<td>Version</td>
<td>1.1.5</td>
</tr>
<tr>
<td>Date</td>
<td>2021-01-12</td>
</tr>
<tr>
<td>License</td>
<td>CC BY-NC-SA 4.0</td>
</tr>
</tbody>
</table>

Index of help topics:

- ClusteringAccuracy
- DBSclustering
- DatabionicSwarm-package
- DefaultColorSequence
- Swarm Intelligence for Self-Organized Clustering
- Default color sequence for plots
Delaunay4Points  Adjacency matrix of the delaunay graph for BestMatches of Points
DelaunayClassificationError  Delaunay Classification Error (DCE)
Delta3DWeightsC  Intern function
DijkstraSSSP  Internal function: Dijkstra SSSP
GeneratePswarmVisualization  Generates the Umatrix for Pswarm algorithm
Hepta  Hepta is part of the Fundamental Clustering Problem Suit (FCPS) [Thrun/Ultsch, 2020].
Lsun3D  Lsun3D is part of the Fundamental Clustering Problem Suit (FCPS) [Thrun/Ultsch, 2020].
ProjectedPoints2Grid  Transforms ProjectedPoints to a grid
Pswarm  A Swarm of Databots based on polar coordinates (Polar Swarm).
PswarmCurrentRadiusC2botsPositive  Intern function, do not use yourself
RelativeDifference  Relative Difference
RobustNorm_BackTrafo  Transforms the Robust Normalization back
RobustNormalization  RobustNormalization
ShortestGraphPathsC  Shortest GraphPaths = geodesic distances
findPossiblePositionsCsingle  Intern function, do not use yourself
getCartesianCoordinates  Intern function: Transformation of Databot indizes to coordinates
getUmatrix4Projection  deprecicated! 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)

## Visualization of Generalized Umatrix,
## 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 topographic 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,
ClusteringAccuracy

    genUmatrixList$Bestmatches, Cls)

    ## End(Not run)

---

**Description**

ClusteringAccuracy

**Usage**

ClusteringAccuracy(PriorCls, CurrentCls, K=9)

**Arguments**

- PriorCls
- CurrentCls: clustering result
- K: Maximal number of classes for computation.

**Details**

Here, accuracy is defined as the normalized sum over all true positive labeled data points of a clustering algorithm. The best of all permutation of labels with the highest accuracy is selected in every trial because algorithms arbitrarily define the labels.

**Value**

Accuracy Between zero and one

**Author(s)**

Michael Thrun

**References**

Michael C. Thrun, Felix Pape, Alfred Ultsch: Benchmarking Cluster Analysis Methods in the Case of Distance and Density-based Structures Defined by a Prior Classification Using PDE-Optimized Violin Plots, ECDA, Potsdam, 2018
Examples

```r
data(Hepta)

InputDistances = as.matrix(dist(Hepta$Data))
projection = Pswarm(InputDistances)
visualization = GeneratePswarmVisualization(Data = Hepta$Data,
                                         projection$ProjectedPoints, projection$LC)

Cls = DBSclustering(k = 7, Hepta$Data, visualization$Bestmatches,
                     visualization$LC, PlotIt = FALSE)

ClusteringAccuracy(Hepta$Cls, Cls, K = 9)
```

---

### DBSclustering

**Databonic swarm clustering (DBS)**

#### 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 [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, DBSclustering (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 DBSclustering 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,
               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
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. 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

```r
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
## thorough 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

Description

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

Usage

Delaunay4Points(Points, IsToroid = TRUE, Grid=NULL, PlotIt=FALSE, Gabriel=FALSE)

Arguments

Points
[1:n,1:3] matrix containing the BMKey, X and Y coordinates of the n, BestMatches 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

Grid
Optional, A vector of length 2, containing the number of lines and columns of the Grid

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

DelaunayClassificationError

*Delaunay Classification Error (DCE)*

**Description**

DCE searches for the k-nearest neighbors of the first delaunay neighbors weighted by the Euclidean Distances of the Inputspace. DCE evaluates these neighbors in the Output space. A low value indicates a better two-dimensional projection of the high-dimensional Input space.

**Usage**

DelaunayClassificationError(Data,ProjectedPoints,Cls,LC=NULL)

**Arguments**

- **Data**: [1:n,1:d]
- **ProjectedPoints**: [1:n,1:2]
- **Cls**: [1:n,1]
- **LC**: Optional, default NULL, Wenn toroid, muss c(Lines,Columns) angeben werden

**Details**

Delaunay classification error (DCE) makes an unbiased evaluation of distance and density-based structure which may be even non-linear separable. First, DCE utilizes the information provided by a prior classification to assess projected structures. Second, DCE applies the insights drawn from graph theory. Details are described in [Thrun/Ultsch, 2018]

**Value**

- list of
  - DCE DelaunayClassificationError NOTE the rest is just for development purposes
  - DCEperPoint(1:n) unnormalized DCE of each point: DCE = mean(DCEperPoint)
  - nn the number of points in a relevant neighborhood: 0.5 * 85percentile(AnzNN)
  - AnzNN(1:n) the number of points with a delaunay graph neighborhood
  - NNdist(1:n,1:nn) the distances within the relevant neighborhood, 0 for inner cluster distances
  - HD(1:nn) HD = HarmonicDecay(nn) i.e weight function for the NNdist: DCEperPoint = HD*NNdist

**Note**

see also chapter 6 of [Thrun, 2018]

**Author(s)**

Michael Thrun
References


Examples

```r
data(Hepta)
InputDistances = as.matrix(dist(Hepta$Data))
projection = Pswarm(InputDistances)
DelaunayClassificationError(Hepta$Data, projection$ProjectedPoints, Hepta$Cls, LC = projection$LC)$DCE
```

---

**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.

**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>int, 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))

Author(s)

Michael Thrun

References

uses a changed code which is inspired by Shreyans Sheth 28.05.2015, see https://ideone.com/qkmt31
**findPossiblePositionsCsingle**

Intern function, do not use yourself

**Description**

Finds all possible jumping position regarding a grid anda Radius for DataBots

**Usage**

```r
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 a integer number.

**Details**

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

**Value**

- `OpenPositions` NumericMatrix, indizes of open positions

**Author(s)**

Michael Thrun

**References**


**See Also**

`setPolarGrid`
GeneratePswarmVisualization

*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 \textit{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 \textit{GeneratePswarmVisualization}, called the generalized U-matrix. The third module is a clustering method with no sensitive parameters \textit{DBSclustering}. The clustering can be verified by the visualization and vice versa. The term DBS refers to the method as a whole.

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

Usage

\texttt{GeneratePswarmVisualization(Data,ProjectedPoints,LC,PlotIt=FALSE,ComputeInR=FALSE)}

Arguments

\begin{itemize}
  \item \texttt{Data} \[1:n,1:d\] array of data: n cases in rows, d variables in columns
  \item \texttt{ProjectedPoints} matrix, \texttt{ProjectedPoints[1:n,1:2]} n by 2 matrix containing coordinates of the Projection: A matrix of the fitted configuration. see output of \textit{Pswarm} for further details
  \item \texttt{LC} size of the grid \(c(Lines,Columns)\), number of Lines and Columns automatic calculated by \texttt{setGridSize} in \textit{Pswarm}
  \item \texttt{PlotIt} Optional, default(\texttt{FALSE}), If TRUE than uses \texttt{plotTopographicMap} of the package \textit{GeneralizedUmatrix} is plotted as a topview in the tiled option, see details for explanation.
  \item \texttt{ComputeInR} Optional, =\texttt{TRUE}: Rcode, =\texttt{FALSE} C++ implementation
\end{itemize}
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: matrix [1:n,1:2], BestMatches of the Umatrix, contrary to ESOM they are always fixed, because predefined by GridPoints.

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

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

GridPoints: 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 plotting 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,Columns**: Size of planar toroid two dimensional grid
- **QuadOrHexa**: Optional, FALSE=If DataPos on hexadiagonal grid, round to 2 decimals after value, Default=TRUE

**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)
library(GeneralizedUmatrix)
plotTopographicMap(genUmatrixList$Umatrix,genUmatrixList$Bestmatches,Cls)
```
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

deprecated! see GeneralizedUmatrix() Generalisierte U-Matrix fuer Projektiionsverfahren

Description
deprecated! see GeneralizedUmatrix()

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

Arguments

Data [1:n,1:d] array of data: 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-Marix 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")
**Lsun3D**

**Details**

Size 212, Dimensions 3, stored in Hepta$Data

Classes 7, stored in Hepta$Cls

**References**


**Examples**

```r
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**

```r
data("Lsun3D")
```

**Details**

Size 404, Dimensions 3

Dataset defined discontinuities, 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**

```r
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**

```r
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**

```r
ProjectedPoints2Grid(ProjectedPoints, Lines, Columns, PlotIt)
```
Arguments

ProjectedPoints
   [1:n,1:2] 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

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=F,Cls=NULL,Silent=T,
          Debug=FALSE,LC=c(NULL,NULL),method= "euclidean",...)

Pswarm

A Swarm of Databots based on polar coordinates (Polar Swarm).
Arguments

DataOrDistance matrix, DataOrDistance[1:n,1:n] symmetric matrix of dissimilarities, if variable unsymmetric DataOrDistance[1:d,1:n] 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

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 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)

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
Reference


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 equilibirium 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 DatsBots can be smelled

DataDists
NumericMatrix, Inputdistances[1:n,1:n]

IndPossibleDBPosR
ComplexVector, see output of **findPossiblePositionsCsingle**
RadiusPositionsschablone
    NumericMatrix, see AllowedDBPosR0 in setPolarGrid

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

Nullpunkt
    NumericVector, equals which(AllowedDBPosR0==0,arr.ind=T), see see AllowedDBPosR0 in setPolarGrid

Lines
    double, small edge length of rectangular grid

Columns
    double, big edge length of rectangular grid

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

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

steigungsverlaufind
    int, internal constant

StressConstAditiv
    double, internal 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

    AllowedDataBotsPos
        ComplexVector, indexes of DataBot Positions after a weak Nash equilibrium is found

    stressverlauf
        NumericVector, internal result, for debugging only

    fokussiertlaufind
        NumericVector, internal result, for debugging only

Author(s)
    Michael Thrun

References
rDistanceToroidCsingle

Intern function for Pswarm

Description

toroid distance calculation

Usage

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

See Also

Pswarm

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 approximately 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 DelaunayClassificationError if a clear baseline is defined.

Author(s)

Michael Thrun

References

RobustNormalization

See Also

DelaunayClassificationError

Examples

```r
x=c(1:5)
y=runif(5,min=1,max=10)
RelativeDifference(x,y)
```

Description

RobustNormalization as described in [Milligan/Cooper, 1988].

Usage

```r
RobustNormalization(Data,Centered=FALSE,Capped=FALSE,
na.rm=TRUE,WithBackTransformation=FALSE,
pmin=0.01,pmax=0.99)
```

Arguments

- **Data**
  
  [1:n,1:d] data matrix of n cases and d features

- **Centered**
  
  centered data around zero by median if TRUE

- **Capped**
  
  TRUE: outliers are capped above 1 or below -1 and set to 1 or -1.

- **na.rm**
  
  If TRUE, infinite values are disregarded

- **WithBackTransformation**
  
  If in the case for forecasting with neural networks a backtransformation is required, this parameter can be set to 'TRUE'.

- **pmin**
  
  defines outliers on the lower end of scale

- **pmax**
  
  defines outliers on the higher end of scale

Details

Normalizes features either between -1 to 1 (Centered=TRUE) or 0-1 (Centered=TRUE) without changing the distribution of a feature itself. For a more precise description please read [Thrun, 2018, p.17].

"[The] scaling of the inputs determines the effective scaling of the weights in the last layer of a MLP with BP neural netowrk, it can have a large effect on the quality of the final solution. At the outset it is besto to standardize all inputs to have mean zero and standard deviation 1 [(or at least the range under 1)]. This ensures all inputs are treated equally in the regularization prozess, and allows to choose a meaningful range for the random starting weights."[Friedman et al., 2012]
**RobustNormalization**

**Value**

- if `WithBackTransformation=FALSE`: TransformedData[1:n,1:d] i.e., normalized data matrix of n cases and d features
- if `WithBackTransformation=TRUE`: List with
  - `TransformedData` [1:n,1:d] normalized data matrix of n cases and d features
  - `MinX` [1:d] numerical vector used for manual back-transformation of each feature
  - `MaxX` [1:d] numerical vector used for manual back-transformation of each feature
  - `Denom` [1:d] numerical vector used for manual back-transformation of each feature
  - `Center` [1:d] numerical vector used for manual back-transformation of each feature

**Author(s)**

Michael Thrun

**References**


**See Also**

`RobustNorm_BackTrafo`

**Examples**

```r
Scaled = RobustNormalization(rnorm(1000, 2, 100), Capped = TRUE)
hist(Scaled)

m = cbind(c(1, 2, 3), c(2, 6, 4))
List = RobustNormalization(m, FALSE, FALSE, FALSE, TRUE)
TransformedData = List$TransformedData

mback = RobustNorm_BackTrafo(TransformedData, List$MinX, List$Denom, List$Center)

sum(m - mback)
```
**RobustNorm_BackTrafo**

*Transforms the Robust Normalization back*

**Description**

Transforms the Robust Normalization back if Capped=True

**Usage**

```
RobustNorm_BackTrafo(TransformedData,
MinX,Denom,Center=0)
```

**Arguments**

- **TransformedData**
  - [1:n,1:d] matrix
- **MinX**
  - scalar
- **Denom**
  - scalar
- **Center**
  - scalar

**Details**

For details see **RobustNormalization**

**Value**

1:n,1:d Data matrix

**Author(s)**

Michael Thrun

**See Also**

**RobustNormalization**

**Examples**

```r
data(Hepta)
Data = Hepta$Data
TransList = RobustNormalization(Data, Centered = TRUE, WithBackTransformation = TRUE)

HeptaData = RobustNorm_BackTrafo(TransList$TransformedData,
   TransList$MinX,
   TransList$Denom,
   TransList$Center)

sum(HeptaData - Data) #<e-15
```
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)

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 =T: Rcode, =F Cpp Codenumner between 1 to x

Details

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

Value

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

Note

Usually not for separated usage!

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.
setPolarGrid

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

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 a 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?
**setRmin**

**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

**Author(s)**

Michael Thrun

**References**


**See Also**

Pswarm

**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]
ShortestGraphPathsC

**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. BEWARE: 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 runs in \(O(n^2 \times \log(V))\), see also [Jungnickel, 2013, p. 87]. Further details can be found in [Jungnickel, 2013, p. 83-87] and [Thrun, 2018, p. 12].
trainstepC

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

See Also
DijkstraSSSP

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
trainstepC(vx, vy, DataSampled, BMUsampled, Lines, Columns, Radius, 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

References

Index

* Accuracy
  ClusteringAccuracy, 7
* BackTransformation_RobustNormalization
  RobustNorm_BackTrafo, 31
* Classification Error
  DelaunayClassificationError, 12
* Clustering
  ClusteringAccuracy, 7
* DBS
  DatabionicSwarm-package, 3
  DBSclustering, 8
  Pswarm, 23
* DCE
  DelaunayClassificationError, 12
* DR
  DatabionicSwarm-package, 3
  GeneratePswarmVisualization, 16
  Pswarm, 23
* DataBionic
  DatabionicSwarm-package, 3
* Databionic swarm
  DatabionicSwarm-package, 3
  GeneratePswarmVisualization, 16
  Pswarm, 23
* Databionic
  DatabionicSwarm-package, 3
* Databionic swarm
  DBSclustering, 8
* Delaunay Classification Error
  DelaunayClassificationError, 12
* Delaunay Graph
  Delaunay4Points, 11
* Delaunay
  Delaunay4Points, 11
* Dijkstra's SSSP
  DijkstraSSSP, 14
* Dijkstra
  DijkstraSSSP, 14
* Dimensionality Reduction
  DatabionicSwarm-package, 3
* ESOM
  GeneratePswarmVisualization, 16
  sESOM4BMUs, 32
* Evaluation of projection methods
  DelaunayClassificationError, 12
* FCPS
  Hepta, 20
  Lsun3D, 21
* GeneratePswarmVisualization
  ProjectedPoints2Grid, 22
  sESOM4BMUs, 32
* Hepta
  Hepta, 20
* Lsun3D
  Lsun3D, 21
* PSwarm
  Pswarm, 23
* Points
  Delaunay4Points, 11
* Polar Swarm
  Pswarm, 23
* Pswarm
  DatabionicSwarm-package, 3
  plotSwarm, 22
* QC
  DelaunayClassificationError, 12
* Quality measurement
  DelaunayClassificationError, 12
* Quality measure
  DelaunayClassificationError, 12
* RelativeDifference
  RelativeDifference, 28
* RobustNorm_BackTrafo
  RobustNorm_BackTrafo, 31
* RobustNormalization
  RobustNorm_BackTrafo, 31
  RobustNormalization, 29
* SOM
GeneratePswarmVisualization, 16
* SSSP
  DijkstraSSSP, 14
* ShortestGraphPaths
  ShortestGraphPathsC, 36
* ShortestPaths
  ShortestGraphPathsC, 36
* Single source shortest path
  DijkstraSSSP, 14
* U-matrix
  GeneratePswarmVisualization, 16
* Umatrix
  GeneratePswarmVisualization, 16
* Voronoi
  Delaunay4Points, 11
* cluster analysis
  DatabionicSwarm-package, 3
  DBSclustering, 8
  GeneratePswarmVisualization, 16
  Pswarm, 23
* clustering
  DatabionicSwarm-package, 3
  DBSclustering, 8
* cluster
  DBSclustering, 8
* datasets
  Lsun3D, 21
* dataset
  Hepta, 20
* difference
  RelativeDifference, 28
* dimensionality reduction
  Pswarm, 23
* distances
  rDistanceToroidCsingle, 27
* emergence
  DatabionicSwarm-package, 3
* equilibrium
  DatabionicSwarm-package, 3
* game theory
  DatabionicSwarm-package, 3
  Pswarm, 23
* generalized Umatrix
  GeneratePswarmVisualization, 16
* generalized Umatrix
  ProjectedPoints2Grid, 22
* graph
  Delaunay4Points, 11
* grid
  setPolarGrid, 34
* hexagonal
  setPolarGrid, 34
* nash
  DatabionicSwarm-package, 3
  PswarmCurrentRadiusC2botsPositive, 25
* pSwarm
  Pswarm, 23
* package
  DatabionicSwarm-package, 3
* polar
  PswarmCurrentRadiusC2botsPositive, 25
* positions
  findPossiblePositionsCsingle, 15
* projection method
  DatabionicSwarm-package, 3
  Pswarm, 23
* projection
  DatabionicSwarm-package, 3
  Pswarm, 23
* pswarm
  Pswarm, 23
* relative
  RelativeDifference, 28
* sESOM
  GeneratePswarmVisualization, 16
* self-organization
  DatabionicSwarm-package, 3
* self-organizing-map
  GeneratePswarmVisualization, 16
* swarm intelligence
  DatabionicSwarm-package, 3
  Pswarm, 23
* swarms
  findPossiblePositionsCsingle, 15
  Pswarm, 23
* swarm
  DatabionicSwarm-package, 3
  DBSclustering, 8
  GeneratePswarmVisualization, 16
  plotSwarm, 22
  Pswarm, 23
  PswarmCurrentRadiusC2botsPositive, 25
  setPolarGrid, 34
INDEX

* toroid
  rDistanceToroidCsingle, 27
* visualization
  DatabionicSwarm-package, 3
  GeneratePswarmVisualization, 16
  Pswarm, 23

ClusteringAccuracy, 7

DatabionicSwarm
  (DatabionicSwarm-package), 3
DatabionicSwarm-package, 3
DBSclustering, 8, 8, 9, 16, 24
DefaultColorSequence, 10
Delaunay4Points, 11
DelaunayClassificationError, 12, 29
Delta3DWeightsC, 13
DijkstraSSSP, 14, 37

findPossiblePositionsCsingle, 15, 25

GeneralizedUmatrix, 18
GeneratePswarmVisualization, 8, 9, 13, 16,
  16, 17, 19, 23, 24, 32
getCartesianCoordinates, 18
getUmatrix4Projection, 19

Hepta, 20

Lsun3D, 21

plotSwarm, 22
plotTopographicMap, 9, 16, 18
ProjectedPoints2Grid, 22
Pswarm, 8, 16–18, 22, 23, 24, 25, 27, 28, 34, 35
pswarm (Pswarm), 23
pswarmCpp (Pswarm), 23
PswarmCurrentRadiusC2botsPositive, 25

quantile, 33

rDistanceToroidCsingle, 27
RelativeDifference, 28
RobustNorm_BackTrafo, 30, 31
RobustNormalization, 29, 31

sESOM4BMUs, 32
setdiffMatrix, 33
setGridSize, 16, 24, 33
setPolarGrid, 15, 18, 26, 34