Package ‘DatabionicSwarm’

February 3, 2020

Type Package
Title Swarm Intelligence for Self-Organized Clustering
Version 1.1.3
Date 2020-02-02
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, matrixStats, 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>1.1.3</td>
</tr>
<tr>
<td>Date</td>
<td>2020-02-02</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 form FCPS
Lsun3D Lsun3D inspired by FCPS
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 depreicated! 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 interactiveGeneralizedUmatrix() 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 applied in


Examples

data('Lsun3D')

#2d projection, without instant visualization of steps
#DistanceMatrix hast to be defined by the user.
InputDistances=as.matrix(dist(Lsun3D$Data))

projection=Pswarm(InputDistances)

#2d projection, with instant visualization
#of steps and DataMatrix (Distance is Euclidean per default)
ClusteringAccuracy

projection=Pswarm(Lsun3D$Data,Cls=Lsun3D$Cls,PlotIt=T)
#
##Computation of Generalized Umatrix
# If Non Euclidean Distances are used, Please Use `code{SammonsMapping}`
# from the ProjectionBasedClustering package with the correct OutputDimension
# to generate a new DataMatrix from the distances (see SheppardDiagram
# or KruskalStress)
visualization=GeneratePswarmVisualization(Data = Lsun3D$Data,
projection$ProjectedPoints,projection$LC)
## Visualizuation of GeneralizedUmatrix,
# Estimation of the Number of Clusters=Number of valleys
library(GeneralizedUmatrix)#install if not installed
GeneralizedUmatrix::plotTopographicMap(visualization$Umatrix,visualization$Bestmatches)
## Automatic Clustering
# number of Cluster from dendrogram (PlotIt=TRUE) or visualization
Cls=DBScustering(k=3, Lsun3D$Data,
visualization$Bestmatches, visualization$LC,PlotIt=FALSE)
## Verification, often its better to mark Outliers manually
GeneralizedUmatrix::plotTopographicMap(visualization$Umatrix,visualization$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 interactivly around high mountain ranges
Imx = ProjectionBasedClustering::interactiveGeneralizedUmatrixIsland(visualization$Umatrix,
visualization$Bestmatches,Cls)
GeneralizedUmatrix::plotTopographicMap(visualization$Umatrix,
visualization$Bestmatches, Cls=Cls,Imx = Imx)
## End(Not run)

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

ClusteringAccuracy

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

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

Automated Clustering approach of the Databonic swarm with abstract U distances, which are the geodesic distances based on high-dimensional distances combined with low dimensional graph paths by using ShortestGraphPathsC, see [Thrun/Ultsch, 2020].

Usage

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=ProjectedPoints, one matrix line per data point

LC  
grid size c(Lines,Columns)

StructureType  
Optional, bool; =TRUE: compact structure of clusters assumed, =FALSE: connected structure of clusters assumed. For the two options vor Clusters, see [Thrun, 2018] or Handl et al. 2006

PlotIt  
Optional, bool, Plots 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

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 (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.
DBSclustering

Value
CIs[1:n] vector with selected classes of the bestmatches. You can use plotTopographicMap(Umatrix,Bestmatches,Cls) for verification.

Note
If you used pswarm with distance matrix instead of a data matrix you may transform your distances into data by using MDS of the ProjectionBasedClustering package. The correct dimension can be found through the Sheppard diagram or kruskals stress.

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 Umatrix package in https://www.uni-marburg.de/fb12/datenbionik/software-en or the function interactiveClustering() of the ProjectionBasedClustering package on CRAN.

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

Author(s)
Michael Thrun

References

Examples
data("Lsun3D")
Data=Lsun3D$Data
InputDistances=as.matrix(dist(Data))

projection=Pswarm(InputDistances)
#automatic Clustering without GeneralizedUmatrix visualization
Cls=DBSclustering(k=3, Data,

projection$ProjectedPoints, projection$LC,PlotIt=TRUE)

## Not run:
visualization=GeneratePswarmVisualization(Data,

projection$ProjectedPoints,projection$LC)
## Sometimes an automatic Clustering can be improved
## thorugh an interactive approach,
## e.g. if Outliers exist (see [Thrun/Ultsch, 2017])
library(ProjectionBasedClustering)
Cls=ProjectionBasedClustering::interactiveClustering(visualization$Umatrix, visualization$Bestmatches, CIs)
DefaultColorSequence  Default color sequence for plots

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

Usage
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
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, 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

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 Deaunay-Graph
DelaunayClassificationError

Author(s)
Michael Thrun

References

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: \( DCE = \text{mean}(DCE\text{PerPoint}) \)
nn the number of points in a relevant neighborhood: \( 0.5 \times 85\text{percentile}(\text{AnzNN}) \)
AnzNN(1:n) the number of points with a delaunay graph neighborhood
NNdists(1:n,1:nn) the distances within the relevant neighborhood, 0 for inner cluster distances
HD(1:nn) \( HD = \text{HarmonicDecay}(nn) \) i.e weight function for the NNdists: \( DCE\text{PerPoint} = HD \times \text{NNdists} \)
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

```r
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:]
**Description**

Dijkstra's SSSP (Single source shortest path) algorithm:

- gets the shortest path (geodesic distance) from source vertex(point) to all other vertices(points) defined by the edges of the adjacency matrix

**Usage**

\[ \text{DijkstraSSSP}(\text{Adj}, \text{Costs}, \text{source}) \]

**Arguments**

- **Adj**: \([1:n,1:n]\) 0/1 adjacency matrix, e.g. from delaunay graph or gabriel graph
- **Costs**: \([1:n,1:n]\) matrix, distances between n points (normally euclidean)
- **source**: int, vertex(point) from which to calculate the geodesic distance to all other points

**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 vertex itself

**Note**

- runs in \(O(E*\log(V))\)

**Author(s)**

Michael Thrun
findPossiblePositionsCsingle

*Intern function, do not use yourself*

**Description**
Finds all possible jumping positions 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, jumping length has to be 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, index of open positions

**Author(s)**
Michael Thrun

**References**

**See Also**
`setPolarGrid`
Generates the U-matrix for Pswarm algorithm

Description
Generates the special case of the generalized U-matrix with the help of an unsupervised neural network (here self-organizing map). From the generalized U-matrix a topographic map with hypsometric tints can be derived. To see this visualization use `plotTopographicMap` of the package `GeneralizedUmatrix`.

Usage
```
GeneratePswarmVisualization(Data, ProjectedPoints, LC, PlotIt=FALSE, ComputeInR=FALSE)
```

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`
- **ComputeInR**: Optional, =TRUE: Rcode, =FALSE C++ implementation

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.

see chapter 8 of [Thrun, 2018] for Pswarm

chapter 5.3 of [Thrun, 2018] for generalized Umatrix and especially the sESOM4BMUs algorithm.
**Value**

- **Bestmatches** matrix \([1:n,1:2]\), BestMatches of the Umatrix, contrary to ESOM they are always fixed, because predefined by GridPoints.
- **Umatrix** matrix \([1:\text{Lines},1:\text{Columns}]\),
- **WeightsOfNeurons** array \([1:\text{Lines},1:\text{Columns},1:d]\), \(d\) 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 pre-defined grid.
- **LC** \(c(\text{Lines,Columns})\), 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.

**Note**

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

**Note**

For interactive Island Generation of an generalized Umatrix see \texttt{interactiveGeneralizedUmatrixIsland} function in the package GeneralizedUmatrix.

The main code of both functions GeneralizedUmatrix and GeneratePswarmVisualization is the same C++ function \texttt{sESOM4BMUs} which is described in [Thrun, 2018].

**Author(s)**

Michael Thrun

**References**


**See Also**

Pswarm and plotTopographicMap and GeneralizedUmatrix of the package GeneralizedUmatrix

**Examples**

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

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-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
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 form FCPS*

**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**

Lsun3D

Description

clearly defined clusters, different variances

Usage

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

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

plotSwarm

Description

Intern function for plotting during the Pswarm annealing process

Usage

plotSwarm(Points,Cls,xlab,ylab,main)
**ProjectedPoints2Grid**

**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

---

**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

---

**Pswarm**

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

**Description**

This projection method is a part of the databionic swarm which uses the nash equilibrium. Using polar coordinates for agents (here Databots) in two dimensions has many advantages, for further details see [Thrun, 2018] and [Thrun/Ultsch, 2020].

**Usage**

```r
Pswarm(DataOrDistance, PlotIt=F, Cls=NULL, Silent=T,
       Debug=FALSE, LC=c(NULL, NULL), method= "euclidean", ...)
```

**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 prozess 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, 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 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

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 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, intern constant, equals round(pp[Radius]*DBAnzahl)
- **limit**  
  int, intern constant, equals ceiling(1/pp[Radius])
- **steigungsverlaufind**  
  int, intern constant
- **StressConstAditiv**  
  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, index 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
RelativeDifference

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

________________________________________________________________________

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 DelaunayClassificationError if a clear baseline is defined.

Author(s)

Michael Thrun

References


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)
```
RobustNormalization

Arguments

Data [1:n,1:d]
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

Range of most of values for each feature is not between zero and one. 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 network, 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 process, and allows to choose a meaningful range for the random starting weights."[Friedman et al., 2012]

Value

if WithBackTransformation=FALSE: TransformedData[1:n,1:d]
if WithBackTransformation=TRUE: List with

TransformedData [1:n,1:d] matrix

MinX scalar
MaxX scalar
Denom scalar
Center scalar

Author(s)

Michael Thrun

References


See Also

RobustNorm_BackTrafo

Examples

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=FALSE

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

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 Codenumber 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
setdiffMatrix

Note

Usually not for separated usage!

Author(s)

Michael Thrun

References


See Also

GeneratePswarmVisualization

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
**Examples**

```r
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)
```

---

### Description

Sets a polar grid for a swarm in an rectangular shape

### Usage

```r
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], # VSAllallowedDBPosPhi0[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. 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

Description

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

Usage

ShortestGraphPathsC(Adj, Cost)

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>Cost</td>
<td>[1:n,1:n] matrix, distances between n points (normally euclidean)</td>
</tr>
</tbody>
</table>

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 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


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

*Topic **Accuracy**
  ClusteringAccuracy, 6
*Topic **BackTransformation_RobustNormalization**
  RobustNorm_BackTrafo, 29
*Topic **Classification Error**
  DelaunayClassificationError, 11
*Topic **Clustering**
  ClusteringAccuracy, 6
*Topic **DBS**
  DatabionicSwarm-package, 3
  Pswarm, 22
*Topic **DCE**
  DelaunayClassificationError, 11
*Topic **DR**
  DatabionicSwarm-package, 3
  GeneratePswarmVisualization, 15
  Pswarm, 22
*Topic **DataBionic**
  DatabionicSwarm-package, 3
*Topic **Databionic Swarm**
  Pswarm, 22
*Topic **Databonic**
  DatabionicSwarm-package, 3
*Topic **Databonic swarm**
  DBSclustering, 8
  GeneratePswarmVisualization, 15
  Pswarm, 22
*Topic **Delaunay Classification Error**
  DelaunayClassificationError, 11
*Topic **Delaunay Graph**
  Delaunay4Points, 10
*Topic **Delaunay**
  Delaunay4Points, 10
*Topic **Dijkstra’s SSSP**
  DijkstraSSSP, 13
*Topic **Dijkstra**
  DijkstraSSSP, 13
*Topic **Dimensionality Reduction**
  DatabionicSwarm-package, 3
*Topic **ESOM**
  GeneratePswarmVisualization, 15
  sESOM4BMUs, 30
*Topic **Evaluation of projection methods**
  DelaunayClassificationError, 11
*Topic **FCPS**
  Hepta, 19
  Lsun3D, 20
*Topic **GeneratePswarmVisualization**
  ProjectedPoints2Grid, 21
  sESOM4BMUs, 30
*Topic **Hepta**
  Hepta, 19
*Topic **Lsun3D**
  Lsun3D, 20
*Topic **PSwarm**
  Pswarm, 22
*Topic **Points**
  Delaunay4Points, 10
*Topic **Pswarm**
  DatabionicSwarm-package, 3
  plotSwarm, 20
*Topic **QM**
  DelaunayClassificationError, 11
*Topic **Quality measurement**
  DelaunayClassificationError, 11
*Topic **Quality measure**
  DelaunayClassificationError, 11
*Topic **RelativeDifference**
  RelativeDifference, 26
*Topic **RobustNorm_BackTrafo**
  RobustNorm_BackTrafo, 29
*Topic **RobustNormalization**
  RobustNorm_BackTrafo, 29
  RobustNormalization, 27
*Topic **SOM**
  GeneratePswarmVisualization, 15
INDEX

*Topic SSSP
  - DijkstraSSSP, 13

*Topic ShortestGraphPaths
  - ShortestGraphPathsC, 35

*Topic ShortestPaths
  - ShortestGraphPathsC, 35

*Topic Single source shortest path
  - DijkstraSSSP, 13

*Topic U-matrix
  - GeneratePswarmVisualization, 15

*Topic U-matrix
  - GeneratePswarmVisualization, 15

*Topic Voronoi
  - Delaunay4Points, 10

*Topic cluster analysis
  - DatabionicSwarm-package, 3
  - DBSclustering, 8
  - GeneratePswarmVisualization, 15
  - Pswarm, 22

*Topic clustering
  - DatabionicSwarm-package, 3
  - DBSclustering, 8

*Topic cluster
  - DBSclustering, 8

*Topic datasets
  - Lsun3D, 20

*Topic dataset
  - Hepta, 19

*Topic difference
  - RelativeDifference, 26

*Topic dimensionality reduction
  - Pswarm, 22

*Topic distances
  - rDistanceToroidCsingle, 25

*Topic emergence
  - DatabionicSwarm-package, 3

*Topic equilibrium
  - DatabionicSwarm-package, 3

*Topic game theory
  - DatabionicSwarm-package, 3

*Topic generalized Umatrix
  - GeneratePswarmVisualization, 15

*Topic generalized Umatrix
  - ProjectedPoints2Grid, 21

*Topic graph
  - Delaunay4Points, 10

*Topic grid
  - setPolarGrid, 33

*Topic hexagonal
  - setPolarGrid, 33

*Topic nash
  - DatabionicSwarm-package, 3
  - PswarmCurrentRadiusC2botsPositive, 24

*Topic pSwarm
  - Pswarm, 22

*Topic package
  - DatabionicSwarm-package, 3

*Topic polar
  - PswarmCurrentRadiusC2botsPositive, 24

*Topic positions
  - findPossiblePositionsCsingle, 14

*Topic projection method
  - DatabionicSwarm-package, 3
  - Pswarm, 22

*Topic projection
  - DatabionicSwarm-package, 3
  - Pswarm, 22

*Topic pswarm
  - Pswarm, 22

*Topic relative
  - RelativeDifference, 26

*Topic sESOM
  - GeneratePswarmVisualization, 15

*Topic self-organization
  - DatabionicSwarm-package, 3

*Topic self-organizing-map
  - GeneratePswarmVisualization, 15

*Topic swarm intelligence
  - DatabionicSwarm-package, 3
  - Pswarm, 22

*Topic swarms
  - findPossiblePositionsCsingle, 14
  - Pswarm, 22

*Topic swarm
  - DatabionicSwarm-package, 3
  - DBSclustering, 8
  - GeneratePswarmVisualization, 15
  - plotSwarm, 20
  - Pswarm, 22
  - PswarmCurrentRadiusC2botsPositive, 24
  - setPolarGrid, 33

*Topic toroid
  - rDistanceToroidCsingle, 25
*Topic visualization
  DatabionicSwarm-package, 3
  GeneratePswarmVisualization, 15
  Pswarm, 22

ClusteringAccuracy, 6

DatabionicSwarm
  (DatabionicSwarm-package), 3
DatabionicSwarm-package, 3
DBSclustering, 8, 8, 15, 23
DefaultColorSequence, 10
Delaunay4Points, 10
DelaunayClassificationError, 11, 27
Delta3DWeightsC, 12
DijkstraSSSP, 13, 36

findPossiblePositionsCsingle, 14, 24
GeneratePswarmVisualization, 8, 12, 15,
  15, 17, 22, 23, 31
getCartesianCoordinates, 17
getUmatrix4Projection, 18

Hepta, 19

Lsun3D, 20

plotSwarm, 20
ProjectedPoints2Grid, 21
Pswarm, 8, 15, 16, 21, 22, 23–26, 32, 34
pswarm (Pswarm), 22
pswarmCpp (Pswarm), 22
PswarmCurrentRadiusC2botsPositive, 24

quantile, 32

rDistanceToroidCsingle, 25
RelativeDifference, 26
RobustNorm_BackTrafo, 29, 29
RobustNormalization, 27, 29, 30

sESOM4BMUs, 30
setdiffMatrix, 31
setGridSize, 15, 22, 23, 32
setPolarGrid, 14, 17, 24, 33
setRmin, 34
ShortestGraphPathsC, 13, 35

trainstepC, 36