A clustering approach applicable to every projection method is proposed here. The two-dimensional scatter plot of any projection method can construct a topographic map which displays unapparent data structures by using distance and density information of the data. The generalized U*-matrix renders this visualization in the form of a topographic map, which can be used to automatically define the clusters of high-dimensional data. The whole system is based on Thrun and Ultsch, "Using Projection based Clustering to Find Distance and Density based Clusters in High-Dimensional Data" <DOI:10.1007/s00357-020-09373-2>. Selecting the correct projection method will result in a visualization in which mountains surround each cluster. The number of clusters can be determined by counting valleys on the topographic map. Most projection methods are wrappers for already available methods in R. By contrast, the neighbor retrieval visualizer (NeRV) is based on C++ source code of the 'dredviz' software package, the t-SNE multicore version is based on C++ source code of Dmitry Ulyanov, and the Curvilinear Component Analysis (CCA) is translated from 'MATLAB' ('SOM Toolbox' 2.0) to R.
URL https://www.deepbionics.org
LazyLoad yes

BugReports https://github.com/Mthrun/ProjectionBasedClustering/issues

Author Michael Thrun [aut, cre, cph],
    Quirin Stier [ctb, rev] (<https://orcid.org/0000-0002-7896-4737>),
    Brinkmann Luca [ctb],
    Florian Lerch [aut],
    Felix Pape [aut],
    Tim Schreier [aut],
    Luis Winckelmann [aut],
    Kristian Nybo [cph],
    Jarkko Venna [cph],
    Ulyanov Dmitry [cph],
    van der Maaten Laurens [cph]

Maintainer Michael Thrun <m.thrun@gmx.net>
Repository CRAN
Date/Publication 2023-10-11 18:00:02 UTC

R topics documented:

- ProjectionBasedClustering-package ........................................ 3
- CCA ................................................................. 5
- ContTrustMeasure ..................................................... 7
- DefaultColorSequence .................................................. 8
- Delaunay4Points ....................................................... 8
- DijkstraSSSP ........................................................ 9
- Hepta ................................................................. 10
- ICA ................................................................. 11
- interactiveClustering .................................................. 12
- interactiveGeneralizedUmatrixIsland .................................. 14
- interactiveProjectionBasedClustering .................................. 15
- Isomap ................................................................. 17
- KLMeasure ............................................................ 18
- KruskalStress .......................................................... 20
- MDS ................................................................. 20
- NeRV ................................................................. 22
- PCA ................................................................. 23
- PlotProjectedPoints .................................................... 25
- PolarSwarm ........................................................... 26
- Projection2Bestmatches ................................................. 27
- ProjectionBasedClustering .............................................. 28
- ProjectionPursuit ...................................................... 30
- SammonsMapping ....................................................... 31
- ShortestGraphPathsC .................................................. 33
- tSNE ................................................................. 34
- UniformManifoldApproximationProjection .............................. 36
Description

The package is based on a conference talk [Thrun/Ultsch, 2017], see <DOI:10.13140/RG.2.2.13124.53124>, and [Thrun/Ultsch, 2020]. The abstract of the conference talk is as follows:

Many data mining methods rely on some concept of the dissimilarity between pieces of information encoded in the data of interest. These methods can be used for cluster analysis. However, no generally accepted definition of clusters exists in the literature [Hennig et al., 2015]. Here, consistent with Bouveyron et al., it is assumed that a cluster is a group of similar objects [Bouveyron et al., 2012]. The clusters are called natural because they do not require a dissection; instead, they are clearly separated in the data [Duda et al., 2001, Theodoridis/Koutroumbas, 2009, pp. 579, 600]. These clusters can be identified by distance or density based high-dimensional structures. Dimensionality reduction techniques are able to reduce the dimensions of the input space to facilitate the exploration of structures in high-dimensional data. If they are used for visualization, they are called projection methods. The generalized U*-matrix technique is applicable for these and can be used to visualize both distance- and density-based structures [Thrun 2018; Ultsch/Thrun, 2017]. The idea that the abstract U*-matrix (AU-matrix) can be used for clustering [Ultsch et al., 2016]. The distances required for hierarchical clustering are defined by the AU-matrix [Lötsch/Ultsch, 2014]. Using this distance we propose a clustering approach for every projection method based on the U*-matrix visualization of a topographic map [Thrun 2018; Thrun/Ultsch, 2017]. The number of clusters and the cluster structure can be estimated by counting the valleys in a topographic map [Thrun et al., 2016]. If the number of clusters and the clustering method are chosen correctly, then the clusters will be well separated by mountains in the visualization. Outliers are represented as volcanoes and can be interactively marked in the visualization after the automated clustering process.

Furthermore, [Thrun et al., 2021] presents an interactive parameter-free approach, that incorporates a human-in-the-loop, for projection-based clustering.

Details

A comparison to 32 common clustering algorithms is provided in [Thrun/Ultsch, 2020].

Note

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

Additionally you can use the standard ShepardScatterPlot or the better approach through the ShepardDensityPlot of the CRAN package DataVisualizations.

Author(s)

Michael Thrun, Felix Pape, Florian Lerch, Tim Schreier, Luis Winckelmann
References


Examples

data('Hepta')
# 2d projection
# Visualization of GeneralizedUmatrix

projectionpoints=NeRV(Hepta$Data)
# Computation of Generalized Umatrix
library(GeneralizedUmatrix)
visualization=GeneralizedUmatrix(Data = Hepta$Data,projectionpoints)
TopviewTopographicMap(visualization$Umatrix,visualization$Bestmatches)
## Curvilinear Component Analysis (CCA)

### Description

CCA Projects data vectors using Curvilinear Component Analysis [Demartines/Herault, 1995],[Demartines/Herault, 1997].

Unknown values (NaN’s) in the data: projections of vectors with unknown components tend to drift towards the center of the projection distribution. Projections of totally unknown vectors are set to unknown (NaN).

### Usage

CCA(DataOrDistances,Epochs,OutputDimension=2,method='euclidean',
alpha0 = 0.5, lambda0,PlotIt=FALSE,Cls)
Arguments

DataOrDistances

Numerical matrix defined as either

Data, i.e., \([1:n,1:d]\), nonsymmetric, and consists of \(n\) cases of \(d\)-dimensional
data points with every case having \(d\) attributes, variables or features,
or

Distances, i.e., \([1:n,1:n]\), symmetric and consists of \(n\) cases, e.g., \(\text{as.matrix(dist(Data, method))}\)

Epochs

Number of eppochs (scalar), i.e, training length

OutputDimension

Number of dimensions in the Outputspace, default=2

method

method specified by distance string. One of: 'euclidean', 'cityblock=manhattan', 'cosine', 'chebychev', 'jaccard', 'minkowski', 'manhattan'

alpha0

(scalar) initial step size, 0.5 by default

lambda0

(scalar) initial radius of influence, 3*max(std(D)) by default

PlotIt

Default: FALSE. If TRUE: Plots the projection as a 2d visualization. OutputDimension>2: only the first two dimensions will be shown

Cls

[1:n,1] Optional,: only relevant if PlotIt=TRUE. Numeric vector, given Classification in numbers: every element is the cluster number of a certain corresponding element of data.

Details

An short overview of different types of projection methods can be found in [Thrun, 2018, p.42, Fig. 4.1] (doi:10.1007/9783658205409).

Value

A \(n\) by OutputDimension matrix containing coordinates of the projected points.

Note

Only Tranfered from matlab to R. Matlabversion: Contributed to SOM Toolbox 2.0, February 2nd, 2000 by Juha Vesanto.

You can use the standard Sheparddiagram or the better approach through the ShepardDensityScatter of the CRAN package DataVisualizations.

Author(s)

Florian Lerch

References


ContTrustMeasure

Examples

```r
data('Hepta')
Data=Hepta$Data

Proj=CCA(Data,Epochs=20)

## Not run:
PlotProjectedPoints(Proj$ProjectedPoints,Hepta$Cls)

## End(Not run)
```

Description

Computes trustworthiness and continuity for projected data (see [Kaski2003]).

Usage

```r
ContTrustMeasure(datamat, projmat, lastNeighbor)
```

Arguments

datamat        numerical matrix of data: n cases in rows, d variables in columns
projmat        numerical matrix of projected data: n cases in rows, k variables in columns, where k is the projection output dimension
lastNeighbor   scalar, maximal size of neighborhood to be considered

Details

C++ source code comes from https://research.cs.aalto.fi/pml/software/dredviz/

Value

numerical [k,7] matrix, where k is the lastNeighbor value. The matrix contains the columns:

- Neighborhood size; worst-case trustworthiness; average trustworthiness; best-case trustworthiness;
- worst-case continuity; average continuity; best-case continuity

where neighborhood size is the size of the neighborhood considered, which ranges from 1:last-Neighbor

Author(s)

Michael Thrun
References


See Also

An alternative measure is the KLMeasure

Examples

```r
data('Hepta')
Data=Hepta$Data
res=MDS(Data)
Proj = res$ProjectedPoints
PlotProjectedPoints(res$ProjectedPoints,Hepta$Cls)

ContTrustMeasure(Hepta$Data, Proj, 10)
```

DefaultColorSequence  Default color sequence for plots

Description

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

Usage

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

Format

A vector with 562 different strings describing colors for plots.

Delaunay4Points  Adjacency matrix of the delaunay graph for BestMatches of Points

Description

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

Usage

```r
Delaunay4Points(Points, IsToroid = TRUE, Grid=NULL, PlotIt=FALSE)
```
DijkstraSSSP

Arguments

| Points          | [1:n,1:3] matrix containing the BMKey, X and Y coordinates of the n, Best-Matches NEED NOT 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 |

Details

as

Value

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

Author(s)

Michael Thrun

References


DijkstraSSSP

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

| 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, vertice(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.

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

References

uses a changed code which is inspired by Shreyans Sheth 28.05.2015, see [https://ideone.com/qkmt31](https://ideone.com/qkmt31)

---

**Hepta**

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

---

Description

clearly defined clusters, different variances

Usage

data("Hepta")

details

Details

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

References


Examples

data(Hepta)
str(Hepta)
ICA

Independent Component Analysis (ICA)

Description

Independent Component Analysis:
Negentropie: difference of entropy to a corresponding normally-distributed random variable
\[ J(y) = |E(G(y) - E(G(v)))|^2 \]

Usage

ICA(Data, OutputDimension=2, Contrastfunction="logcosh", Alpha=1, Iterations=200, PlotIt=FALSE,Cls)

Arguments

Data numerical matrix of n cases in rows, d variables in columns, matrix is not symmetric.
OutputDimension Number of dimensions in the Outputspace, default=2
Contrastfunction Maximierung der Negentropie ueber geeignete geeignete Kontrastfunktion Default: 'logcosh' \( G(u) = 1/e^a \log \cosh(a^2) \) 'exp': \( G(u) = -e^{u^2/2} \)
Alpha constant with 1<=alpha<=2 used in approximation to neg-entropy when fun == "logcosh"
Iterations maximum number of iterations to perform.
PlotIt Default: FALSE, If TRUE: Plots the projection as a 2d visualization. OutputDimension>2: only the first two dimensions will be shown
Cls [1:n,1] Optional: only relevant if PlotIt=TRUE. Numeric vector, given Classification in numbers: every element is the cluster number of a certain corresponding element of data.

Details

An short overview of different types of projection methods can be found in [Thrun, 2018, p.42, Fig. 4.1] (doi:10.1007/9783658205409).

Value

ProjectedPoints [1:n, OutputDimension], n by OutputDimension matrix containing coordinates of the Projection
Mixing [1:OutputDimension,1:d] Mischungsmatrix s.d gilt Data=MixingMatrix*ProjectedPoints
Unmixing Entmischungsmatrix with Data*Unmixing=ProjectedPoints
PCMatrix pre-whitening matrix that projects data onto the first n.comp principal components.
Note
A wrapper for fastICA
You can use the standard ShepardScatterPlot or the better approach through the ShepardDensityPlot of the CRAN package DataVisualizations.

Author(s)
Michael Thrun

Examples
```r
data('Hepta')
Data=Hepta$Data
Proj=ICA(Data)

## Not run:
PlotProjectedPoints(Proj$ProjectedPoints,Hepta$Cls)

## End(Not run)
```

interactiveClustering  GUI for interactive cluster analysis

Description
This tool is an interactive shiny tool that visualizes a given generalized Umatrix and allows the user to select areas and mark them as clusters to improve a projection based clustering.

Arguments
- **Umatrix** [1:Lines,1:Columns] Matrix of Umatrix Heights
- **Bestmatches** [1:n,1:2] Array with positions of Bestmatches
- **Cls** [1:n] Classification of the Bestmatches

Details
Clicking on "Quit" returns theCls vector to the workspace.

Value
List of
- **EnlargedBestmatches** [1:n,1:2] Array with positions of Bestmatches according to the enlarged umatrix.
**interactiveClustering**

EnlargedCls  [1:n] Classification of the Bestmatches according to the enlarged umatrix.


Bestmatches  [1:n,1:2] Array with positions of Bestmatches

Cls  [1:n] Classification of the Bestmatches

TopView_TopographicMap

Plot of a topographic map.

**Note**

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

**Author(s)**

Florian Lerch, Michael Thrun

**References**


**Examples**

data('Hepta')

#2d projection

# Visualization of GeneralizedUmatrix

projectionpoints=NeRV(Hepta$Data)

library(GeneralizedUmatrix)

visualization=GeneralizedUmatrix(Data = Hepta$Data,projectionpoints)

## Semi-Automatic Clustering done interactively in a shiny gui

## Not run:

Cls = interactiveClustering(visualization$Umatrix, visualization$Bestmatches)

## Plotting

plotTopographicMap(visualization$Umatrix,visualization$Bestmatches,Cls)

## End(Not run)
interactiveGeneralizedUmatrixIsland

GUI for cutting out an Island.

Description

The toroid Umatrix is usually drawn 4 times, so that connected areas on borders can be seen as a whole. An island is a manual cutout of such a tiled visualization, that is selected such that all connected areas stay intact. This shiny tool allows the user to do this manually.

Usage

interactiveGeneralizedUmatrixIsland(Umatrix, Bestmatches=NULL, Cls=NULL, Plotter="plotly")

Arguments

Bestmatches: [1:n, 1:2] Matrix with positions of Bestmatches for n datapoints, first column is the position in Lines and second column in Columns
Cls: Classification of the Bestmatches
Plotter: Choose between plotting frameworks: "plotly" and "ggplot2"

Details

Clicking on "Quit" returns the Imx matrix to the workspace. Details can bee read in [Thrun et al, 2016, Thrun/Ultsch, 2017].

Value

Boolean Matrix that represents the island within the tiled Umatrix.

Note

This function is a deprecated version of a function from the Umatrix packages created by Florian Lerch and Michael Thrun

Author(s)

Michael Thrun

References


Examples

data("Hepta")
Data=Hepta$Data
Cls=Hepta$Cls
InputDistances=as.matrix(dist(Data))
res=cmdscale(d=InputDistances, k = 2, eig = TRUE, add = FALSE, x.ret = FALSE)
ProjectedPoints=as.matrix(res$points)
#see also ProjectionBasedClustering package for other common projection methods

library(GeneralizedUmatrix)
resUmatrix=GeneralizedUmatrix(Data,ProjectedPoints)
TopviewTopographicMap(resUmatrix$Umatrix,resUmatrix$Bestmatches,Cls)
#if in 3D if rgl package exists
#plotTopographicMap(resUmatrix$Umatrix,resUmatrix$Bestmatches,Cls)

##Interactive Island Generation
## from a tiled Umatrix (toroidal assumption)

## Not run:
Imx = interactiveGeneralizedUmatrixIsland(resUmatrix$Umatrix, resUmatrix$Bestmatches)
plotTopographicMap(resUmatrix$Umatrix, resUmatrix$Bestmatches, Imx = Imx)

## End(Not run)

---

**interactiveProjectionBasedClustering**

*Interactive Projection-Based Clustering (IPBC)*

Description

An interactive clustering tool published in [Thrun et al., 2020] that uses the topographic map visualizations of the generalized U-matrix and a variety of different projection methods. This function receives a dataset and starts a shiny interface where one is able to choose a projection method and generate a plotly visualization of the topographic map [Thrun et al., 2016] of the generalized U-matrix [Ultsch/Thrun, 2017] combined with projected points. It includes capabilities for interactive clustering within the interface as well as automatic projection-based clustering based on [Thrun/Ultsch, 2020].

Usage

`interactiveProjectionBasedClustering(Data, Cls=NULL)`

`IPBC(Data, Cls=NULL)`
interactiveProjectionBasedClustering

**Arguments**

Data
The dataset [1:n,1:d] of n cases and d variables with which the U-matrix and the projection will be calculated. Please see also the note below.

Cls
Optional: Prior Classification of the data for the [1:n] cases of k classes.

**Details**

To cluster data interactively, i.e., select specific data points and create a cluster), first generate the visualization. Thereafter, switch in the menu to clustering, hold the left mouse button and then frame a valley. Simple mouse clicks will not start the lasso functionality of plotly.

The resulting clustering is stored in Cls which is a numerical vector of the length n (number of cases) with the integer elements of numbers from 1 to k if k is the number of groups in the data. Each element of Cls as an unambiguous mapping to a case of Data indicating by the rownames of Data. If Data has no rownames a vector from 1:n is generated and then Cls is named by it.

**Value**

Returns a List of:

Cls
[1:n] numerical vector of the clustering of the dataset for then cases of k clusters

Umatrix
[1:Lines,1:Columns] generalized Umatrix to be plotted, numerical matrix storing the U-heights, see [Thrun, 2018] for definition.

Bestmatches

LastProjectionMethodUsed
name of last projection method that was used as a string

TopView_TopographicMap
The final plot generated by plot.ly when closing the tool

**Note**

Some dimensionality reduction methods will assume data without missing values, some other DR methods assume unique data points, i.e., no distance=0 for any two cases(rows) of data. In these cases the IPBC method will crash.

**Author(s)**

Tim Schreier, Felix Pape, Luis Winckelmann, Michael Thrun

**References**


Isomap


Examples

```r
if(interactive()){
  data('Hepta')
  Data=Hepta$Data
  V=interactiveProjectionBasedClustering(Data)
  # with prior classification
  Cls=Hepta$Cls
  V=IPBC(Data,Cls)
}
```

Isomap

Description

Isomap projection as introduced in 2000 by Tenenbaum, de Silva and Langford

Even with a manifold structure, the sampling must be even and dense so that dissimilarities along a manifold are shorter than across the folds. If data do not have such a manifold structure, the results are very sensitive to parameter values.

Usage

Isomap(Distances,k,OutputDimension=2,PlotIt=FALSE,Cls)

Arguments

Distances: Symmetric [1:n,1:n] distance matrix, e.g. as.matrix(dist(Data,method))
k: number of k nearest neighbors, if the data is fragmented choose an higher k
OutputDimension: Number of dimensions in the output space, default = 2
PlotIt: Default: FALSE, If TRUE: Plots the projection as a 2d visualization. If OutputDimension > 2 only the first two dimensions will be shown.
Cls: Optional and only relevant if PlotIt=TRUE. Numeric vector, given Classification in numbers: every element is the cluster number of a certain corresponding element of data.
Details
An short overview of different types of projection methods can be found in [Thrun, 2018, p.42, Fig. 4.1] (doi:10.1007/9783658205409).

Value
ProjectedPoints[1:n,OutputDimension] n by OutputDimension matrix containing coordinates of the Projection: A matrix of the fitted configuration.

Note
A wrapper enabling a planar projection of the manifold learning method based on the isomap of the package vegan
if Data fragmented choose an higher k
You can use the standard ShepardScatterPlot or the better approach through the ShepardDensityPlot of the CRAN package DataVisualizations.

Author(s)
Michael Thrun

Examples

data('Hepta')
Data=Hepta$Data
Proj=Isomap(as.matrix(dist(Data)),k=7)

## Not run:
PlotProjectedPoints(Proj$ProjectedPoints,Hepta$Cls)

## End(Not run)

---

**KLMMeasure**

*Rank-based smoothed precision/recall measure for projection.*

Description
Computes rank-based smoothed precision/recall, with cost function based on Kullback-Leibler-divergence (see [Venna2010]).

Usage

KLMMeasure(Data, pData, NeighborhoodSize = 20L)
**Arguments**

- **Data**
  - numerical matrix of data: n cases in rows, d variables in columns

- **pData**
  - numerical matrix of projected data: n cases in rows, k variables in columns, where k is the projection output dimension

- **NeighborhoodSize**
  - Number of points in neighborhood to be considered. Default is 20

**Value**

- **SmoothedPrecision**
  - Scalar, smoothed precision value

- **SmoothedRecall**
  - Scalar, smoothed recall value

**Note**

C++ source code comes from [https://research.cs.aalto.fi/pml/software/dredviz/](https://research.cs.aalto.fi/pml/software/dredviz/)

**Author(s)**

Michael Thrun

**References**


**See Also**

An alternative measure is the **ContTrustMeasure**

**Examples**

```r
data('Hepta')
Data=Hepta$Data
res=MDS(Data)
Proj = res$ProjectedPoints

kl_m = KLMeasure(Hepta$Data, Proj)
# Smoothed precision
print(kl_m[[1]])
# Smoothed recall
print(kl_m[[2]])
```
KruskalStress \hspace{1cm} Kruskal stress calculation

**Description**
Calculates the stress as defined by Kruskal for 2 distance matrices

**Usage**
KruskalStress(InputDistances, OutputDistances)

**Arguments**
- InputDistances: Distance matrix of the original Data
- OutputDistances: Distance matrix of the projected Data

**Details**
An short overview of different types of quality measures can be found in [Thrun, 2018, p.68, Fig. 6.3] (doi:10.1007/9783658205409).

**Value**
A single numerical representing the Kruskal stress of the distance matrices.

**Author(s)**
Felix Pape

---

MDS \hspace{1cm} Multidimensional Scaling (MDS)

**Description**
Classical multidimensional scaling of a data matrix. Also known as principal coordinates analysis

**Usage**
MDS(DataOrDistances, method='euclidean', OutputDimension=2, PlotIt=FALSE, Cls)
Arguments

DataOrDistances
Numerical matrix defined as either
\( \text{Data} \), i.e., \([1:n,1:d]\), nonsymmetric, and consists of \( n \) cases of \( d \)-dimensional
data points with every case having \( d \) attributes, variables or features,
or
\( \text{Distances} \), i.e., \([1:n,1:n]\), symmetric and consists of \( n \) cases, e.g.,
\( \text{as.matrix(dist(Data,method))} \)

method
method specified by distance string: 'euclidean','cityblock=manhattan','cosine','chebychev','jaccard','minkowski','manhattan','binary'

OutputDimension
Number of dimensions in the Outputspace, default=2

PlotIt
Default: FALSE, If TRUE: Plots the projection as a 2d visualization.

Cls
[1:n,1] Optional, only relevant if PlotIt=TRUE, Numeric vector, given Classification in numbers: every element is the cluster number of a certain corresponding element of data.

Details
An short overview of different types of projection methods can be found in [Thrun, 2018, p.42, Fig. 4.1] (doi:10.1007/9783658205409).

Value

ProjectedPoints
[1:n,OutputDimension], \( n \) by OutputDimension matrix containing coordinates of the Projection

Eigenvalues
the eigenvalues of MDSvalues*MDSvalues'

Stress
Shephard-Kruskal Stress

Note
A wrapper for \text{cmdscale}
You can use the standard \text{ShepardScatterPlot} or the better approach through the \text{ShepardDensityPlot}
of the CRAN package \text{DataVisualizations}.

Author(s)
Michael Thrun

Examples

data('Hepta')
Data=Hepta$Data

Proj=MDS(Data)

## Not run:
PlotProjectedPoints(Proj$ProjectedPoints,Hepta$Cls)

## End(Not run)
NeRV

*Neighbor Retrieval Visualizer (NeRV)*

**Description**

Projection is done by the neighbor retrieval visualizer (NeRV)

**Usage**

```
NeRV(Data, lambda = 0.1, neighbors = 20, iterations = 10,
   cg_steps = 2, cg_steps_final = 40, randominit = T, OutputDimension = 2,
   PlotIt = FALSE, Cls)
```

**Arguments**

- **Data**
  - Numerical matrix of the Data to be projected, \([1:n,1:d]\), nonsymmetric, and consists of \(n\) cases of \(d\)-dimensional data points with every case having \(d\) attributes, variables or features
- **lambda**
  - Optional: Controls the trustworthiness-continuity tradeoff. Default = 0.1
- **neighbors**
  - Optional: Set the number of nearest neighbours that each point should have. Should be positive. Default = 20
- **iterations**
  - Optional: The number of iterations to perform. Default = 10
- **cg_steps**
  - Optional: The number of conjugate gradient steps to perform per iteration in NeRV’s optimization scheme. Default = 2
- **cg_steps_final**
  - Optional: The number of conjugate gradient steps to perform on the final iteration in NeRV’s optimization scheme. Default = 40
- **randominit**
  - Optional: TRUE: Random Initialization (default), FALSE: PCA initialization
- **OutputDimension**
  - Optional: Number of dimensions in the Output space, default=2
- **PlotIt**
  - Optional: Should the projected points be plotted? Default: FALSE. Note: this is only useful if OutputDimension = 2.
- **Cls**
  - Optional: Vector containing the number of the class for each row in Data. This is only used to color the points according to their classes if PlotIt = T

**Details**

Uses the NeRV projection with matrix Data and lambda. Lambda controls the trustworthiness-continuity tradeoff.

An short overview of different types of projection methods can be found in [Thrun, 2018, p.42, Fig. 4.1] (doi:10.1007/9783658205409).
PCA

Principal Component Analysis (PCA)

Description

Performs a principal components analysis on the given data matrix projection=SammonsMapping(Data)

Usage

PCA(Data, OutputDimension=2, Scale=FALSE, Center=FALSE, PlotIt=FALSE, Cls)
Arguments

Data
numerical matrix of data: n cases in rows, d variables in columns

OutputDimension
Number of dimensions in the Outputspace, default=2

Scale
a logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place. The default is FALSE for consistency with S, but in general scaling is advisable. Alternatively, a vector of length equal the number of columns of x can be supplied. The value is passed to scale.

Center
a logical value indicating whether the variables should be shifted to be zero centered. Alternately, a vector of length equal the number of columns of x can be supplied. The value is passed to scale.

PlotIt
Default: FALSE, If TRUE: Plots the projection as a 2d visualization. OutputDimension>2: only the first two dimensions will be shown

Cls
[1:n,1] Optional,: only relevant if PlotIt=TRUE. Numeric vector, given Classification in numbers: every element is the cluster number of a certain corresponding element of data.

Details

An short overview of different types of projection methods can be found in [Thrun, 2018, p.42, Fig. 4.1] (doi:10.1007/9783658205409).

Value

ProjectedPoints
[1:n,OutputDimension], n by OutputDimension matrix containing coordinates of the Projectio

Rotation
the matrix of variable loadings (i.e., a matrix whose columns contain the eigenvectors)

sDev
the standard deviations of the principal components (i.e., the square roots of the eigenvalues of the covariance/correlation matrix, though the calculation is actually done with the singular values of the data matrix)

TransformedData
matrix with PCA transformed Data

Center
the centering used, or FALSE

Scale
the scaling used, or FALSE

Note

A wrapper for prcomp
You can use the standard ShepardScatterPlot or the better approach through the ShepardDensityPlot of the CRAN package DataVisualizations.

Author(s)

Michael Thrun
### Examples

```r
data('Hepta')
Data=Hepta$Data

Proj=PCA(Data)

## Not run:
PlotProjectedPoints(Proj$ProjectedPoints,Hepta$Cls)

## End(Not run)
```

### Description

plots XY data colored by Cls with ggplot2

### Usage

```r
PlotProjectedPoints(Points,Cls,BMUorProjected=F,PlotLegend=FALSE,
  xlab='X',ylab='Y',main="Projected Points",PointSize=2.5)
```

### Arguments

- **Points** `[1:n,1:2]` xy cartesian coordinates of a projection
- **Cls** numeric vector, given Classification in numbers: every element is the cluster number of a certain corresponding element of data.
- **BMUorProjected** Default ==FALSE, If TRUE assuming BestMatches of ESOM instead of Projected Points
- **PlotLegend** ...
- **xlab** Optional: Label of the x axis
- **ylab** Optional: Label of the y axis
- **main** Optional: title
- **PointSize** Optional: size of points

### Value

`ggobject of ggplot2`

### Author(s)

Michael Thrun
**PolarSwarm**  

**Polar Swarm (Pswarm)**

**Description**


**Usage**

PolarSwarm(DataOrDistances, method = "euclidean", PlotIt = FALSE, Cls)

**Arguments**

- **DataOrDistances**  
  Numerical matrix defined as either  
  - Data, i.e., [1:n,1:d], nonsymmetric, and consists of n cases of d-dimensional  
    data points with every case having d attributes, variables or features,  
  - Distances, i.e.,[1:n,1:n], symmetric and consists of n cases, e.g., as.matrix(parallelDist::parallelDist(Data,method))

- **method**  
  If Data is given the method to computing the distances can be specified here. Please see the documentation of package `parallelDist` for the types that are possible.

- **PlotIt**  
  Default: FALSE. If TRUE: Plots the projection as a 2d visualization. OutputDimension>2: only the first two dimensions will be shown

- **Cls**  
  Optional,: only relevant if PlotIt=TRUE. Numeric vector, given Classification in numbers: every element is the cluster number of a certain corresponding element of data.

**Details**

By exploiting swarm intelligence and game theory no parameter have to be set.

**Value**

List of

- **ProjectedPoints**  
  [1:n,2], n by 2 matrix containing coordinates of the Projection

- **ModelObject**  
  output of Pswarm

**Author(s)**

Michael Thrun

**References**

Projection2Bestmatches

Projection to Bestmatches

Description
Transformation of projected points to bestmatches defined by generalized Umatrix

Usage
Projection2Bestmatches(ProjectedPoints)

Arguments
ProjectedPoints
[1:n,1:2] n projected points in two-dimensional space.

Details
It is assumed that an unambiguous assignment between projected points and data points is given.

Value
Bestmatches [1:n,1:2] Positions of GridConverted Projected Points, which can be used for the generalized Umatrix, to the predefined Grid by Lines and Columns. First Columns has the content of the Line No and second Column of the Column number.

LCA [1:2] vector if Line No. and ColumnNo. which defines the size of the grid of the generalized Umatrix

Note
Details of the equations used are written down in [Thrun, 2018, p. 47].
Author(s)
Michael Thrun

References

See Also
XYcoords2LinesColumns

Examples
```r
data('Hepta')
ProjList=MDS(Hepta$Data)
trafo=Projection2Bestmatches(ProjList$ProjectedPoints)
```

Description
Three steps are necessary for PBC. First, a projection method has to be chosen to generate projected points of high-dimensional data points. Second, the generalized U*-matrix has to be applied to the projected points by using a simplified emergent self-organizing map (ESOM) algorithm which is an unsupervised neural network [Thrun, 2018]. The resulting generalized U-matrix can be visualized by the topographic map [Thrun et al., 2016]. Third, the clustering itself is built on top of the generalized U-matrix using the concept of the abstract U-Matrix and shortest graph paths using ShortestGraphPathsC.

Usage
```r
ProjectionBasedClustering(k, DataOrDistances, BestMatches, LC,
StructureType = TRUE, PlotIt = FALSE, method = "euclidean")
```

Arguments
- **k**
  number of clusters, how many to you see in the 3d landscape?
- **DataOrDistances**
  Numerical matrix that will be used for clustering with one DataPoint per row, defined as either as
  Data, i.e., [1:n,1:d], nonsymmetric, and consists of n cases of d-dimensional data points with every case having d attributes, variables or features,
or Distances, i.e., [1:n,1:n], symmetric and consists of n cases, e.g., as.matrix(dist(Data,method))

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, 2017] or Handl et al. 2006

PlotIt Optional, bool, Plots Dendrogramm

method Optional, distance method used in parallelDist if Data given.

Details

ProjectionBasedClustering is a flexible and robust clustering framework based on a chose projection method and projection method a parameter-free high-dimensional data visualization technique. The visualization combines projected points with a topographic map with hypsometric colors, defined by the generalized U-matrix (see package GeneralizedUmatrix function GeneralizedUmatrix).

The clustering method with no sensitive parameters is done in this function and the algorithm is introduced in detail in [Thrun/Ultsch, 2020]. The clustering can be verified by the visualization and vice versa. If you want to verify your clustering result externally, you can use Heatmap or SilhouettePlot of the CRAN package DataVisualizations.

If parallelDist is not installed, function automatically falls back to dist.

Value

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

Note

Often it is better to mark the outliers manually after the process of clustering; 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 interactiveClustering function.

Author(s)

Michael Thrun

References


Examples

```r
data('Hepta')

# Step I: 2d projection
projectionpoints=NeRV(Hepta$Data)

# Step II (Optional): Computation of Generalized Umatrix
library(GenUmatrix)
visualization=GenUmatrix(Data = Hepta$Data,projectionpoints)

# Visualization of Generalized Umatrix
library(GenUmatrix)
TopviewTopographicMap(visualization$Umatrix,visualization$Bestmatches)

# or in 3D if rgl package exists
#plotTopographicMap(visualization$Umatrix,visualization$Bestmatches)

# Step III: Automatic Clustering
trafo=Projection2Bestmatches(projectionpoints)
Cls=ProjectionBasedClustering(k=7, Hepta$Data, trafo$Bestmatches, trafo$LC, PlotIt=TRUE)

# Verification of Clustering
TopviewTopographicMap(visualization$Umatrix,visualization$Bestmatches,Cls)
# or in 3D if rgl package exists
#plotTopographicMap(visualization$Umatrix,visualization$Bestmatches,Cls)
```

---

**ProjectionPursuit**

**Projection Pursuit**

**Description**

In the absence of a generative model for the data the algorithm can be used to find the projection pursuit directions. Projection pursuit is a technique for finding 'interesting' directions in multidimensional datasets.

**Usage**

```r
ProjectionPursuit(Data, OutputDimension=2, Indexfunction="logcosh",
Alpha=1, Iterations=200, PlotIt=FALSE, Cls)
```
Arguments

Data array of data: n cases in rows, d variables in columns, matrix is not symmetric or distance matrix, in this case matrix has to be symmetric

OutputDimension Number of dimensions in the Outputspace, default=2

Indexfunction Criterium for Minimization:
  default: 'logcosh’ G(u)=1/a*log cosh(a*u) (ICA) ‘exp': G(u)=-exp(u^2/2) 'kernel’ 1/(1* pi )*exp(r/2)

Alpha constant with 1<=alpha<=2 used in approximation to neg-entropy when fun == "logcosh"

Iterations maximum number of iterations to perform.

PlotIt Default: FALSE. If TRUE: Plots the projection as a 2d visualization. OutputDimension>2: only the first two dimensions will be shown

Cls [1:n,1] Optional: only relevant if PlotIt=TRUE. Numeric vector, given Classification in numbers: every element is the cluster number of a certain corresponding element of data.

Details

An short overview of different types of projection methods can be found in [Thrun, 2018, p.42, Fig. 4.1] (doi:10.1007/9783658205409).

Value

ProjectedPoints
  [1:n,OutputDimension], n by OutputDimension matrix containing coordinates of the Projection

Note

You can use the standard ShepardScatterPlot or the better approach through the ShepardDensityPlot of the CRAN package DataVisualizations.

Author(s)

Michael Thrun

SammonsMapping Sammons Mapping

Description

Improved MDS through a normalization of the Input space
Usage

SammonsMapping(DataOrDistances, method='euclidean', OutputDimension=2, PlotIt=FALSE, Cls)

Arguments

DataOrDistances
Numerical matrix defined as either
Data, i.e., [1:n,1:d], nonsymmetric, and consists of n cases of d-dimensional
data points with every case having d attributes, variables or features,
or
Distances, i.e., [1:n,1:n], symmetric and consists of n cases, e.g., as.matrix(dist(Data, method))

method
method specified by distance string: 'euclidean', 'cityblock=manhatten', 'cosine', 'chebychev', 'jaccard', 'minkowski', 'manhattan'

OutputDimension
Number of dimensions in the Outputspace, default=2

PlotIt
Default: FALSE, If TRUE: Plots the projection as a 2d visualization.

Cls
[1:n,1] Optional,: only relevant if PlotIt=TRUE. Numeric vector, given Classification in numbers: every element is the cluster number of a certain corresponding element of data.

Details

An short overview of different types of projection methods can be found in [Thrun, 2018, p.42, Fig. 4.1] (doi:10.1007/9783658205409).

Value

ProjectedPoints
[1:n,OutputDimension], n by OutputDimension matrix containing coordinates of the Projection

Stress
Shephard-Kruskal Stress

Note

A wrapper for sammon

You can use the standard ShepardScatterPlot or the better approach through the ShepardDensityPlot of the CRAN package DataVisualizations.

Author(s)

Michael Thrun

Examples

data('Hepta')
Data=Hepta$Data
Proj=SammonsMapping(Data)
## Not run:
PlotProjectedPoints(Proj$ProjectedPoints,Hepta$Cls)
## End(Not run)

**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 adjacency matrix, e.g. from delaunay graph or gabriel graph

- **Cost**
  
  [1:n,1:n] matrix, distances between n points (normally euclidean)

### Details
Vertices are the points, edges have the costs defined by weights (normally a distance). The algorithm runs in \( O(n^2 \cdot \text{E} \cdot \log(V)) \), see also [Jungnickel, 2013, p. 87]. Further details can be found in [Jungnickel, 2013, p. 83-87].

### 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


**tSNE**

*T-distributed Stochastic Neighbor Embedding (t-SNE)*

**Description**

T-distributed Stochastic Neighbor Embedding

\[ \text{res} = \text{tSNE}(\text{Data}, \text{KNN}=30, \text{OutputDimension}=2) \]

**Usage**

\[ \text{tSNE}(\text{DataOrDistances}, k, \text{OutputDimension}=2, \text{Algorithm}='tsne\_cpp', \]

\[ \text{method}='\text{euclidean}', \text{Whitening}=\text{FALSE}, \text{Iterations}=1000, \text{PlotIt}=\text{FALSE}, \text{Cls}, \text{num\_threads}=1, \ldots) \]

**Arguments**

- **DataOrDistances**
  Numerical matrix defined as either
  \[ \text{Data}, \text{i.e.,}[1:n,1:d], \text{nonsymmetric, and consists of n cases of d-dimensional} \]
  \[ \text{data points with every case having d attributes, variables or features,} \]
  \[ \text{or} \]
  \[ \text{Distances, i.e.,}[1:n,1:n], \text{symmetric and consists of n cases, e.g., as.matrix(dist(\text{Data}, method))} \]

- **k**
  number of k nearest neighbors=number of effective nearest neighbors("perplexity");
  Important parameter. If not given, settings of packages of t-SNE will be used depending Algorithm

- **OutputDimension**
  Number of dimensions in the Outputspace, default=2

- **Algorithm**
  'tsne\_cpp': T-Distributed Stochastic Neighbor Embedding using a Barnes-HutImplementation in C++ of **Rtsne.** Requires Version >= 0.15 of **Rtsne** for multicore parallelisation.
  'tsne\_opt\_cpp': T-Distributed Stochastic Neighbor Embedding with automated optimized parameters using a Barnes-HutImplementation in C++ of [Ulyanov, 2016].
  'tsne\_r': pure R implementation of the t-SNE algorithm of of **tsne**

- **method**
  method specified by distance string: 'euclidean', 'cityblock=manhatten', 'cosine', 'chebychev', 'jaccard', 'minkowski', 'manhattan', 'binary'

- **Whitening**
  A boolean value indicating whether the matrix data should be whitened (tsne\_r)
  or if pca should be used priorly (tsne\_cpp)

- **Iterations**
  maximum number of iterations to perform.

- **PlotIt**
  Default: FALSE, If TRUE: Plots the projection as a 2d visualization. OutputDimension>2: only the first two dimensions will be shown
**Details**

An short overview of different types of projection methods can be found in [Thrun, 2018, p.42, Fig. 4.1], doi:10.1007/9783658205409.

**Value**

List of

- `ProjectedPoints` 
  [1:n,OutputDimension], n by OutputDimension matrix containing coordinates of the Projection
- `ModelObject` NULL for tsne_r, further information if tsne_cpp is selected

**Note**

A wrapper for **Rtsne** (Algorithm='tsne_cpp'),

**Multicore-opt-tSNE** (Algorithm='tsne_opt_cpp'),

or for **tsne** (Algorithm='tsne_r')

You can use the standard ShepardScatterPlot or the better approach through the ShepardDensityPlot of the CRAN package DataVisualizations.

**Author(s)**

Michael Thrun, Luca Brinkmann

**References**


**Examples**

```r
data('Hepta')
Data=Hepta$Data

## Not run:
Proj=tSNE(Data,k=7)
PlotProjectedPoints(Proj$ProjectedPoints,Hepta$Cls)
## End(Not run)
```

---

**Uniform Manifold Approximation and Projection**

**Description**

Uniform manifold approximation and projection is a technique for dimension reduction. The algorithm was described by [McInnes et al., 2018].

**Usage**

```r
UniformManifoldApproximationProjection(DataOrDistances, k,
Epochs, OutputDimension=2, Algorithm='umap_pkg', PlotIt=FALSE, Cls,...)
```

**Arguments**

- **DataOrDistances**: Numerical matrix defined as either
  - `Data`, i.e., `[1:n,1:d]`, nonsymmetric, and consists of n cases of d-dimensional data points with every case having d attributes, variables or features,
  - `Distances`, i.e., `[1:n,1:n]`, symmetric and consists of n cases, e.g., `as.matrix(dist(Data, method))`

- **k**: number of k nearest neighbors, Important parameter, if not given, settings of package `umap` will be used, default of package `umap` is currently 15

- **Epochs**: Number of epochs (scalar), i.e, training length, default of package `umap` is currently 200

- **OutputDimension**: Number of dimensions in the Outputspace, default=2

- **Algorithm**: "umap_pkg": provides an interface for two implementations. One is written from scratch other one requires python `umap`
  - "uwot_pkg": complete re-implementation in R (and C++, via the 'Rcpp' package) of `uwot`
**PlotIt**
Default: FALSE. If TRUE: Plots the projection as a 2d visualization. OutputDimension>2: only the first two dimensions will be shown.

**Cls**
Optional.: only relevant if PlotIt=TRUE. Numeric vector, given Classification in numbers: every element is the cluster number of a certain corresponding element of data.

... one of the other 21 parameters that can be specified, please see `umap.defaults` of package `umap` for details or parameters to be set in package `uwot` depending on the choice of Algorithm.

**Details**
To the knowledge of the author of this function no peer-reviewed publication of the method exists. Use with great care.

**Value**
List of

- **ProjectedPoints**
  [1:n,OutputDimension], n by OutputDimension matrix containing coordinates of the Projection

- **ModelObject**
  output of `umap` or of package `uwot` depending on Algorithm

- **Setting**
  specific settings used in `UniformManifoldApproximationProjection`

**Note**
Uniform Manifold Approximation and Projection and U-matrix [Ultsch/Siemon, 1990] are both sometimes abbreviated with Umap. Hence the abbreviation is omitted here.

**Author(s)**
Michael Thrun

**References**


**See Also**
- `umap` of `umap`
- `umap` of `uwot`
Examples

```r
data('Hepta')
Data=Hepta$Data

Proj=UniformManifoldApproximationProjection(Data)

## Not run:
PlotProjectedPoints(Proj$ProjectedPoints,Hepta$Cls)

## End(Not run)
```
Index

* CCA
  CCA, 5
  NeRV, 22
  PCA, 23
  PolarSwarm, 26
  ProjectionPursuit, 30
  SammonsMapping, 31
  tSNE, 34
  UniformManifoldApproximationProjection, 36
* Classical multidimensional scaling
  MDS, 20
* Cluster Analysis
  interactiveProjectionBasedClustering, 15
* Clustering
  interactiveProjectionBasedClustering, 15
* Continuity
  ContTrustMeasure, 7
* Curvilinear Component Analysis
  CCA, 5
  ContTrustMeasure, 7
* DR
  CCA, 5
  ICA, 11
  interactiveProjectionBasedClustering, 15
  Isomap, 17
  MDS, 20
  NeRV, 22
  PCA, 23
  PolarSwarm, 26
  ProjectionPursuit, 30
  SammonsMapping, 31
  tSNE, 34
* Delaunay
  Delaunay4Points, 8
* Dijkstra's SSSP
  DijkstraSSSP, 9
* Dijkstra
  DijkstraSSSP, 9
* Dimensionality Reduction
  CCA, 5
  ICA, 11
  interactiveProjectionBasedClustering, 15
  MDS, 20
* FCPS
  Hepta, 10
* Hepta
  Hepta, 10
* ICA
  ICA, 11
* IPBC
  interactiveProjectionBasedClustering, 15
* Independent Component Analysis
  ICA, 11
* Isomap
  Isomap, 17
* Kullback-Leibler-divergence
  KLMeasure, 18
* MDS
  MDS, 20
* Manifold Learning
  Isomap, 17
* Measure
  ContTrustMeasure, 7
  KLMeasure, 18
* NeRV
  NeRV, 22
* PBC
  ProjectionBasedClustering, 28
* PCA
  PCA, 23
* Points
  Delaunay4Points, 8
* Polar Swarm
<table>
<thead>
<tr>
<th><em>PolarSwarm</em>, 26</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>PolarSwarm</em></td>
</tr>
<tr>
<td>PolarSwarm, 26</td>
</tr>
<tr>
<td><em>Precision</em></td>
</tr>
<tr>
<td>KLMeasure, 18</td>
</tr>
<tr>
<td><em>Principal component analysis</em></td>
</tr>
<tr>
<td>PCA, 23</td>
</tr>
<tr>
<td><em>Projection Method</em></td>
</tr>
<tr>
<td>CCA, 5</td>
</tr>
<tr>
<td>ICA, 11</td>
</tr>
<tr>
<td>interactiveProjectionBasedClustering, 15</td>
</tr>
<tr>
<td>MDS, 20</td>
</tr>
<tr>
<td>NeRV, 22</td>
</tr>
<tr>
<td>PCA, 23</td>
</tr>
<tr>
<td>PolarSwarm, 26</td>
</tr>
<tr>
<td>ProjectionPursuit, 30</td>
</tr>
<tr>
<td>SammonsMapping, 31</td>
</tr>
<tr>
<td>tSNE, 34</td>
</tr>
<tr>
<td>UniformManifoldApproximationProjection, 36</td>
</tr>
<tr>
<td><em>Projection Pursuit</em></td>
</tr>
<tr>
<td>ProjectionPursuit, 30</td>
</tr>
<tr>
<td><em>Projection-based Clustering</em></td>
</tr>
<tr>
<td>ProjectionBasedClustering, 28</td>
</tr>
<tr>
<td><em>ProjectionPursuit</em></td>
</tr>
<tr>
<td>ProjectionPursuit, 30</td>
</tr>
<tr>
<td><em>Projection</em></td>
</tr>
<tr>
<td>ContTrustMeasure, 7</td>
</tr>
<tr>
<td>KLMeasure, 18</td>
</tr>
<tr>
<td><em>Pswarm</em></td>
</tr>
<tr>
<td>PolarSwarm, 26</td>
</tr>
<tr>
<td><em>Recall</em></td>
</tr>
<tr>
<td>KLMeasure, 18</td>
</tr>
<tr>
<td><em>SSSP</em></td>
</tr>
<tr>
<td>DijkstraSSSP, 9</td>
</tr>
<tr>
<td><em>Sammons Mapping</em></td>
</tr>
<tr>
<td>SammonsMapping, 31</td>
</tr>
<tr>
<td><em>SammonsMapping</em></td>
</tr>
<tr>
<td>SammonsMapping, 31</td>
</tr>
<tr>
<td><em>ShortestGraphPaths</em></td>
</tr>
<tr>
<td>ShortestGraphPathsC, 33</td>
</tr>
<tr>
<td><em>ShortestPaths</em></td>
</tr>
<tr>
<td>ShortestGraphPathsC, 33</td>
</tr>
<tr>
<td><em>Single source shortest path</em></td>
</tr>
<tr>
<td>DijkstraSSSP, 9</td>
</tr>
<tr>
<td><em>T-distributed Stochastic Neighbor Embedding</em></td>
</tr>
<tr>
<td>tSNE, 34</td>
</tr>
<tr>
<td><em>Trustworthiness</em></td>
</tr>
<tr>
<td>ContTrustMeasure, 7</td>
</tr>
<tr>
<td><em>Uniform Manifold Approximation Projection</em></td>
</tr>
<tr>
<td>UniformManifoldApproximationProjection, 36</td>
</tr>
<tr>
<td><em>UniformManifoldApproximationProjection</em></td>
</tr>
<tr>
<td>UniformManifoldApproximationProjection, 36</td>
</tr>
<tr>
<td><em>cluster analysis</em></td>
</tr>
<tr>
<td>ProjectionBasedClustering, 28</td>
</tr>
<tr>
<td><em>clustering</em></td>
</tr>
<tr>
<td>ProjectionBasedClustering, 28</td>
</tr>
<tr>
<td><em>cluster</em></td>
</tr>
<tr>
<td>ProjectionBasedClustering, 28</td>
</tr>
<tr>
<td><em>datasets</em></td>
</tr>
<tr>
<td>Hepta, 10</td>
</tr>
<tr>
<td><em>interactive Projection Based Clustering</em></td>
</tr>
<tr>
<td>interactiveProjectionBasedClustering, 15</td>
</tr>
<tr>
<td><em>neighbor retrieval visualizer</em></td>
</tr>
<tr>
<td>NeRV, 22</td>
</tr>
<tr>
<td><em>swarm</em></td>
</tr>
<tr>
<td>PolarSwarm, 26</td>
</tr>
<tr>
<td><em>t-SNE</em></td>
</tr>
<tr>
<td>tSNE, 34</td>
</tr>
<tr>
<td><em>tSNE</em></td>
</tr>
<tr>
<td>tSNE, 34</td>
</tr>
<tr>
<td>CCA, 5</td>
</tr>
<tr>
<td>ContTrustMeasure, 7, 19</td>
</tr>
<tr>
<td>DefaultColorSequence, 8</td>
</tr>
<tr>
<td>Delaunay4Points, 8</td>
</tr>
<tr>
<td>DijkstraSSSP, 9, 34</td>
</tr>
<tr>
<td>dist, 29</td>
</tr>
<tr>
<td>fastICA, 12</td>
</tr>
<tr>
<td>Hepta, 10</td>
</tr>
<tr>
<td>ICA, 11</td>
</tr>
<tr>
<td>interactiveClustering, 12</td>
</tr>
<tr>
<td>interactiveGeneralizedUmatrixIsland, 14</td>
</tr>
<tr>
<td>interactiveProjectionBasedClustering, 15</td>
</tr>
</tbody>
</table>
INDEX

IPBC
  (interactiveProjectionBasedClustering), 15
Isomap, 17
KLMeasure, 8, 18
KruskalStress, 20
MDS, 20
NeRV, 22
PCA, 23
PlotProjectedPoints, 25
PolarSwarm, 26
prcomp, 24
Projection2Bestmatches, 27
ProjectionBasedClustering, 28
ProjectionBasedClustering-package, 3
ProjectionPursuit, 30
Pswarm, 26, 27
Rtsne, 35
sammon, 32
SammonsMapping, 31
ShortestGraphPathsC, 33
tSNE, 34
ttse, 35
umap, 37
umap.defaults, 37
UniformManifoldApproximationProjection, 36
XYcoords2LinesColumns, 28