Package ‘stream’

May 24, 2015

Version 1.1-4
Date 2015-05-24
Title Infrastructure for Data Stream Mining
Description A framework for data stream modeling and associated data mining tasks such as clustering and classification. The development of this package was supported in part by NSF IIS-0948893 and NIH R21HG005912.
Depends R (>= 2.13.0), methods, proxy (>= 0.4-7)
Imports animation, clue, cluster, clusterGeneration, fpc, MASS, mlbench, Rcpp (>= 0.11.4)
Suggests DBI, RSQLite, testthat
LinkingTo Rcpp
License GPL-3
NeedsCompilation yes
Author Michael Hahsler [aut, cre, cph], Matthew Bolanos [aut, cph], John Forrest [aut, cph]
Maintainer Michael Hahsler <mhahsler@lyle.smu.edu>
Repository CRAN
Date/Publication 2015-05-24 21:23:43

R topics documented:

  animation .......................................................... 3
  DSC .............................................................. 4
  DSClassify ....................................................... 5
  DSC_DBSCAN ..................................................... 5
  DSC_DBSTREAM .................................................. 6
  DSC_DStream .................................................... 8
  DSC_Hierarchical ............................................... 11
  DSC_Kmeans .................................................... 12
  DSC_Marco ...................................................... 14
R topics documented:

DSC_Mirco ................................................................. 14
DSC_Reachability ...................................................... 14
DSC_Sample ............................................................ 16
DSC_Static .............................................................. 17
DSC_TwoStage ......................................................... 18
DSC_Window ............................................................ 19
DSD ................................................................. 20
DSD_BarsAndGaussians .............................................. 21
DSD_Benchmark ....................................................... 22
DSD_Cubes ............................................................. 23
DSD_Gaussians ......................................................... 24
DSD_Memory ........................................................... 25
DSD_MG ............................................................... 26
DSD_mlbenchData ..................................................... 28
DSD_mlbenchGenerator ............................................... 29
DSD_ReadCSV .......................................................... 30
DSD_ReadDB ........................................................... 32
DSD_ScaleStream ...................................................... 33
DSD_Target ............................................................ 34
DSD_UniformNoise .................................................... 35
DSFP ................................................................. 36
DSO ................................................................. 36
DSO_Sample ........................................................... 37
DSO_Window .......................................................... 38
DST ................................................................. 39
evaluate ............................................................... 39
get_assignment ......................................................... 43
get_centers ........................................................... 44
get_copy ............................................................... 45
get_points ............................................................ 46
get_weights ........................................................... 47
MGC ................................................................. 48
microToMacro ......................................................... 50
nclusters ............................................................... 52
plot ................................................................. 53
prune_clusters ........................................................ 54
recluster ............................................................. 55
reset_stream .......................................................... 56
update ............................................................... 57
write_stream .......................................................... 58

Index 60
animation

Animates the plotting of a DSD and the clustering process

Description
Generates an animation of a data stream or a data stream clustering.

Usage

animate_data(dsd, n=1000, wait=.1, horizon=100, ...)
animate_cluster(dsc, dsd, macro=NULL, n=1000, wait=.1, horizon=100,
  evaluationMeasure=NULL, evaluationType="auto", evaluationAssign="micro",
  evaluationAssignmentMethod = "auto", ...)

Arguments

dsd a DSD object
dsc a DSC object
macro a DSC_macro object used for reclustering when performing evaluations.
n the number of points to be plotted
wait the time interval between each frame
horizon the number of points displayed at once/used for evaluation.
evaluationMeasure the evaluation measure that should be graphed below the animation
evaluationType evaluate "micro" or "macro"-clusters? "auto" chooses micro if dsd is of class DSC_micro and no macro is given. Otherwise macro is used.
evaluationAssign assign new points to the closest "micro" or "macro"-cluster to calculate the evaluation measure.
evaluationAssignmentMethod how to assign data points to micro-clusters. Options are "model" and "nn" (nearest neighbor). "auto" uses model if available and nn otherwise.
... extra arguments for plot.

Details
Animations are recorded using the library animation and can be replayed (which gives a smoother experience since the is no more computation done) and saved in various formats (see Examples section below).

See Also
evaluate_cluster for stream evaluation without animation. See ani.replay for replaying and saving animations.
Examples

```r
## Not run:
stream <- DSD_Benchmark()
animate_data(stream, n=5000, xlim=c(0,1), ylim=c(0,1), horizon=100)

### animations can be replayed with the animation package
library(animation)
animation::ani.options(interval=.1) ## change speed
ani.replay()

### animations can also be saved as HTML, animated gifs, etc.
saveHTML(ani.replay())

### animate the clustering process with evaluation
reset_stream(stream)
dbstream <- DSC_DBSTREAM(r=.05, lambda=.1, shared_density=TRUE,
                           alpha=.2, noise=.1, gap_time=100)

animate_cluster(dbstream, stream, n=5000, horizon=100,
                evaluationMeasure="crand", evaluationType="macro",
                evaluationAssign="micro",
                xlim=c(0,1), ylim=c(0,1), shared=TRUE)

## End(Not run)
```

---

**DSC**

*Data Stream Clusterer Base Classes*

**Description**

Abstract base classes for all DSC (Data Stream Clusterer) and DSC\_R classes.

**Details**

The DSC and DSC\_R classes cannot be instantiated (calling DSC() or DSC\_R() produces errors), but they serve as a base class from which other DSC classes inherit.

Class DSC provides several generic functions that can operate on all DSC subclasses: `print()`, `plot()`, `nclusters()` to get the current number of clusters, `get_centers()` to get the cluster centers, and `get_weights()` to get the cluster weights (if implemented). `get_centers()` and `get_weights()` are typically overwritten by subclasses of DSC. DSC\_R provides these functions for R-based DSC implementations.

**See Also**

`animate_cluster, cluster, evaluate, get_assignment, get_centers, get_weights, get_copy, microToMacro, nclusters, plot, prune_clusters, recluster`
DSClassify

Abstract Class for Data Stream Classifiers

Description

Abstract class for data stream classifiers. Currently, stream does not implement classification algorithms.

See Also

DST

DSC_DBSCAN

DBSCAN Macro-clusterer

Description

Implements the DBSCAN algorithm for reclustering micro-clusterings.

Usage

DSC_DBSCAN(eps, MinPts = 5, weighted = TRUE, description=NULL)

Arguments

eps            radius of the eps-neighborhood.
MinPts          minimum number of points required in the eps-neighborhood.
weighted        logical indicating if a weighted version of DBSCAN should be used.
description     optional character string to describe the clustering method.

Details

DBSCAN is a weighted extended version of the implementation in fpc where each micro-cluster center considered a pseudo point. For weighting we use in the MinPts comparison the sum of weights of the micro-cluster instead of the number.

DBSCAN first finds core points based on the number of other points in its eps-neighborhood. Then core points are joined into clusters using reachability (overlapping eps-neighborhoods).

Note that this clustering cannot be updated iteratively and every time it is used for (re)clustering, the old clustering is deleted.

Value

An object of class DSC_DBSCAN (a subclass of DSC, DSC_R, DSC_Macro).
References


See Also

DSC, DSC_Macro

Examples

# 3 clusters with 5% noise
stream <- DSD_Gaussians(k=3, d=2, noise=0.05)

# Use DBSCAN to recluster micro clusters (a sample)
sample <- DSC_Sample(k=100)
update(sample, stream, 500)

dbscan <- DSC_DBSCAN(eps = .05)
recluster(dbscan, sample)
plot(dbscan, stream, type="both")

# For comparison we can cluster some data with DBSCAN directly
# Note: DBSCAN is not suitable for data streams since it passes over the data
# several times.
dbscan <- DSC_DBSCAN(eps = .05)
update(dbscan, stream, 500)
plot(dbscan, stream)

DSC_DBSTREAM

DBSTREAM clustering algorithm

Description

Implements a simple density-based stream clustering algorithm that assigns data points to micro-clusters with a given radius and implements shared-density-based reclustering.

Usage

DSC_DBSTREAM(r, lambda = 0.001, gaptime = 1000L,
noise = 0.1, shared_density = FALSE, alpha=0.1, k=0, minweight = 0)
get_shared_density(x, use_alpha = TRUE)
change_alpha(x, alpha)
DSC_DBSTREAM

Arguments

- **r**  The radius of micro-clusters.
- **lambda**  The lambda used in the fading function.
- **gapt ime**  weak micro-clusters (and weak shared density entries) are removed every gaptime points.
- **noise**  The proportion of noise points in the data that will be removed while clustering.
- **shared_density**  Record shared density information. If set to TRUE then shared density is used for reclustering, otherwise reachability is used (overlapping clusters with less than \( r \times (1 - \alpha) \) distance are clustered together).
- **k**  The number of macro clusters to be returned if macro is true.
- **alpha**  For shared density: The minimum proportion of shared points between to clusters to warrant combining them (a suitable value for 2D data is .3). For reachability clustering it is a distance factor.
- **minweight**  The minimum number of weight a micro-cluster needs to have.
- **x**  A DSC_DBSTREAM object to get the shared density information from.
- **use_alpha**  only return shared density if it exceeds alpha.

Details

The DBSTREAM algorithm checks for each new data point in the incoming stream, if it is below the threshold value of dissimilarity value of any existing micro-clusters, and if so, merges the point with the micro-cluster. Otherwise, a new micro-cluster is created to accommodate the new data point.

Although DSC_DBSTREAM is a micro clustering algorithm, macro clusters and weights are available.

plot() for DSC_DBSTREAM has two extra logical parameters called assignment and shared_density which show the assignment area and the shared density graph, respectively.

Note that DSC_DBSTREAM can at this point not be saved to disk using save() or saveRDS(). This functionality will be added later!

Value

An object of class DSC_DBSTREAM (subclass of DSC, DSC_R, DSC_Micro).

See Also

DSC, DSC_Micro

Examples

```r
set.seed(0)
stream <- DSD_Gaussians(k = 3, noise = 0.05)

# create clusterer with r = .1
dbstream <- DSC_DBSTREAM(r = .1, noise = 0.05)
update(dbstream, stream, 500)
```
DSC_DStream

# D-Stream Data Stream Clustering Algorithm

Description

Implements the D-Stream data stream clustering algorithm.
Usage

```python
DSC_DStream(gridsize, lambda = 1e-3, gaptime=1000L, 
    Cm=3, Cl=.8, attraction=FALSE, epsilon=.3, Cm2=Cm, k=NULL, N = 0)
get_attraction(x, relative=FALSE, grid_type = "dense", dist=FALSE)
```

Arguments

- **gridsize**: Size of grid cells.
- **lambda**: Fading constant used function to calculate the decay factor $2^{-\lambda}$. (Note: in the paper the authors use lambda to denote the decay factor and not the fading constant!)
- **gaptime**: Sporadic grids are removed every gaptime number of points.
- **Cm**: Density threshold used to detect dense grids as a proportion of the average expected density ($Cm > 1$). The average density is given by the total weight of the clustering over $N$, the number of grid cells.
- **Cl**: Density threshold to detect sporadic grids ($0 > Cl > Cm$). Transitional grids have a density between $Cl$ and $Cm$.
- **attraction**: Compute and store information about the attraction between adjacent grids. If TRUE then attraction is used to create macro-clusters, otherwise macro-clusters are created by merging adjacent dense grids.
- **epsilon**: Overlap parameter for attraction as a proportion of gridsize.
- **Cm2**: Threshold on attraction to join two dense grid cells (as a proportion on the average expected attraction). In the original algorithm $Cm2$ is equal to $Cm$.
- **k**: Alternative to $Cm2$ (not in the original algorithm). Create $k$ clusters based on attraction. In case of more than $k$ unconnected components, closer groups of MCs are joined.
- **N**: Fix the number of grid cells used for the calculation of the density thresholds with $Cl$ and $Cm$. If $N$ is not given (0) then the algorithm tries to determine $N$ from the data. Note that this means that $N$ potentially increases over time and outliers might produce an extremely large value which will lead to a sudden creation of too many dense micro-clusters. The original paper assumed that $N$ is known a priori.
- **x**: DSC_DStream object to get attraction values from.
- **relative**: Calculates relative attraction (normalized by the cluster weight).
- **grid_type**: The attraction between what grid types should be returned?
- **dist**: Make attraction symmetric and transform into a distance.

Details

D-Stream creates an equally spaced grid and estimates the density in each grid cell using the count of points falling in the cells. Grid cells are classified based on density into dense, transitional and sporadic cells. The density is faded after every new point by a factor of $2^{-\lambda}$. Every gaptime number of points sporadic grid cells are removed.

For reclustering D-Stream (2007 version) merges adjacent dense grids to form macro-clusters and then assigns adjacent transitional grids to macro-clusters. This behavior is implemented as attraction=FALSE.
The 2009 version of the algorithm adds the concept of attraction between grids cells. If attraction = TRUE is used then the algorithm produces macro-clusters based on attraction between dense adjacent grids (uses \text{cm2} which in the original algorithm is equal to \text{cm}).

For many functions (e.g., get\_centers(), plot()), D-Stream adds a parameter grid\_type with possible values of "dense", "transitional", "sparse", "all" and "used". This only returns the selected type of grid cells. "used" includes dense and adjacent transitional cells which are used in D-Stream for reclustering.

For plot D-Stream also provides extra parameters "grid" and "grid\_type" to show micro-clusters as grid cells (density represented by gray values).

Note that DSC\_DStream can at this point not be saved to disk using save() or save\_RDS(). This functionality will be added later!

**Value**

An object of class DSC\_DStream (subclass of DSC, DSC\_R, DSC\_Micro).

**References**


**See Also**

DSC, DSC\_Micro

**Examples**

```r
stream <- DSD\_BarsAndGaussians(noise=.05)
plot(stream)

# we set Cm=.8 to pick up the lower density clusters
dstream1 <- DSC\_DStream(gridsize=1, Cm=1.5)
update(dstream1, stream, 1000)
dstream1

# micro-clusters (these are "used" grid cells)
nclusters(dstream1)
head(get\_centers(dstream1))

# plot (DStream provides additional grid visualization)
plot(dstream1, stream)
plot(dstream1, stream, grid=TRUE)

# look only at dense grids
nclusters(dstream1, grid\_type="dense")
plot(dstream1, stream, grid=TRUE, grid\_type="dense")
```
DSC_Hierarchical

Hierarchical Micro-Cluster Reclusterer

Description

Implementation of hierarchical clustering to recluster a set of micro-clusters.

Usage

DSC_Hierarchical(k=NULL, h=NULL, method = "complete", min_weight=NULL, description=NULL)

Arguments

- **k**
  - The number of desired clusters.
- **h**
  - Height where to cut the dendrogram.
- **method**
  - the agglomeration method to be used. This should be (an unambiguous abbreviation of) one of "ward", "single", "complete", "average", "mcquitty", "median" or "centroid".
- **min_weight**
  - micro-clusters with a weight less than this will be ignored for reclustering.
- **description**
  - optional character string to describe the clustering method.

Details

Please refer to hclust for more details on the behavior of the algorithm.

Note that this clustering cannot be updated iteratively and every time it is used for (re)clustering, the old clustering is deleted.
Value

A list of class DSC, DSC_R, DSC_Macro, and DSC_Hierarchical. The list contains the following items:

- **description**: The name of the algorithm in the DSC object.
- **Robj**: The underlying R object.

See Also

DSC, DSC_Macro

Examples

```r
# Cassini dataset
stream <- DSD_mlbenchGenerator("cassini")

# Use hierarchical clustering to recluster micro-clusters
dbstream <- DSC_DBSTREAM(r=0.05)
update(dbstream, stream, 500)

# reclustering using single-link and specifying k
hc <- DSC_Hierarchical(k=3, method="single")
recluster(hc, dbstream)
hc
plot(hc, stream, type="both")

# reclustering by specifying height
hc <- DSC_Hierarchical(h=0.2, method="single")
recluster(hc, dbstream)
hc
plot(hc, stream, type="both")

# For comparison we use hierarchical clustering directly on the data
# Note: hierarchical clustering is not a data stream clustering algorithm!
hc <- DSC_Hierarchical(k=3, method="single")
update(hc, stream, 500)
plot(hc, stream)
```

---

**DSC_Kmeans**  
*Kmeans Macro-clusterer*

Description

Class implements the k-means algorithm for reclustering a set of micro-clusters.

Usage

```r
DSC_Kmeans(k, weighted = TRUE, iter.max = 10, nstart = 1,
algorithm = c("Hartigan-Wong", "Lloyd", "Forgy", "MacQueen"),
min_weight = NULL, description=NULL)
```
**DSC_Kmeans**

**Arguments**

- **k**: either the number of clusters, say k, or a set of initial (distinct) cluster centers. If a number, a random set of (distinct) rows in x is chosen as the initial centers.
- **weighted**: use a weighted k-means (algorithm is ignored).
- **iter.max**: the maximum number of iterations allowed.
- **nstart**: if centers is a number, how many random sets should be chosen?
- **algorithm**: character: may be abbreviated.
- **min_weight**: micro-clusters with a weight less than this will be ignored for reclustering.
- **description**: optional character string to describe the clustering method.

**Details**

Please refer to function `kmeans` in **stats** for more details on the algorithm.

Note that this clustering cannot be updated iteratively and every time it is used for (re)clustering, the old clustering is deleted.

**Value**

An object of class `DSC_Kmeans` (subclass of `DSC`, `DSC_R`, `DSC_Macro`)

**See Also**

`DSC`, `DSC_Macro`

**Examples**

```r
stream <- DSD_Gaussians(k=3, noise=0)

# create micro-clusters via sampling
sample <- DSC_Sample(k=20)
update(sample, stream, 500)
sample

# recluster micro-clusters
kmeans <- DSC_Kmeans(k=3)
recluster(kmeans, sample)
plot(kmeans, stream, type="both")

# For comparison we use k-means directly to cluster data
# Note: k-means is not a data stream clustering algorithm
kmeans <- DSC_Kmeans(k=3)
update(kmeans, stream, 500)
plot(kmeans, stream)
```
Abstract Class for Macro Clusterers

Description

Abstract class for all DSC Macro Clusterers.

Details

DSC_Macro cannot be instantiated. Calling DSC_Macro() results in an error.

See Also

DSC

Abstract Class for Micro Clusterers

Description

Abstract class for all DSC Micro Clusterers.

Details

DSC_Micro cannot be instantiated. Calling DSC_Micro() results in an error.

See Also

DSC

Reachability Micro-Cluster Reclusterer

Description

Implementation of reachability clustering (based on DBSCAN’s concept of reachability) to recluster a set of micro-clusters. Two micro-clusters are directly reachable if they are within each other’s epsilon-neighborhood (i.e., the distance between the centers is less than epsilon). Two micro-clusters are reachable if they are connected by a chain of pairwise directly reachable micro-clusters. All mutually reachable micro-clusters are put in the same cluster.

Usage

DSC_Reachability(epsilon, min_weight=NULL, description=NULL)
**Arguments**

- **epsilon**
  - radius of the epsilon-neighborhood.
- **min_weight**
  - micro-clusters with a weight less than this will be ignored for reclustering.
- **description**
  - optional character string to describe the clustering method.

**Details**

Reachability uses internally `dsc_hierarchical` with single link.

Note that this clustering cannot be updated iteratively and every time it is used for (re)clustering, the old clustering is deleted.

**Value**

An object of class `DSC_Reachability`. The object contains the following items:

- **description**
  - The name of the algorithm in the DSC object.
- **Robj**
  - The underlying R object.

**References**


**See Also**

`DSC`, `DSC_Macro`

**Examples**

```r
stream <- DSD_mlbenchGenerator("cassini")

# Recluster micro-clusters from DSC_Sample with reachability
sample <- DSC_Sample(k = 200)
update(sample, stream, 1000)
reach <- DSC_Reachability(epsilon=0.3)
recluster(reach, sample)
plot(reach, stream, type="both")

# For comparison we using reachability clustering directly on data points
# Note: reachability is not a data stream clustering algorithm taking O(n^2) time and space.
reach <- DSC_Reachability(epsilon=0.2)
update(reach, stream, 500)
reach
plot(reach, stream)
```
DSC_Sample

Extract a Fixed-size Sample from a Data Stream

Description

Extracts a sample from a data stream using Reservoir Sampling. The sample is stored as a set of micro-clusters to be compatible with other data DSC stream clustering algorithms.

Usage

DSC_Sample(k = 100, biased = FALSE)

Arguments

k
  the number of points to be sampled from the stream.

biased
  if FALSE then a regular (unbiased) reservoir sampling is used. If true then the sample is biased towards keeping more recent data points (see Details section).

Details

If biased=FALSE then the reservoir sampling algorithm by McLeod and Bellhouse (1983) is used. This sampling makes sure that each data point has the same chance to be sampled. All sampled points will have a weight of 1. Note that this might not be ideal for an evolving stream since very old data points have the same chance to be in the sample as newer points.

If bias=TRUE then sampling prefers newer points using the modified reservoir sampling algorithm 2.1 by Aggarwal (2006). New points are always added. They replace a random point in the reservoir with a probability of reservoir size over k. This an exponential bias function of $2^{-\lambda}$ with $\lambda = 1/k$.

Value

An object of class DSC_Sample (subclass of DSC, DSC_R, DSC_Micro).

References


See Also

DSC, DSC_Micro
DSC_Static

Examples

```r
stream <- DSD_Gaussians(k=3, noise=0.05)
sample <- DSC_Sample(k=20)
update(sample, stream, 500)
sample

# plot micro-clusters
plot(sample, stream)

# reclustering (Kmeans is sufficient because DSC_Sample does not return weights)
kmeans <- DSC_Kmeans()
recluster(kmeans, sample)
plot(kmeans, stream, type="both")

# sample from an evolving stream
stream <- DSD_Benchmark(1)
sample <- DSC_Sample(k=100)
update(sample, stream, 2000)
plot(sample, stream)
# Note: the clusters move from left to right and the sample keeps many outdated points

# use a biased sample to keep more recent data points
stream <- DSD_Benchmark(1)
sample <- DSC_Sample(k=100, biased=TRUE)
update(sample, stream, 2000)
plot(sample, stream)
```

---

DSC_Static

Create as Static Copy of a Clustering

Description

This representation cannot perform clustering anymore, but it also does not need the supporting data structures. It only stores the cluster centers and weights.

Usage

```r
DSC_Static(x, type=c("auto", "micro", "macro"), k_largest=NULL, min_weight=NULL)
```

Arguments

- `x`: The clustering (a DSD object) to copy.
- `type`: which clustering to copy.
- `k_largest`: only copy the k largest (highest weight) clusters.
- `min_weight`: only copy clusters with a weight larger or equal to `min_weight`.  
Value
An object of class DSC_Static (sub class of DSC, DSC_R). The list also contains either DSC_Micro or DSC_Macro depending on what type of clustering was copied.

See Also
DSC, DSC_Micro, DSC_Macro

Examples
stream <- DSD_Gaussians(k=3, noise=0.05)
dstream <- DSC_DStream(gridsize=0.05)
update(dstream, stream, 500)
dstream
plot(dstream, stream)

# create a static copy of the clustering
static <- DSC_Static(dstream)
static
plot(static, stream)

# copy only the 5 largest clusters
static2 <- DSC_Static(dstream, k_largest=5)
static2
plot(static2, stream)

# copy all clusters with a weight of at least .3
static3 <- DSC_Static(dstream, min_weight=.3)
static3
plot(static3, stream)
Details

`update()` runs the micro-clustering stage and if centers/weights are requested the reclustering is automatically performed.

Value

An object of class `DSC_TwoStage` (subclass of `DSC`, `DSC_Macro`).

See Also

`DSC`, `DSC_Macro`

Examples

```r
stream <- DSD_Gaussians(k=3)

# Create a clustering process that uses a window for the online stage and # k-means for the offline stage (reclustering)
win_km <- DSC_TwoStage(
  micro=DSC_Window(horizon=100),
  macro=DSC_Kmeans(k=3)
)

update(win_km, stream, 200)

win_km
plot(win_km, stream, type="both")
evaluate(win_km, stream, assign="macro")
```

---

**DSC_Window**

A sliding window from a Data Stream

Description

Implements a sliding window which keeps a fixed amount (window length) of the most recent data points of the stream.

Usage

`DSC_Window(horizon = 100, lambda=0)`

Arguments

- `horizon` the window length.
- `lambda` decay factor damped window model. `lambda=0` means no dampening.
Details

If $\lambda$ is greater than 0 then the weight uses a damped window model (Zhu and Shasha, 2002). The weight for points in the window follows $2^{-\lambda t}$ where $t$ is the age of the point.

Value

An object of class `DSC_Window` (subclass of `DSC`, `DSC_R`, `DSC_Micro`).

References


See Also

`DSC`, `DSC_Micro`

Examples

```r
stream <- DSD_Gaussians(k=3, d=2, noise=0.05)

window <- DSC_Window(horizon=100)
window

update(window, stream, 200)
window

# plot micro-clusters
plot(window, stream)

# animation for a window using a damped window model. The weight decays
# with a half-life of 25
## Not run:
window <- DSC_Window(horizon=25, lambda=1/25)
animate_cluster(window, stream, horizon=1, n=100, xlim=c(0,1), ylim=c(0,1))

## End(Not run)
```
Details

The DSD class cannot be instantiated, but it serves as a abstract base class from which all DSD objects inherit.

DSD_R inherits form DSD and is the abstract parent class for DSD implemented in R. To create a new R-based implementation there are only two function that needs to be implemented for a new DSD subclass: A creator function (the name should start with DSD_) and a method get_points() for that class.

DSD provides common functionality like print(), plot(), etc.

Note that calling DSD() or DSD_R() results in an error since both are abstract classes.

See Also

animate_data, animate_cluster, cluster, evaluate, get_points, plot, recluster, write_stream

Examples

# create data stream with three clusters in 3-dimensional space
stream <- DSD_Gaussians(k=3, d=3)

# get points from stream
get_points(stream, n=5)

# get points with true cluster assignment
p <- get_points(stream, n=5, cluster=TRUE)
attr(p, "cluster")

# plotting the data (scatter plot matrix, first and third dimension, and first # two principal components)
plot(stream)
plot(stream, dim=c(1,3))
plot(stream, method="pc")

---

DSD_BarsAndGaussians  Data Stream Generator for Bars and Gaussians

Description

A data stream generator which creates the shape of two bars and two Gaussians clusters with different density.

Usage

DSD_BarsAndGaussians(angle = NULL, noise = 0)

Arguments

angle  rotation in degrees. NULL will produce a random rotation.
noise  The amount of noise that should be added to the output.
Value

Returns a DSD_BarsAndGaussians object.

See Also

DSD

Examples

# create data stream with three clusters in 2D
stream <- DSD_BarsAndGaussians(noise=0.1)

# plotting the data
plot(stream)

DSD_Benchmark  Data Stream Generator for Benchmark Data

Description

A data stream generator that generates several dynamic streams intended to be benchmarks to compare data stream clustering algorithms.

Usage

DSD_Benchmark(i=1)

Arguments

i  number of the benchmark.

Details

Currently available benchmarks are 1 and 2.

Value

Returns a DSD object.

See Also

DSD

Examples

stream <- DSD_Benchmark(i=1)
## Not run:
animate_data(stream, n=10000, horizon=100, xlim=c(0,1), ylim=c(0,1))
## End(Not run)
**DSD_Cubes**

**Static Cubes Data Stream Generator**

**Description**

A data stream generator that produces a data stream with static (hyper) cubes filled uniformly with data points.

**Usage**

```
DSD_Cubes(k=2, d=2, center, size, p, noise = 0, noise_range)
```

**Arguments**

- **k**
  Determines the number of clusters.

- **d**
  Determines the number of dimensions.

- **center**
  A matrix of means for each dimension of each cluster.

- **size**
  A k times d matrix with the cube dimensions.

- **p**
  A vector of probabilities that determines the likelihood of generated a data point from a particular cluster.

- **noise**
  Noise probability between 0 and 1. Noise is uniformly distributed within noise range (see below).

- **noise_range**
  A matrix with d rows and 2 columns. The first column contains the minimum values and the second column contains the maximum values for noise.

**Value**

Returns a `DSD_Cubes` object (subclass of `DSD_R`, `DSD`).

**See Also**

- `DSD`

**Examples**

```
# create data stream with three clusters in 3D
stream <- DSD_Cubes(k=3, d=3)

# plotting the data
plot(stream)
```
Description

A data stream generator that produces a data stream with a mixture of static Gaussians.

Usage

DSD_Gaussians(k=2, d=2, mu, sigma, p, separation=0.2, noise=0, noise_range)

Arguments

- **k**: Determines the number of clusters.
- **d**: Determines the number of dimensions.
- **mu**: A matrix of means for each dimension of each cluster.
- **sigma**: A list of length k of covariance matrices.
- **p**: A vector of probabilities that determines the likelihood of generated a data point from a particular cluster.
- **separation**: Minimum distance between cluster centers to reduce overlap between clusters (0-.8).
- **noise**: Noise probability between 0 and 1. Noise is uniformly distributed within noise range (see below).
- **noise_range**: A matrix with d rows and 2 columns. The first column contains the minimum values and the second column contains the maximum values for noise.

Details

DSD_Gaussians creates a mixture of k d-dimensional static Gaussians in approximately unit space. The centers mu and the covariance matrices sigma can be supplied or will be randomly generates. The probability vector p defines for each cluster the probability that the next data point will be chosen from it (defaults to equal probability).

The generation method is similar to the one suggested by Jain and Dubes (1988).

Value

Returns a DSD_Gaussians object (subclass of DSD_R, DSD) which is a list of the defined params. The params are either passed in from the function or created internally. They include:

- **description**: A brief description of the DSD object.
- **k**: The number of clusters.
- **d**: The number of dimensions.
- **mu**: The matrix of means of the dimensions in each cluster.
- **sigma**: The covariance matrix.
- **p**: The probability vector for the clusters.
- **noise**: A flag that determines if or if not noise is generated.
DSD_Memory

References


See Also

DSD

Examples

# create data stream with three clusters in 3-dimensional data space
stream1 <- DSD_Gaussians(k=3, d=3)
plot(stream1)

# create data stream with specified cluster positions, 
# 20% noise in a given bounding box and
# with different densities (1 to 9 between the two clusters)
stream2 <- DSD_Gaussians(k=2, d=2,
  mu=rbind(c(-.5,-.5), c(.5,.5)),
  noise=.2, noise_range=rbind(c(-1,1),c(-1,1)),
  p=c(.1,.9))
plot(stream2)

DSD_Memory

A Data Stream Interface for Data Stored in Memory

Description

This class provides a data stream interface for data stored in memory as matrix-like objects (including data frames). All or a portion of the stored data can be replayed several times.

Usage

DSD_Memory(x, n, k=NA, loop=FALSE,
  class = NULL, description=NULL)

Arguments

x A matrix-like object containing the data. If x is a DSD object then a data frame for n data points from this DSD is created.
n Number of points used if x is a DSD object. If x is a matrix-like object then n is ignored.
k Optional: The known number of clusters in the data
loop Should the stream start over when it reaches the end?
class Vector with the class/cluster label (only used if x is not a DSD object).
description character string with a description.
Details

In addition to regular data.frames other matrix-like objects that provide subsetting with the bracket operator can be used. This includes ffdf (large data.frames stored on disk) from package `ff` and big.matrix from `bigmemory`.

Value

Returns a DSD_Memory object (subclass of DSD_R, DSD).

See Also

`DSD.reset_stream`

Examples

```r
# store 1000 points from a stream
stream <- DSD_Gaussians(k=3, d=2)
replayer <- DSD_Memory(stream, k=3, n=1000)
replayer
plot(replayer)

# creating 2 clusterers of different algorithms
dsc1 <- DSC_DBSTREAM(r=0.1)
dsc2 <- DSC_DStream(gridsize=0.1, Cm=1.5)

# clustering the same data in 2 DSC objects
reset_stream(replayer) # resetting the replayer to the first position
update(dsc1, replayer, 500)
reset_stream(replayer)
update(dsc2, replayer, 500)

# plot the resulting clusterings
reset_stream(replayer)
plot(dsc1, replayer, main="DBSTREAM")
reset_stream(replayer)
plot(dsc2, replayer, main="D-Stream")

### use a data.frame to create a stream (3rd col. contains the assignment)
df <- data.frame(x=runif(100), y=runif(100),
    class=sample(1:3, 100, replace=TRUE))
head(df)

stream <- DSD_Memory(df[,c("x", "y")], class=as.character("class"))
stream
```
**Description**

Creates an evolving DSD that consists of several MGCs.

**Usage**

```r
DSD_MG(dimension = 2, ..., labels=NULL, description=NULL)
```

```r
add_cluster(x, c, label=NULL)
get_clusters(x)
remove_cluster(x, i)
```

**Arguments**

- `dimension` the dimension of the DSD object
- `...` initial set of MGCs
- `x` A `DSD_MG` object.
- `c` The cluster that should be added to the `DSD_MG` object.
- `i` The index of the cluster that should be removed from the `DSD_MG` object.
- `label, labels` integer representing the cluster label. NA represents noise. If labels are not specified, then each new cluster gets a new label.
- `description` An optional string used by `print` to describe the data generator.

**Details**

This DSD is able to generate complex datasets that are able to evolve over a period of time. Its behavior is determined by the MGCs it is composed of.

**See Also**

`MGC_Function`, `MGC_Linear`, `MGC_Noise`, `MGC_Random` for details on the different MGC objects.

**Examples**

```r
## create an empty DSD_MG
stream <- DSD_MG(dim = 2)
stream

## add two clusters
c1 <- MGC_Random(density=50, center=c(50,50), parameter=1, randomness = )
add_cluster(stream, c1)
stream
c2 <- MGC_Noise(density=1, range=rbind(c(-20,120), c(-20,120)))
add_cluster(stream, c2)
stream
get_clusters(stream)
plot(stream, xlim=c(-20,120), ylim=c(-20,120))
```
## DSD_mlbenchData

Stream Interface for Data Sets From mlbench

```r
## Not run:
animate_data(stream, n=5000, xlim=c(-20,120), ylim=c(-20,120))

## End(Not run)

### remove cluster 1
remove_cluster(stream, 1)

get_clusters(stream)
plot(stream, xlim=c(-20,120), ylim=c(-20,120))

### create a more complicated cluster structure (using 2 clusters with the same
### label to form an L shape)
stream <- DSD_MG(dim=2,
    MGC_Static(density=10, center=c(.5,.2), par=c(.4,.2), shape=MGC_Shape_Block),
    MGC_Static(density=10, center=c(.6,.5), par=c(.2,.4), shape=MGC_Shape_Block),
    MGC_Static(density=5, center=c(.39,.53), par=c(.16,.35), shape=MGC_Shape_Block),
    MGC_Noise(density=1, range=rbind(c(0,1), c(0,1)), labels= c(1, 1, 2, NA))
)

plot(stream, xlim=c(0,1), ylim=c(0,1))

### simulate the clustering of a splitting cluster
c1 <- MGC_Linear(dim = 2, keyframelist = list(  
    keyframe(time = 1, dens = 20, center = c(0,0), param = 10),
    keyframe(time = 50, dens = 10, center = c(50,50), param = 10),
    keyframe(time = 100,dens = 10, center = c(50,100),param = 10)
))

### Note: Second cluster appearch at time=50
c2 <- MGC_Linear(dim = 2, keyframelist = list(  
    keyframe(time = 50, dens = 10, center = c(50,50), param = 10),
    keyframe(time = 100,dens = 10, center = c(100,50),param = 10)
))

stream <- DSD_MG(dim = 2, c1, c2)
stream
dbstream <- DSC_DBSTREAM(r=10, lambda=0.1)

## Not run:
purity <- animate_cluster(dbstream, stream, n=2500, type="both",
    xlim=c(-10,120), ylim=c(-10,120), evaluationMeasure="purity", horizon=100)

## End(Not run)
```
Description

Provides a convenient stream interface for data sets from the mlbench package.

Usage

DSD_mlbenchData(data=NULL, loop = FALSE, random = FALSE, scale = FALSE)

Arguments

data	The name of the dataset from mlbench. If missing then a list of all available data sets is shown and returned.
loop	A flag that tells the stream to loop or not to loop over the data frame.
random	A flag that determines if the data should be in a random order.
scale	A flag that determines if the data should be scaled.

Details

The DSD_mlbenchData class is designed to be a wrapper class for data that is held in memory in either a data frame or matrix form. It is a subclass of DSD_Memory.

Call DSD_mlbenchData with a missing value for data to get a list of all available data sets.

Value

Returns a DSD_mlbenchData object which is also of class DSD_Memory.

See Also

DSD, DSD_Memory, reset_stream

Examples

stream <- DSD_mlbenchData("Shuttle")
stream

plot(stream, n=100)

DSD_mlbenchGenerator mlbench Data Stream Generator

Description

A data stream generator class that interfaces data generators found in mlbench.

Usage

DSD_mlbenchGenerator(method, ...)
**Arguments**

```
method       The name of the mlbench data generator.
...
```

**Details**

The `DSD_mlbenchGenerator` class is designed to be a wrapper class for data created by data generators in the mlbench library.

Call `DSD_mlbenchGenerator` with missing method to get a list of available methods.

**Value**

Returns a `DSD_mlbenchGenerator` object (subclass of `DSD_R`, `DSD`) which is a list of the defined parameters. The parameters are either passed in from the function or created internally. They include:

```
description       The name of the class of the DSD object.
method            The name of the mlbench data generator.
variables         The variables for the mlbench data generator.
```

**See Also**

`DSD`

**Examples**

```r
stream <- DSD_mlbenchGenerator(method="cassini")
plot(stream, n=500)
```

---

**Description**

A DSD class that reads a data stream from a file or any R connection.

**Usage**

```r
DSD_ReadCSV(file, k=NA, d=NA, take=NULL,
            class=NULL, loop=FALSE, sep="", header=FALSE, skip=0, ...)
close_stream(dsd)
```
**Arguments**

- **file**: A file/URL or an open connection.
- **k**: Number of true clusters, if known.
- **d**: Number of dimensions (only used for print).
- **take**: Indices of columns to extract from the file.
- **class**: Column index for the class attribute/cluster label.
- **loop**: If enabled, the object will loop through the stream when the end has been reached. If disabled, the object will warn the user upon reaching the end.
- **sep**: The character string that separates dimensions in data points in the stream.
- **header**: Does the first line contain variable names?
- **skip**: The number of lines of the data file to skip before beginning to read data.
- **...**: Further arguments are passed on to readNtable. This can for example be used for encoding, quotes, etc.
- **dsd**: A object of class DSD_ReadCSV.

**Details**

DSD_ReadCSV uses read.table() to read in data from an R connection. The connection is responsible for maintaining where the stream is currently being read from. In general, the connections will consist of files stored on disk but have many other possibilities (see connection).

The position in the file can be reset to the beginning using reset_stream(). The connection can be closed using close_stream().

**Value**

An object of class DSD_ReadCSV (subclass of DSD_R, DSD).

**See Also**

DSD, reset_stream, read.table.

**Examples**

```r
# creating data and writing it to disk
stream <- DSD_Gaussians(k=3, d=5)
write_stream(stream, "data.txt", n=100, sep=" ")

# reading the same data back (as a loop)
stream2 <- DSD_ReadCSV("data.txt", sep=" ", loop=TRUE)
stream2

# clean up
close_stream(stream2)
file.remove("data.txt")

# example with a part of the kddcup1999 data (take only cont. variables)
file <- system.file("examples", "kddcup10000.data.gz", package="stream")
```
stream <- DSD_ReadCSV(gzfile(file),
take=c(1, 5, 6, 8:11, 13:20, 23:41), class=42, k=7)
stream
get_points(stream, 5)

# plot 100 points (projected on the first two principal components)
plot(stream, n=100, method="pc")
close_stream(stream)

---

DSD_ReadDB

Read a Data Stream from an open DB Query

Description

A DSD class that reads a data stream from an open DB result set from a relational database with using R’s database interface (DBI).

Usage

DSD_ReadDB(result, k=NA, class=NULL, description=NULL)

Arguments

result An open DBI result set.

k Number of true clusters, if known.

class column index for the class/cluster assignment.

description a character string describing the data.

Details

This class provides a streaming interface for result sets from a database with via DBI. You need to connect to the database and submit a SQL query using dbGetQuery() to obtain a result set. Make sure that your query only includes the columns that should be included in the stream. Do not forget to close the result set and the database connection.

Value

An object of class DSD_ReadDB (subclass of DSD_R, DSD).

See Also

DSD, dbGetQuery
**Examples**

```r
### create a data base with a table with 3 Gaussians
library("RSQLite")
con <- dbConnect(RSQLite::SQLite(), ":memory:"

points <- get_points(DSD_Gaussians(k=3, d=2), 600, class = TRUE)
head(points)

dbWriteTable(con, "gaussians", points)

### prepare a query result set
res <- dbSendQuery(con, "SELECT V1, V2, class FROM gaussians")
res

### create a stream interface to the result set
stream <- DSD_ReadDB(res, k=3, class = 3)

### get points
get_points(stream, 5)
plot(stream)

### clean up
dbClearResult(res)
dbDisconnect(con)
```

---

**DSD_ScaleStream**: Scale a Stream from a DSD

**Description**

Make an unscaled data stream into a scaled data stream.

**Usage**

```r
DSD_ScaleStream(dsd, center=TRUE, scale=TRUE, n=1000, reset=FALSE)
```

**Arguments**

- `dsd`: A object of class `DSD` that will be scaled.
- `center`, `scale`: logical or a numeric vector of length equal to the number of columns used for centering/scaling (see function `scale`).
- `n`: The number of points used to creating the centering/scaling
- `reset`: Try to reset the stream to its beginning after taking `n` points for scaling.

**Details**

`sparse_stream()` estimates the values for centering and scaling (see `scale` in `base`) using `n` points from the stream.
Value

An object of class DSD_ScaleStream (subclass of DSD_R, DSD).

See Also

DSD, reset_stream, scale in base.

Examples

stream <- DSD_Gaussians(k=3, d=3)
plot(stream)

# scale stream using 100 points
stream_scaled <- DSD_ScaleStream(stream, n=100)
plot(stream_scaled)

---

DSD_Target  Target Data Stream Generator

Description

A data stream generator that generates a data stream in the shape of a target. It has a single Gaussian cluster in the center and a ring that surrounds it.

Usage

DSD_Target(center_sd = 0.05, center_weight = 0.5, ring_r = 0.2,
            ring_sd = 0.02, noise = 0)

Arguments

center_sd standard deviation of center
center_weight proportion of points in center
ring_r average ring radius
ring_sd standard deviation of ring radius
noise proportion of noise

Details

DSD_Target is a DSD generator for stream data. It has been implemented entirely in R, so there is no computational overhead with communicating to the Java Runtime Interface (JRI) or native C code. This DSD will produce a singular Gaussian cluster in the center with a ring around it.
Value

Returns a DSD_Target object which is a list of the defined params. The params are either passed in from the function or created internally. They include:

- **description**: A brief description of the DSD object.
- **k**: The number of clusters.
- **d**: The number of dimensions.

See Also

DSD

Examples

```r
# create data stream with three clusters in 2D
stream <- DSD_Target()
# plotting the data
plot(stream)
```

DSD_UniformNoise

Uniform Noise Data Stream Generator

Description

This generator produces uniform noise in a d-dimensional unit (hyper) cube.

Usage

```
DSD_UniformNoise(d=2, range=NULL)
```

Arguments

- **d**: Determines the number of dimensions.
- **range**: A matrix with two columns and d rows giving the minimum and maximum for each dimension. Defaults to the range of [0, 1].

Value

Returns a DSD_UniformNoise object (subclass of DSD_R, DSD).

See Also

DSD
Examples

```r
# create data stream with three clusters in 2D
stream <- DSD.UniformNoise(d=2)
plot(stream, n=100)

# specify a different range for each dimension
stream <- DSD.UniformNoise(d=3, range=rbind(c(0,1), c(0,10), c(0,5)))
plot(stream, n=100)
```

---

**DSFP**  
*Abstract Class for Frequent Pattern Mining Algorithms for Data Streams*

---

**Description**

Abstract class for frequent pattern mining algorithms for data streams. Currently, `stream` does not implement frequent pattern mining algorithms.

**See Also**

`DST`

---

**DSO**  
*Data Stream Operator Base Classes*

---

**Description**

Abstract base classes for all DSO (Data Stream Operator) classes.

**Details**

The DSO class cannot be instantiated (calling DSO() produces errors), but it serve as a base class from which other DSO classes inherit.

Data stream operators use update() to process new data from the stream. The result of the operator can be obtained via get_points() and get_weights().

**See Also**

`update, get_points, get_weights, DSO_Window, DSO_Sample`
**DSO_Sample**  
*Sampling from a Data Stream (Data Stream Operator)*

**Description**

Extracts a sample form a data stream using Reservoir Sampling.

**Usage**

```r
DSO_Sample(k = 100, biased = FALSE)
```

**Arguments**

- `k`  
  the number of points to be sampled from the stream.

- `biased`  
  if `false` then a regular (unbiased) reservoir sampling is used. If `true` then the sample is biased towards keeping more recent data points (see Details section).

**Details**

If `biased=False` then the reservoir sampling algorithm by McLeod and Bellhouse (1983) is used. This sampling makes sure that each data point has the same chance to be sampled. All sampled points will have a weight of 1. Note that this might not be ideal for an evolving stream since very old data points have the same chance to be in the sample as newer points.

If `bias=TRUE` then sampling prefers newer points using the modified reservoir sampling algorithm 2.1 by Aggarwal (2006). New points are always added. They replace a random point in the reservoir with a probability of reservoir size over `k`. This an exponential bias function of $2^{-\lambda} \lambda = 1/k$.

**Value**

An object of class `DSO_Sample` (subclass of `DSO`).

**References**


**See Also**

`DSO`
Examples

```r
stream <- DSD_Gaussians(k=3, noise=0.05)
sample <- DSO_Sample(k=20)
update(sample, stream, 500)
sample

# plot points in sample
plot(get_points(sample))
```

**Sliding Window (Data Stream Operator)**

**Description**

Implements a sliding window data stream operator which keeps a fixed amount (window length) of the most recent data points of the stream.

**Usage**

```r
DSO_Window(horizon = 100, lambda=0)
```

**Arguments**

- `horizon` the window length.
- `lambda` decay factor damped window model. `lambda=0` means no dampening.

**Details**

If `lambda` is greater than 0 then the weight uses a damped window model (Zhu and Shasha, 2002). The weight for points in the window follows $2^{-\lambda t}$ where $t$ is the age of the point.

**Value**

An object of class `DSO_Window` (subclass of `DSO`).

**References**


**See Also**

`DSO`
Examples

stream <- DSD.Gaussians(k=3, d=2, noise=0.05)

window <- DSO_Window(horizon=100)

update(window, stream, 200)

# plot points in window
plot(get_points(window))

---

**DST**

*Abstract Base Class for All Data Stream Mining Tasks*

---

**Description**

Abstract base class for all data stream mining tasks. Current tasks are data stream clustering DSC, classification on data streams DSClassify and frequent pattern mining on data streams DSFP.

**See Also**

DSC, DSClassify, DSFP

---

**evaluate**

*Evaluate Clusterings*

---

**Description**

Gets evaluation measures for micro or macro-clusters from a DSC object given the original DSD object.

**Usage**

```r
evaluate(dsc, dsd, measure, n = 100, type=c("auto", "micro", "macro"),
assign="micro", assignmentMethod=c("auto", "model", "nn"), ...)
evaluate_cluster(dsc, dsd, macro=NULL, measure, n = 1000,
type=c("auto", "micro", "macro"),
assign="micro", assignmentMethod=c("auto", "model", "nn"),
horizon=100, verbose=FALSE, ...)
```
Arguments

dsc  The DSC object that the evaluation measure is being requested from.
dsd  The DSD object that holds the initial training data for the DSC.
measure Evaluation measure(s) to use. If missing then all available measures are returned.
n   The number of data points being requested.
type Use micro- or macro-clusters for evaluation. Auto used the class of dsc to decide.
assign Assign points to micro or macro-clusters?
assignmentMethod How are points assigned to clusters for evaluation (see get_assignment)?
macro A DSC_macro object for reclustering.
horizon Evaluation is done using horizon many previous points (see detail section).
verbose report progress?
... Unused arguments are ignored.

Details

For evaluation each data points are assigned to its nearest cluster using Euclidean distance to the cluster centers. Then for each cluster the majority class is determined. Based on the majority class several evaluation measures can be computed.

For evaluate_cluster the most commonly used measure introduced by Aggarwal et al (2003) is applied. The data points used for evaluation are taken from the previously clustered horizon. Many evaluation measures are calculated with code from the packages cluster, clue and fpc. Detailed documentation can be found in these packages (see Section See Also.)

The following information items are available:

- "numMicroClusters" number of micro-clusters
- "numMacroClusters" number of macro-clusters
- "numClasses" number of classes

The following internal evaluation measures are available:

- "SSQ" sum of squares (actual noise points are excluded)
- "silhouette" average silhouette width (points that are actual noise and predicted to be noise are excluded) (cluster)
- "average.between" average distance between clusters (fpc)
- "average.within" average distance within clusters (fpc)
- "max.diameter" maximum cluster diameter (fpc)
- "min.separation" minimum cluster separation (fpc)
- "ave.within.cluster.ss" a generalization of the within clusters sum of squares (half the sum of the within cluster squared dissimilarities divided by the cluster size) (fpc)
- "g2" Goodman and Kruskal’s Gamma coefficient (fpc)
• "pearson_gamma" correlation between distances and a 0-1-vector where 0 means same cluster, 1 means different clusters (fpc)
• "dunn" Dunn index (minimum separation / maximum diameter) (fpc)
• "dunn2" minimum average dissimilarity between two cluster / maximum average within cluster dissimilarity (fpc)
• "entropy" entropy of the distribution of cluster memberships (fpc)
• "wb.ratio" average.within/average.between (fpc)

The following external evaluation measures are available:

• "precision", "recall", "F1" F1. A true positive (TP) decision assigns two points in the same true cluster also to the same cluster, a true negative (TN) decision assigns two points from two different true clusters to two different clusters. A false positive (FP) decision assigns two points from the same true cluster to two different clusters. A false negative (FN) decision assigns two points from the same true cluster to different clusters.
  
  precision = TP/(TP+FP)
  recall = TP/(TP+FN)
  The F1 measure is the harmonic mean of precision and recall.
• "purity" Average purity of clusters. The purity of each cluster is the proportion of the points of the majority true group assigned to it (see Cao et al. (2006))
• "Euclidean" Euclidean dissimilarity of the memberships (see Dimitriadou, Weingessel and Hornik (2002)) (clue)
• "Manhattan" Manhattan dissimilarity of the memberships (clue)
• "Rand" Rand index (see Rand (1971)) (clue)
• "cRand" Adjusted Rand index (see Hubert and Arabie (1985)) (clue)
• "NMI" Normalized Mutual Information (see Strehl and Ghosh (2002)) (clue)
• "KP" Katz-Powell index (see Katz and Powell (1953)) (clue)
• "angle" maximal cosine of the angle between the agreements (clue)
• "diag" maximal co-classification rate (clue)
• "FM" Fowlkes and Mallows’s index (see Fowlkes and Mallows (1983)) (clue)
• "Jaccard" Jaccard index (clue)
• "PS" Prediction Strength (see Tibshirani and Walter (2005)) (clue)
• "vi" variation of information (VI) index (fpc)

Many measures are the average over all clusters. For example, purity is the average purity over all clusters.

For DSC_Micro objects, data points are assigned to micro-clusters and then each micro-cluster is evaluated. For DSC_Macro objects, data points by default (assign="micro") also assigned to micro-clusters, but these assignments are translated to macro-clusters. The evaluation is here done for macro-clusters. This is important when macro-clustering is done with algorithms which do not create spherical clusters (e.g. hierarchical clustering with single-linkage or DBSCAN) and this assignment to the macro-clusters directly (i.e., their center) does not make sense.

Using type and assign, the user can select how to assign data points and ad what level (micro or macro) to evaluate.
Many of the above measures are implemented package clue in function cl_agreement().
evaluate_cluster() is used to evaluate an evolving data stream using the method described by Wan et al. (2009). Of the n data points horizon many points are clustered and then the evaluation measure is calculated on the same data points. The idea is to find out if the clustering algorithm was able to adapt to the changing stream.

Value

evaluate returns an object of class stream_eval which is a numeric vector of the values of the requested measures and two attributes, “type” and “assign”, to see at what level the evaluation was done.

References

L. Katz and J. H. Powell (1953). A proposed index of the conformity of one sociometric measurement to another. Psychometrika, 18, 249-256.
L Wan, W.K. Ng, X.H. Dang, P.S. Yu and K. Zhang (2009). Density-Based Clustering of Data Streams at Multiple Resolutions, 3(3).

See Also

animate_cluster, cl_agreement in clue, cluster.stats in fpc, silhouette in cluster.

Examples

stream <- DSD_Gaussians(k=3, d=2)
dstream <- DSC_DStream(gridsize=0.05, Cm=1.5)
update(dstream, stream, 500)
plot(dstream, stream)
# Evaluate micro-clusters
# Note: we use here only n=500 points for evaluation to speed up execution
evaluate(dstream, stream, measure=c("numMicro", "numMacro", "purity", "crand", "SSQ"), n=100)

# DStream also provides macro clusters. Evaluate macro clusters with type="macro"
plot(dstream, stream, type="macro")
evaluate(dstream, stream, type = "macro",
  measure=c("numMicro", "numMacro", "purity", "crand", "SSQ"), n=100)

# Points are by default assigned to the closest micro clusters for evaluation.
# However, points can also be assigned to the closest macro-cluster using
# assign="macro".
evaluate(dstream, stream, type = "macro", assign = "macro",
  measure=c("numMicro", "numMacro", "purity", "crand", "SSQ"), n=100)

# Evaluate an evolving data stream
stream <- DSD_Benchmark()
dstream <- DSC_DStream(gridsize=0.05, lambda=0.1)
evaluate_cluster(dstream, stream, type="macro", assign=micro",
  measure=c("numMicro", "numMacro", "purity", "crand"),
  n=600, horizon=100)

## Not run:
# animate the clustering process
reset_stream(stream)
dstream <- DSC_DStream(gridsize=0.05, lambda=0.1)
animate_cluster(dstream, stream, n=5000, horizon=100,
  evaluationMeasure=c("crand"), evaluationType="macro", evaluationAssign="macro",
  type="both", xlim=c(0,1), ylim=c(0,1))

## End(Not run)

---

**get_assignment**

**Assignment Data Points to Clusters**

**Description**

Get the assignment of data points to clusters in a DSC (nearest-neighbor).

**Usage**

```r
get_assignment(dsc, points, type=c("auto", "micro", "macro"),
  method="auto", ...)
```

**Arguments**

- **dsc**
  The DSC object with the clusters for assignment.
- **points**
  The points to be assigned as a data.frame.
- **type**
  Use micro- or macro-clusters in DSC for assignment. Auto used the class of dsc to decide.
get_centers

method assignment method "model" uses the assignment model of the underlying algorithm (unassigned points return NA). "nn" performs nearest neighbor assignment using Euclidean distance. "auto" uses model if it is available and defaults to nn otherwise.

Additional arguments are passed on.

Details

Each data point is assigned either using the original model’s assignment rule or Euclidean nearest neighbor assignment.

Value

A vector containing the assignment of each point. NA means that a data point was not assigned to a cluster.

See Also
dsc

Examples

stream <- DSD_Gaussians(k=3, d=2, noise=.05)
dbstream <- DSC_DBSTREAM(r=.1, noise=.05)
update(dbstream, stream, n=100)

# find the assignment for the next 100 points to
# micro-clusters in dsc. This uses the model’s assignment function
points <- get_points(stream, n=100)
a <- get_assignment(dbstream, points)
a

# show the MC assignment areas. Assigned points as blue circles and
# the unassigned points as red dots
plot(dbstream, stream, assignment= TRUE, type="none")
points(points[!is.na(a),], col="blue")
points(points[is.na(a),], col="red", pch=20)

# use nearest neighbor assignment instead
get_assignment(dbstream, points, method="nn")

get_centers

Get Cluster Centers from a DSC

Description

Gets the cluster centers (micro- or macro-clusters) from a DSC object.
get_copy

Usage

get_centers(x, type=c("auto", "micro", "macro"), ...)

Arguments

x The DSC object the centers are being requested from.
type get centers for micro- or macro-clusters. Auto used the class of dsc to decide.
... Additional parameters.

Value

A data.frame with the micro- or macro-cluster centers as rows.

See Also

DSC

Examples

stream <- DSD_Gaussians(k=3, d=2)
dstream <- DSC_DStream(gridsize=.1)
update(dstream, stream, 500)
dstream

# getting the micro-cluster centers
get_centers(dstream)

# D-Stream also has macro-clusters
get_centers(dstream, type="macro")

get_copy Create a Deep Copy of a DSC Object

Description

DSC objects contain reference classes or Java data structures (for MOA). Therefore, we provide a mechanism to create deep copies.

Usage

get_copy(x)

Arguments

x The DSC object being copied.
get_points

Value

A deep copy of the original DSC.

See Also

DSC

get_points

Get Points from a Data Stream Generator

Description

Gets points from a DSD object.

Usage

get_points(x, n=1, outofpoints=c("stop", "warn", "ignore"), ...)

Arguments

x
The DSD object.

n
Request up to n points from the stream.

outofpoints
Action taken if less than n data points are available. The default is to stop with an error. For warn and ignore all available (possibly zero) points are returned.

... Additional parameters to pass to get_points() implementations.

Details

Each DSD object has a unique way for returning data points, but they all are called through the generic function, get_points(). This is done by using the S3 class system. See the man page for the specific DSD class on the semantics for each implementation of get_points().

Value

Returns a matrix of $x \times d$ columns and n rows.

See Also

DSD

Examples

stream <- DSD_Gaussians()
get_points(stream, 100)
get_weights  Get Cluster Weights

Description

Get the weights of the clusters in the DSC

Usage

get_weights(x, type=c("auto", "micro", "macro"), scale=NULL, ...)

Arguments

x The DSC object the weights are being requested from.
type Return weights of micro- or macro-clusters in x. Auto uses the class of x to
decide.
scale a range (from, to) to scale the weights. Returns by default the raw weights.
... Additional arguments are passed on.

Details

The cluster weights are typically a function of how many points were assigned to each cluster.

Value

A vector containing the weight of each micro-cluster or macro-cluster. Internally the call is dele-
gated to the appropriate get_microweights() or get_macroweights() method.

See Also

DSC

Examples

stream <- DSD_Gaussians(k=3, d=2)
dstream <- DSC_DStream(gridsize=1)
update(dstream, stream, 500)
dstream

# getting the micro-cluster weights
get_weights(dstream)

# D-Stream also has macro-clusters
get_weights(dstream, type="macro")
Description

Creates an evolving cluster for a DSD_MG.

Usage

MGC_Static(density, center, parameter, shape = NULL)
MGC_Function(density, center, parameter, shape = NULL)
MGC_Random(density, center, parameter, randomness = 1, shape = NULL)
MGC_Noise(density, range)

MGC_Linear(dimension = 2, keyframelist = NULL, shape = NULL)
keyframe(time, density, center, parameter, reset = FALSE)
add_keyframe(x, time, density, center, parameter, reset = FALSE)
get_keyframes(x)
remove_keyframe(x, time)

Arguments

center A list that defines the center of the cluster. The list should have a length equal to the dimensionality. For MGC_Function, this list consists of functions that define the movement of the cluster. For MGC_Random, this attribute defines the beginning location for the MGC before it begins moving.
density The density of the cluster. For MGC_Function, this attribute is a function and defines the density of a cluster at a given timestamp.
dimension Dimensionality of the data stream.
keyframelist a list of keyframes to initialize the MGC_Linear object with.
parameter Parameters for the shape. For the default shape MGC_Shape_Gaussian the parameter is the standard deviation, one per dimension. If a single value is specified then it is recycled for all dimensions.
randomness The maximum amount the cluster will move during one time step.
range The area in which the noise should appear.
reset Should the cluster reset to the first keyframe (time 0) after this keyframe is finished?
shape A function creating the shape of the cluster. It gets passed on the parameters argument from above. Available functions are MGC_Shape_Gaussian (the parameters are a vector containing standard deviations) and MGC_Shape_Block (parameters are the dimensions of the uniform block).
time The time stamp the keyframe should be located or which keyframe should be removed.
x An object of class MGC_Linear.
Details

An MGC describes a single cluster for use within a DSD_MG. There are currently four different MGCs that allow a user to express many different behaviors within a single data stream.

An MGC_Linear creates an evolving Gaussian cluster for a DSD_MG who’s behavior is determined by several keyframes. Keyframes can be added and removed.

An MGC_Function allows for a creation of a DSD_MG that is defined by functions of time.

An MGC_Random allows for a creation of a DSD_MG that moves randomly.

An MGC_Noise allows for a creation of noise within a DSD_MG.

See Also

DSD_MG for details on how to use an MGC within a DSD

Examples

```r
### Two static clusters
stream <- DSD_MG(dim=2,
  MGC_Static(den = 1, center=c(1, 0), par=.1),
  MGC_Static(den = 1, center=c(2, 0), par=.4, shape=MGC_Shape_Block)
)

plot(stream)

### Example of several MGC_Randoms
stream <- DSD_MG(dimension=2,
  MGC_Random(den = 100, center=c(1, 0), par=.1, rand=.1),
  MGC_Random(den = 100, center=c(2, 0), par=.4, shape=MGC_Shape_Block, rand=.1)
)

## Not run:
  animate_data(stream, 2500, xlim=c(0,3), ylim=c(-2,2), horizon=100)

## End(Not run)

### Example of several MGC_Functions
stream <- DSD_MG(dim = 2)

### block-shaped cluster moving from bottom-left to top-right increasing size
c1 <- MGC_Function(
  density = function(t){100},
  parameter = function(t){1*t},
  center = function(t) c(t,t),
  shape = MGC_Shape_Block
)
add_cluster(stream,c1)

### cluster moving in a circle (default shape is Gaussian)
c2 <- MGC_Function(
  density = function(t){25},
  parameter = function(t){sin(2*pi*t)},
  center = function(t) c(t,t),
  shape = MGC_Shape_Circle
)
add_cluster(stream,c2)
```
parameter = function(t){
    center= function(t) c(sin(t/10)*50+50, cos(t/10)*50+50)
}
add_cluster(stream,c2)

## Not run:
animate_data(stream,10000,xlim=c(-20,120),ylim=c(-20,120), horizon=100)

## End(Not run)

### Example of several mgc_linears: A single cluster splits at time 50 into two.
### Note that c2 starts at time=50!
stream <- DSD_MG(dim = 2)
c1 <- MGC_Linear(dim = 2)
add_keyframe(c1, time=1, dens=50, par=5, center=c(0,0))
add_keyframe(c1, time=50, dens=50, par=5, center=c(50,50))
add_keyframe(c1, time=100, dens=50, par=5, center=c(50,100))
add_cluster(stream,c1)

c2 <- MGC_Linear(dim = 2, shape=MGC_Shape_Block)
add_keyframe(c2, time=50, dens=25, par=c(10,10), center=c(50,50))
add_keyframe(c2, time=100, dens=25, par=c(30,30), center=c(100,50))
add_cluster(stream,c2)

## Not run:
animate_data(stream,5000,xlim=c(0,100),ylim=c(0,100), horiz=100)

## End(Not run)

### two fixed and a moving cluster
stream <- DSD_MG(dim = 2,
    MGC_Static(dens=1, par=.1, center=c(0,0)),
    MGC_Static(dens=1, par=.1, center=c(1,1)),
    MGC_Linear(dim=2,list(
        keyframe(time = 0, dens=1, par=.1, center=c(0,0)),
        keyframe(time = 1000, dens=1, par=.1, center=c(1,1)),
        keyframe(time = 2000, dens=1, par=.1, center=c(0,0), reset=TRUE)
    )))
noise <- MGC_Noise(dens=.1, range=rbind(c(-.2,1.2),c(-.2,1.2)))
add_cluster(stream, noise)

## Not run:
animate_data(stream, n=2000*3.1, xlim=c(-.2,1.2), ylim=c(-.2,1.2), horiz=200)

## End(Not run)
microToMacro

Description

Translates micro-cluster ids into macro-cluster ids for a DSC_Macro object.

Usage

microToMacro(x, micro=NULL)

Arguments

x a DSC_Macro object that also contains information about micro-clusters.
micro A vector with micro-cluster ids. If NULL then the assignments for all micro-clusters in x are returned.

Value

A vector of the same length as micro with the macro-cluster ids.

See Also

DSC_Macro

Examples

stream <- DSD_Gaussians(k=3, d=2, noise=0.05, p=c(.2,.4,.6))

# recluster a micro-clusters
micro <- DSC_DStream(gridsize=0.05)
update(micro, stream, 500)

macro <- DSC_Kmeans(k=3)
recluster(macro, micro)

# translate all micro-cluster ids
microToMacro(macro)

# plot some data points in gray
plot(stream, col="gray", cex=.5, xlim=c(0,1), ylim=c(0,1))

# add micro-clusters and use the macro-cluster ids as color and weights as size
points(get_centers(macro, type="micro"),

  col=microToMacro(macro),
  cex=get_weights(macro, type="micro", scale=c(.5,3)))

# add macro-cluster centers (size is weight)
points(get_centers(macro, type="macro"),

  cex = get_weights(macro, type="macro", scale=c(2,5)),
  pch=3,lwd=3, col=1:3)
Description

Returns the number of micro-clusters from the DSC object.

Usage

nclusters(x, type=c("auto", "micro", "macro"), ...)

Arguments

x

A DSC object.

type

Return the number of micro- or macro-clusters in DSC. Auto used the class of dsc to decide.

... Additional arguments are passed on.

Value

An integer; the number of micro- or macro-clusters in the clustering.

See Also

DSC

Examples

# setting up the objects
stream <- DSD_Gaussians(k=3)
dstream <- DSC_DStream(gridsize=0.1)
update(dstream, stream, 500)

# retrieving the results
get_centers(dstream)
nclusters(dstream)
Description

Methods to plot data stream data and clusterings.

Usage

```r
## S3 method for class 'DSD'
plot(x, n = 500, col = NULL,
     pch = NULL, ..., method = "pairs", dim = NULL, alpha = 0.6)
## S3 method for class 'DSC'
plot(x, dsd = NULL, n = 500,
     col_points = NULL, col_clusters = c("red", "blue"),
     weights = TRUE, scale = c(1, 5), cex = 1, pch = NULL,
     method = "pairs", dim = NULL, type = c("auto", "micro", "macro", "both"),
     assignment = FALSE, ...)
```

Arguments

- `x` the DSD or DSC object to be plotted.
- `dsd` a DSD object to plot the data in the background.
- `n` number of plots taken from the dsd to plot.
- `col, col_points, col_clusters` colors used for plotting.
- `weights` the size of the symbols for micro- and macro-clusters represents its weight.
- `scale` range for the symbol sizes used.
- `cex` size factor for symbols.
- `pch` symbol type.
- `method` method used for plotting: "pairs" (pairs plot), "scatter" (scatter plot) or "pc" (plot first 2 principal components).
- `dim` an integer vector with the dimensions to plot. If NULL then for methods "pairs" and "pc" all dimensions are used and for "scatter" the first two dimensions are plotted.
- `alpha` alpha shading used to plot the points.
- `type` Plot micro- or macro-clusters. Auto used the class of dsc to decide.
- `assignment` logical; show assignment area of micro-clusters.
- `...` further arguments are passed on to plot or pairs in `graphics`.

See Also

`DSC`, `DSD`
Examples

stream <- DSD_Gaussians(k=3, d=3)

## plot data
plot(stream, n=500)
plot(stream, method="pc", n=500)
plot(stream, method="scatter", dim=c(1,3), n=500)

## create and plot micro-clusters
dstream <- DSC_DStream(gridsize=0.1)
update(dstream, stream, 500)
plot(dstream)

## plot with data, projected on the first two principal components
## and dimensions 2 and 3
plot(dstream, stream)
plot(dstream, stream, method="pc")
plot(dstream, stream, dim=c(2,3))

## plot micro and macro-clusters
plot(dstream, stream, type="both")


---

prune_clusters  
Prune Clusters from a Clustering

Description

Creates a (static) copy of a clustering where a fraction of the weight or the number of clusters with the lowest weights were pruned.

Usage

prune_clusters(dsc, threshold= 0.05, weight = TRUE)

Arguments

dsc  
The DSC object to be pruned.

threshold  
The numeric vector of probabilities for the quantile.

weight  
should a fraction of the total weight in the clustering be pruned? Otherwise a fraction of clusters is pruned.

Value

Returns an object of class DSC_Static.

See Also

DSC_Static
**recluster**

**Examples**

```r
# # 3 clusters with 10% noise
stream <- DSD_Gaussians(k=3, noise=0.1)

dbstream <- DSC_DBSTREAM(r=0.1)
update(dbstream, stream, 500)
plot(dbstream)

# prune lightest micro-clusters for 20% of the weight of the clustering
static <- prune_clusters(dbstream, threshold=0.2)
plot(static, stream)
```

---

**Description**

Use a macro clustering algorithm to reclustering micro-clusters into a final clustering.

**Usage**

```r
recluster(macro, dsc, type="auto", ...)
```

**Arguments**

- `macro`: a macro clustering algorithm (class "DSC_Macro")
- `dsc`: a DSC object containing micro-clusters.
- `type`: controls which clustering is used from `dsc` (typically micro-clusters).
- `...`: additional arguments passed on.

**Details**

Takes centers and weights of the micro-clusters and applies the macro clustering algorithm.

**Value**

The object macro is altered and contains the clustering.

**Examples**

```r
set.seed(0)
### create a data stream and a micro-clustering
stream <- DSD_Gaussians(k=3, d=3)

sample <- DSC_Sample(k=50)
update(sample, stream, 500)
```
sample

### recluster using k-means
kmeans <- DSC_Kmeans(k=3)
recluster(kmeans, sample)

### plot clustering
plot(kmeans, stream, main="Macro-clusters (Sampling + k-means")

---

**reset_stream**

**Reset a Data Stream to its Beginning**

**Description**
Resets the counter in a DSD object to the beginning or any other position in the stream.

**Usage**
reset_stream(dsd, pos = 1)

**Arguments**
- `dsd`: An object of class a subclass of DSD which implements a reset function.
- `pos`: Position in the stream (the beginning of the stream is position 1).

**Details**
Resets the counter of the stream object. For example, for DSD_Memory, the counter stored in the environment variable is moved back to 1. For DSD_ReadCSV objects, this is done by calling `seek()` on the underlying connection.

**See Also**
- `DSD_ReadCSV`, `DSD_MG`, `DSD_ScaleStream`, `DSD_Memory`

**Examples**

```r
# initializing the objects
stream <- DSD_Gaussians(k=3, d=2)
replayer <- DSD_Memory(stream, 100)
replayer

p <- get_points(replayer, 50)
replayer

# reset replayer to the begining of the stream
reset_stream(replayer)
replayer
```
**update**

# set replayer to position 21
reset_stream(replayer, pos=21)
replayer

---

**update**  
**Update a Data Stream Clustering Model**

**Description**

Update a clustering model by clustering a number of input points from a data stream into a clustering object.

**Usage**

```r
## S3 method for class 'DSC_R'
update(object, dsd, n = 1, verbose = FALSE, block=10000L, ...)
## S3 method for class 'DSC_TwoStage'
update(object, dsd, n = 1, verbose = FALSE,
       block=10000L, ...)
## S3 method for class 'DSO_Sample'
update(object, dsd, n = 1, verbose = FALSE, ...)
## S3 method for class 'DSO_Window'
update(object, dsd, n = 1, verbose = FALSE, ...)
```

**Arguments**

- `object`: an object of a subclass of DST (data stream mining task).
- `dsd`: a DSD object (data stream).
- `n`: number of points to cluster.
- `verbose`: report progress.
- `block`: maximal number of data points passed on at once to the algorithm. This only is used since R loops are very slow.
- `...`: extra arguments for clusterer.

**Details**

`update` takes `n` times a single data points out of the DSD updates the model in `object`. Note that `update` directly modifies the object (which is a reference class) and thus the result does not need to be reassigned to the object name.

**Value**

The updated model is returned invisibly for reassignment (however, this is not necessary).

To obtain the updated model for a `DSC` (data stream clustering model), call `get_centers()` on the `DSC` object.
See Also

DSC, DSD, and animation for producing an animation of the clustering process.

Examples

```r
stream <- DSD_Gaussians(k=3)
dstream <- DSC_DStream(gridsize=.05)

update(dstream, stream, 500)
plot(dstream, stream)
```

---

**write_stream**  

**Write a Data Stream to a File**

**Description**

Writes points from a data stream DSD object to a file or a connection.

**Usage**

```r
write_stream(dsd, file, n=100, block=100000L,
class=FALSE, sep=",",
col.names=FALSE, row.names=FALSE, ...)
```

**Arguments**

- `dsd`  
  The DSD object that will generate the data points for output.
- `file`  
  A file name or a R connection to be written to.
- `n`  
  The number of data points to be written.
- `block`  
  Write stream in blocks to improve file I/O speed.
- `class`  
  Save the class/cluster labels of the points as the last column.
- `sep`  
  The character that will separate attributes in a data point.
- `col.names`  
  A flag that determines if column names will be output.
- `row.names`  
  A flag that determines if row names will be output.
- `...`  
  Additional parameters that are passed to `write.table()`.

**Value**

There is no value returned from this operation.

**See Also**

write.table, DSD
Examples

# creating data and writing it to disk
stream <- DSD_Gaussians(k=3, d=5)
write_stream(stream, file="data.txt", n=100, class=TRUE)

# clean up
file.remove("data.txt")
Index

add_cluster (DSD_MG), 26
add_keyframe (MGC), 48
ani.replay, 3
animate (animation), 3
animate_cluster, 4, 21, 42
animate_cluster (animation), 3
animate_data, 21
animate_data (animation), 3
animation, 3, 58

change_alpha (DSC_DBSTREAM), 6
c1_agreement, 42
close_stream (DSD_ReadCSV), 30
cluster, 4, 21
cluster.stats, 42
connection, 31

d-Stream (DSC_DStream), 8
d-stream (DSC_DStream), 8
dbGetQuery, 32
DBSTREAM (DSC_DBSTREAM), 6
dbstream (DSC_DBSTREAM), 6
DSC, 4, 6, 7, 10, 12–16, 18–20, 39, 44–47, 52, 53, 58
DSC_DBSCAN, 5
DSC_DBSTREAM, 6
DSC_DStream, 8
DSC_Hierarchical, 11
DSC_Kmeans, 12
DSC_Macro, 6, 12, 13, 15, 18, 19, 51
DSC_Macro (DSC_Macro), 14
DSC_Marco, 14
DSC_Micro, 7, 10, 16, 18, 20
DSC_Micro (DSC_Micro), 14
DSC_Mirco, 14
DSC_R (DSC), 4
DSC_Reachability, 14
DSC_Sample, 16
DSC_Static, 17, 54
DSC_TwoStage, 18

DSC_Window, 19
DSClassify, 5, 39
DSD, 20, 22, 23, 25, 26, 29–32, 34, 35, 46, 53, 58
DSD_BarsAndGaussians, 21
DSD_Benchmark, 22
DSD_Cubes, 23
DSD_Gaussians, 24
DSD_Memory, 25, 29, 56
DSD_MG, 26, 49, 56
DSD_mlbenchData, 28
DSD_mlbenchGenerator, 29
DSD_MovingGenerator (DSD_MG), 26
DSD_R (DSD), 20
DSD_ReadCSV, 30, 56
DSD_ReadDB, 32
DSD_ScaleStream, 33, 56
DSD_Target, 34
DSD_UniformNoise, 35
DSFP, 36, 39
DSO, 36, 37, 38
DSO_Sample, 36, 37
DSO_Window, 36, 38
DST, 5, 36, 39
dstream (DSC_DStream), 8
evaluate, 4, 21, 39
evaluate_cluster, 3
evaluate_cluster (evaluate), 39

get_assignment, 4, 43
get_attraction (DSC_DStream), 8
get_centers, 4, 44
get_clusters (DSD_MG), 26
get_copy, 4, 45
get_keyframes (MGC), 48
get_macroclusters (DSC), 4
get_macroweights (DSC), 4
get_microclusters (DSC), 4
get_weights (DSC), 4
INDEX

get_points, 21, 36, 46
get_shared_density (DSC_DBSTREAM), 6
get_weights, 4, 36, 47
keyframe (MGC), 48

MGC, 48
MGC_Function, 27
MGC_Function (MGC), 48
MGCLinear, 27
MGCLinear (MGC), 48
MGCNoise, 27
MGCNoise (MGC), 48
MGCRandom, 27
MGCRandom (MGC), 48
MGCShapeBlock (MGC), 48
MGCShapeGaussian (MGC), 48
MGCStatic (MGC), 48
microToMacro, 4, 50
MovingGenerator (DSD_MG), 26

nclusters, 4, 52
plot, 4, 21, 53
print.stream_eval (evaluate), 39
prune_clusters, 4, 54

read.table, 31
recluster, 4, 21, 55
remove_cluster (DSD_MG), 26
remove_keyframe (MGC), 48
reset_stream, 26, 29, 31, 34, 56

scale, 34
scale_stream (DSD_ReadCSV), 30
silhouette, 42

update, 36, 57

write.table, 58
write_stream, 21, 58