# Package 'stream'

October 14, 2022

**Version** 2.0-0

**Date** 2022-09-01

**Encoding** UTF-8

**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, NSF CMMI 1728612, and NIH R21HG005912. Hahsler et al (2017) <doi:10.18637/jss.v076.i14>.

**Depends** R (>= 3.5.0), methods, proxy (>= 0.4-7), magrittr

**Imports** clue, cluster, clusterGeneration, dbscan (>= 1.0-0), fpc, graphics, grDevices, MASS, mlbench, Rcpp (>= 0.11.4), stats, utils

**Suggests** animation, DBI, dplyr, rJava, RSQLite, testthat, knitr

**URL** https://github.com/mhahsler/stream

**BugReports** https://github.com/mhahsler/stream/issues

**LinkingTo** Rcpp, BH

**License** GPL-3

**VignetteBuilder** knitr

**RoxygenNote** 7.2.1

**NeedsCompilation** yes

**Author** Michael Hahsler [aut, cre, cph] (<https://orcid.org/0000-0003-2716-1405>), Matthew Bolaños [ctb], John Forrest [ctb], Matthias Carnein [ctb], Dennis Assenmacher [ctb], Dalibor Krleža [ctb]

**Maintainer** Michael Hahsler <mhahsler@lyle.smu.edu>

**Repository** CRAN

**Date/Publication** 2022-09-02 23:00:02 UTC

# R topics documented:

---

`stream-package` *stream: 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, NSF CMMI 1728612, and NIH R21HG005912. Hahsler et al (2017) doi:10.18637/jss.v076.i14.

## Author(s)

Michael Hahsler

agreement *Agreement-based Measures for Clustering*

## Description

Calculates the agreement between two partitions, typically the known actual cluster labels and the predicted cluster labels.

## Usage

```
agreement(predicted, actual, method = "cRand", na_as_cluster = TRUE)
```

## Arguments

| | |
|---|---|
| predicted | a vector with predicted cluster labels. |
| actual | the known cluster labels (ground truth). |
| method | the used method (see `clue::cl_agreement()`). |
| na_as_cluster | logical; should NA labels (noise points) be considered its own cluster? |

## Details

This convenience function is an interface to `clue::cl_agreement()`. See `methods` in that man page for a list of available methods. A measure typically used for clustering is the corrected Rand index (also called adjusted Rand index). Numbers close to 1 indicate a very good agreement.

## References

Hornik K (2005). A CLUE for CLUster Ensembles. *Journal of Statistical Software*, *14*(12). doi:10.18637/jss.v014.i12

## Examples

```
# Perfect agreement (1 and 2 are just switched)
actual <- c(2, 2, 1, 3, 2, NA)
predicted <- c(1, 1, 2, 3, 1, NA)
agreement(actual, predicted)

# No agreement
predicted <- sample(predicted)
agreement(actual, predicted)
```

`animate_cluster` *Animates Plots of the Clustering Process*

## Description

Generates an animation of a data stream clustering process.

## Usage

```
animate_cluster(
  dsc,
  dsd,
  measure = NULL,
  horizon = 100,
  n = 1000,
  type = c("auto", "micro", "macro"),
  assign = "micro",
  assignmentMethod = c("auto", "model", "nn"),
  excludeNoise = FALSE,
  wait = 0.1,
  plot.args = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `dsc` | a DSC |
| `dsd` | a DSD |
| `measure` | the evaluation measure that should be graphed below the animation (see `evaluate_stream()`.) |
| `horizon` | the number of points displayed at once/used for evaluation. |
| `n` | the number of points to be plotted |
| `type, assign, assignmentMethod, excludeNoise` | are passed on to `evaluate_stream()` to calculate the evaluation measure. |
| `wait` | the time interval between each frame |
| `plot.args` | a list with plotting parameters for the clusters. |
| `...` | extra arguments are added to `plot.args`. |

## 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).

**Note:** You need to install package **animation** and its system requirements.

### Author(s)

Michael Hahsler

### See Also

`animation::ani.replay()` for replaying and saving animations.

Other DSC: `DSC_Macro()`, `DSC_Micro()`, `DSC_R()`, `DSC_Static()`, `DSC_TwoStage()`, `DSC()`, `evaluate.DSC`, `get_assignment()`, `plot.DSC()`, `predict()`, `prune_clusters()`, `read_saveDSC`, `recluster()`

Other plot: `animate_data()`, `plot.DSC()`, `plot.DSD()`

Other evaluation: `evaluate.DSC`, `evaluate`

### Examples

```
if (interactive()) {
stream <- DSD_Benchmark(1)

### animate the clustering process with evaluation
### Note: we choose to exclude noise points from the evaluation
###       measure calculation, even if the algorithm would assign
###       them to a cluster.
dbstream <- DSC_DBSTREAM(r = .04, lambda = .1, gaptime = 100, Cm = 3,
  shared_density = TRUE, alpha = .2)

animate_cluster(dbstream, stream, horizon = 100, n = 5000,
  measure = "crand", type = "macro", assign = "micro",
  plot.args = list(xlim = c(0, 1), ylim = c(0, 1), shared = TRUE))
}
```

---

## animate_data *Animates the Plotting of a Data Streams*

---

### Description

Generates an animation of a data stream.

### Usage

```
animate_data(dsd, horizon = 100, n = 1000, wait = 0.1, plot.args = NULL, ...)
```

### Arguments

| | |
|---|---|
| `dsd` | a DSD object |
| `horizon` | the number of points displayed at once/used for evaluation. |
| `n` | the number of points to be plotted |
| `wait` | the time interval between each frame |
| `plot.args` | a list with plotting parameters for the clusters. |
| `...` | extra arguments are added to `plot.args`. |

### 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).

**Note:** You need to install package **animation** and its system requirements.

### Author(s)

Michael Hahsler

### See Also

`animation::ani.replay()` for replaying and saving animations.

Other DSD: `DSD_BarsAndGaussians()`, `DSD_Benchmark()`, `DSD_Cubes()`, `DSD_Gaussians()`, `DSD_MG()`, `DSD_Memory()`, `DSD_Mixture()`, `DSD_NULL()`, `DSD_ReadDB()`, `DSD_ReadStream()`, `DSD_Target()`, `DSD_UniformNoise()`, `DSD_mlbenchData()`, `DSD_mlbenchGenerator()`, `DSD()`, `DSF()`, `close_stream()`, `get_points()`, `plot.DSD()`, `reset_stream()`

Other plot: `animate_cluster()`, `plot.DSC()`, `plot.DSD()`

### Examples

```
if (interactive()) {

stream <- DSD_Benchmark(1)
animate_data(stream, horizon = 100, n = 5000, xlim = c(0,1), ylim = c(0,1))

### 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())
}
```

---

## close_stream *Close a Data Stream*

---

### Description

Close a data stream that needs closing (e.g., a file or a connection).

### Usage

```
close_stream(dsd, ...)
```

### Arguments

| | |
|---|---|
| dsd | An object of class a subclass of DSD which implements a reset function. |
| ... | further arguments. |

### Details

close_stream() is implemented for:

- DSD
- DSD_ReadCSV
- DSD_ReadDB
- DSD_ReadStream
- DSF

### Author(s)

Michael Hahsler

### See Also

Other DSD: DSD_BarsAndGaussians(), DSD_Benchmark(), DSD_Cubes(), DSD_Gaussians(), DSD_MG(), DSD_Memory(), DSD_Mixture(), DSD_NULL(), DSD_ReadDB(), DSD_ReadStream(), DSD_Target(), DSD_UniformNoise(), DSD_mlbenchData(), DSD_mlbenchGenerator(), DSD(), DSF(), animate_data(), get_points(), plot.DSD(), reset_stream()

---

## DSAggregate *Data Stream Aggregator Base Classes*

---

### Description

Abstract base classes for all DSAggregate (Data Stream Aggregator) classes to aggregate streams.

### Usage

```
DSAggregate(...)

## S3 method for class 'DSAggregate'
update(object, dsd, n = 1, ...)

## S3 method for class 'DSAggregate'
get_points(x, ...)

## S3 method for class 'DSAggregate'
get_weights(x, ...)
```

### Arguments

| | |
|---|---|
| ... | Further arguments. |
| dsd | a data stream object. |
| n | the number of data points used for the update. |
| x, object | a concrete implementation of DSAggregate. |

### Details

The DSAggreagate class cannot be instantiated, but it serve as a base class from which other DSAggregate subclasses inherit.

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

### Author(s)

Michael Hahsler

### See Also

Other DST: DSClassifier(), DSC(), DSF_Scale(), DSOutlier(), DST_Runner(), DST_WriteStream(), DST(), evaluate, predict(), update()

Other DSAggregate: DSAggregate_Sample(), DSAggregate_Window()

### Examples

```
DSAggregate()
```

---

## DSAggregate_Sample *Sampling from a Data Stream (Data Stream Operator)*

---

### Description

Extracts a sample form a data stream using Reservoir Sampling.

### Usage

```
DSAggregate_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 thre 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 `DSAggregate_Sample` (subclass of `DSAggregate`).

## Author(s)

Michael Hahsler

## References

Vitter, J. S. (1985): Random sampling with a reservoir. ACM Transactions on Mathematical Software, 11(1), 37-57.

McLeod, A.I., Bellhouse, D.R. (1983): A Convenient Algorithm for Drawing a Simple Random Sample. Applied Statistics, 32(2), 182-184.

Aggarwal C. (2006) On Biased Reservoir Sampling in the Presence of Stream Evolution. International Conference on Very Large Databases (VLDB'06). 607-618.

## See Also

Other DSAggregate: `DSAggregate_Window()`, `DSAggregate()`

## Examples

```
set.seed(1500)

stream <- DSD_Gaussians(k = 3, noise = 0.05)

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

head(get_points(sample))

# apply k-means clustering to the sample (data without info columns)
km <- kmeans(get_points(sample, info = FALSE), centers = 3)
plot(get_points(sample, info = FALSE))
points(km$centers, col = "red", pch = 3, cex = 2)
```

DSAggregate_Window *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

```
DSAggregate_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 DSAggregate_Window (subclass of DSAggregate).

## Author(s)

Michael Hahsler

## References

Zhu, Y. and Shasha, D. (2002). StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time, Intl. Conference of Very Large Data Bases (VLDB'02).

## See Also

Other DSAggregate: DSAggregate_Sample(), DSAggregate()

## Examples

```
set.seed(1500)

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

window <- DSAggregate_Window(horizon = 10)
window

# update with only two points. The window is mostly empty (NA)
```

```
update(window, stream, 2)
get_points(window)

# update window
update(window, stream, 100)
get_points(window)
```

---

DSC *Data Stream Clustering Base Class*

---

## Description

Abstract base classes for Data Stream Clustering (DSC). Concrete implementations are functions starting with DSC_ (RStudio use auto-completion with Tab to select one).

## Usage

```
DSC(...)

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

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

get_copy(x)

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

get_microclusters(x, ...)

get_microweights(x, ...)

get_macroclusters(x, ...)

get_macroweights(x, ...)
```

## Arguments

| | |
|---|---|
| ... | further parameter |
| x | a DSC object. |
| 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. |

## Details

The DSC class cannot be instantiated (calling DSC() produces only a message listing the available implementations), but they serve as a base class from which other DSC classes inherit.

Data stream clustering has typically an

- **online clustering component** (see DSC_Micro), and an
- **offline reclustering component** (see DSC_Macro).

Class DSC provides several generic functions that can operate on all DSC subclasses. See Usage and Functions sections for methods. Additional, separately documented methods are:

- update() adds new data points from a stream to a clustering.
- predict() predicts the cluster assignment for new data points.
- plot() plots cluster centers (see plot.DSC()).

get_centers() and get_weights() are typically overwritten by subclasses of DSC.

Since DSC objects often contain external pointers, regular saving and reading operations will fail. Use saveDSC() and readDSC() which will serialize the objects first appropriately.

## Functions

- get_centers(): Gets the cluster centers (micro- or macro-clusters) from a DSC object.
- get_weights(): Get the weights of the clusters in the DSC (returns 1s if not implemented by the clusterer)
- get_copy(): Create a Deep Copy of a DSC Object that contain reference classes (e.g., Java data structures for MOA).
- nclusters(): Returns the number of micro-clusters from the DSC object.
- get_microclusters(): Used as internal interface.
- get_microweights(): Used as internal interface.
- get_macroclusters(): Used as internal interface.
- get_macroweights(): Used as internal interface.

## Author(s)

Michael Hahsler

## See Also

Other DST: DSAggregate(), DSClassifier(), DSF_Scale(), DSOutlier(), DST_Runner(), DST_WriteStream(), DST(), evaluate, predict(), update()

Other DSC: DSC_Macro(), DSC_Micro(), DSC_R(), DSC_Static(), DSC_TwoStage(), animate_cluster(), evaluate.DSC, get_assignment(), plot.DSC(), predict(), prune_clusters(), read_saveDSC, recluster()

## Examples

```
DSC()

set.seed(1000)
stream <- DSD_Gaussians(k = 3, d = 2, noise = 0.05)
dstream <- DSC_DStream(gridsize = .1, gaptime = 100)
update(dstream, stream, 500)
dstream

# get micro-cluster centers
get_centers(dstream)

# get the number of clusters
nclusters(dstream)

# get the micro-cluster weights
get_weights(dstream)

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

# plot the clustering result
plot(dstream, stream)
plot(dstream, stream, type = "both")

# predict macro clusters for new points (see predict())
points <- get_points(stream, n = 5)
points

predict(dstream, points, type = "macro")
```

---

# DSClassifier *Abstract Class for Data Stream Classifiers*

---

## Description

Abstract class for data stream classifiers. Currently, **stream** does not implement classification algorithms. Implementations can be found in package **streamMOA**.

## Usage

```
DSClassifier(...)
```

## Arguments

| | |
|---|---|
| ... | Further arguments. |

## Author(s)

Michael Hahsler

## See Also

Other DST: DSAggregate(), DSC(), DSF_Scale(), DSOutlier(), DST_Runner(), DST_WriteStream(), DST(), evaluate, predict(), update()

## Examples

```
DSClassifier()
```

---

DSC_BICO *BICO - Fast computation of k-means coresets in a data stream*

---

## Description

Micro Clusterer. BICO maintains a tree which is inspired by the clustering tree of BIRCH. Each node in the tree represents a subset of these points. Instead of storing all points as individual objects, only the number of points, the sum and the squared sum of the subset's points are stored as key features of each subset. Points are inserted into exactly one node.

## Usage

```
DSC_BICO(formula = NULL, k = 5, space = 10, p = 10, iterations = 10)
```

## Arguments

| | |
|---|---|
| formula | NULL to use all features in the stream or a model formula of the form ~ X1 + X2 to specify the features used for clustering. Only ., + and - are currently supported in the formula. |
| k | number of centers |
| space | coreset size |
| p | number of random projections used for nearest neighbor search in first level |
| iterations | number of repetitions for the kmeans++ procedure in the offline component |

## Details

In this implementation, the nearest neighbor search on the first level of the tree is sped up by projecting all points to random 1-d subspaces. The first estimation of the optimal clustering cost is computed in a buffer phase at the beginning of the algorithm. This implementation interfaces the original C++ implementation available here: http://ls2-www.cs.tu-dortmund.de/grav/de/bico. For micro-clustering, the algorithm computes the coreset of the stream. Reclustering is performed by using the kmeans++ algorithm on the coreset.

## Author(s)

R-Interface: Matthias Carnein (<Matthias.Carnein@uni-muenster.de>), Dennis Assenmacher. C-Implementation: Hendrik Fichtenberger, Marc Gille, Melanie Schmidt, Chris Schwiegelshohn, Christian Sohler.

## References

Hendrik Fichtenberger, Marc Gille, Melanie Schmidt, Chris Schwiegelshohn, Christian Sohler: BICO: BIRCH Meets Coresets for k-Means Clustering. *ESA 2013:* 481-492.

## See Also

Other DSC_Micro: DSC_BIRCH(), DSC_DBSTREAM(), DSC_DStream(), DSC_Micro(), DSC_Sample(), DSC_Window(), DSC_evoStream()

## Examples

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

BICO <- DSC_BICO(k = 3, p = 10, space = 100, iterations = 10)
update(BICO, stream, n = 500)

plot(BICO,stream)
```

---

# DSC_BIRCH *Balanced Iterative Reducing Clustering using Hierarchies*

---

## Description

Micro Clusterer. BIRCH builds a balanced tree of Clustering Features (CFs) to summarize the stream.

## Usage

```
DSC_BIRCH(
  formula = NULL,
  threshold,
  branching,
  maxLeaf,
  maxMem = 0,
  outlierThreshold = 0.25
)
```

## Arguments

| | |
|---|---|
| formula | NULL to use all features in the stream or a model formula of the form ~ X1 + X2 to specify the features used for clustering. Only ., + and - are currently supported in the formula. |
| threshold | threshold used to check whether a new data point can be absorbed or not. |
| branching | branching factor (maximum amount of child nodes for a non-leaf node) of the CF-Tree. |
| maxLeaf | maximum number of entries within a leaf node |
| maxMem | memory limitation for the whole CFTree in bytes. Default is 0, indicating no memory restriction. |
| outlierThreshold | threshold for identifying outliers when rebuilding the CF-Tree. |

## Details

A CF in the calanced tree is a tuple (n, LS, SS) which represents a cluster by storing the number of elements (n), their linear sum (LS) and their squared sum (SS). Each new observation descends the tree by following its closest CF until a leaf node is reached. It is either merged into its closest leaf-CF or inserted as a new one. All leaf-CFs form the micro-clusters. Rebuilding the tree is realized by inserting all leaf-CF nodes into a new tree structure with an increased threshold.

## Author(s)

Dennis Assenmacher (<Dennis.Assenmacher@uni-muenster.de>), Matthias Carnein (<Matthias.Carnein@uni-muenste

## References

Zhang T, Ramakrishnan R and Livny M (1996), "BIRCH: An Efficient Data Clustering Method for Very Large Databases", *In Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data.* Montreal, Quebec, Canada , pp. 103-114. ACM.

Zhang T, Ramakrishnan R and Livny M (1997), "BIRCH: A new data clustering algorithm and its applications", *Data Mining and Knowledge Discovery.* Vol. 1(2), pp. 141-182.

## See Also

Other DSC_Micro: DSC_BICO(), DSC_DBSTREAM(), DSC_DStream(), DSC_Micro(), DSC_Sample(), DSC_Window(), DSC_evoStream()

## Examples

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

BIRCH <- DSC_BIRCH(threshold = .1, branching = 8, maxLeaf = 20)
update(BIRCH, stream, n = 500)
BIRCH

plot(BIRCH, stream)
```

DSC_DBSCAN *DBSCAN Macro-clusterer*

## Description

Macro Clusterer. Implements the DBSCAN algorithm for reclustering micro-clusterings.

## Usage

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

## Arguments

| | |
|---|---|
| formula | NULL to use all features in the stream or a model formula of the form ~ X1 + X2 to specify the features used for clustering. Only ., + and - are currently supported in the formula. |
| 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).

update() and recluster() invisibly return the assignment of the data points to clusters.

**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).

## Author(s)

Michael Hahsler

## References

Martin Ester, Hans-Peter Kriegel, Joerg Sander, Xiaowei Xu (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In Evangelos Simoudis, Jiawei Han, Usama M. Fayyad. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96).* AAAI Press. pp. 226-231.

## See Also

Other DSC_Macro: DSC_EA(), DSC_Hierarchical(), DSC_Kmeans(), DSC_Macro(), DSC_Reachability()

## Examples

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

# Use a moving window for "micro-clusters and recluster with DBSCAN (macro-clusters)
cl <- DSC_TwoStage(
  micro = DSC_Window(horizon = 100),
  macro = DSC_DBSCAN(eps = .05)
)

update(cl, stream, 500)
cl

plot(cl, stream)
```

---

# DSC_DBSTREAM *DBSTREAM Clustering Algorithm*

---

## Description

Micro Clusterer with reclustering. 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(
  formula = NULL,
  r,
  lambda = 0.001,
  gaptime = 1000L,
  Cm = 3,
  metric = "Euclidean",
  noise_multiplier = 1,
  shared_density = FALSE,
  alpha = 0.1,
  k = 0,
```

```
  minweight = 0
)

get_shared_density(x, use_alpha = TRUE)

change_alpha(x, alpha)

## S3 method for class 'DSC_DBSTREAM'
plot(
  x,
  dsd = NULL,
  n = 500,
  col_points = NULL,
  dim = NULL,
  method = "pairs",
  type = c("auto", "micro", "macro", "both", "none"),
  shared_density = FALSE,
  use_alpha = TRUE,
  assignment = FALSE,
  ...
)

DSOutlier_DBSTREAM(
  formula = NULL,
  r,
  lambda = 0.001,
  gaptime = 1000L,
  Cm = 3,
  metric = "Euclidean",
  outlier_multiplier = 2
)
```

## Arguments

| | |
|---|---|
| formula | NULL to use all features in the stream or a model formula of the form ~ X1 + X2 to specify the features used for clustering. Only ., + and - are currently supported in the formula. |
| r | The radius of micro-clusters. |
| lambda | The lambda used in the fading function. |
| gaptime | weak micro-clusters (and weak shared density entries) are removed every gaptime points. |
| Cm | minimum weight for a micro-cluster. |
| metric | metric used to calculate distances. |
| noise_multiplier, outlier_multiplier | multiplier for radius r to declare noise or outliers. |
| 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 $r * (1 - alpha)$ distance are clustered together). |

| | |
|---|---|
| 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. |
| k | The number of macro clusters to be returned if macro is true. |
| minweight | The proportion of the total weight a macro-cluster needs to have not to be noise (between 0 and 1). |
| x | A DSC_DBSTREAM object to get the shared density information from. |
| use_alpha | only return shared density if it exceeds alpha. |
| dsd | a data stream object. |
| n | number of plots taken from the dsd to plot. |
| col_points | color used for plotting. |
| 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. |
| method | plot method. |
| type | Plot micro clusters (type="micro"), macro clusters (type="macro"), both micro and macro clusters (type="both"), outliers(type="outliers"), or everything together (type="all"). type="auto" leaves to 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. |

## 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.

update() invisibly return the assignment of the data points to clusters. The columns are .class with the index of the strong micro-cluster and .mc_id with the permanent id of the strong micro-cluster.

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.

predict() can be used to assign new points to clusters. Points are assigned to a micro-cluster if they are within its assignment area (distance is less then r times noise_multiplier).

DSOutlier_DBSTREAM classifies points as outlier/noise if they that cannot be assigned to a micro-cluster representing a dense region as a outlier/noise. Parameter outlier_multiplier specifies how far a point has to be away from a micro-cluster as a multiplyer for the radius r. A larger value means that outliers have to be farther away from dense regions and thus reduce the chance of misclassifying a regular point as an outlier.

## Value

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

## Author(s)

Michael Hahsler and Matthew Bolanos

## References

Michael Hahsler and Matthew Bolanos. Clustering data streams based on shared density between micro-clusters. *IEEE Transactions on Knowledge and Data Engineering,* 28(6):1449–1461, June 2016

## See Also

Other DSC_Micro: DSC_BICO(), DSC_BIRCH(), DSC_DStream(), DSC_Micro(), DSC_Sample(), DSC_Window(), DSC_evoStream()

Other DSC_TwoStage: DSC_DStream(), DSC_TwoStage(), DSC_evoStream()

Other DSOutlier: DSC_DStream(), DSOutlier()

## Examples

```
set.seed(1000)
stream <- DSD_Gaussians(k = 3, d = 2, noise = 0.05)

# create clusterer with r = .05
dbstream <- DSC_DBSTREAM(r = .05)
update(dbstream, stream, 500)
dbstream

# check micro-clusters
nclusters(dbstream)
head(get_centers(dbstream))
plot(dbstream, stream)

# plot micro-clusters with assignment area
plot(dbstream, stream, type = "none", assignment = TRUE)


# DBSTREAM with shared density
dbstream <- DSC_DBSTREAM(r = .05, shared_density = TRUE, Cm = 5)
update(dbstream, stream, 500)
dbstream

plot(dbstream, stream)
# plot the shared density graph (several options)
plot(dbstream, stream, type = "micro", shared_density = TRUE)
plot(dbstream, stream, type = "none", shared_density = TRUE, assignment = TRUE)

# see how micro and macro-clusters relate
# each micro-cluster has an entry with the macro-cluster id
```

```
# Note: unassigned micro-clusters (noise) have an NA
microToMacro(dbstream)

# do some evaluation
evaluate_static(dbstream, stream, measure = "purity")
evaluate_static(dbstream, stream, measure = "cRand", type = "macro")

# use DBSTREAM also returns the cluster assignment
# later retrieve the cluster assignments for each point)
data("iris")
dbstream <- DSC_DBSTREAM(r = 1)
cl <- update(dbstream, iris[,-5], assignments = TRUE)
dbstream

head(cl)

# micro-clusters
plot(iris[,-5], col = cl$.class, pch = cl$.class)

# macro-clusters (2 clusters since reachability cannot separate two of the three species)
plot(iris[,-5], col = microToMacro(dbstream, cl$.class))

# use DBSTREAM with a formula (cluster all variables but X2)
stream <- DSD_Gaussians(k = 3, d = 4, noise = 0.05)
dbstream <- DSC_DBSTREAM(formula = ~ . - X2, r = .2)

update(dbstream, stream, 500)
get_centers(dbstream)
```

---

DSC_DStream *D-Stream Data Stream Clustering Algorithm*

---

## Description

Micro Clusterer with reclustering. Implements the grid-based D-Stream data stream clustering algorithm.

## Usage

```
DSC_DStream(
  formula = NULL,
  gridsize,
  lambda = 0.001,
  gaptime = 1000L,
  Cm = 3,
  Cl = 0.8,
  attraction = FALSE,
  epsilon = 0.3,
  Cm2 = Cm,
```

```
  k = NULL,
  N = 0
)

get_attraction(x, relative = FALSE, grid_type = "dense", dist = FALSE)

## S3 method for class 'DSC_DStream'
plot(
  x,
  dsd = NULL,
  n = 500,
  type = c("auto", "micro", "macro", "both"),
  grid = FALSE,
  grid_type = "used",
  assignment = FALSE,
  ...
)

DSOutlier_DStream(
  formula = NULL,
  gridsize,
  lambda = 0.001,
  gaptime = 1000L,
  Cm = 3,
  Cl = 0.8,
  outlier_multiplier = 2
)
```

## Arguments

| | |
|---|---|
| formula | NULL to use all features in the stream or a model formula of the form ~ X1 + X2 to specify the features used for clustering. Only ., + and - are currently supported in the formula. |
| gridsize | Size of grid cells. |
| lambda | Fading constant used function to calculate the decay factor $2^{-}lambda$. (Note: in the paper the authors use lamba 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. |
| dsd | a DSD data stream object. |
| n | number of plots taken from dsd to plot. |
| type | Plot micro clusters (type = "micro"), macro clusters (type = "macro"), both micro and macro clusters (type = "both"), outliers(type = "outliers"), or everything together (type = "all"). type = "auto" leaves to the class of DSC to decide. |
| grid | logical; show the D-Stream grid instead of circles for micro-clusters. |
| assignment | logical; show assignment area of micro-clusters. |
| ... | further argument are passed on. |
| outlier_multiplier | multiplier for assignment grid width to declare outliers. |

## 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 Cm2 which in the original algorithm is equal to 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).

DSOutlier_DStream classifies points that do not fall into a dense grid cell as outlier/noise. Parameter outlier_multiplier specifies how far the point needs to be away from a dense cell to be classified as an outlier by multiplying the grid size.

Note that DSC_DStream currently cannot be saved to disk using save() or saveRDS(). This functionality will be added later!

## Value

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

## Author(s)

Michael Hahsler

## References

Yixin Chen and Li Tu. 2007. Density-based clustering for real-time stream data. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '07).* ACM, New York, NY, USA, 133-142.

Li Tu and Yixin Chen. 2009. Stream data clustering based on grid density and attraction. *ACM Transactions on Knowledge Discovery from Data,* 3(3), Article 12 (July 2009), 27 pages.

## See Also

Other DSC_Micro: DSC_BICO(), DSC_BIRCH(), DSC_DBSTREAM(), DSC_Micro(), DSC_Sample(), DSC_Window(), DSC_evoStream()

Other DSC_TwoStage: DSC_DBSTREAM(), DSC_TwoStage(), DSC_evoStream()

Other DSOutlier: DSC_DBSTREAM(), DSOutlier()

## Examples

```
stream <- DSD_BarsAndGaussians(noise = .05)
plot(stream)

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")
```

```
# look at transitional and sparse cells
plot(dstream1, stream, grid = TRUE, grid_type = "transitional")
plot(dstream1, stream, grid = TRUE, grid_type = "sparse")

### Macro-clusters
# standard D-Stream uses reachability
nclusters(dstream1, type = "macro")
get_centers(dstream1, type = "macro")
plot(dstream1, stream, type = "macro")
evaluate_static(dstream1, stream, measure = "crand", type = "macro")

# use attraction for reclustering
dstream2 <- DSC_DStream(gridsize = 1, attraction = TRUE, Cm = 1.5)
update(dstream2, stream, 1000)
dstream2

plot(dstream2, stream, grid = TRUE)
evaluate_static(dstream2, stream, measure = "crand", type = "macro")
```

---

DSC_EA *Reclustering using an Evolutionary Algorithm*

---

## Description

Macro Clusterer.

## Usage

```
DSC_EA(
  formula = NULL,
  k,
  generations = 2000,
  crossoverRate = 0.8,
  mutationRate = 0.001,
  populationSize = 100
)
```

## Arguments

| | |
|---|---|
| formula | NULL to use all features in the stream or a model formula of the form ~ X1 + X2 to specify the features used for clustering. Only ., + and - are currently supported in the formula. |
| k | number of macro-clusters |
| generations | number of EA generations performed during reclustering |
| crossoverRate | cross-over rate for the evolutionary algorithm |
| mutationRate | mutation rate for the evolutionary algorithm |
| populationSize | number of solutions that the evolutionary algorithm maintains |

## Details

Reclustering using an evolutionary algorithm. This approach was designed for `evoStream` (see DSC_evoStream) but can also be used for other micro-clustering algorithms.

The evolutionary algorithm uses existing clustering solutions and creates small variations of them by combining and randomly modifying them. The modified solutions can yield better partitions and thus can improve the clustering over time. The evolutionary algorithm is incremental, which allows to improve existing macro-clusters instead of recomputing them every time.

## Author(s)

Matthias Carnein <Matthias.Carnein@uni-muenster.de>

## References

Carnein M. and Trautmann H. (2018), "evoStream - Evolutionary Stream Clustering Utilizing Idle Times", Big Data Research.

## See Also

Other DSC_Macro: `DSC_DBSCAN()`, `DSC_Hierarchical()`, `DSC_Kmeans()`, `DSC_Macro()`, `DSC_Reachability()`

## Examples

```
stream <- DSD_Gaussians(k = 3, d = 2) %>% DSD_Memory(n = 1000)

## online algorithm
dbstream <- DSC_DBSTREAM(r = 0.1)

## offline algorithm (note: we use a small number of generations
##                           to make this run faster.)
EA <- DSC_EA(k = 3, generations = 100)

## create pipeline and insert observations
two <- DSC_TwoStage(dbstream, EA)
update(two, stream, n = 1000)
two

## plot result
reset_stream(stream)
plot(two, stream)

## if we have time, evaluate additional generations. This can be
## called at any time, also between observations.
two$macro$RObj$recluster(100)

## plot improved result
reset_stream(stream)
plot(two, stream)


## alternatively: do not create twostage but apply directly
```

```
reset_stream(stream)
update(dbstream, stream, n = 1000)
recluster(EA, dbstream)
reset_stream(stream)
plot(EA, stream)
```

---

`DSC_evoStream` *evoStream - Evolutionary Stream Clustering*

---

## Description

Micro Clusterer with reclustering. Stream clustering algorithm based on evolutionary optimization.

## Usage

```
DSC_evoStream(
  formula = NULL,
  r,
  lambda = 0.001,
  tgap = 100,
  k = 2,
  crossoverRate = 0.8,
  mutationRate = 0.001,
  populationSize = 100,
  initializeAfter = 2 * k,
  incrementalGenerations = 1,
  reclusterGenerations = 1000
)
```

## Arguments

| | |
|---|---|
| `formula` | `NULL` to use all features in the stream or a model formula of the form `~ X1 + X2` to specify the features used for clustering. Only `.`, `+` and `-` are currently supported in the formula. |
| `r` | radius threshold for micro-cluster assignment |
| `lambda` | decay rate |
| `tgap` | time-interval between outlier detection and clean-up |
| `k` | number of macro-clusters |
| `crossoverRate` | cross-over rate for the evolutionary algorithm |
| `mutationRate` | mutation rate for the evolutionary algorithm |
| `populationSize` | number of solutions that the evolutionary algorithm maintains |
| `initializeAfter` | number of micro-cluster required for the initialization of the evolutionary algorithm. |
| `incrementalGenerations` | number of EA generations performed after each observation |
| `reclusterGenerations` | number of EA generations performed during reclustering |

## Details

The online component uses a simplified version of DBSTREAM to generate micro-clusters. The micro-clusters are then incrementally reclustered using an evolutionary algorithm. Evolutionary algorithms create slight variations by combining and randomly modifying existing solutions. By iteratively selecting better solutions, an evolutionary pressure is created which improves the clustering over time. Since the evolutionary algorithm is incremental, it is possible to apply it between observations, e.g. in the idle time of the stream. Whenever there is idle time, we can call the `recluster()` function of the reference class to improve the macro-clusters (see example). The evolutionary algorithm can also be applied as a traditional reclustering step, or a combination of both. In addition, this implementation also allows to evaluate a fixed number of generations after each observation.

## Author(s)

Matthias Carnein <Matthias.Carnein@uni-muenster.de>

## References

Carnein M. and Trautmann H. (2018), "evoStream - Evolutionary Stream Clustering Utilizing Idle Times", Big Data Research.

## See Also

Other DSC_Micro: `DSC_BICO()`, `DSC_BIRCH()`, `DSC_DBSTREAM()`, `DSC_DStream()`, `DSC_Micro()`, `DSC_Sample()`, `DSC_Window()`

Other DSC_TwoStage: `DSC_DBSTREAM()`, `DSC_DStream()`, `DSC_TwoStage()`

## Examples

```
stream <- DSD_Gaussians(k = 3, d = 2) %>% DSD_Memory(n = 500)

## init evoStream
evoStream <- DSC_evoStream(r = 0.05, k = 3,
  incrementalGenerations = 1, reclusterGenerations = 500)

## insert observations
update(evoStream, stream, n = 500)

## micro clusters
get_centers(evoStream, type = "micro")

## micro weights
get_weights(evoStream, type = "micro")

## macro clusters
get_centers(evoStream, type = "macro")

## macro weights
get_weights(evoStream, type = "macro")
```

```
## plot result
reset_stream(stream)
plot(evoStream, stream)

## if we have time, then we can evaluate additional generations.
## This can be called at any time, also between observations.
## by default, 1 generation is evaluated after each observation and
## 1000 generations during reclustering but we set it here to 500
evoStream$RObj$recluster(500)

## plot improved result
reset_stream(stream)
plot(evoStream, stream)

## get assignment of micro to macro clusters
microToMacro(evoStream)
```

---

DSC_Hierarchical *Hierarchical Micro-Cluster Reclusterer*

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| formula | NULL to use all features in the stream or a model formula of the form ~ X1 + X2 to specify the features used for clustering. Only ., + and - are currently supported in the formula. |
| 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.

`update()` and `recluster()` invisibly return the assignment of the data points to clusters.

**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. |

### Author(s)

Michael Hahsler

### See Also

Other DSC_Macro: `DSC_DBSCAN()`, `DSC_EA()`, `DSC_Kmeans()`, `DSC_Macro()`, `DSC_Reachability()`

### Examples

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

# Use a moving window for "micro-clusters and recluster with HC (macro-clusters)
cl <- DSC_TwoStage(
  micro = DSC_Window(horizon = 100),
  macro = DSC_Hierarchical(h = .1, method = "single")
)

update(cl, stream, 500)
cl

plot(cl, stream)
```

---

## `DSC_Kmeans` *Kmeans Macro-clusterer*

---

### Description

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

## Usage

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

## Arguments

| | |
|---|---|
| formula | NULL to use all features in the stream or a model formula of the form ~ X1 + X2 to specify the features used for clustering. Only ., + and - are currently supported in the formula. |
| 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

update() and recluster() invisibly return the assignment of the data points to clusters.

Please refer to function stats::kmeans() 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)

## Author(s)

Michael Hahsler

## See Also

Other DSC_Macro: DSC_DBSCAN(), DSC_EA(), DSC_Hierarchical(), DSC_Macro(), DSC_Reachability()

## Examples

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

# Use a moving window for "micro-clusters and recluster with k-means (macro-clusters)
cl <- DSC_TwoStage(
  micro = DSC_Window(horizon = 100),
  macro = DSC_Kmeans(k = 3)
)

update(cl, stream, 500)
cl

plot(cl, stream)
```

---

# DSC_Macro *Abstract Class for Macro Clusterers (Offline Component)*

---

## Description

Abstract class for all DSC Macro Clusterers which recluster micro-clusters **offline** into final clusters called macro-clusters.

## Usage

```
DSC_Macro(...)

microToMacro(x, micro = NULL)
```

## Arguments

| | |
|---|---|
| ... | further 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. |

## Details

Data stream clustering algorithms typically consists of an **online component** that creates micro-clusters (implemented as DSC_Micro) and and **offline components** which is used to recluster micro-clusters into final clusters called macro-clusters. The function `recluster()` is used extract micro-clusters from a DSC_Micro and create macro-clusters with a DSC_Macro.

Available clustering methods can be found in the See Also section below.

`microToMacro()` returns the assignment of Micro-cluster IDs to Macro-cluster IDs.

For convenience, a DSC_Micro and DSC_Macro can be combined using DSC_TwoStage.

DSC_Macro cannot be instantiated.

## Value

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

## Author(s)

Michael Hahsler

## See Also

Other DSC_Macro: `DSC_DBSCAN()`, `DSC_EA()`, `DSC_Hierarchical()`, `DSC_Kmeans()`, `DSC_Reachability()`

Other DSC: `DSC_Micro()`, `DSC_R()`, `DSC_Static()`, `DSC_TwoStage()`, `DSC()`, `animate_cluster()`, `evaluate.DSC`, `get_assignment()`, `plot.DSC()`, `predict()`, `prune_clusters()`, `read_saveDSC`, `recluster()`

---

# DSC_Micro *Abstract Class for Micro Clusterers (Online Component)*

---

## Description

Abstract class for all clustering methods that can operate **online** and result in a set of micro-clusters.

## Usage

```
DSC_Micro(...)
```

## Arguments

| | |
|---|---|
| `...` | further arguments. |

## Details

Micro-clustering algorithms are data stream mining tasks DST which implement the **online component of data stream clustering.** The clustering is performed sequentially by using `update()` to add new points from a data stream to the clustering. The result is a set of micro-clusters that can be retrieved using `get_clusters()`.

Available clustering methods can be found in the See Also section below.

Many data stream clustering algorithms define both, the online and an offline component to recluster micro-clusters into larger clusters called macro-clusters. This is implemented here as class DSC_TwoStage.

`DSC_Micro` cannot be instantiated.

## Author(s)

Michael Hahsler

## See Also

Other DSC_Micro: DSC_BICO(), DSC_BIRCH(), DSC_DBSTREAM(), DSC_DStream(), DSC_Sample(), DSC_Window(), DSC_evoStream()

Other DSC: DSC_Macro(), DSC_R(), DSC_Static(), DSC_TwoStage(), DSC(), animate_cluster(), evaluate.DSC, get_assignment(), plot.DSC(), predict(), prune_clusters(), read_saveDSC, recluster()

## Examples

```
stream <- DSD_BarsAndGaussians(noise = .05)

# Use a DStream to create micro-clusters
dstream <- DSC_DStream(gridsize = 1, Cm = 1.5)
update(dstream, stream, 1000)
dstream
nclusters(dstream)
plot(dstream, stream, main = "micro-clusters")
```

---

# DSC_R *Abstract Class for Implementing R-based Clusterers*

---

## Description

Abstract class for implementing R-based clusterers.

## Usage

```
DSC_R(...)

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

## Arguments

| | |
|---|---|
| ... | further arguments. |
| object | a DSC object. |
| dsd | a data stream object. |
| n | number of data points taken from the stream. |

| | |
|---|---|
| verbose | logical; show progress? |
| block | process blocks of data to improve speed. |
| assignment | logical; return a vector with cluster assignments? |

## Details

DSC_R cannot be instantiated.

### Implementing new Classes

To implement a new clusterer you need to create an S3 class with elements description and RObj. RObj needs to be a reference class with methods:

- cluster(newdata, ...)
- get_microclusters(...)
- get_microweights(...)
- get_macroclusters(...)
- get_macroweights(...)
- microToMacro(micro, ...)

See DSC for details and parameters.

DSC_R cannot be instantiated.

## Author(s)

Michael Hahsler

## See Also

Other DSC: DSC_Macro(), DSC_Micro(), DSC_Static(), DSC_TwoStage(), DSC(), animate_cluster(), evaluate.DSC, get_assignment(), plot.DSC(), predict(), prune_clusters(), read_saveDSC, recluster()

---

# DSC_Reachability *Reachability Micro-Cluster Reclusterer*

---

## Description

Macro Clusterer. Implementation of reachability clustering (based on DBSCAN's concept of reachability) to recluster a set of micro-clusters.

## Usage

```
DSC_Reachability(
  formula = NULL,
  epsilon,
  min_weight = NULL,
  description = NULL
)
```

## Arguments

| | |
|---|---|
| formula | NULL to use all features in the stream or a model formula of the form ~ X1 + X2 to specify the features used for clustering. Only ., + and - are currently supported in the formula. |
| 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

Two micro-clusters are directly reachable if they are within each other's epsilon-neighborhood (i.e., the distance between the centers is less then 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.

Reachability uses internally DSC_Hierarchical with single link.

update() and recluster() invisibly return the assignment of the data points to clusters.

**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. |

## Author(s)

Michael Hahsler

## References

Martin Ester, Hans-Peter Kriegel, Joerg Sander, Xiaowei Xu (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In Evangelos Simoudis, Jiawei Han, Usama M. Fayyad. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96).* AAAI Press. pp. 226-231.

## See Also

Other DSC_Macro: DSC_DBSCAN(), DSC_EA(), DSC_Hierarchical(), DSC_Kmeans(), DSC_Macro()

## Examples

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

# Use a moving window for "micro-clusters and recluster with DBSCAN (macro-clusters)
cl <- DSC_TwoStage(
```

```
  micro = DSC_Window(horizon = 100),
  macro = DSC_Reachability(eps = .05)
)

update(cl, stream, 500)
cl

plot(cl, stream)
```

---

DSC_Sample *Extract a Fixed-size Sample from a Data Stream*

---

## Description

Micro Clusterer. Extracts a sample form a data stream using Reservoir Sampling (DSAggregate_Sample). 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).

## Author(s)

Michael Hahsler

## References

Vitter, J. S. (1985): Random sampling with a reservoir. *ACM Transactions on Mathematical Software,* 11(1), 37-57.

McLeod, A.I., Bellhouse, D.R. (1983): A Convenient Algorithm for Drawing a Simple Random Sample. *Applied Statistics,* 32(2), 182-184.

Aggarwal C. (2006) On Biased Reservoir Sampling in the Presence of Stream Evolution. *International Conference on Very Large Databases (VLDB'06).* 607-618.

## See Also

Other DSC_Micro: DSC_BICO(), DSC_BIRCH(), DSC_DBSTREAM(), DSC_DStream(), DSC_Micro(), DSC_Window(), DSC_evoStream()

## Examples

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

sample <- DSC_Sample(k = 20)
update(sample, stream, 500)
sample

# plot micro-clusters
plot(sample, stream)

# recluster the sample with k-means
kmeans <- DSC_Kmeans(k = 3)
recluster(kmeans, sample)
plot(kmeans, stream)

# sample from an evolving stream
stream <- DSD_Benchmark(1)
sample <- DSC_Sample(k = 20)
update(sample, stream, 1000)

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 = 20, biased = TRUE)
update(sample, stream, 1000)
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

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

## Arguments

| | |
|---|---|
| x | The clustering (a DSD object) to copy or a list with components centers (a data frame or matrix) and weights (a vector with cluster weights). |
| 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.

## Author(s)

Michael Hahsler

## See Also

Other DSC: DSC_Macro(), DSC_Micro(), DSC_R(), DSC_TwoStage(), DSC(), animate_cluster(), evaluate.DSC, get_assignment(), plot.DSC(), predict(), prune_clusters(), read_saveDSC, recluster()

## Examples

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

# create a manual clustering
static4 <- DSC_Static(list(
             centers = data.frame(X1 = c(1, 2), X2 = c(1, 2)),
             weights = c(1, 2)),
             type = "macro")
static4
plot(static4)
```

---

`DSC_TwoStage` *TwoStage Clustering Process*

---

## Description

Combines an **online clustering component** (DSC_Micro) and an **offline reclustering component** (DSC_Macro) into a single process.

## Usage

```
DSC_TwoStage(micro, macro)
```

## Arguments

| | |
|---|---|
| `micro` | Clustering algorithm used in the online stage (DSC_Micro) |
| `macro` | Clustering algorithm used for reclustering in the offline stage (DSC_Macro) |

## Details

`update()` runs the online micro-clustering stage and only when macro cluster centers/weights are requested using `get_centers()` or `get_weights()`, then the offline stage reclustering is automatically performed.

Available clustering methods can be found in the See Also section below.

## Value

An object of class `DSC_TwoStage` (subclass of DSC, DSC_Macro) which is a named list with elements:

- `description`: a description of the clustering algorithms.

- micro: The DSD used for creating micro clusters in the online component.
- macro: The DSD for offline reclustering.
- state: an environment storing state information needed for reclustering.

with the two clusterers. The names are “

## Author(s)

Michael Hahsler

## See Also

Other DSC_TwoStage: DSC_DBSTREAM(), DSC_DStream(), DSC_evoStream()

Other DSC: DSC_Macro(), DSC_Micro(), DSC_R(), DSC_Static(), DSC(), animate_cluster(), evaluate.DSC, get_assignment(), plot.DSC(), predict(), prune_clusters(), read_saveDSC, recluster()

## Examples

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

# 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)
  )
win_km

update(win_km, stream, 200)
win_km
win_km$micro
win_km$macro

plot(win_km, stream)
evaluate_static(win_km, stream, assign = "macro")
```

---

# DSC_Window *A sliding window from a Data Stream*

---

## Description

Interface for DSO_Window. Represents the points in the sliding window as micro-clusters.

## 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).

## Author(s)

Michael Hahsler

## References

Zhu, Y. and Shasha, D. (2002). StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time, *International Conference of Very Large Data Bases (VLDB'02).*

## See Also

Other DSC_Micro: DSC_BICO(), DSC_BIRCH(), DSC_DBSTREAM(), DSC_DStream(), DSC_Micro(), DSC_Sample(), DSC_evoStream()

## Examples

```
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)
```

DSD *Data Stream Data Generator Base Classes*

## Description

Abstract base classes for DSD (Data Stream Data Generator).

## Usage

```
DSD(...)

DSD_R(...)
```

## Arguments

| | |
|---|---|
| ... | further arguments. |

## Details

The DSD class cannot be instantiated, but it serves as a abstract base class from which all DSD objects inherit. Implementations can be found in the See Also section below.

DSD provides common functionality like:

- get_points()
- print()
- plot()
- reset_stream() (if available)
- close_stream() (if needed)

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 called Foo would be:

1. A creator function DSD_Foo(...) and
2. a method get_points.DSD_Foo(x, n = 1L) for that class.

For details see vignette()

## Author(s)

Michael Hahsler

## See Also

Other DSD: DSD_BarsAndGaussians(), DSD_Benchmark(), DSD_Cubes(), DSD_Gaussians(), DSD_MG(), DSD_Memory(), DSD_Mixture(), DSD_NULL(), DSD_ReadDB(), DSD_ReadStream(), DSD_Target(), DSD_UniformNoise(), DSD_mlbenchData(), DSD_mlbenchGenerator(), DSF(), animate_data(), close_stream(), get_points(), plot.DSD(), reset_stream()

## Examples

```
DSD()

# 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)

# 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 = "pca")
```

---

# 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.

## Author(s)

Michael Hahsler

## See Also

DSD

Other DSD: DSD_Benchmark(), DSD_Cubes(), DSD_Gaussians(), DSD_MG(), DSD_Memory(), DSD_Mixture(), DSD_NULL(), DSD_ReadDB(), DSD_ReadStream(), DSD_Target(), DSD_UniformNoise(), DSD_mlbenchData(), DSD_mlbenchGenerator(), DSD(), DSF(), animate_data(), close_stream(), get_points(), plot.DSD(), reset_stream()

## Examples

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

get_points(stream, n = 10)
plot(stream)
```

---

DSD_Benchmark *Data Stream Generator for Dynamic Data Stream Benchmarks*

---

## Description

A data stream generator that generates several dynamic streams indented to be benchmarks to compare data stream clustering algorithms. The benchmarks can be used to test if a clustering algorithm can follow moving clusters, and merging and separating clusters.

## Usage

```
DSD_Benchmark(i = 1)
```

## Arguments

| | |
|---|---|
| i | integer; the number of the benchmark. |

## Details

Currently available benchmarks are:

- 1: two tight clusters moving across the data space with noise and intersect in the middle.
- 2: two clusters are located in two corners of the data space. A third cluster moves between the two clusters forth and back.

The benchmarks are created using DSD_MG.

## Value

Returns a DSD object.

## Author(s)

Michael Hahsler

## See Also

Other DSD: DSD_BarsAndGaussians(), DSD_Cubes(), DSD_Gaussians(), DSD_MG(), DSD_Memory(), DSD_Mixture(), DSD_NULL(), DSD_ReadDB(), DSD_ReadStream(), DSD_Target(), DSD_UniformNoise(), DSD_mlbenchData(), DSD_mlbenchGenerator(), DSD(), DSF(), animate_data(), close_stream(), get_points(), plot.DSD(), reset_stream()

## Examples

```
stream <- DSD_Benchmark(i = 1)
get_points(stream, n = 5)

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

stream <- DSD_Benchmark(i = 2)
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).

## Author(s)

Michael Hahsler

## See Also

Other DSD: DSD_BarsAndGaussians(), DSD_Benchmark(), DSD_Gaussians(), DSD_MG(), DSD_Memory(), DSD_Mixture(), DSD_NULL(), DSD_ReadDB(), DSD_ReadStream(), DSD_Target(), DSD_UniformNoise(), DSD_mlbenchData(), DSD_mlbenchGenerator(), DSD(), DSF(), animate_data(), close_stream(), get_points(), plot.DSD(), reset_stream()

## Examples

```
# create data stream with three clusters in 3D
stream <- DSD_Cubes(k = 3, d = 3, noise = 0.05)

get_points(stream, n = 5)

plot(stream)
```

---

# DSD_Gaussians *Mixture of Gaussians Data Stream Generator*

---

## Description

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

## Usage

```
DSD_Gaussians(
  k = 3,
  d = 2,
  p,
  mu,
  sigma,
  variance_limit = c(0.001, 0.002),
  separation = 6,
  space_limit = c(0, 1),
  noise = 0,
  noise_limit = space_limit,
  noise_separation = 3,
  separation_type = c("Euclidean", "Mahalanobis"),
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| k | Determines the number of clusters. |
| d | Determines the number of dimensions. |
| p | A vector of probabilities that determines the likelihood of generated a data point from a particular cluster. |

| | |
|---|---|
| mu | A matrix of means for each dimension of each cluster. |
| sigma | A list of length k of covariance matrices. |
| variance_limit | Lower and upper limit for the randomly generated variance when creating cluster covariance matrices. |
| separation | Minimum separation distance between clusters (measured in standard deviations according to separation_type). |
| space_limit | Defines the space bounds. All constructs are generated inside these bounds. For clusters this means that their centroids must be within these space bounds. |
| noise | Noise probability between 0 and 1. Noise is uniformly distributed within noise range (see below). |
| noise_limit | 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. |
| noise_separation | Minimum separation distance between cluster centers and noise points (measured in standard deviations according to separation_type). 0 means separation is ignored. |
| separation_type | The type of the separation distance calculation. It can be either Euclidean distance or Mahalanobis distance. |
| verbose | Report cluster and outlier generation process. |

## Details

DSD_Gaussians creates a mixture of k static clusters in a d-dimensional space. The cluster centers mu and the covariance matrices sigma can be supplied or will be randomly generated. The probability vector p defines for each cluster the probability that the next data point will be chosen from it (defaults to equal probability). Separation between generated clusters (and outliers; see below) can be imposed by using Euclidean or Mahalanobis distance, which is controlled by the separation_type parameter. Separation value then is supplied in the separation parameter. The generation method is similar to the one suggested by Jain and Dubes (1988).

Noise points which are uniformly chosen from noise_limit can be added.

Outlier points can be added. The outlier spatial positions predefined_outlier_space_positions and the outlier stream positions predefined_outlier_stream_positions can be supplied or will be randomly generated. Cluster and outlier separation distance is determined by and outlier_virtual_variance parameters. The outlier virtual variance defines an empty space around outliers, which separates them from their surrounding. Unlike noise, outliers are data points of interest for end-users, and the goal of outlier detectors is to find them in data streams. For more details, read the "Introduction to **stream**" vignette.

## Value

Returns a object of class DSD_Gaussian (subclass of DSD_R, DSD).

## Author(s)

Michael Hahsler

## References

Jain and Dubes (1988) Algorithms for clustering data, Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

## See Also

Other DSD: DSD_BarsAndGaussians(), DSD_Benchmark(), DSD_Cubes(), DSD_MG(), DSD_Memory(), DSD_Mixture(), DSD_NULL(), DSD_ReadDB(), DSD_ReadStream(), DSD_Target(), DSD_UniformNoise(), DSD_mlbenchData(), DSD_mlbenchGenerator(), DSD(), DSF(), animate_data(), close_stream(), get_points(), plot.DSD(), reset_stream()

## Examples

```
# Example 1: create data stream with three clusters in 3-dimensional data space
#            with 5 times sqrt(variance_limit) separation.
set.seed(1)
stream1 <- DSD_Gaussians(k = 3, d = 3)
stream1

get_points(stream1, n = 5)
plot(stream1, xlim = c(0, 1), ylim = c(0, 1))


# Example 2: create data stream with specified cluster positions,
# 5% 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)),
    p = c(.1, .9),
    variance_limit = c(0.02, 0.04),
    noise = 0.05,
    noise_limit = rbind(c(-1, 1), c(-1, 1)))

get_points(stream2, n = 5)
plot(stream2, xlim = c(-1, 1), ylim = c(-1, 1))


# Example 3: create 4 clusters and noise separated by a Mahalanobis
# distance. Distance to noise is increased to 6 standard deviations to make them
# easier detectable outliers.
stream3 <- DSD_Gaussians(k = 4, d = 2,
  separation_type = "Mahalanobis",
  space_limit = c(5, 20),
  variance_limit = c(1, 2),
  noise = 0.05,
  noise_limit = c(0, 25),
  noise_separation = 6
  )
plot(stream3)
```

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,
  outofpoints = c("warn", "ignore", "stop"),
  loop = FALSE,
  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 |
| outofpoints | Action taken if less than n data points are available. The default is to return the available data points with a warning. Other supported actions are:<br>• warn: return the available points (maybe an empty data.frame) with a warning.<br>• ignore: silently return the available points.<br>• stop: stop with an error. |
| loop | Should the stream start over when it reaches the end? |
| 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**.

**Reading the whole stream** By using n = -1 in get_points(), the whole stream is returned.

## Value

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

## Author(s)

Michael Hahsler

## See Also

Other DSD: DSD_BarsAndGaussians(), DSD_Benchmark(), DSD_Cubes(), DSD_Gaussians(), DSD_MG(), DSD_Mixture(), DSD_NULL(), DSD_ReadDB(), DSD_ReadStream(), DSD_Target(), DSD_UniformNoise(), DSD_mlbenchData(), DSD_mlbenchGenerator(), DSD(), DSF(), animate_data(), close_stream(), get_points(), plot.DSD(), reset_stream()

## Examples

```
# Example 1: 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")


# Example 2: 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))

# add some outliers
out <- runif(100) > .95
df[['.outlier']] <- out
df[['.class']] <- NA
head(df)

stream <- DSD_Memory(df)
stream

reset_stream(stream)
get_points(stream, n = 5)

# get the remaining points
```

```
rest <- get_points(stream, n = -1)
nrow(rest)

# plot all available points with n = -1
reset_stream(stream)
plot(stream, n = -1)
```

---

DSD_MG *DSD Moving Generator*

---

## Description

Creates an evolving DSD that consists of several MGC, each representing a moving cluster.

## Usage

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

add_cluster(x, c, label = NULL)

get_clusters(x)

remove_cluster(x, i)

## S3 method for class 'DSD_MG'
add_cluster(x, c, label = NULL)
```

## Arguments

| | |
|---|---|
| dimension | the dimension of the DSD object |
| ... | initial set of MGCs |
| description | An optional string used by print() to describe the data generator. |
| x | A DSD_MG object. |
| c | The cluster that should be added to 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. |
| i | The index of the cluster that should be removed from the DSD_MG object. |

## Details

This DSD is able to generate complex datasets that are able to evolve over a period of time. Its behavior is determined by as set of MGCs, each representing a moving cluster.

## Author(s)

Matthew Bolanos

## See Also

MGC for types of moving clusters.

Other DSD: DSD_BarsAndGaussians(), DSD_Benchmark(), DSD_Cubes(), DSD_Gaussians(), DSD_Memory(), DSD_Mixture(), DSD_NULL(), DSD_ReadDB(), DSD_ReadStream(), DSD_Target(), DSD_UniformNoise(), DSD_mlbenchData(), DSD_mlbenchGenerator(), DSD(), DSF(), animate_data(), close_stream(), get_points(), plot.DSD(), reset_stream()

## Examples

```
### 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)
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)
get_points(stream, n = 5)
plot(stream, xlim = c(-20,120), ylim = c(-20, 120))

if (interactive()) {
animate_data(stream, n = 5000, xlim = c(-20, 120), ylim = c(-20, 120))
}

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

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 = Shape_Block),
  MGC_Static(density = 10, center = c(.6, .5),   par = c(.2, .4),   shape = Shape_Block),
  MGC_Static(density = 5,  center = c(.39, .53), par = c(.16, .35), shape = Shape_Block),
  MGC_Noise( density = 1,  range = rbind(c(0,1), c(0,1))),
  labels = c(1, 1, 2, NA)
  )
stream

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 = 20, lambda = 0.1)
if (interactive()) {
purity <- animate_cluster(dbstream, stream, n = 2500, type = "micro",
  xlim = c(-10, 120), ylim = c(-10, 120), measure = "purity", horizon = 100)
}
```

---

DSD_Mixture *Mixes Data Points from Several Streams into a Single Stream*

---

## Description

This generator mixes multiple streams given specified probabilities. The streams have to contain the same number of dimensions.

## Usage

```
DSD_Mixture(..., prob = NULL)
```

## Arguments

| | |
|---|---|
| ... | DSD objects. |
| prob | a numeric vector with the probability for each stream that the next point will be drawn from that stream. |

## Value

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

## Author(s)

Michael Hahsler

## See Also

Other DSD: DSD_BarsAndGaussians(), DSD_Benchmark(), DSD_Cubes(), DSD_Gaussians(), DSD_MG(), DSD_Memory(), DSD_NULL(), DSD_ReadDB(), DSD_ReadStream(), DSD_Target(), DSD_UniformNoise(), DSD_mlbenchData(), DSD_mlbenchGenerator(), DSD(), DSF(), animate_data(), close_stream(), get_points(), plot.DSD(), reset_stream()

## Examples

```
# create data stream with three clusters in 2D
stream1 <- DSD_Gaussians(d = 2, k = 3)
stream2 <- DSD_UniformNoise(d = 2,  range = rbind(c(-.5, 1.5), c(-.5, 1.5)))

combinedStream <- DSD_Mixture(stream1, stream2, prob = c(.9, .1))
combinedStream

get_points(combinedStream, n = 20)
plot(combinedStream, n = 200)
```

---

# DSD_mlbenchData *Stream Interface for Data Sets From mlbench*

---

## 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 | logical; loop or not to loop over the data frame. |
| random | logical; should the data be used a random order? |
| scale | logical; apply scaling to the data? |

## Details

The DSD_mlbenchData class is designed to be a wrapper class for data from the mlbench package.

All data is held in memory in either data frame or matrix form. It is served as a stream using the DSD_Memory class. The stream can be reset to position 1 using reset_stream().

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.

### Author(s)

Michael Hahsler and Matthew Bolanos

### See Also

Other DSD: DSD_BarsAndGaussians(), DSD_Benchmark(), DSD_Cubes(), DSD_Gaussians(), DSD_MG(), DSD_Memory(), DSD_Mixture(), DSD_NULL(), DSD_ReadDB(), DSD_ReadStream(), DSD_Target(), DSD_UniformNoise(), DSD_mlbenchGenerator(), DSD(), DSF(), animate_data(), close_stream(), get_points(), plot.DSD(), reset_stream()

### Examples

```
DSD_mlbenchData()

stream <- DSD_mlbenchData("Shuttle")
stream

get_points(stream, n = 5)

plot(stream, n = 100)
```

---

## DSD_mlbenchGenerator *mlbench Data Stream Generator*

---

### Description

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

### Usage

```
DSD_mlbenchGenerator(method, ...)
```

### Arguments

| | |
|---|---|
| method | The name of the mlbench data generator. If missing then a list of all available generators is shown and returned. |
| ... | Parameters for 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)

## Author(s)

John Forrest

## See Also

Other DSD: DSD_BarsAndGaussians(), DSD_Benchmark(), DSD_Cubes(), DSD_Gaussians(), DSD_MG(), DSD_Memory(), DSD_Mixture(), DSD_NULL(), DSD_ReadDB(), DSD_ReadStream(), DSD_Target(), DSD_UniformNoise(), DSD_mlbenchData(), DSD(), DSF(), animate_data(), close_stream(), get_points(), plot.DSD(), reset_stream()

## Examples

```
DSD_mlbenchGenerator()

stream <- DSD_mlbenchGenerator(method = "cassini")
stream

get_points(stream, n = 5)

plot(stream, n = 500)
```

---

# DSD_NULL *Placeholder for a DSD Stream*

---

## Description

Placeholder for a DSD. DSD_NULL does not produce points and creates an error for get_points().

## Usage

```
DSD_NULL()
```

## Value

Returns a DSD_NULL object (subclass of DSD).

## Author(s)

Michael Hahsler

## See Also

Other DSD: DSD_BarsAndGaussians(), DSD_Benchmark(), DSD_Cubes(), DSD_Gaussians(), DSD_MG(), DSD_Memory(), DSD_Mixture(), DSD_ReadDB(), DSD_ReadStream(), DSD_Target(), DSD_UniformNoise(), DSD_mlbenchData(), DSD_mlbenchGenerator(), DSD(), DSF(), animate_data(), close_stream(), get_points(), plot.DSD(), reset_stream()

## Examples

```
nullstream <- DSD_NULL()
nullstream

## This will produce an error
## Not run:
get_points(nullstream)
## End(Not run)
```

---

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 data base interface (DBI).

## Usage

```
DSD_ReadDB(
  result,
  k = NA,
  outofpoints = c("warn", "ignore", "stop"),
  description = NULL
)

## S3 method for class 'DSD_ReadDB'
close_stream(dsd, disconnect = TRUE, ...)
```

## Arguments

| | |
|---|---|
| result | An open DBI result set. |
| k | Number of true clusters, if known. |
| outofpoints | Action taken if less than n data points are available. The default is to return the available data points with a warning. Other supported actions are:<br>• warn: return the available points (maybe an empty data.frame) with a warning.<br>• ignore: silently return the available points.<br>• stop: stop with an error. |
| description | a character string describing the data. |
| dsd | a stream. |
| disconnect | logical; disconnect from the database? |
| ... | further arguments. |

## Details

This class provides a streaming interface for result sets from a data base with via DBI::DBI. You need to connect to the data base and submit a SQL query using `DBI::dbGetQuery()` to obtain a result set. Make sure that your query only includes the columns that should be included in the stream (including class and outlier marking columns).

### Closing and resetting the stream

Do not forget to clear the result set and disconnect from the data base connection. `close_stream()` clears the query result with `DBI::dbClearResult()` and the disconnects from the database with `DBI::dbDisconnect()`. Disconnecting can be prevented by calling `close_stream()` with `disconnect = FALSE`.

`reset_stream()` is not available for this type of stream.

### Additional information

If additional information is available (e.g., class information), then the SQL statement needs to make sure that the columns have the appropriate name starting with `.`. See Examples section below.

## Value

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

## Author(s)

Michael Hahsler

## See Also

`DBI::dbGetQuery()`

Other DSD: `DSD_BarsAndGaussians()`, `DSD_Benchmark()`, `DSD_Cubes()`, `DSD_Gaussians()`, `DSD_MG()`, `DSD_Memory()`, `DSD_Mixture()`, `DSD_NULL()`, `DSD_ReadStream()`, `DSD_Target()`, `DSD_UniformNoise()`, `DSD_mlbenchData()`, `DSD_mlbenchGenerator()`, `DSD()`, `DSF()`, `animate_data()`, `close_stream()`, `get_points()`, `plot.DSD()`, `reset_stream()`

## Examples

```
### create a data base with a table with 3 Gaussians
if(require("RSQLite")) {

library("RSQLite")
con <- dbConnect(RSQLite::SQLite(), ":memory:")

points <- get_points(DSD_Gaussians(k = 3, d = 2), n = 110)
head(points)

dbWriteTable(con, "Gaussians", points)

### prepare a query result set. Make sure that the additional information
### column starts with .
res <- dbSendQuery(con, "SELECT X1, X2, `.class` AS '.class' FROM Gaussians")
res
```

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

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

plot(stream, n = 100)

close_stream(stream)
}
```

---

DSD_ReadStream *Read a Data Stream from a File or a Connection*

---

## Description

A DSD class that reads a data stream (text format) from a file or any R connection.

## Usage

```
DSD_ReadStream(
  file,
  k = NA,
  take = NULL,
  sep = ",",
  header = FALSE,
  skip = 0,
  col.names = NULL,
  colClasses = NA,
  outofpoints = c("warn", "ignore", "stop"),
  ...
)

DSD_ReadCSV(
  file,
  k = NA,
  take = NULL,
  sep = ",",
  header = FALSE,
  skip = 0,
  col.names = NULL,
  colClasses = NA,
  outofpoints = c("warn", "ignore", "stop"),
  ...
)
```

```
## S3 method for class 'DSD_ReadStream'
close_stream(dsd, ...)

## S3 method for class 'DSD_ReadCSV'
close_stream(dsd, ...)
```

## Arguments

| | |
|---|---|
| file | A file/URL or an open connection. |
| k | Number of true clusters, if known. |
| take | indices of columns to extract from the file. |
| 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. |
| col.names | A vector of optional names for the variables. The default is to use "V" followed by the column number. Additional information (e.g., class labels) need to have names starting with .. |
| colClasses | A vector of classes to be assumed for the columns passed on to read.table(). |
| outofpoints | Action taken if less than n data points are available. The default is to return the available data points with a warning. Other supported actions are:<br>• warn: return the available points (maybe an empty data.frame) with a warning.<br>• ignore: silently return the available points.<br>• stop: stop with an error. |
| ... | Further arguments are passed on to read.table(). This can for example be used for encoding, quotes, etc. |
| dsd | A object of class DSD_ReadCSV. |

## Details

DSD_ReadStream uses readLines() and read.table() to read data from an R connection line-by-line and convert it into a data.frame. 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 implementation tries to gracefully deal with slightly corrupted data by dropping points with inconsistent reading and producing a warning. However, this might not always be possible resulting in an error instead.

### Column names

If the file has column headers in the first line, then they can be used by setting header = TRUE. Alternatively, column names can be set using col.names or a named vector for take. If no column names are specified then default names will be created.

Columns with names that start with . are considered information columns and are ignored by DSTs. See get_points() for details.

Other information columns are are used by various functions.

**Reading the whole stream** By using `n = -1` in `get_points()`, the whole stream is returned.

**Resetting and closing a stream**

The position in the file can be reset to the beginning or another position using `reset_stream()`. This fails of the underlying connection is not seekable (see connection).

`DSD_ReadStream` maintains an open connection to the stream and needs to be closed using `close_stream()`.

`DSD_ReadCSV` reads a stream from a comma-separated values file.

## Value

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

## Author(s)

Michael Hahsler

## See Also

`readLines()`, `read.table()`.

Other DSD: `DSD_BarsAndGaussians()`, `DSD_Benchmark()`, `DSD_Cubes()`, `DSD_Gaussians()`, `DSD_MG()`, `DSD_Memory()`, `DSD_Mixture()`, `DSD_NULL()`, `DSD_ReadDB()`, `DSD_Target()`, `DSD_UniformNoise()`, `DSD_mlbenchData()`, `DSD_mlbenchGenerator()`, `DSD()`, `DSF()`, `animate_data()`, `close_stream()`, `get_points()`, `plot.DSD()`, `reset_stream()`

## Examples

```
# Example 1: creating data and writing it to disk
stream <- DSD_Gaussians(k = 3, d = 2)
write_stream(stream, "data.txt", n = 100, info = TRUE, header = TRUE)
readLines("data.txt", n = 5)

# reading the same data back
stream2 <- DSD_ReadStream("data.txt", header = TRUE)
stream2

# get points
get_points(stream2, n = 5)
plot(stream2, n = 20)

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

# Example 2:  Read part of the kddcup1999 data (take only cont. variables)
# col 42 is the class variable
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 = "pca")

close_stream(stream)
```

---

`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

This DSD will produce a singular Gaussian cluster in the center with a ring around it.

## Value

Returns a `DSD_Target` object.

## Author(s)

Michael Hahsler

## See Also

Other DSD: DSD_BarsAndGaussians(), DSD_Benchmark(), DSD_Cubes(), DSD_Gaussians(), DSD_MG(), DSD_Memory(), DSD_Mixture(), DSD_NULL(), DSD_ReadDB(), DSD_ReadStream(), DSD_UniformNoise(), DSD_mlbenchData(), DSD_mlbenchGenerator(), DSD(), DSF(), animate_data(), close_stream(), get_points(), plot.DSD(), reset_stream()

## Examples

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

get_points(stream, n = 5)

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).

## Author(s)

Michael Hahsler

## See Also

Other DSD: DSD_BarsAndGaussians(), DSD_Benchmark(), DSD_Cubes(), DSD_Gaussians(), DSD_MG(), DSD_Memory(), DSD_Mixture(), DSD_NULL(), DSD_ReadDB(), DSD_ReadStream(), DSD_Target(), DSD_mlbenchData(), DSD_mlbenchGenerator(), DSD(), DSF(), animate_data(), close_stream(), get_points(), plot.DSD(), reset_stream()

## Examples

```
# create data stream with three clusters in 2D
stream <- DSD_UniformNoise(d = 2)
get_points(stream, n = 5)
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)
```

---

DSF *Data Stream Filter Base Classes*

---

## Description

Abstract base classes for all data stream filter (DSF) classes. Data stream filters transform a data stream (DSD).

## Usage

```
DSF(...)

## S3 method for class 'DSF'
reset_stream(dsd, pos = 1)

## S3 method for class 'DSF'
close_stream(dsd, ...)
```

## Arguments

| | |
|---|---|
| ... | Further arguments. |
| dsd | a stream object of class DSD. |
| pos | position in the stream. |

## Details

The DSF class cannot be instantiated, but it serve as a base class from which other DSF classes inherit.

Data stream filters transform a DSD data stream. DSF implementations inherit from DSD and have the same basic interface.

reset_stream() resets the source stream.

It is convenient to use the pipe (magrittr::%>%) to apply filters to data streams (see Examples section).

### Methods (by generic)

- `reset_stream(DSF)`: reset the attached stream if reset is supported.
- `close_stream(DSF)`: close the attached stream if close is supported.

### Author(s)

Michael Hahsler

### See Also

Other DSF: `DSF_Convolve()`, `DSF_Downsample()`, `DSF_ExponentialMA()`, `DSF_Func()`, `DSF_dplyr()`

Other DSD: `DSD_BarsAndGaussians()`, `DSD_Benchmark()`, `DSD_Cubes()`, `DSD_Gaussians()`, `DSD_MG()`, `DSD_Memory()`, `DSD_Mixture()`, `DSD_NULL()`, `DSD_ReadDB()`, `DSD_ReadStream()`, `DSD_Target()`, `DSD_UniformNoise()`, `DSD_mlbenchData()`, `DSD_mlbenchGenerator()`, `DSD()`, `animate_data()`, `close_stream()`, `get_points()`, `plot.DSD()`, `reset_stream()`

### Examples

```
DSF()

stream <- DSD_Gaussians(k = 3, d = 2) %>%
  DSF_Func(function(x) cbind(x, Xsum = x$X1 + x$X2))
stream

get_points(stream, n = 5)
```

---

## 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.

### Usage

```
DSFP(...)
```

### Arguments

| | |
|---|---|
| ... | Further arguments. |

### Author(s)

Michael Hahsler

## See Also

DST

## Examples

```
DSFP()
```

---

DSF_Convolve *Apply a Filter to a Data Stream*

---

## Description

Applies a filter (i.e., a convolution with a filter kernel) to a data stream.

## Usage

```
DSF_Convolve(
  dsd,
  dim = NULL,
  kernel = NULL,
  pre = NULL,
  post = NULL,
  na.rm = FALSE,
  replace = TRUE,
  name = NULL
)

filter_MA(width)

filter_Hamming(width)

filter_diff(lag)

filter_Sinc(fc, fs, width = NULL, bw = NULL)

pow2(x)
```

## Arguments

| | |
|---|---|
| dsd | A object of class DSD. |
| dim | columns to which the filter should be applied. Default is all columns. |
| kernel | filter kernel as a numeric vector of weights. |
| pre, post | functions to be applied before and after the convolution. |
| na.rm | logical; should NAs be ignored? |
| replace | logical; should the column be replaced or a column with the convolved column added? |

| | |
|---|---|
| name | character; the new column will be name with the old column name + _ + name. |
| width | filter width. |
| lag | an integer indicating which time lag to use. |
| fc | cutoff frequency. |
| fs | sampling frequency. |
| bw | transition bandwidth. |
| x | values to be squared. |

## Details

A filter kernel is a vector with kernel weights. A few filter are provided.

- filter_MA(width) creates a moving average.
- filter_diff(lag) calculates laged differences. Note that na.rm = TRUE will lead to artifacts and should not be used.
- filter_Hamming(width) creates a Hamming window.
- filter_Sinc(fc, fs, width, bw) creates a windowed-sinc filter. One of width (filter length) or bw (transition bandwidth can be used to control the filter roll-off. The relationship is $width = 4/bw$. See Chapter 16 in Smith (1997).

pre and post are functions that are called before and after the convolution. For example, to calculate RMS, you can use pre = pow2 and post = sqrt. pow2() is a convenience function.

## Value

An object of class DSF_Convolve (subclass of DSF and DSD).

## Author(s)

Michael Hahsler

## References

Steven W. Smith, The Scientist and Engineer's Guide to Digital Signal Processing, California Technical Pub; 1st edition (January 1, 1997). ISBN 0966017633, URL: https://www.dspguide.com/

## See Also

stats::filter provides non-streaming convolution.

Other DSF: DSF_Downsample(), DSF_ExponentialMA(), DSF_Func(), DSF_dplyr(), DSF()

## Examples

```
data(presidents)

## Example 1: Create a data stream with three copies of president approval ratings.
## We will use several convolutions.
stream <- data.frame(
    approval_orig = presidents,
    approval_MA = presidents,
    approval_diff1 = presidents,
    .time = time(presidents)) %>%
  DSD_Memory()

plot(stream, dim = 1, n = 120, method = "ts")

## apply a moving average filter to dimension 1 (using the column name) and diff to dimension 3
filteredStream <- stream %>%
  DSF_Convolve(kernel = filter_MA(5), dim = "approval_orig", na.rm = TRUE) %>%
  DSF_Convolve(kernel = filter_diff(1), dim = 3)
filteredStream

## resetting the filtered stream also resets the original stream
reset_stream(filteredStream)
ps <- get_points(filteredStream, n = 120)
head(ps)

year <- ps[[".time"]]
approval <- remove_info(ps)
matplot(year, approval, type = "l", ylim = c(-20, 100))
legend("topright", colnames(approval), col = 1:3, lty = 1:3, bty = "n")

## Example 2: Create a stream with a constant sine wave and apply
## a moving average, an RMS envelope and a differences
stream <- DSD_Memory(data.frame(y = sin(seq(0, 2 * pi - (2 * pi / 100) ,
  length.out = 100))), loop = TRUE)
plot(stream, n = 200, method = "ts")

filteredStream <- stream %>%
  DSF_Convolve(kernel = filter_MA(100), dim = 1,
    replace = FALSE, name = "MA") %>%
  DSF_Convolve(kernel = filter_MA(100), pre = pow2, post = sqrt, dim = 1,
    replace = FALSE, name = "RMS") %>%
  DSF_Convolve(kernel = filter_diff(1), dim = 1,
    replace = FALSE, name = "diff1")
filteredStream

ps <- get_points(filteredStream, n = 500)
head(ps)

matplot(ps, type = "l")
legend("topright", colnames(ps), col = 1:4, lty = 1:4)

## Note that MA and RMS use a window of length 200 and are missing at the
```

```
##   beginning of the stream the window is full.

## Filters: look at different filters
filter_MA(5)
filter_diff(1)
plot(filter_Hamming(20), type = "h")
plot(filter_Sinc(10, 100, width = 20), type = "h")
```

---

`DSF_Downsample` *Downsample a Data Stream*

---

## Description

Creates a new stream that reduces the frequency of a given stream by a given factor.

## Usage

```
DSF_Downsample(dsd, factor = 1)
```

## Arguments

| | |
|---|---|
| `dsd` | The input stream as an DSD object. |
| `factor` | the downsampling factor. |

## Value

An object of class `DSF_Downsample` (subclass of DSF and DSD).

## Author(s)

Michael Hahsler

## See Also

Other DSF: `DSF_Convolve()`, `DSF_ExponentialMA()`, `DSF_Func()`, `DSF_dplyr()`, `DSF()`

## Examples

```
# Simple downsampling example
stream <- DSD_Memory(data.frame(rownum = seq(100))) %>% DSF_Downsample(factor = 10)
stream

get_points(stream, n = 2)
get_points(stream, n = 1)
get_points(stream, n = 5)

# DSD_Memory supports getting the remaining points using n = -1
get_points(stream, n = -1)
```

```
# Downsample a time series
data(presidents)

stream <- data.frame(
    presidents,
    .time = time(presidents)) %>%
  DSD_Memory()

plot(stream, dim = 1, n = 120, method = "ts")

# downsample by taking only every 3rd data point (quarters)
downsampledStream <- stream %>% DSF_Downsample(factor = 3)

reset_stream(downsampledStream)
plot(downsampledStream, dim = 1, n = 40, method = "ts")
```

---

DSF_dplyr *Apply a dplyr Transformation to a Data Stream*

---

## Description

Applies dplyr transformations to a data stream.

## Usage

```
DSF_dplyr(dsd, func = NULL, info = FALSE)
```

## Arguments

| | |
|---|---|
| dsd | A object of class DSD. |
| func | a dplyr expression. |
| info | logical; does the function also receive and modify the info columns? |

## Details

**dplyr** needs to be installed and loaded with `library(dplyr)` before `DSF_dplyr` can be used.

Since streams are processed one point or block at a time, only dplyr::dplyr operations that work on individual rows are allowed on streams. Examples are:

- `dplyr::select()`
- `dplyr::mutate()`
- `dplyr::rename()`
- `dplyr::transmute()`
- `dplyr::filter()`

Summary functions can be used, but will only be applied to the requested part of the stream of length `n`.

`DSF_dplyr()` calls the function using `points %>% <func>` and multiple `dplyr` functions can be applied by using `%>%` between them.

## Value

An object of class DSF_dplyr (subclass of DSF and DSD).

## Author(s)

Michael Hahsler

## See Also

Other DSF: DSF_Convolve(), DSF_Downsample(), DSF_ExponentialMA(), DSF_Func(), DSF()

## Examples

```
if (require(dplyr)) {

library(dplyr)

stream <- DSD_Gaussians(k = 3, d = 3)
plot(stream, xlim = c(0, 1), ylim = c(0, 1))

# 1. Select only columns X1 and X2
# 2. filter points by X1 > .5 (Note that the info columns also need to be filtered!)
# 3. Add a sum columns

stream2 <- stream %>%
  DSF_dplyr(select(X1, X2)) %>%
  DSF_dplyr(filter(X1 > .5), info = TRUE) %>%
  DSF_dplyr(mutate(Xsum = X1 + X2))
stream2

# Note: you get fewer points because of the filter operation.
get_points(stream2, n = 10)
plot(stream2, xlim = c(0, 1), ylim = c(0, 1))

}
```

---

# DSF_ExponentialMA *Exponential Moving Average over a Data Stream*

---

## Description

Applies an exponential moving average to components of a data stream.

## Usage

```
DSF_ExponentialMA(dsd, dim = NULL, alpha = 0.5)
```

## Arguments

| | |
|---|---|
| dsd | The input stream as an DSD object. |
| dim | columns to which the filter should be applied. Default is all columns. |
| alpha | smoothing coefficient in $[0, 1]$. Larger means discounting older observations faster. |

## Details

The exponential moving average is calculated by:

$S_t = \alpha Y_t + (1 - \alpha)\ S_{i-1}$

with $S_0 = Y_0$.

## Value

An object of class DSF_ExponentialMA (subclass of DSF and DSD).

## Author(s)

Michael Hahsler

## See Also

Other DSF: DSF_Convolve(), DSF_Downsample(), DSF_Func(), DSF_dplyr(), DSF()

## Examples

```
# Smooth a time series
data(presidents)

stream <- data.frame(
    presidents,
    .time = time(presidents)) %>%
  DSD_Memory()

plot(stream, dim = 1, n = 120, method = "ts", main = "Original")

smoothStream <- stream %>% DSF_ExponentialMA(alpha = .7)
smoothStream

reset_stream(smoothStream)
plot(smoothStream, dim = 1, n = 120, method = "ts", main = "With ExponentialMA(.7)")
```

DSF_Func *Apply a Function to Transformation to a Data Stream*

## Description

Applies an R function to transform to a data stream.

## Usage

```
DSF_Func(dsd, func = NULL, ..., info = FALSE)
```

## Arguments

| | |
|---|---|
| dsd | A object of class DSD. |
| func | a function that takes a data.frame as the first argument and returns the transformed data.frame. |
| ... | further arguments are passed on to the function specified in func. |
| info | logical; does the function also receive and modify the info columns? |

## Details

The function's first argument needs to be a data.frame representing points of the data stream. The function will be called as ps %>% your_function(), where ps is the data.frame with some points obtained using get_points() on the data stream source.

## Value

An object of class DSF_Func (subclass of DSF and DSD).

## Author(s)

Michael Hahsler

## See Also

Other DSF: DSF_Convolve(), DSF_Downsample(), DSF_ExponentialMA(), DSF_dplyr(), DSF()

## Examples

```
stream <- DSD_Gaussians(k = 3, d = 3)
get_points(stream, n = 5)

## Example 1: rename the columns
rename <- function(x, names) {
  colnames(x) <-  names
  x
}
```

```
# By default, the info columns starting with . are not affected.
stream2 <- stream %>% DSF_Func(rename, names = c("A", "B", "C"))
stream2
get_points(stream2, n = 5)

## Example 2: add a sum columns
stream3 <- stream2 %>% DSF_Func(function(x) {
  x$sum = rowSums(x)
  x
})
stream3
get_points(stream3, n = 5)

## Example 3: Project the stream on its first 2 PCs (using a sample)
pr <- princomp(get_points(stream, n = 100, info = FALSE))
pca_trans <- function(x) predict(pr, x[, c("X1", "X2", "X3")])[, 1:2 , drop = FALSE]
pca_trans(get_points(stream, n = 3, info = FALSE))

stream4 <- stream %>% DSF_Func(pca_trans)
stream4

get_points(stream4, n = 3)
plot(stream4)

## Example 4: Change a class labels using info = TRUE. We redefine class 3 as noise (NA)
stream5 <- stream %>% DSF_Func(
  function(x) { x[['.class']][x[['.class']] == 3] <- NA; x },
  info = TRUE)
stream5

get_points(stream5, n = 5)
plot(stream5)
```

---

DSF_Scale *Scale a Data Stream*

---

## Description

Make an unscaled data stream into a scaled data stream.

## Usage

```
DSF_Scale(dsd, dim = NULL, center = TRUE, scale = TRUE, n = 100)

DSD_ScaleStream(dsd, dim = NULL, center = TRUE, scale = TRUE, n = 100)
```

## Arguments

| | |
|---|---|
| dsd | A object of class DSD that will be scaled. |
| dim | integer vector or names of dimensions that should be scaled? Default is all. |

| | |
|---|---|
| center, scale | logical or a numeric vector of length equal to the number of columns (selected with dim) used for centering/scaling (see function scale). |
| n | The number of points used by scale_stream() to creating the centering/scaling |

## Details

If center and scale are not vectors with scaling factors, then scale_stream() estimates the values for centering and scaling (see scale in **base**) using n points from the stream and the stream is reset if reset = TRUE and the DSD object supports resetting.

## Value

An object of class DSF_Scale (subclass of DSF and DSD).

## Deprecated

DSD_ScaleStream is deprecated. Use DSF_Scale instead.

## Author(s)

Michael Hahsler

## See Also

scale in **base**

Other DST: DSAggregate(), DSClassifier(), DSC(), DSOutlier(), DST_Runner(), DST_WriteStream(), DST(), evaluate, predict(), update()

## Examples

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

# scale with manually calculated scaling factors
points <- get_points(stream, n = 100, info = FALSE)
center <- colMeans(points)
scale <- apply(points, MARGIN = 2, sd)

scaledStream <- stream %>%  DSF_Scale(center = center, scale = scale)
colMeans(get_points(scaledStream, n = 100, info = FALSE))
apply(get_points(scaledStream, n = 100, info = FALSE), MARGIN = 2, sd)

# let DSF_Scale calculate the scaling factors
scaledStream <- stream %>% DSF_Scale(n = 100)
colMeans(get_points(scaledStream, n = 100, info = FALSE))
apply(get_points(scaledStream, n = 100, info = FALSE), MARGIN = 2, sd)

## scale only X2
scaledStream <- stream %>% DSF_Scale(n = 100, dim = "X2")
colMeans(get_points(scaledStream, n = 100, info = FALSE))
apply(get_points(scaledStream, n = 100, info = FALSE), MARGIN = 2, sd)
```

DSOutlier *Abstract Class for Data Stream Outlier Detectors*

## Description

The abstract class for all data stream outlier detectors. Cannot be instantiated. Some DSC implementations also implement outlier/noise detection.

## Usage

```
DSOutlier(...)
```

## Arguments

| | |
|---|---|
| ... | further arguments. |

## Details

plot() has an extra logical argument to specify if outliers should be plotted as red crosses.

## Author(s)

Michael Hahsler

## See Also

Other DST: DSAggregate(), DSClassifier(), DSC(), DSF_Scale(), DST_Runner(), DST_WriteStream(), DST(), evaluate, predict(), update()

Other DSOutlier: DSC_DBSTREAM(), DSC_DStream()

## Examples

```
DSOutlier()

#' @examples
set.seed(1000)
stream <- DSD_Gaussians(k = 3, d = 2, noise = 0.1, noise_separation = 5)

outlier_detector <- DSOutlier_DBSTREAM(r = .05, outlier_multiplier = 2)
update(outlier_detector, stream, 500)
outlier_detector

points <- get_points(stream, 20)
points

# Outliers are predicted as class NA
predict(outlier_detector, points)

# Plot new points from the stream. Predicted outliers are marked with a red x.
```

```
plot(outlier_detector, stream)

evaluate_static(outlier_detector, stream, measure =
  c("noiseActual", "noisePredicted", "noisePrecision", "outlierJaccard"))

# use a different detector
outlier_detector2 <- DSOutlier_DStream(gridsize = .05, Cl = 0.5, outlier_multiplier = 2)
update(outlier_detector2, stream, 500)
plot(outlier_detector2, stream)

evaluate_static(outlier_detector2, stream, measure =
  c("noiseActual", "noisePredicted", "noisePrecision", "outlierJaccard"))
```

---

DST *Conceptual Base Class for All Data Stream Mining Tasks*

---

## Description

Conceptual base class for all data stream mining tasks.

## Usage

```
DST(...)

description(x, ...)

## S3 method for class 'DST'
description(x, ...)
```

## Arguments

| | |
|---|---|
| ... | Further arguments. |
| x | an object of a concrete implementation of a DST. |

## Details

Base class for data stream mining tasks. Types of DST are

- DSC for data stream clustering.
- DSAggregate to aggregate data streams (e.g., with a sliding window).
- DSFP frequent pattern mining for data stream clustering.
- DSClassifier classification for data streams.
- DSOutlier outlier detection for data streams.

The common interface for all DST classes consists of

- update()
- predict()

and the methods in the Methods Section below.

### Methods (by generic)

- description(DST): Get a description of the task as a character string.

### Author(s)

Michael Hahsler

### See Also

Other DST: DSAggregate(), DSClassifier(), DSC(), DSF_Scale(), DSOutlier(), DST_Runner(), DST_WriteStream(), evaluate, predict(), update()

### Examples

```
DST()
```

---

## DST_Multi *Apply Multiple Task to the Same Data Stream*

---

### Description

Apply multiple task (DST) to the same data stream. The tasks can be accessed as a list as $dsts.

### Usage

```
DST_Multi(dsts)
```

### Arguments

| | |
|---|---|
| dsts | a list of DST objects. |

### Author(s)

Michael Hahsler

### Examples

```
set.seed(1500)

stream <- DSD_Gaussians(k = 3, d = 2)

## define multiple tasks as a list
tasks <- DST_Multi(list(
   DSAggregate_Window(horizon = 10),
   DSC_DStream(gridsize = 0.1)
))
tasks

## update both tasks with the same stream
```

```
update(tasks, stream, n = 1000)

## inspect the results of the tasks
tasks$dsts[[1]]
get_points(tasks$dsts[[1]])

tasks$dsts[[2]]
plot(tasks$dsts[[2]])
```

---

DST_Runner *Create a Data Stream Pipeline*

---

## Description

Define a complete data stream pipe line consisting of a data stream, filters and a data mining task using %>%.

## Usage

```
DST_Runner(dsd, dst)
```

## Arguments

| | |
|---|---|
| dsd | A data stream (subclass of DSD) typically provided using a %>% (pipe). |
| dst | A data stream mining task (subclass of DST). |

## Details

A data stream pipe line consisting of a data stream, filters and a data mining task:

DSD %>% DSF %>% DST

Once the pipeline is defined, it can be run using update() where points are taken from the DSD, filtered through a sequence of DSFs and then used to update the task DST.

## Author(s)

Michael Hahsler

## See Also

Other DST: DSAggregate(), DSClassifier(), DSC(), DSF_Scale(), DSOutlier(), DST_WriteStream(), DST(), evaluate, predict(), update()

## Examples

```
set.seed(1500)

# Set up a pipeline with a DSD data source, DSF Filters and then a DST task
cluster_pipeline <- DSD_Gaussians(k = 3, d = 2) %>%
                    DSF_Scale() %>%
                    DST_Runner(DSC_DBSTREAM(r = .05))

cluster_pipeline

# the DSD and DST can be accessed directly
cluster_pipeline$dsd
cluster_pipeline$dst

# update the DST using the pipeline
update(cluster_pipeline, n = 1000)

cluster_pipeline$dst
get_centers(cluster_pipeline$dst)
plot(cluster_pipeline$dst)
```

---

# DST_WriteStream *Task to Write a Stream to a File or a Connection*

---

## Description

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

## Usage

```
DST_WriteStream(file, append = TRUE, ...)
```

## Arguments

| | |
|---|---|
| file | A file name or a R connection to be written to. |
| append | Append the data to an existing file. |
| ... | further arguments are passed on to write_stream(). |

## Author(s)

Michael Hahsler

## See Also

Other DST: DSAggregate(), DSClassifier(), DSC(), DSF_Scale(), DSOutlier(), DST_Runner(), DST(), evaluate, predict(), update()

## Examples

```
set.seed(1500)

stream <- DSD_Gaussians(k = 3, d = 2)
writer <- DST_WriteStream(file = "data.txt", info = TRUE, header = TRUE)

update(writer, stream, n = 2)
readLines("data.txt")
update(writer, stream, n = 3)
readLines("data.txt")

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

---

# evaluate *Evaluate a Data Stream Mining Task*

---

## Description

Calculate evaluation measures for a data stream mining task DST using a data stream DSD object.

## Usage

```
evaluate_static(object, dsd, measure, n, ...)

evaluate_stream(object, dsd, measure, n, horizon, ..., verbose = FALSE)
```

## Arguments

| | |
|---|---|
| object | The DST object that the evaluation measure is being requested from. |
| dsd | The DSD object used to create the test data. |
| measure | Evaluation measure(s) to use. If missing then all available measures are returned. |
| n | The number of data points being requested. |
| ... | Further arguments. |
| horizon | Evaluation is done using horizon many previous points (see detail section). |
| verbose | Report progress? |

## Details

We provide two evaluation methods:

- `evaluate_static()` evaluates the current DST model on new data without updating the model.

- `evaluate_stream()` evaluates the DST model using *prequential error estimation* (see Gama, Sebastiao and Rodrigues; 2013). The data points in the horizon are first used to calculate the evaluation measure and then they are used for updating the cluster model. A horizon of ‘ means that each point is evaluated and then used to update the model.

The evaluation measures depend on the task.

## Value

`evaluate` returns an object of class `stream_eval` which is a numeric vector of the values of the requested measures.

## Author(s)

Michael Hahsler

## References

Joao Gama, Raquel Sebastiao, Pedro Pereira Rodrigues (2013). On evaluating stream learning algorithms. *Machine Learning,* March 2013, Volume 90, Issue 3, pp 317-346.

## See Also

Other DST: `DSAggregate()`, `DSClassifier()`, `DSC()`, `DSF_Scale()`, `DSOutlier()`, `DST_Runner()`, `DST_WriteStream()`, `DST()`, `predict()`, `update()`

Other evaluation: `animate_cluster()`, `evaluate.DSC`

---

# evaluate.DSC *Evaluate Stream Clustering*

---

## Description

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

## Usage

```
## S3 method for class 'DSC'
evaluate_static(
  object,
  dsd,
  measure,
  n = 100,
  type = c("auto", "micro", "macro"),
  assign = "micro",
  assignmentMethod = c("auto", "model", "nn"),
  excludeNoise = FALSE,
  callbacks = list(),
```

```
  ...
)

## S3 method for class 'DSC'
evaluate_stream(
  object,
  dsd,
  measure,
  n = 1000,
  horizon = 100,
  type = c("auto", "micro", "macro"),
  assign = "micro",
  assignmentMethod = c("auto", "model", "nn"),
  excludeNoise = FALSE,
  callbacks = NULL,
  ...,
  verbose = FALSE
)
```

## Arguments

| | |
|---|---|
| object | 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 predict())? |
| excludeNoise | logical; Should noise points in the data stream be excluded from the calculation? |
| callbacks | A named list of functions to calculate custom evaluation measures. |
| ... | Unused arguments are ignored. |
| horizon | Evaluation is done using horizon many previous points (see detail section). |
| verbose | logical; Report progress? |

## Details

For evaluation, each data point is 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.

We provide two evaluation methods:

- evaluate_static() evaluates the current static clustering using new data without updating the model.

- `evaluate_stream()` evaluates the clustering process using *prequential error estimation* (see Gama, Sebastiao and Rodrigues; 2013). The current model is first applied to the data points in the horizon to calculate the evaluation measures. Then, the cluster model is updated with the points.

**Evaluation Measures**

Many evaluation measures are available using code from other packages including `cluster::silhouette()`, `clue:: cl_agreement()`, and `fpc::cluster.stats()`.

The following information items are available:

- `"numPoints"` number of points used for evaluation.
- `"numMicroClusters"` number of micro-clusters
- `"numMacroClusters"` number of macro-clusters
- `"numClasses"` number of classes

The following noise-related/outlier items are available:

- `"noisePredicted"` Number data points predicted as noise
- `"noiseActual"` Number of data points which are actually noise
- `"noisePrecision"` Precision of the predicting noise (i.e., number of correctly predicted noise points over the total number of points predicted as noise)
- `"outlierJaccard"` - A variant of the Jaccard index used to assess outlier detection accuracy (see Krleza et al (2020)). Outlier Jaccard index is calculated as `TP / (TP + FP + UNDETECTED)`.

The following internal evaluation measures are available:

- `"SSQ"` within cluster sum of squares. Assigns each point to its nearest center from the clustering and calculates the sum of squares. Noise points in the data stream are always ignored.
- `"silhouette"` average silhouette width. Actual noise points which stay unassigned by the clustering algorithm are ignored; regular points that are unassigned by the clustering algorithm form their own noise cluster) (**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**)
- `"pearsongamma"` 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)).
- "classPurity" (of real clusters; see Wan et al (2009)).
- "fpr" false positive rate.
- "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**) %
- "corrected.rand" corrected Rand index (**fpc**)
- "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.

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.

**Custom Evaluation Measures**

The parameter `callbacks` can be supplied with a named list with functions with the signature `function(actual, predict, points, centers, dsc)` as elements. See the Examples sections for details.

## 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.

## Author(s)

Michael Hahsler, Matthew Bolanos, John Forrest, and Dalibor Krleža

## References

Joao Gama, Raquel Sebastiao, Pedro Pereira Rodrigues (2013). On evaluating stream learning algorithms. *Machine Learning,* March 2013, Volume 90, Issue 3, pp 317-346.

F. Cao, M. Ester, W. Qian, A. Zhou (2006). Density-Based Clustering over an Evolving Data Stream with Noise. *Proceeding of the 2006 SIAM Conference on Data Mining,* 326-337.

E. Dimitriadou, A. Weingessel and K. Hornik (2002). A combination scheme for fuzzy clustering. *International Journal of Pattern Recognition and Artificial Intelligence,* 16, 901-912.

E. B. Fowlkes and C. L. Mallows (1983). A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association,* 78, 553-569.

L. Hubert and P. Arabie (1985). Comparing partitions. *Journal of Classification,* 2, 193-218.

W. M. Rand (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association,* 66, 846-850.

L. Katz and J. H. Powell (1953). A proposed index of the conformity of one sociometric measurement to another. *Psychometrika,* 18, 249-256.

A. Strehl and J. Ghosh (2002). Cluster ensembles - A knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research,* 3, 583-617.

R. Tibshirani and G. Walter (2005). Cluster validation by Prediction Strength. *Journal of Computational and Graphical Statistics,* 14/3, 511-528.

L Wan, W.K. Ng, X.H. Dang, P.S. Yu and K. Zhang (2009). Density-Based Clustering of Data Streams at Multiple Resolutions, *ACM Transactions on Knowledge Discovery from Data,* 3(3).

D. Krleža, B. Vrdoljak, and M. Brčić (2020). Statistical Hierarchical Clustering Algorithm for Outlier Detection in Evolving Data Streams, *Springer Machine Learning*.

## See Also

cluster::silhouette(), clue:: cl_agreement(), and fpc::cluster.stats().

Other DSC: DSC_Macro(), DSC_Micro(), DSC_R(), DSC_Static(), DSC_TwoStage(), DSC(), animate_cluster(), get_assignment(), plot.DSC(), predict(), prune_clusters(), read_saveDSC, recluster()

Other evaluation: animate_cluster(), evaluate

## Examples

```
# Example 1: Static Evaluation
set.seed(0)
stream <- DSD_Gaussians(k = 3, d = 2)

dstream <- DSC_DStream(gridsize = 0.05, Cm = 1.5)
update(dstream, stream, 500)
plot(dstream, stream)

# Evaluate the micro-clusters in the clustering
# Note: we use here only n = 100 points for evaluation to speed up execution
evaluate_static(dstream, stream, n = 100)

evaluate_static(dstream, stream,
  measure = c("numMicro", "numMacro", "purity", "crand", "SSQ"),
  n = 100)

# DStream also provides macro clusters. Evaluate macro clusters with type = "macro"
# Note that SSQ and cRand increase.
plot(dstream, stream, type = "macro")
evaluate_static(dstream, stream, type = "macro",
  measure = c("numMicro", "numMacro", "purity", "crand", "SSQ"),
  n = 100)

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

# Example 2: Evaluate with Noise/Outliers
stream <- DSD_Gaussians(k = 3, d = 2, noise = .05)
dstream <- DSC_DStream(gridsize = 0.05, Cm = 1.5)
update(dstream, stream, 500)

# For cRand, noise is its own group, for SSQ, actual noise is always
# excluded.
plot(dstream, stream, 500)
evaluate_static(dstream, stream, n = 100,
  measure = c("numPoints", "noisePredicted", "noiseActual",
    "noisePrecision", "outlierJaccard", "cRand", "SSQ"))
```

```
# Note that if noise is excluded, the number of used points is reduced.
evaluate_static(dstream, stream, n = 100,
  measure = c("numPoints", "noisePredicted", "noiseActual",
    "noisePrecision", "outlierJaccard", "cRand", "SSQ"), excludeNoise = TRUE)


# Example 3: Evaluate an evolving data stream
stream <- DSD_Benchmark(1)
dstream <- DSC_DStream(gridsize = 0.05, lambda = 0.1)

evaluate_stream(dstream, stream, type = "macro", assign = "micro",
  measure = c("numMicro", "numMacro", "purity", "cRand"),
  n = 600, horizon = 100)

if (interactive()){
# animate the clustering process
reset_stream(stream)
dstream <- DSC_DStream(gridsize = 0.05, lambda = 0.1)
animate_cluster(dstream, stream, horizon = 100, n = 5000,
  measure = "cRand", type = "macro", assign = "micro",
  plot.args = list(type = "both", xlim = c(0,1), ylim = c(0,1)))
}

# Example 4: Add a custom measure as a callback
callbacks <- list(
   noisePercentage = function(actual, predict, points, centers, dsc) {
     sum(actual == 0L) / length(actual)
   },
   noiseFN = function(actual, predict, points, centers, dsc) {
     sum(actual == 0L & predict != 0L)
   },
   noiseFP = function(actual, predict, points, centers, dsc) {
     sum(actual != 0L & predict == 0L)
   }
 )

stream <- DSD_Gaussians(k = 3, d = 2, noise = .2)
dstream <- DSC_DStream(gridsize = 0.05, Cm = 1.5)
update(dstream, stream, 500)

evaluate_static(dstream, stream,
  measure = c("numPoints", "noiseActual", "noisePredicted",
    "noisePercentage", "noiseFN", "noiseFP"),
  callbacks = callbacks, n = 100)

evaluate_static(dstream, stream, callbacks = callbacks)
```

---

`get_assignment` *Assignment Data Points to Clusters deprecated*

---

## Description

**Deprecation Notice:** use `predict()` for a more general interface to apply a data stream model to new data. `get_assignment()` is deprecated.

## Usage

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

## S3 method for class 'DSC'
get_assignment(
  dsc,
  points,
  type = c("auto", "micro", "macro"),
  method = c("auto", "nn", "model"),
  ...
)
```

## 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. |
| `method` | assignment method<br>• `"model"` uses the assignment method of the underlying algorithm (unassigned points return `NA`). Not all algorithms implement this option.<br>• `"nn"` performs nearest neighbor assignment using Euclidean distance.<br>• `"auto"` uses the model assignment method. If this method is not implemented/available then method `"nn"` is used instead. |
| `...` | Additional arguments are passed on. |

## Details

Get the assignment of data points to clusters in a `DSC` using the model's assignment rules or nearest neighbor assignment. The clustering is not modified.

Each data point is assigned either using the original model's assignment rule or Euclidean nearest neighbor assignment. If the user specifies the model's assignment strategy, but is not available, then nearest neighbor assignment is used and a warning is produced.

## Value

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

### Author(s)

Michael Hahsler

### See Also

Other DSC: DSC_Macro(), DSC_Micro(), DSC_R(), DSC_Static(), DSC_TwoStage(), DSC(), animate_cluster(), evaluate.DSC, plot.DSC(), predict(), prune_clusters(), read_saveDSC, recluster()

### Examples

```
stream <- DSD_Gaussians(k = 3, d = 2, noise = .05)

dbstream <- DSC_DBSTREAM(r = .1)
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 <- predict(dbstream, points)
head(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[, ".class"]),], col = "blue")
points(points[is.na(a[, ".class"]),], col = "red", pch = 20)

# use nearest neighbor assignment instead
a <- predict(dbstream, points, method = "nn")
head(a)
```

---

## get_points *Get Points from a Data Stream Generator*

---

### Description

Gets points from a DSD object.

### Usage

```
get_points(x, ...)

## S3 method for class 'DSD'
get_points(x, n = 1L, info = TRUE, ...)

remove_info(points)
```

## Arguments

| | |
|---|---|
| x | A DSD object. |
| ... | Additional parameters to pass to the get_points() implementations. |
| n | integer; request up to n points from the stream. n = -1 returns all remaining points from limited streams. |
| info | return additional columns with information about the data point (e.g., a known cluster assignment). |
| points | a data.frame with points. |

## Details

Each DSD object has a unique way for creating/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().

### Additional Point Information

Additional point information (e.g., known cluster/class assignment, noise status) can be requested with info = TRUE. This information is returned as additional columns. The column names start with . and are ignored by DST implementations. remove_info() is a convenience function to remove the information columns. Examples are

- .id for point IDs
- .class for known cluster/class labels used for plotting and evaluation
- .time a time stamp for the point (can be in seconds or an index for ordering)

### Resetting a Stream

Many streams can be reset using reset_stream().

## Value

Returns a data.frame with (up to) n rows and as many columns as x produces.

## Author(s)

Michael Hahsler

## See Also

Other DSD: DSD_BarsAndGaussians(), DSD_Benchmark(), DSD_Cubes(), DSD_Gaussians(), DSD_MG(), DSD_Memory(), DSD_Mixture(), DSD_NULL(), DSD_ReadDB(), DSD_ReadStream(), DSD_Target(), DSD_UniformNoise(), DSD_mlbenchData(), DSD_mlbenchGenerator(), DSD(), DSF(), animate_data(), close_stream(), plot.DSD(), reset_stream()

## Examples

```
stream <- DSD_Gaussians()
points <- get_points(stream, n = 5)
points

remove_info(points)
```

---

MGC *Moving Generator Cluster*

---

## Description

Creates an evolving cluster for use as a component of a DSD_MG data stream.

## Usage

```
MGC(...)

MGC_Function(density, center, parameter, shape = Shape_Gaussian)

MGC_Linear(dimension = 2, keyframelist = NULL, shape = Shape_Gaussian)

keyframe(time, density, center, parameter, reset = FALSE)

add_keyframe(x, time, density, center, parameter, reset = FALSE)

get_keyframes(x)

remove_keyframe(x, time)

MGC_Noise(density, range)

MGC_Random(density, center, parameter, randomness = 1, shape = Shape_Gaussian)

Shape_Gaussian(center, parameter)

Shape_Block(center, parameter)

MGC_Static(density = 1, center, parameter, shape = Shape_Gaussian)
```

## Arguments

| | |
|---|---|
| ... | Further arguments. |
| density | The density of the cluster. For 'MGC_Function, this attribute is a function and defines the density of a cluster (i.e., how many points it creates) at each given timestamp. |

| | |
|---|---|
| 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. |
| parameter | Parameters for the shape. For the default shape Shape_Gaussian the parameter is the standard deviation, one per dimension. If a single value is specified then it is recycled for all dimensions. |
| shape | A function creating the shape of the cluster. It gets passed on the parameters argument from above. Available functions are Shape_Gaussian (the parameters are a vector containing standard deviations) and Shape_Block (parameters are the dimensions of the uniform block). |
| dimension | Dimensionality of the data stream. |
| keyframelist | a list of keyframes to initialize the MGC_Linear object with. |
| time | The time stamp the keyframe should be located or which keyframe should be removed. |
| reset | Should the cluster reset to the first keyframe (time 0) after this keyframe is finished? |
| x | An object of class MGC_Linear. |
| range | The area in which the noise should appear. |
| randomness | The maximum amount the cluster will move during one time step. |

## Details

An MGC describes a single cluster for use as a component in a DSD_MG. Different MGCs allow the user to express different cluster behaviors within a single data stream. Static, (i.e., not moving) clusters are defined as:

- MGC_Static cluster positions are fixed
- MGC_Noise allows to add random noise.

Moving (evolving) clusters are defined as:

- MGC_Linear creates an evolving cluster for a who's behavior is determined by keyframes. Several keyframe functions are provided to create, add and remove keyframes. See Examples section for details.
- MGC_Function allows to specify density, center, and parameter as a function of time.
- MGC_Random allows for a creation of a cluster that follows a random walk.

Cluster shapes can be specified using the functions:

- Shape_Gaussian
- Shape_Block

New shapes can be defined as a function with parameters center and parameter that return a single new point. Here is an example:

```
Shape_Gaussian <- function(center, parameter)
   rnorm(length(center), mean = center, sd = parameter)
```

## Author(s)

Matthew Bolanos

## See Also

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

## Examples

```
MGC()

### Two static clusters (Gaussian with sd of .1 and a Block with width .4)
###   with added noise
stream <- DSD_MG(dim = 2,
  MGC_Static(den = .45, center = c(1, 0), par = .1, shape = Shape_Gaussian),
  MGC_Static(den = .45, center = c(2, 0), par = .4, shape = Shape_Block),
  MGC_Noise( den = .1, range = rbind(c(0, 3), c(-1,1)))
)
stream

plot(stream)

### Example of several MGC_Randoms which define clusters that randomly move.
stream <- DSD_MG(dim = 2,
  MGC_Random(den = 100, center=c(1, 0), par = .1, rand = .2),
  MGC_Random(den = 100, center=c(2, 0), par = .4, shape = Shape_Block, rand = .2)
)

## Not run:
  animate_data(stream, 2500, xlim = c(0,3), ylim = c(-1,1), horizon = 100)

## End(Not run)


### Example of several MGC_Functions

### a 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 = Shape_Block
  )

### a cluster moving in a circle (default shape is Gaussian)
c2 <- MGC_Function(
  density = function(t){ 25 },
  parameter = function(t){ 5 },
  center= function(t) c(sin(t / 10) * 50 + 50, cos(t / 10) * 50 + 50)
)

stream <- DSD_MG(dim = 2, c1, c2)
```

```
## adding noise after the stream was created
add_cluster(stream, MGC_Noise(den = 10, range = rbind(c(-20, 120), c(-20, 120))))

stream

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

## End(Not run)

### Example of several MGC_Linear: 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 = 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)
```

---

plot.DSC *Plot Results of a Data Stream Clustering*

---

## Description

Method to plot the result of data stream data clustering. To plot DSD see `plot.DSD()`.

## Usage

```
## S3 method for class 'DSC'
plot(
  x,
  dsd = NULL,
  n = 500,
  col_points = NULL,
  col_clusters = c("red", "blue", "green"),
  weights = TRUE,
  scale = c(1, 5),
  cex = 1,
  pch = NULL,
  method = c("pairs", "scatter", "pca"),
  dim = NULL,
  type = c("auto", "micro", "macro", "both", "none"),
  assignment = FALSE,
  transform = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | the DSC object to be plotted. |
| `dsd` | a DSD object to plot the data in the background. |
| `n` | number of plots taken from `dsd` to plot. |
| `col_points, col_clusters` | colors used for plotting. |
| `weights` | if `TRUE` then the cluster weight is used for symbol size. Alternatively, a vector with the size of the symbols for micro- and macro-clusters can be supplied. |
| `scale` | range for the symbol sizes used. |
| `cex` | size factor for symbols. |
| `pch` | symbol type for points. |
| `method` | method used for plotting: `"pairs"` (pairs plot), `"scatter"` (scatter plot), `"pca"` (plot first 2 principal components). |
| `dim` | an integer vector with the dimensions to plot. If `NULL` then for methods `pairs` and `"pca"` all dimensions are used and for `"scatter"` the first two dimensions are plotted. |
| `type` | Plot micro clusters (`type = "micro"`), macro clusters (`type = "macro"`), both micro and macro clusters (`type = "both"`). |
| `assignment` | logical; show assignment area of micro-clusters. |
| `transform` | a function that maps data stream points onto a 2-D plane for plotting. |

... further arguments are passed on to graphics::plot.default() or graphics::pairs(). **graphics**.

## Author(s)

Michael Hahsler

## See Also

Other DSC: DSC_Macro(), DSC_Micro(), DSC_R(), DSC_Static(), DSC_TwoStage(), DSC(), animate_cluster(), evaluate.DSC, get_assignment(), predict(), prune_clusters(), read_saveDSC, recluster()

Other plot: animate_cluster(), animate_data(), plot.DSD()

## Examples

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

## Example 1: Plot data
plot(stream)

## Example 2: Plot a clustering
dstream <- DSC_DStream(gridsize = 0.1)
update(dstream, stream, 500)
dstream
plot(dstream, stream)

## plot micro or macro-clusters only
plot(dstream, stream, type = "micro")
plot(dstream, stream, type = "macro")

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

## D-Stream has a special implementation to show assignment areas
plot(dstream, stream, assignment = TRUE)

## Example 4: Use a custom transformation for plotting.
##     We fit PCA using 100 points and create a transformation
##     function to project the stream to the first two PCs.
pr <- princomp(get_points(stream, n = 100, info = FALSE))
trans <- function(x) predict(pr, x)[, 1:2 , drop = FALSE]

trans(get_points(stream, n = 3))

plot(dstream, stream, transform = trans)
```

plot.DSD *Plot Data Stream Data*

## Description

Method to plot data stream data. To plot DSC see `plot.DSC()`.

## Usage

```
## S3 method for class 'DSD'
plot(
  x,
  n = 500,
  col = NULL,
  pch = NULL,
  ...,
  method = c("pairs", "scatter", "pca", "ts"),
  dim = NULL,
  alpha = 0.6,
  transform = NULL
)
```

## Arguments

| | |
|---|---|
| `x` | the DSD object to be plotted. |
| `n` | number of plots taken from `x` to plot. |
| `col` | colors used for points. |
| `pch` | symbol type. |
| `...` | further arguments are passed on to `graphics::plot.default()` or `graphics::pairs()`. |
| `method` | method used for plotting: `"pairs"` (pairs plot), `"scatter"` (scatter plot), `"pca"` (plot first 2 principal components), or `"ts"` (time series). |
| `dim` | an integer vector with the dimensions to plot. If `NULL` then for methods `pairs` and `"pca"` all dimensions are used and for `"scatter"` the first two dimensions are plotted. |
| `alpha` | alpha shading used to plot the points. |
| `transform` | a function that maps data stream points onto a 2-D plane for plotting. |

## Author(s)

Michael Hahsler

## See Also

Other DSD: DSD_BarsAndGaussians(), DSD_Benchmark(), DSD_Cubes(), DSD_Gaussians(), DSD_MG(), DSD_Memory(), DSD_Mixture(), DSD_NULL(), DSD_ReadDB(), DSD_ReadStream(), DSD_Target(), DSD_UniformNoise(), DSD_mlbenchData(), DSD_mlbenchGenerator(), DSD(), DSF(), animate_data(), close_stream(), get_points(), reset_stream()

Other plot: animate_cluster(), animate_data(), plot.DSC()

## Examples

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

## plot data
plot(stream, n = 500)
plot(stream, method = "pca", 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 = "pca")
plot(dstream, stream, dim = c(2, 3))

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

## plot a time series using the AirPassenger data with the total monthly
## passengers from 1949 to 1960) a as a stream
AirPassengers
stream <- DSD_Memory(data.frame(
  .time = time(AirPassengers),
  passengers = AirPassengers))

get_points(stream, n = 10)
plot(stream, n = 100, method = "ts")
```

---

# predict *Make a Prediction for a Data Stream Mining Task*

---

## Description

predict() for data stream mining tasks DST.

## Usage

```
## S3 method for class 'DST'
predict(object, newdata, ...)

## S3 method for class 'DSC'
predict(
  object,
  newdata,
  type = c("auto", "micro", "macro"),
  method = "auto",
  ...
)
```

## Arguments

| | |
|---|---|
| object | The DST object. |
| newdata | The points to make predictions for as a data.frame. |
| ... | Additional arguments are passed on. |
| type | Use micro- or macro-clusters in DSC for assignment. |
| method | assignment method<br>• "model" uses the assignment method of the underlying algorithm (unassigned points return NA). Not all algorithms implement this option.<br>• "nn" performs nearest neighbor assignment using Euclidean distance.<br>• "auto" uses the model assignment method. If this method is not implemented/available then method "nn" is used instead. |

## Value

A data.frame with columns containing the predictions. The columns depend on the type of the data stream mining task.

## Author(s)

Michael Hahsler

## See Also

Other DST: DSAggregate(), DSClassifier(), DSC(), DSF_Scale(), DSOutlier(), DST_Runner(), DST_WriteStream(), DST(), evaluate, update()

Other DSC: DSC_Macro(), DSC_Micro(), DSC_R(), DSC_Static(), DSC_TwoStage(), DSC(), animate_cluster(), evaluate.DSC, get_assignment(), plot.DSC(), prune_clusters(), read_saveDSC, recluster()

## Examples

```
set.seed(1500)
stream <- DSD_Gaussians(k = 3, d = 2, noise = .1)
```

```
dbstream <- DSC_DBSTREAM(r = .1)
update(dbstream, stream, n = 100)
plot(dbstream, stream, type = "both")

# 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 = 10)
points

pr <- predict(dbstream, points, type = "macro")
pr

# Note that the clusters are labeled in arbitrary order. Check the
# agreement.
agreement(pr[,".class"], points[,".class"])
```

---

`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`.

## Author(s)

Michael Hahsler

## See Also

`DSC_Static`

Other DSC: `DSC_Macro()`, `DSC_Micro()`, `DSC_R()`, `DSC_Static()`, `DSC_TwoStage()`, `DSC()`, `animate_cluster()`, `evaluate.DSC`, `get_assignment()`, `plot.DSC()`, `predict()`, `read_saveDSC`, `recluster()`

## Examples

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

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

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

---

# read_saveDSC *Save and Read DSC Objects*

---

## Description

Save and Read DSC objects safely (serializes the underlying data structure). This also works for **streamMOA** DSC objects.

## Usage

```
saveDSC(object, file, ...)

readDSC(file)
```

## Arguments

| | |
|---|---|
| object | a DSC object. |
| file | filename. |
| ... | further arguments. |

## Author(s)

Michael Hahsler

## See Also

saveRDS and readRDS.

Other DSC: DSC_Macro(), DSC_Micro(), DSC_R(), DSC_Static(), DSC_TwoStage(), DSC(), animate_cluster(), evaluate.DSC, get_assignment(), plot.DSC(), predict(), prune_clusters(), recluster()

## Examples

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

# create clusterer with r = 0.05
dbstream1 <- DSC_DBSTREAM(r = .05)
update(dbstream1, stream, 1000)
dbstream1

saveDSC(dbstream1, file="dbstream.Rds")

dbstream2 <- readDSC("dbstream.Rds")
dbstream2

## cleanup
unlink("dbstream.Rds")
```

---

# recluster *Re-clustering micro-clusters*

---

## Description

Use an ***offline** macro clustering algorithm to recluster micro-clusters into a final clusters.

## Usage

```
recluster(macro, micro, type = "auto", ...)

## S3 method for class 'DSC_Macro'
recluster(macro, micro, type = "auto", ...)
```

## Arguments

| | |
|---|---|
| `macro` | an empty DSC_Macro. |
| `micro` | an updated DSC_Micro with micro-clusters. |
| `type` | controls which clustering is used from `micro`. Typically `auto`. |
| `...` | additional arguments passed on. |

## Details

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

See DSC_TwoStage for a convenient combination of micro and macro clustering.

## Value

The object `macro` is altered in place and contains the clustering.

### Author(s)

Michael Hahsler

### See Also

Other DSC: DSC_Macro(), DSC_Micro(), DSC_R(), DSC_Static(), DSC_TwoStage(), DSC(), animate_cluster(), evaluate.DSC, get_assignment(), plot.DSC(), predict(), prune_clusters(), read_saveDSC

### Examples

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

### sample can be seen as a simple online clusterer where the sample points
### are the micro clusters.
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, type = "both", main = "Macro-clusters (Sampling + k-means)")
```

---

## reset_stream *Reset a Data Stream to its Beginning*

---

### Description

Resets the position in a DSD object to the beginning or, if available, 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.

`reset_stream()` is implemented for:

- DSD
- DSD_MG
- DSD_Memory
- DSD_ReadStream
- DSF
- DSF_Convolve

## Author(s)

Michael Hahsler

## See Also

Other DSD: `DSD_BarsAndGaussians()`, `DSD_Benchmark()`, `DSD_Cubes()`, `DSD_Gaussians()`, `DSD_MG()`, `DSD_Memory()`, `DSD_Mixture()`, `DSD_NULL()`, `DSD_ReadDB()`, `DSD_ReadStream()`, `DSD_Target()`, `DSD_UniformNoise()`, `DSD_mlbenchData()`, `DSD_mlbenchGenerator()`, `DSD()`, `DSF()`, `animate_data()`, `close_stream()`, `get_points()`, `plot.DSD()`

## Examples

```
# initializing the objects
stream <- DSD_Gaussians()
replayer <- DSD_Memory(stream, 100)
replayer

p <- get_points(replayer, 50)
replayer

# reset replayer to the beginning of the stream
reset_stream(replayer)
replayer

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

update *Update a Data Stream Mining Task Model with Points from a Stream*

## Description

update() for data stream mining tasks DST.

## Usage

```
## S3 method for class 'DST'
update(object, dsd, n = 1L, ...)
```

## Arguments

| | |
|---|---|
| object | The DST object. |
| dsd | A DSD object with the data stream. |
| n | number of points from dsd to use for the update. Some DSD dsd accept n = -1 to update with all remaining points in the stream. |
| ... | Additional arguments are passed on. |

## Value

NULL or a data.frame n rows containing update information for each data point.

## Author(s)

Michael Hahsler

## See Also

Other DST: DSAggregate(), DSClassifier(), DSC(), DSF_Scale(), DSOutlier(), DST_Runner(), DST_WriteStream(), DST(), evaluate, predict()

## Examples

```
set.seed(1500)
stream <- DSD_Gaussians(k = 3, d = 2, noise = .1)

dbstream <- DSC_DBSTREAM(r = .1)
assignment <- update(dbstream, stream, n = 100, assignment = TRUE)
plot(dbstream, stream, type = "both")

# DBSTREAM returns cluster assignments (see DSC_DBSTREAM).
head(assignment)
```

`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

```
write_stream(
  dsd,
  file,
  n = 100,
  block = 100000L,
  info = FALSE,
  append = FALSE,
  sep = ",",
  header = FALSE,
  row.names = FALSE,
  close = TRUE,
  ...
)
```

## 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. |
| `info` | Save the class/cluster labels and other information columns with the data. |
| `append` | Append the data to an existing file. If `FALSE`, then the file will be overwritten. |
| `sep` | The character that will separate attributes in a data point. |
| `header` | A flag that determines if column names will be output (equivalent to `col.names` in `write.table()`). |
| `row.names` | A flag that determines if row names will be output. |
| `close` | close stream after writing. |
| `...` | Additional parameters that are passed to `write.table()`. |

## Value

There is no value returned from this operation.

## Author(s)

Michael Hahsler

## See Also

write.table

## Examples

```
# creating data and writing it to disk
stream <- DSD_Gaussians(k = 3, d = 5)
write_stream(stream, file="data.txt", n = 10, header = TRUE, info = TRUE)

readLines("data.txt")

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

# Index