Package ‘dtwclust’

September 12, 2015

Type    Package
Title   Time Series Clustering with Dynamic Time Warping
Version 1.1.0
Date    2015-09-12
Depends flexclust, proxy, dtw, ggrepplot
Imports stats, methods, caTools, reshape2, modeltools
Suggests TSdist
Author  Alexis Sarda-Espinosa
Maintainer Alexis Sarda <alexis.sarda@gmail.com>
Description Time series clustering using different techniques related to the Dynamic Time Warping distance and its corresponding lower bounds. Additionally, an implementation of k-Shape clustering is available.
URL     https://github.com/asardaes/dtwclust
License GPL-3
LazyData TRUE
NeedsCompilation no
Repository CRAN
Date/Publication 2015-09-12 17:20:03

R topics documented:

dtwclust-package .................................................. 2
DBA ................................................................. 3
dtwclust ............................................................ 5
dtwclust-class ..................................................... 9
dtw_lb ............................................................... 10
lb_improved ........................................................ 11
lb_keogh ........................................................... 13
NCCc ............................................................... 14
dtwclust-package

Time series clustering under Dynamic Time Warping (DTW) distance.

Description

Perform time series clustering using different techniques related to the DTW distance and its corresponding lower bounds (LB). Additionally, an implementation of k-Shape clustering is available.

Details

This package tries to consolidate the different procedures available to perform clustering of time series under DTW. Most of the optimizations require time series to have equal lengths. DTW itself doesn’t, but it’s much slower to compute. The shape-based distance (SBD) can also be used for series of different lengths, and it could be faster. If series have different lengths, it is debatable whether it makes sense to cluster them directly. If possible, reinterpolating them could be one way of speeding up calculations (see Ratanamahatana and Keogh, 2004; also reinterpolate).

Please see the documentation for dtwclust, which serves as the main entry point.

Other packages that are particularly leveraged here are the flexclust package for partitional clustering, the proxy package for distance matrix calculations, and the dtw package for the core DTW calculations.

Four distances are registered via pr_DB: "LB_Keogh", "LB_Improved", "SBD" and "DTW2". See lb_keogh, lb_improved and SBD for more details on the first 3. The last one is done with dtw using L2 norm, but it differs from the result you would obtain if you specify L2 as dist.method: with DTW2, pointwise distances (the local cost matrix) are calculated with L1 norm, each element of the matrix is squared and the result is fed into dtw, which finds the optimum warping path. The square root of the resulting distance is then computed.

Please note that the dist function in the proxy package accepts one or two arguments for data objects. Users should usually use the two-input list version, even if there is just one dataset (i.e. proxy::dist(x=data, y=data, ...)), because otherwise it sometimes fails to detect a whole time series as a single object and, instead, calculates distances between each observation of each time series.

Author(s)

Alexis Sarda-Espinosa
References


Ratanamahatana A and Keogh E (2004). “Everything you know about dynamic time warping is wrong.” In 3rd Workshop on Mining Temporal and Sequential Data, in conjunction with 10th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD-2004), Seattle, WA.


See Also
dtwclust, kcca, dist, dtw

DTW Barycenter Averaging

Description

A global averaging method for time series under DTW (Petitjean, Ketterlin and Gancarski, 2011).
Usage

DBA(X, center = NULL, max.iter = 20, norm = "L1", window.size = NULL, delta = 1e-06, error.check = TRUE, trace = FALSE)

Arguments

X A data matrix where each row is a time series. Optionally, a list where each element is a time series.

center Optionally, a time series to use as reference. It must be a numeric vector. Defaults to a random series of X if NULL.

max.iter Maximum number of iterations allowed.

norm Norm for the local cost matrix of DTW. Either "L1" for Manhattan distance or "L2" for Euclidean distance.

window.size Window constraint for the DTW calculations. NULL means no constraint.

delta At iteration i, if all(abs(center[i] - center[i-1]) < delta), convergence is assumed.

error.check Should inconsistencies in the data be checked?

trace If TRUE, the current iteration is printed to screen.

Details

This function tries to find the optimum average series between a group of time series in DTW space. Refer to the cited article for specific details on the algorithm.

If a given series reference is provided in center, the algorithm should always converge to the same result provided the rows of X keep the same values, although their order may change.

Value

The average time series.

References


Examples

# Sample data
data(uciCT)

# Obtain an average for the first 5 time series
dtw.avg <- DBA(CharTraj[1:5], CharTraj[[1]], trace = TRUE)
plot(dtw.avg, type="l")

# Change the provided order
dtwclust

\[
\text{dtw.avg2} \leftarrow \text{DBA(CharTraj[5:1], CharTraj[1:1], trace = TRUE)} \\
\text{all(dtw.avg == dtw.avg2)}
\]

dtwclust  Time series clustering under DTW

Description
This function uses the DTW distance and related lower bounds to cluster time series. For now, all series must be univariate.

Usage
\[
dtwclust(data = \text{NULL}, \text{type} = \text{"partitional"}, k = 2, \text{method} = \text{"average"}, \\
\text{distance} = \text{"dtw"}, \text{centroid} = \text{"pam"}, \text{preproc} = \text{NULL}, \\
\text{window.size} = \text{NULL}, \text{norm} = \text{"l1"}, \text{dc} = \text{NULL}, \text{dba.iter} = 50, \\
\text{control} = \text{NULL}, \text{save.data} = \text{FALSE}, \text{seed} = \text{NULL}, \text{trace} = \text{FALSE}, \ldots)
\]

Arguments

- **data**: A list where each element is a time series, or a numerical matrix where each row is a time series. All series must have equal lengths in case of type = "tadpole".
- **type**: What type of clustering method to use, partitional, hierarchical or tadpole.
- **k**: Number of desired clusters in partitional methods.
- **method**: Which linkage method to use in hierarchical methods. See \text{hclust}.
- **distance**: One of the supported distance definitions (see Distance section). Ignored for type = "tadpole".
- **centroid**: Either a supported string or an appropriate function to calculate centroids when using partitional methods (see Centroid section).
- **preproc**: Function to preprocess data. Defaults to zscore only if centroid = "shape", but will be replaced by a custom function if provided. See Preprocessing section.
- **window.size**: Window constraint for DTW and LB calculations. See Sakoe-Chiba section.
- **norm**: Pointwise distance for DTW, DBA and the LB. Either L1 for Manhattan distance or L2 for Euclidean. Ignored for distance = "DTW" (which always uses L1) and distance = "DTW2" (which always uses L2).
- **dc**: Cutoff distance for TADPole algorithm.
- **dba.iter**: Maximum number of iterations for DBA centroids.
- **control**: Parameters for partitional clustering algorithms. See \text{flexclustControl}.
- **save.data**: Return a copy of the data in the returned object? Ignored for hierarchical clustering.
- **seed**: Random seed for reproducibility of partitional algorithms.
- **trace**: Boolean flag. If true, more output regarding the progress is printed to screen.
- **...**: Additional arguments to pass to \text{dist} or a custom function.
Details

Partitional algorithms are implemented via \texttt{kcca}. Hierarchical algorithms use the \texttt{hclust} function. The \texttt{tadpole} algorithm uses the \texttt{TADPole} function.

The data may be a matrix or a list, but the matrix will be coerced to a list. A matrix input requires that all time series have equal lengths. If the lengths vary slightly between time series, reinterpolating them to a common length is most likely an acceptable approach (Ratanamahatana and Keogh, 2004). If this is not the case, then clustering them directly is probably ill-advised. See the examples.

Value

An object with formal class \texttt{dtwclust-class} if \texttt{type} = "partitional" | "tadpole". Otherwise an object with class \texttt{hclust} as returned by \texttt{hclust}.

Distance

If a custom distance function is provided, it will receive the data as the first argument. For partitional algorithms, the second argument will be the cluster centers (i.e. other time series). If \texttt{data} is a matrix, the cluster centers will also be given in the form of a matrix where each row is a center series; if \texttt{data} is a list of series, so will the centers.

If hierarchical algorithms are used, the function will also receive the elements of \ldots.

For partitional algorithms, the function \textit{could} make use of the \texttt{window.size} and \texttt{norm} parameters, which \textit{should} be detected thanks to R’s lexical scoping, however this cannot be guaranteed.

The function should return a distance matrix, ideally of class \texttt{crossdist}. In case of partitional algorithms, the time series in the data should be along the rows, and the cluster centers along the columns of the distance matrix.

The other option is to provide a string. The string can represent a compatible registered distance of \texttt{dist}. In the case of hierarchical algorithms, extra parameters can be provided in \ldots.

Additionally, with either type of algorithm, it can be one of the following custom implementations:

- "dtw": DTW with L1 norm and optionally a Sakoe-Chiba constraint.
- "dtw2": DTW with L2 norm and optionally a Sakoe-Chiba constraint.
- "dtw_lb": DTW with L1 or L2 norm and optionally a Sakoe-Chiba constraint. Some computations are avoided by first estimating the distance matrix with Lemire’s lower bound and then iteratively refining with DTW. See \texttt{dtw_lb}.
- "lk": Keogh’s lower bound with either L1 or L2 norm for the Sakoe-Chiba constraint.
- "li": Lemire’s lower bound with either L1 or L2 norm for the Sakoe-Chiba constraint.
- "sbd": Shape-based distance. Each series is z-normalized in this case. As a result, the cluster centers (for partitional methods) are also z-normalized. See \texttt{SBD} for more details.

Note that only \texttt{dtw}, \texttt{dtw2} and \texttt{sbd} support series of different lengths.
Centroid

In the case of partitional algorithms, a suitable function should calculate the cluster centers. In this case, the centers are themselves time series.

If a custom function is provided, it will receive different inputs depending on the format of data:

- For matrix input, it will receive a matrix as single input. Each row will be a series that belongs to a given cluster. The function should return a numeric vector representing the centroid series.
- For a list input, the function will receive three inputs in the following order: the whole data list; a numeric vector with length equal to the number of series in data, indicating which cluster a series belongs to; the current number of total clusters.

The other option is to provide a character string. The following options are available:

- "mean": The average along each dimension. In other words, the average of all $x_j^i$ among the $j$ series that belong to the same cluster for all time points $t_i$.
- "median": The median along each dimension. Similar to mean.
- "shape": Shape averaging. See shape_extraction for more details.
- "dba": DTW Barycenter Averaging. See DBA for more details.
- "pam": Partition around medoids. This basically means that the cluster centers are always one of the time series in the data. In this case, the distance matrix is pre-computed once using all time series in the data and then re-used at each iteration.

Note that only dba and pam support series of different lengths

Sakoe-Chiba Constraint

A global constraint to speed up the DTW calculation is the Sakoe-Chiba band (Sakoe and Chiba, 1978). To use it, a window width must be defined via window.size.

The windowing constraint uses a centered window. The calculations expect a value in window.size that represents the distance between the point considered and one of the edges of the window. Therefore, if, for example, window.size = 10, the warping for an observation $x_i$ considers the points between $x_{i-10}$ and $x_{i+10}$, resulting in $10*2 + 1 = 21$ observations falling within the window.

Preprocessing

It is strongly advised to use z-normalization in case of centroid = "shape", because the resulting series have this normalization (see shape_extraction). The user can, however, specify a custom function that performs any transformation on the data, but the user must make sure that the format stays consistent, i.e. a matrix where each row is a series or a list of time series. For example, the z-normalization could be implemented as `t(apply(data, 1, zscore))` or `lapply(data, zscore)` respectively.

The function will receive the data as first argument and, in case hierarchical methods are used, the contents of ... as the second argument.
Notes

In order to ensure that the parameter values are detected correctly by the included functions when partitional clustering is used, the environment of the dtwclust function is assigned as an attribute of data via attr(data, "env") <- environment(). If the user alters the dataset with a preprocessing function, it should make sure that this attribute is maintained.

Notice that the lower bounds are defined only for time series of equal lengths. DTW and DTW2 don’t require this, but they are much slower to compute.

The lower bounds are not symmetrical, and DTW is only symmetrical if series are of equal lengths.

Specifying distance = "sbd" and centroid = "shape" is equivalent to the k-Shape algorithm (Papparizos and Gravano, 2015). See sbd and shape_extraction for more info.

Author(s)

Alexis Sarda-Espinosa

References


Ratanamahatana A and Keogh E (2004). “Everything you know about dynamic time warping is wrong.” In 3rd Workshop on Mining Temporal and Sequential Data, in conjunction with 10th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD-2004), Seattle, WA.


See Also

Please check the brief description in dtwclust-package.

Additionally: plot-dtwclust, dtwclust-class.

Examples

# Load data
data(uciCT)

# Reinterpolate to same length and coerce as matrix
data <- t(sapply(CharTraj, reinterpolate, newLength = 205))

# Simple partitional clustering with L2 distance and PAM
kc.12 <- dtwclust(data, k = 20, distance = "L2", centroid = "pam",
seed = 3247, trace = TRUE, save.data = TRUE)
cat("Rand index for L2+PAM: ", randIndex(kc.12, CharTrajLabels), "\n\n")

## Not run:
# Saving and modifying the ggplot object with custom time
dtwclust-class

Class definition for dtwclust

Description
Formal S4 class to know how to handle data for plotting.

Details
It contains the following specific slots:
• type: A string indicating one of the supported clustering types of `dtwclust`.
• distance: A string indicating the distance used with `dtwclust`.
• centroid: A string indicating the centroid used with `dtwclust`.
• preproc: A string indicating the preprocessing used with `dtwclust`.
• datalist: The provided data in the form of a list, where each element is a time series.

Additionally, the class inherits from `kccasimple-class`, so most related slots and methods are also supported.

dtw_lb

**Description**

Calculation of a distance matrix with the Dynamic Time Warping (DTW) distance guided by Lemire’s lower bound (LB).

**Usage**

```r
dtw_lb(x, y = NULL, window.size = NULL, norm = "l1", error.check = TRUE)
```

**Arguments**

- `x`: A matrix where rows are time series, or a list of time series.
- `y`: An object similar to `x`.
- `window.size`: Window size to use with the LB and DTW calculation. See details.
- `norm`: Pointwise distance. Either `l1` for Manhattan distance or `l2` for Euclidean.
- `error.check`: Should inconsistencies in the data be checked?

**Details**

This function first calculates an initial estimate of a distance matrix between two sets of time series using Lemire’s improved lower bound. Afterwards, it uses the estimate to calculate the true DTW distances between only the nearest neighbors of each series in `x` found in `y`. If only `x` is provided, the distance matrix is calculated between all its time series. This could be useful in case one is interested in only the nearest neighbor of one or more series among a dataset.

The windowing constraint uses a centered window. The calculations expect a value in `window.size` that represents the distance between the point considered and one of the edges of the window. Therefore, if, for example, `window.size = 10`, the warping for an observation `x_i` considers the points between `x_{i-10}` and `x_{i+10}`, resulting in `10*2 + 1 = 21` observations falling within the window.

**Value**

The distance matrix with class `crossdist`. 
Note

This function uses a lower bound that is only defined for time series of equal lengths.

References


See Also

lb_improved

Examples

# Load data
data(uciCT)

# Reinterpolate to same length
data <- lapply(CharTraj, reinterpolate, newLength = 205)

# Calculate the DTW distance between a certain subset aided with the lower bound
system.time(d <- dtw_lb(data[1:5], data[6:50], window.size = 20))

# Nearest neighbors
NN1 <- apply(d, 1, which.min)

# Calculate the DTW distances between all elements (about seven times slower)
system.time(d2 <- proxy::dist(data[1:5], data[6:50], method = "DTW",
                           window.type = "slantedband", window.size = 20))

# Nearest neighbors
NN2 <- apply(d2, 1, which.min)

# Same results?
all(NN1 == NN2)

lb_improved  

Lemire’s improved DTW lower bound

Description

This function calculates a lower bound (LB) on the Dynamic Time Warp (DTW) distance between two time series. It uses a Sakoe-Chiba constraint.

Usage

lb_improved(x, y, window.size = NULL, norm = "L1", lower.env = NULL, upper.env = NULL)
Arguments

- **x**: A time series.
- **y**: A time series with the same length as x.
- **window.size**: Window size for envelope calculation. See details.
- **norm**: Pointwise distance. Either \( l_1 \) for Manhattan distance or \( l_2 \) for Euclidean.
- **lower.env**: Optionally, a pre-computed lower envelope for y can be provided.
- **upper.env**: Optionally, a pre-computed upper envelope for y can be provided.

Details

The lower bound is defined for time series of equal length only.

The windowing constraint uses a centered window. The calculations expect a value in `window.size` that represents the distance between the point considered and one of the edges of the window. Therefore, if, for example, `window.size = 10`, the warping for an observation \( x_i \) considers the points between \( x_{i-10} \) and \( x_{i+10} \), resulting in \( 10 \times 2 + 1 = 21 \) observations falling within the window.

Value

The improved lower bound for the DTW distance.

Note

If you wish to calculate the lower bound between several time series, it would be better to use the version registered with the ‘proxy’ package, since it includes some small optimizations. See the examples.

References


Examples

```r
# Sample data
data(uciCT)

data(uciCT)

# Lower bound distance between two series
d.lbi <- lb_improved(CharTraj[[1]], CharTraj[[2]], window.size = 20)

# Corresponding true DTW distance
d.dtw <- dtw(CharTraj[[1]], CharTraj[[2]],
             window.type = "slantedband", window.size = 20)$distance

d.lbi <= d.dtw

# Calculating the LB between several time series using the 'proxy' package
```
# Notice how both arguments must be lists)
D.lbi <- proxy::dist(CharTraj[1], CharTraj[2:5], method = "LB_Improved",
  window.size = 20, norm = "L2")

# Corresponding true DTW distance
# (see dtwclust-package description for an explanation of DTW2)
D.dtw <- proxy::dist(CharTraj[1], CharTraj[2:5], method = "DTW2",
  window.type = "slantedband", window.size = 20)

D.lbi <- D.dtw

---

## Description

This function calculates a lower bound (LB) on the Dynamic Time Warp (DTW) distance between two time series. It uses a Sakoe-Chiba constraint.

## Usage

```r
lb_keogh(x, y, window.size = NULL, norm = "L1", lower.env = NULL,
  upper.env = NULL)
```

## Arguments

- **x**: A time series.
- **y**: A time series with the same length as `x`.
- **window.size**: Window size for envelope calculation. See details.
- **norm**: Pointwise distance. Either `L1` for Manhattan distance or `L2` for Euclidean.
- **lower.env**: Optionally, a pre-computed lower envelope for `y` can be provided.
- **upper.env**: Optionally, a pre-computed upper envelope for `y` can be provided.

## Details

The lower bound is defined for time series of equal length only.

The windowing constraint uses a centered window. The calculations expect a value in `window.size` that represents the distance between the point considered and one of the edges of the window. Therefore, if, for example, `window.size = 10`, the warping for an observation `x_i` considers the points between `x_{i-10}` and `x_{i+10}`, resulting in `10*2 + 1 = 21` observations falling within the window.

## Value

A list with:
- **d**: The lower bound of the DTW distance.
- **upper.env**: The time series of `y`'s upper envelope.
- **lower.env**: The time series of `y`'s lower envelope.
Note

If you wish to calculate the lower bound between several time series, it would be better to use the version registered with the 'proxy' package, since it includes some small optimizations. See the examples.

References


Examples

```r
# Sample data
data(uciCT)

# Lower bound distance between two series
d.lbk <- lb_keogh(CharTraj[[1]], CharTraj[[2]], window.size = 20)

d.dt <- dtw(CharTraj[[1]], CharTraj[[2]],
window.type = "slantedband", window.size = 20)

d.lbk <= d.dt

# Calculating the LB between several time series using the 'proxy' package
# (notice how both arguments must be lists)
D.lbk <- proxy::dist(CharTraj[1], CharTraj[2:5], method = "LB_Keogh",
window.size = 20, norm = "L2")

# Corresponding true DTW distance
# (see dtwclust-package description for an explanation of DTW2)
D.dt <- proxy::dist(CharTraj[1], CharTraj[2:5], method = "DTW2",
window.type = "slantedband", window.size = 20)

D.lbk <= D.dt
```

NCCc

Cross-correlation with coefficient normalization

Description

This function uses FFT to compute the cross-correlation sequence between two series. They need not be of equal length.

Usage

NCCc(x, y)
Arguments

- **x**: A time series.
- **y**: Another time series.

Value

The cross-correlation sequence with length \( \text{length}(x) + \text{length}(y) - 1 \).

References


See Also

SBD

---

**plot-dtwclust**  
*Plot the result of dtwclust*

Description

Plots the time series of each cluster along with the obtained centroid. It uses ggplot2 plotting system.

Usage

```r
## S4 method for signature 'dtwclust,missing'
plot(x, y, clus = seq_len(x@k),
     labs.arg = NULL, data = NULL, time = NULL, plot = TRUE, ...)
```

Arguments

- **x**: An object of class `dtwclust-class` as returned by `dtwclust`.
- **y**: Ignored.
- **clus**: Which clusters to plot.
- **labs.arg**: Arguments to change the title and/or axis labels. See `labs` for more information.
- **data**: The data in the same format as it was provided to `dtwclust`.
- **time**: Optional values for the time axis. If series have different lengths, provide the time values of the longest series.
- **plot**: Boolean flag. You can set this to `FALSE` in case you want to save the ggplot object without printing anything to screen.
- **...**: Further arguments to pass to `geom_line` for the plotting of the cluster centers. Default values are: `linetype = "dashed", size = 1.5, colour = "black", alpha = 0.5`.
Details
The flag save_data must be set to TRUE when running `dtwclust` to be able to use this.
Optionally, you can manually provide the clustering result as well as the data in data.
The function returns the gg object invisibly, in case you want to modify it to your liking. You might want to look at `ggplot_build` if that’s the case.

See Also
- `dtwclust-class`, `dtwclust`, `ggplot`

---

### reinterpolate

**Wrapper for simple linear reinterpolation**

**Description**
This function is just a wrapper for the native function `approx` to do simple linear reinterpolation.

**Usage**

```r
reinterpolate(ts, newLength)
```

**Arguments**
- `ts` A time series.
- `newLength` Desired length of the output series.

**Value**
Reinterpolated time series

---

### SBD

**Shape-based distance**

**Description**
Distance based on coefficient-normalized cross-correlation as proposed by Papparizos and Gravano, 2015, for the k-Shape clustering algorithm.

**Usage**

```r
SBD(x, y, znorm = FALSE)
```
**Arguments**

- **x**: A time series.
- **y**: Another time series.
- **znorm**: Should each series be z-normalized before calculating the distance?

**Details**

This function works best if the series are z-normalized. If not, at least they should have corresponding amplitudes, since the values of the signal do affect the outcome.

If x and y do not have the same length, it would be best if the longer sequence is provided in y, because it will be shifted to match x. Anything before the matching point is discarded and the series is padded with trailing zeros as needed.

The output values lie between 0 and 2, with 0 indicating perfect similarity.

**Value**

A list with:

- **dist**: The distance between x and y.
- **yshift**: A shifted version of y so that it optimally matches x.

**References**


**See Also**

`NCCc, shape_extraction`

**Examples**

```r
# load data
data(uciICT)

# distance between series of different lengths
sbd <- SBD(CharTraj[[1]], CharTraj[[10]], znorm = TRUE)$dist

# cross-distance matrix for series subset (notice the two-list input)
sBD <- proxy::dist(CharTraj[1:10], CharTraj[1:10], method = "SBD", znorm = TRUE)
```
shape_extraction

Shape average of several time series

Description
Time-series shape extraction based on optimal alignments as proposed by Papparizos and Gravano, 2015, for the k-Shape clustering algorithm.

Usage
shape_extraction(X, cz = NULL, znorm = TRUE)

Arguments
- **X**: Numeric matrix where each row is a time series.
- **cz**: Center to use as basis. It should already be normalized. Calculation uses all X if cz = NULL.
- **znorm**: Boolean flag. Should z-scores be calculated for X before processing?

Details
This works only if the signals are z-normalized, since the output will also have this normalization.
This centroid computation is casted as an optimization problem called maximization of Rayleigh Quotient. See the cited article for more details.

Value
Centroid time series.

References

See Also
SBD, zscore

Examples
```r
# Sample data
data(uciCT)

# Subset of interest, normalized
X <- t(sapply(CharTraj[1:5], zscore))
```
Description

Time-series Anytime Density Peaks Clustering as proposed by Begum et al., 2015.

Usage

TADPole(data, window.size = NULL, k = 2, dc, error.check = TRUE)

Arguments

data The data matrix where each row is a time series. Optionally a list with each time series.
window.size Window size constraint for DTW. See details.
k The number of desired clusters.
dc The cutoff distance.
error.check Should the data be checked for inconsistencies?

Details

This function can be called either directly or through dtwclust.

TADPole clustering adopts a relatively new clustering framework and adapts it to time series clustering with DTW. See the cited article for the details of the algorithm.

Because of the way the algorithm works, it can be considered a kind of Partitioning Around Medoids (PAM). This means that the cluster centers are always elements of the data.

The algorithm first uses the DTW’s upper and lower bounds to find series with many close neighbors (in DTW space). Anything below the cutoff distance (dc) is considered a neighbor. Aided with this information, the algorithm then tries to prune as many DTW calculations as possible in order to accelerate the clustering procedure. The series that lie in dense areas (i.e. that have lots of neighbors) are taken as cluster centers.

The algorithm relies on the DTW bounds, which are only defined for time series of equal lengths. The windowing constraint uses a centered window. The calculations expect a value in window.size that represents the distance between the point considered and one of the edges of the window. Therefore, if, for example, window.size = 10, the warping for an observation \( x_i \) considers the points between \( x_{i-10} \) and \( x_{i+10} \), resulting in \( 10 \times 2 + 1 = 21 \) observations falling within the window.
Value

A list with:

- `cl`: Cluster indices.
- `centers`: Indices of the centers.
- `distCalcPercentage`: Percentage of distance calculations that were actually performed.

References


uciCT

Subset of character trajectories data set

Description

Subset: only 5 examples of X velocity. See details.

Format

A list with 100 elements. Each element is a time series. Labels included as factor vector.

Details

Quoting the source:

"Multiple, labelled samples of pen tip trajectories recorded whilst writing individual characters. All samples are from the same writer, for the purposes of primitive extraction. Only characters with a single pen-down segment were considered."

The subset included here (CharTraj) has only 5 examples of the X velocity for each character. A vector with labels is also loaded in CharTrajLabels.

Source

https://archive.ics.uci.edu/ml/datasets/Character+Trajectories
Description

Wrapper for function `scale` that returns zeros instead of NaN.

Usage

```
zscore(x, ...)
```

Arguments

- `x` Data to normalize.
- `...` Further arguments to pass to `scale`.

Value

Normalized data.
Index

approx, 16
CharTraj (uciCT), 20
CharTrajLabels (uciCT), 20

DBA, 3, 5, 7
dist, 2, 3, 5, 6
dtw, 2, 3
dtw_lb, 6, 10
dtwclust, 2, 3, 5, 10, 15, 16, 19
dtwclust-class, 9
dtwclust-package, 2

flexclustControl, 5
gem_line, 15
ggplot, 16
ggplot_build, 16

hclust, 5, 6

kcca, 3, 6

labs, 15
lb_improved, 2, 11, 11
lb_keogh, 2, 13

NCCc, 14, 17

plot, dtwclust, missing-method
   (plot-dtwclust), 15
plot-dtwclust, 15
pr_DB, 2
reinterpolate, 2, 16

SBD, 2, 6, 8, 15, 16, 18
scale, 21
shape_extraction, 7, 8, 17, 18

TADPole, 6, 19

uciCT, 20
ucict (uciCT), 20
zscore, 18, 21