Package 'IncDTW'

January 3, 2019

Type Package

Title Incremental Calculation of Dynamic Time Warping

Version 1.0.5

Author Maximilian Leodolter

Maintainer Maximilian Leodolter <maximilian.leodolter@gmail.com>

Description The Dynamic Time Warping (DTW) distance for time series allows non-linear alignments of time series to match similar patterns in time series of different lengths and or different speeds. Beside the traditional implementation of the DTW algorithm, the specialities of this package are, (1) the incremental calculation, which is specifically useful for life data streams due to computationally efficiency, (2) the vector based implementation of the traditional DTW algorithm which is faster because no matrices are allocated and is especially useful for computing distance matrices of pairwise DTW distances for many time series and (3) the combination of incremental and vector-based calculation. C++ in the heart. For details about DTW see the original paper ``Dynamic programming algorithm optimization for spoken word recognition" by Sakoe and Chiba (1978) <DOI:10.1109/TASSP.1978.1163055>.

License GPL (>= 2)

Encoding UTF-8

LazyData true

Depends R (>= 2.10)

Imports Rcpp (>= 0.12.8), RcppParallel, ggplot2, scales, parallel, stats, data.table

LinkingTo Rcpp, RcppParallel, RcppArmadillo

NeedsCompilation yes

RoxygenNote 6.0.1

Suggests knitr, dtw, rmarkdown, gridExtra, testthat, dtwclust, parallelDist, microbenchmark, rucrdtw, proxy

VignetteBuilder knitr

SystemRequirements GNU make

Repository CRAN

Date/Publication 2019-01-03 15:20:09 UTC

R topics documented:

`IncDTW-package` *Incremental Dynamic Time Warping*

Description

The Dynamic Time Warping (DTW) distance for time series allows non-linear alignments of time series to match similar patterns in time series of different lengths and or different speeds. Beside the traditional implementation of the DTW algorithm, the specialities of this package are, (1) the incremental calculation, which is specifically useful for life data streams due to computationally efficiency, (2) the vector based implementation of the traditional DTW algorithm which is faster because no matrices are allocated and is especially useful for computing distance matrices of pairwise DTW distances for many time series and (3) the combination of incremental and vector-based calculation.

Details

Main features:

- C++ in the heart thanks to Rcpp
- Incremental Calculation, `idtw()` and `idtw2vec()`
- Matrix-based `dtw()` and Vector-based `dtw2vec()` implemtantion of the DTW algorithm
- Sakoe Chiba Warping Window
- Early abandoning
- support for multivariate time series
- `distDTW()` for fast calculation of a Distance Matrix of pairwise DTW distances for clustering or classification of many multivariate time series
- `DBA()` for averaging a centroid of a found cluster

Author(s)

Maximilian Leodolter

Maintainer: Maximilian Leodolter <maximilian.leodolter@gmail.com>

References

Dynamic programming algorithm optimization for spoken word recognition by Sakoe and Chiba published in 1978 (DOI:10.1109/TASSP.1978.1163055)

See Also

https://ieeexplore.ieee.org/document/1163055/

https://en.wikipedia.org/wiki/Dynamic_time_warping

https://github.com/maxar/IncDTW

DBA *Dynamic Time Warping Barycenter Averaging*

Description

Average multiple time series that are non-linearly aligned by Dynamic Time Warping. Find the centroid of a list of time series.

Usage

```
DBA(lot, m0 = NULL, iterMax = 10, eps = NULL,
                dist_method = c("norm1", "norm2", "norm2_square"),
                step_pattern = c("symmetric2", "symmetric1"),
                ws = NULL,
                iter_dist_method = c("dtw_norm1", "dtw_norm2",
                                     "norm1","norm2", "max", "min"),
                plotit = FALSE)

centroid(lot, dist_method = c("norm1", "norm2", "norm2_square"),
              step_pattern = c("symmetric2", "symmetric1"),
              normalize = TRUE, ws = NULL, ncores = NULL,
              useRcppParallel = TRUE)
```

Arguments

`lot`	List of time series. Each entry of the list is a time series as described in `dtw2vec`.
`m0`	time series as vector or matrix. If `m0` is `NULL`, the initial time series `m0` is determined by `centroid` as the centroid of `lot`, which is the one time series of `lot` with the minimum average DTW distance to all other time series of `lot`.
`iterMax`	integer, number of maximum iterations

eps	numeric, threshold parameter that causes the algorithm to break if the distance of two consecutive barycenters are closer than eps
dist_method	character, describes the method of distance measure. See also dtw.
step_pattern	character, describes the step pattern. See also dtw.
ws	integer, describes the window size for the sakoe chiba window. If NULL, then no window is applied. (default = NULL)
iter_dist_method	character, that describes how the distance between two consecutive barycenter iterations are defined (default = "dtw")
plotit	logical, if the iterations should be plotted or not (only possible for univariate time series)
normalize	logical, default is TRUE, passed to dtw_dismat
ncores	integer, default = NULL, passed to dtw_dismat
useRcppParallel	logical, default is TRUE, passed to dtw_dismat

Details

The parameter iter_dist_method describes the method to measure the progress between two iterations. For two consecutive centroid candidates m1 and m2 the following methods are implemented:

'dtw_norm1': `dtw2vec(m1, m2, dist_method = "norm1", step_pattern = "symmetric2")$normalized_distance`

'idm_dtw2': `dtw2vec(m1, m2, dist_method = "norm2", step_pattern = "symmetric2")$normalized_distance`

'idm_norm1': `sum(abs(m1-m2))/(ncol(m1) * 2 * nrow(m1))`

'idm_norm2': `sqrt(sum((m1-m2)^2))/(ncol(m1) * 2 * nrow(m1))`

'idm_max': `max(abs(m1-m2))`

'idm_min': `min(abs(m1-m2))`

Value

call	function call
m1	new centroid/ bary center of the list of time series
iterations	list of time series that are the best centroid of the respective iteration
iterDist_m2lot	list of distances of the iterations to lot
iterDist_m2lot_norm	list of normalized distances of the iterations to lot
iterDist_m2m	vector of distances of the iterations to their ancestors
centroid_index	integer giving the index of the centroid time series of lot
dismat_result	list of results of dtw_dismat called by centroid

Author(s)

Maximilian Leodolter

References

Sakoe, H.; Chiba, S., Dynamic programming algorithm optimization for spoken word recognition, Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on , vol.26, no.1, pp. 43-49, Feb 1978. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1163055

Petitjean, F; Ketterlin, A; Gancarski, P, A global averaging method for dynamic time warping, with applications to clustering, Pattern Recognition, Volume 44, Issue 3, 2011, Pages 678-693, ISSN 0031-3203

Examples

```
## Not run:
data("drink_glass")
# initialize with any time series
m1 <- DBA(lot = drink_glass[1:10], m0 = drink_glass[[1]],
          dist_method = "norm2", iterMax = 20)

# initialize with the centroid

tmp <- centroid(drink_glass)
cent <- drink_glass[[tmp$centroid_index]]
m1 <- DBA(lot = drink_glass[1:10], m0 = cent,
          dist_method = "norm2", iterMax = 20)
plot(sapply(m1$iterDist_m2lot, mean), xlab = "Iterations",
         ylab = "mean distance",
         main = "Distance of updated bary center to lot")
plot(m1$iterDist_m2m, ylab = "distance", xlab = "Iterations",
     main = "Distance of iterations of bary center updates")

plot(m1)
plot(m1, type = "m2m")
plot(m1, type = "m2lot")



## End(Not run)
```

---

dec_dm *Decrement the Warping Path*

Description

Update the warping path to omit observations of the alignment of two time series.

Usage

```
dec_dm(dm, Ndec, diffM = NULL)
```

Arguments

dm	direction matrix, output from dtw(Q=Q, C=C, ws=ws)
Ndec	integer, number of observations (columns) to be reduced
diffM	matrix of differences

Value

wp	warping path
ii	indices of C of the optimal path
jj	indices of Q of the optimal path
diffp	path of differences (only returned if diffM is not NULL)

Author(s)

Maximilian Leodolter

References

Sakoe, H.; Chiba, S., Dynamic programming algorithm optimization for spoken word recognition, Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on , vol.26, no.1, pp. 43-49, Feb 1978. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1163055

Examples

```
Q <- cos(1:100)
C <- cumsum(rnorm(80))
# the ordinary calculation
result_base <- dtw(Q=Q, C=C, return_diffM = FALSE)

# the ordinary calculation without the last 4 observations
result_decr <- dtw(Q=Q, C=C[1:(length(C) - 4)], return_diffM = FALSE)
# the decremental step: reduce C for 4 observation
result_decr2 <- dec_dm(result_base$dm, Ndec = 4)

# compare ii, jj and wp of result_decr and those of
result_decr$ii
result_decr2$ii
identical(result_decr$ii, result_decr2$ii)

result_decr$jj
result_decr2$jj
identical(result_decr$jj, result_decr2$jj)

result_decr$wp
result_decr2$wp
identical(result_decr$wp, result_decr2$wp)
```

drink_glass *Acceleromter: drink a glass, walk, brush teeth.*

Description

3-dimensional acceleration time series recorded during the activites of walking, drinking a glass or brushing teeth.

Usage

```
data("drink_glass")
```

Format

A list of matrices, where each matrix has 3 columns (x, y, and z axis of the accelerometer). The number of rows differ.

Details

list of 3-dimensional time series stored as matrix. The data is recorded with 32Hz.

Source

UCI Machine Learning Repository https://archive.ics.uci.edu/ml/datasets/Dataset+for+ADL+Recognition+with+Wrist-worn+Accelerometer

Examples

```
## Not run:
data(drink_glass)
class(drink_glass)
length(drink_glass)
dim(drink_glass[[1]])
matplot(drink_glass[[1]], type="l")

data(Walk)
class(Walk)
length(Walk)
dim(Walk[[1]])
matplot(Walk[[1]], type="l")

data(brush_teeth)
class(brush_teeth)
length(brush_teeth)
dim(brush_teeth[[1]])
matplot(brush_teeth[[1]], type="l")

## End(Not run)
```

dtw *Dynamic Time Warping*

Description

Calculate the DTW distance, cost matrices and direction matrices including the warping path two multivariate time series.

Usage

```
dtw(Q, C, dist_method = c("norm1", "norm2", "norm2_square"),
    step_pattern = c("symmetric2", "symmetric1"), ws = NULL,
    return_cm = FALSE,
    return_diffM = FALSE,
    return_wp = FALSE,
    return_diffp = FALSE,
    return_QC = FALSE)

cm(Q, C, dist_method = c("norm1", "norm2", "norm2_square"), ws = NULL)
```

Arguments

Q	Query time series. Q needs to be one of the following: (1) a one dimensional vector, (2) a matrix where each row is one observations and each column is one dimension of the time seris, or (3) a matrix of differences/ costs (diffM, cm). If Q and C are matrices they need to have the same number of columns.
C	Candidate time series. C needs to be one of the following: (1) a one dimensional vector, (2) a matrix where each row is one observations and each column is one dimension of the time seris, or (3) if Q is a matrix of differences or costs C needs to be the respective character string 'diffM' or 'cm'.
dist_method	character, describes the method of distance measure for multivariate time series (this parameter is ignored for univariate time series). Currently supported methods are 'norm1' (default, is the manhatten distance), 'norm2' (is the Euclidean distance) and 'norm2_square' For the function cm the parameter dist_method can also be a user defined distance function.
step_pattern	character, describes the step pattern. Currently implemented are the patterns symmetric1 and symmetric2, see details.
ws	integer, describes the window size for the sakoe chiba window. If NULL, then no window is applied. (default = NULL)
return_cm	boolean, if TRUE then the Matrix of costs (the absolute value) is returned. (default = FALSE)
return_diffM	boolean, if TRUE then the Matrix of differences (not the absolute value) is returned. (default = FALSE)
return_wp	boolean, if TRUE then the warping path is returned. (default = FALSE) If return_diffp == TRUE, then return_wp is set to TRUE as well.

`return_diffp` boolean, if TRUE then the path of differences (not the absolute value) is returned. (default = FALSE)

`return_QC` boolean, if TRUE then the input vectors Q and C are appended to the returned list. This is useful for the `plot.idtw` function. (default = FALSE)

Details

The dynamic time warping distance is the element in the last row and last column of the global cost matrix.

For the multivariate case where Q is a matrix of n rows and k columns and C is a matrix of m rows and k columns the dist_method parameter defines the following distance measures:

norm1:

$$dist(Q_{i,.}, C_{j,.}) = \sum_{l=1}^{k} |Q_{i,l} - C_{j,l}|$$

norm2:

$$dist(Q_{i,.}, C_{j,.}) = \sqrt{\sum_{l=1}^{k} (Q_{i,l} - C_{j,l})^2}$$

norm2_square:

$$dist(Q_{i,.}, C_{j,.}) = \sum_{l=1}^{k} (Q_{i,l} - C_{j,l})^2$$

The parameter `step_pattern` describes how the two time series are aligned. If `step_pattern == "symmetric1"` then

$$gcm_{i,j} = cmi, j + min(gcm_{i-1,j}, gcmi - 1, j - 1, gcmi, i - 1$$

.

If `step_pattern == "symmetric2"` then

$$gcm_{i,j} = cmi, j + min(gcm_{i-1,j}, cmi, j + gcmi - 1, j - 1, gcmi, i - 1$$

.

To make DTW distances comparable for many time seires of differnt lengths use the `normlized_distance` with the setting `step_method = 'symmetric2'`. Please find a more detailed discussion and further references here: http://dtw.r-forge.r-project.org/.

Value

`distance` the DTW distance, that is the element of the last row and last column of gcm

`normalized_distance` the normalized DTW distance, that is the distance divided by `N+M`, where N and M are the lengths of the time series Q and C, respectively. If `step_pattern == 'symmetric1'` no normaliziation is performed and `NA` is returned (see details).

`gcm` global cost matrix

`dm` direction matrix (3=up, 1=diagonal, 2=left)

wp	warping path
ii	indices of C of the optimal path
jj	indices of Q of the optimal path
cm	Matrix of costs
diffM	Matrix of differences
diffp	path of differences
Q	input Q
C	input C

Author(s)

Maximilian Leodolter

References

Sakoe, H.; Chiba, S., Dynamic programming algorithm optimization for spoken word recognition, Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on , vol.26, no.1, pp. 43-49, Feb 1978. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1163055

Examples

```
# multivariate
Q <- matrix(rnorm(100), ncol=2)
C <- matrix(rnorm(80), ncol=2)
tmp <- dtw(Q = Q, C = C, ws = 30)


# univariate
Q <- cumsum(rnorm(100))
C <- Q[11:100] + rnorm(90, 0, 0.5)
tmp <- dtw(Q = Q, C = C, ws = 15, return_diffM = FALSE)
names(tmp)
tmp$distance



# compare different input variations
dtw_base <- dtw(Q = Q, C = C, ws = 15, return_diffM = TRUE)
dtw_diffM <- dtw(Q = dtw_base$diffM, C = "diffM", ws = 15, return_diffM = TRUE)
dtw_cm <- dtw(Q = abs(dtw_base$diffM), C = "cm", ws = 15, return_diffM = TRUE)

identical(dtw_base$gcm, dtw_cm$gcm)
identical(dtw_base$gcm, dtw_diffM$gcm)

# of course no diffM is returned in the 'cm'-case
dtw_cm$diffM

# multivariate case
Q <- matrix(rnorm(100), ncol=2)
```

```
C <- matrix(rnorm(80), ncol=2)
dtw(Q = Q, C = C, ws = 30, dist_method = "norm2")
```

---

dtw2vec *Fast vector-based Dynamic Time Warping*

Description

Calculates the Dynamic Time Warping distance by hand of a vector-based implementation and is much faster than the traditional method `dtw()`. Also allows early abandoning and sakoe chiba warping window, both for univariate and multivariate time series.

Usage

```
dtw2vec(Q, C, dist_method = c("norm1", "norm2", "norm2_square"),
         step_pattern = c("symmetric2", "symmetric1"),
         ws = NULL, threshold = NULL)
```

Arguments

`Q`	Either Q is (a) a time series (vector or matrix for multivariate time series) or (b) Q is a cost matrix, so a matrix storing the local distances of the time series Q and C. If Q and C are matrices, they need to have the same number of columns. If Q is a cost matrix, C needs to be equal the charachter string "cm".
`C`	time series as vector or matrix, or for case (b) C equals "cm"
`dist_method`	character, describes the method of distance measure. See also `dtw`. If Q is a cost matrix, the dist_method parameter is not necessary.
`step_pattern`	character, describes the step pattern. See also `dtw`.
`ws`	integer, describes the window size for the sakoe chiba window. If NULL, then no window is applied. (default = NULL)
`threshold`	numeric, the threshold for early abandoning. In the calculation of the global cost matrix a possible path stops as soon as the threshold is reached. Facilitates faster calculations incase of low threshold. The threshold relates to the non-normalized distance measure. (default = NULL, no early abandoning)

Details

no matrices are allocated, no matrices are returned

Value

`distance`	the DTW distance
`normalized_distance`	the normalized DTW distance, see also `dtw`

Author(s)

Maximilian Leodolter

References

Sakoe, H.; Chiba, S., Dynamic programming algorithm optimization for spoken word recognition, Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on , vol.26, no.1, pp. 43-49, Feb 1978. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1163055

Examples

```
Q <- cumsum(rnorm(100))
C <- Q[11:100] + rnorm(90, 0, 0.5)
dtw2vec(Q = Q, C = C)
dtw2vec(Q = Q, C = C, ws = 30)
dtw2vec(Q = Q, C = C, threshold = 100)
dtw2vec(Q = Q, C = C, ws = 30, threshold = 100)

cm0 <- cm(Q, C)
dtw2vec(Q = cm0, C = "cm", ws = 30, threshold = 100)
```

---

dtw_dismat *DTW Distance Matrix*

Description

Calculates a matrix of pairwise DTW distances for many multivariate time series, or a vector of DTW distances all relative to one query time series. A parallel computation is possible.

Usage

```
dtw_dismat(lot, dist_method = c("norm1", "norm2", "norm2_square"),
         step_pattern = c("symmetric2", "symmetric1"), normalize = TRUE,
         ws = NULL, threshold = NULL,
         return_matrix = TRUE, ncores = NULL, useRcppParallel = TRUE)

dtw_disvec(Q, lot, dist_method = c("norm1", "norm2", "norm2_square"),
         step_pattern = c("symmetric2", "symmetric1"), normalize = TRUE,
         ws = NULL, threshold = NULL, ncores = NULL)
```

Arguments

Q	time series, vector or matrix
lot	List of time series. Each entry of the list is a time series as described in dtw2vec.
dist_method	character, describes the method of distance measure. See also dtw.
step_pattern	character, describes the step pattern. See also dtw.

normalize	logical, whether to return normalized pairwise distances or not. If `step_pattern == 'symmetric1'` only non-normalized distances can be returned (default = TRUE)
ws	integer, describes the window size for the sakoe chiba window. If NULL, then no window is applied. (default = NULL)
threshold	numeric, the threshold for early abandoning. In the calculation of the global cost matrix a possible path stops as soon as the threshold is reached. Facilitates faster calculations incase of low threshold. (default = FALSE)
return_matrix	boolean, If TRUE (default) a symmetric matrix of differences is returned. If FALSE the distance matrix is returned as `dist` object.
ncores	integer, number of cores to be used for parallel computation of the distance matrix. If `ncores = NULL` (default) then ncores is set to the number of available cores minus 1. If `ncores = 0` then no parallel computation is performed and standard `sapply` instead of `parallel::parSapply` is applied.
useRcppParallel	boolean, if the package RcppParallel (TRUE, default) or parallel (FALSE) is used for parallel computation

Details

no matrices are allocated, no matrices are returned

Value

input	the function input parameters
dismat	the matrix of pairwise DTW distances, either as matrix or dist object
disvec	the vector DTW distances

Author(s)

Maximilian Leodolter

Examples

```
lot <- lapply(1:10, function(i){matrix(rnorm(100), ncol=2)})

## Not run:
# per default all minus 1 available cores are used
result <- dtw_dismat(lot = lot, dist_method = "norm2", ws = 20,
                  return_matrix = FALSE)

## End(Not run)

# for few time series you can set ncores=0 for NON-parallel computation
result <- dtw_dismat(lot = lot, dist_method = "norm2", ws = 20,
                  return_matrix = FALSE, ncores = 1)
```

`dtw_partial` *Partial Dynamic Time Warping*

Description

Get the cheapest partial open end alignment of two time series

Usage

```
dtw_partial(x, partial_Q = TRUE, partial_C = TRUE, reverse = FALSE)
```

Arguments

`x`	result object of either `dtw()` or `idtw2vec()`
`partial_Q`	logical (default = TRUE), whether Q is aligned completly to C or open ended.
`partial_C`	logical (default = TRUE), whether C is aligned completly to Q or open ended.
`reverse`	logical (default = FALSE), whether Q and C are in original or reverse order.

Details

Q is the time series that describes the vertical dimension of the global cost matrix, so `length(Q)` is equal to `nrow(x$gcm)`. So C describes the horizontal dimension of the global cost matrix, `length(C)` is equal to `ncol(x$gcm)`.

`dtw_partial()` returns the open-end alignment of Q and C with the minimal normalized distance. If `partial_Q` and `partial_C` both are `TRUE` the partial alignmnet with the smaller normalized distance is returned.

If Q and C are in reverse order, then the optimal solution for the reverse problem is found, that is the aligmnet with minial normalized distance allowing and open-start alignment.

Value

`rangeQ`	Vector of initial and ending index for best alignment
`rangeC`	Vector of initial and ending index for best alignment
`normalized_distance`	the normalized DTW distance (see details in `dtw`).

Author(s)

Maximilian Leodolter

Examples

```
# open-end alignment for multivariate time series
# first simulate a 2-dim time series Q
Q <- matrix(cumsum(rnorm(100)), ncol=2)

# simulate C as noisy and warped version of Q, and append noise at the end
C <- Q[sort(sample(50, 40, replace=FALSE)), ] + rnorm(80)
C <- rbind(C, matrix(rnorm(30), ncol=2))

tmp <- dtw(Q = Q, C = C, ws = 30, return_QC = TRUE, return_wp = TRUE)
par <- dtw_partial(tmp, partial_C = TRUE)
par
plot(tmp, partial = par, type = "QC")
plot(tmp, partial = par, type = "warp")
plot(tmp, partial = par, type = "QC", selDim = 2)



# open-start is possible as well (open-end is the regular case) since DTW is reversible:
Q <- sin(1:100)
C <- c(rnorm(50), Q)
tmp <- dtw(Q = rev(Q), C = rev(C))
dtw_partial(tmp, reverse = TRUE)
```

---

idtw *Incremental DTW*

Description

Update the DTW distance, cost matrices and direction matrices including the warping path for new observations of two time series.

Usage

```
idtw(Q, C, newObs, gcm, dm, dist_method = c("norm1", "norm2", "norm2_square"),
      step_pattern = c("symmetric2", "symmetric1"),
      diffM = NULL, ws = NULL,
      return_cm = FALSE,
      return_diffM = FALSE,
      return_wp = FALSE,
      return_diffp = FALSE,
      return_QC = FALSE)
```

Arguments

Q	numeric vector, or matrix (see also dtw)
C	numeric vector, or matrix

newObs	vector or matrix of new observations to be appended to C
gcm	global cost matrix, output from dtw(Q=Q, C=C, ws=ws)
dm	direction matrix, output from dtw(Q=Q, C=C, ws=ws)
dist_method	character, describes the method of distance measure. See also dtw.
step_pattern	character, describes the step pattern. See also dtw.
diffM	differences matrix, output from dtw(Q=Q, C=C, ws=ws). This matrix is an optional input parameter (default = NULL) that is necessary to return the path of differences. It makes no sense in the multivariate case.
ws	integer, describes the window size for the sakoe chiba window. If NULL, then no window is applied. (default = NULL)
return_cm	boolean, if TRUE then the Matrix of costs (the absolute value) is returned. (default = FALSE)
return_diffM	boolean, if TRUE then the Matrix of differences (not the absolute value) is returned. (default = FALSE)
return_wp	boolean, if TRUE then the warping path is returned. (default = FALSE) If return_diffp == TRUE, then return_wp is set to TRUE as well.
return_diffp	boolean, if TRUE then the path of differences (not the absolute value) is returned. (default = FALSE)
return_QC	boolean, if TRUE then the input vectors Q and C are appended to the returned list. This is useful for the plot.idtw function. (default = FALSE)

Details

The dynamic time warping distance is the element in the last row and last column of the global cost matrix.

Value

distance	the DTW distance, that is the element of the last row and last column of gcm
gcm	global cost matrix
dm	direction matrix (3=up, 1=diagonal, 2=left)
wp	warping path
ii	indices of C of the optimal path
jj	indices of Q of the optimal path
cm	Matrix of costs
diffM	Matrix of differences
diffp	path of differences
Q	input Q
C	input C
normalized_distance	the normalized DTW distance, see also link{dtw}

Author(s)

Maximilian Leodolter

References

Sakoe, H.; Chiba, S., Dynamic programming algorithm optimization for spoken word recognition, Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on , vol.26, no.1, pp. 43-49, Feb 1978. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1163055

Examples

```
Q <- cumsum(rnorm(100))
C <- Q[11:100] + rnorm(90, 0, 0.5)
newObs <-  c(2,3)# new observation
base <- dtw(Q=Q, C=C, ws = 15, return_diffM = TRUE) # the ordinary calculation


#--- recalculation from scratch with new observations
result0 <- dtw(Q=Q, C=c(C, newObs), ws = 15,  return_diffM = TRUE) # the ordinary calculation

#--- the incremental step with new observations
result1 <- idtw(Q, C, ws = 15, newO = newObs, gcm=base$gcm, dm=base$dm, diffM = base$diffM,
               return_diffp = TRUE,  return_diffM = TRUE)

#--- the incremental step with new observations,
#     but already calculated additive costMatrix cm_add
mQ <- matrix(Q, ncol = length(newObs), nrow = length(Q), byrow = FALSE)
mC <- matrix(newObs, ncol = length(newObs), nrow = length(Q), byrow = TRUE)
cm_add <- matrix(abs(mQ - mC), ncol = length(newObs))
result2 <- idtw(Q=cm_add, C="cm_add", ws = 15, newO = newObs, gcm=base$gcm, dm=base$dm)

c(result0$distance, result1$distance, result2$distance)


#--- now with integers
Q <- 1:10
C <- 2:20
base <- dtw(Q=Q, C=C, ws = 15, return_diffM = TRUE) # the ordinary calculation
# new observation
newObs <-  c(2L,3L)
# the incremental step with new observations
tmp1 <- idtw(Q, C, newO = newObs, gcm=base$gcm, dm=base$dm, diffM = base$diffM,
      return_diffp = TRUE, ws = 15, return_diffM = TRUE)
str(tmp1)
```

idtw2vec *Incremental vector-based DTW*

Description

Update the DTW distance for new observations of two time series.

Usage

```
idtw2vec(Q, newObs, dist_method = c("norm1", "norm2", "norm2_square"),
         step_pattern = c("symmetric2", "symmetric1"),
         gcm_lc = NULL, gcm_lr = NULL, nC = NULL, ws = NULL)
```

Arguments

Q	Either Q is (a) a time series (vector or matrix for multivariate time series) or (b) Q is a cost matrix, so a matrix storing the local distances of the time series Q and newObs. If Q and newObs are matrices, they need to have the same number of columns. If Q is a cost matrix, see details...
newObs	time series as vector or matrix, or if Q is a cost matrix newObs must equals "cm". If newObs is a time series, see details...
dist_method	character, describes the method of distance measure. See also dtw.
step_pattern	character, describes the step pattern. See also dtw.
gcm_lc	vector, last column of global cost matrix of previous calculation. If NULL (necessary for the initial calculation), then DTW is calculated and the last column and last row are returned to start upcoming incremental calculations. (default = NULL)
gcm_lr	vector, last row of global cost matrix of previous calculation (default = NULL).
nC	integer, is the length of the original time series C, of which newObs are the new observations. Length of time series C exclusive new observations, such that length(c(C, newObs)) = nC + length(newObs). Necessary if ws is not NULL. (default = NULL)
ws	integer, describes the window size for the sakoe chiba window. If NULL, then no window is applied. (default = NULL)

Details

If new observations are recorded only for C and the only interest is a fast update of the DTW distance, the last row is not required, neither for the current nor for future incremental calculations.

If Q is a cost matrix, it needs to store either the distances of Q and new observations of C (running calculations, in that case gcm_lc != NULL), or it stores the distances of Q and the entire time series C (initial calculation, in that case gcm_lc = NULL).

If newObs is a time series, it stores either new Observations of C (running calculations) or the complete time series C (initial calculation).

no matrices are allocated, no matrices are returned

Value

distance	the DTW distance
gcm_lc_new	the last column of the new global cost matrix
gcm_lr_new	the last row of the new global cost matrix. Only if the input vector gcm_lr is not NUll and represents the last row of the previous global cost matrix, gcm_lr_new actually is the last row of the updated global cost matrix. Otherwise, if gcm_lr is NULL then gcm_lr_new is only the last row of the new part (concerning the new observations) of the global cost matrix.
normalized_distance	the normalized DTW distance, see also link{dtw}

Author(s)

Maximilian Leodolter

References

Sakoe, H.; Chiba, S., Dynamic programming algorithm optimization for spoken word recognition, Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on , vol.26, no.1, pp. 43-49, Feb 1978. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1163055

Examples

```
Q <- cumsum(rnorm(100))
C <- Q[11:100] + rnorm(90, 0, 0.5)

# initial calculation
res0 <- idtw2vec(Q=Q, newObs = C, gcm_lc = NULL)

# incremental calculation for new observations
res1 <- idtw2vec(Q, newObs = rnorm(10), gcm_lc = res0$gcm_lc_new)



# perform an incremental DTW calculation with a customized distance function
d_cos <- function(x, y){
   sum(x * y)/(sqrt(sum(x^2)) * sqrt(sum(y^2)))
}

x <- matrix(rnorm(100), ncol=5, nrow=20)
y <- matrix(rnorm(150), ncol=5, nrow=30)

cm1 <- function(x,y){
   nx <- nrow(x); ny <- nrow(y)
   mat <- matrix(0, ncol=ny, nrow=nx)
   for(i in 1:nx){
      for(j in 1:ny){
         mat[i,j] <- d_cos(x[i,], y[j, ])
      }
   }
```

```
   return(mat)
}

dtw2vec(Q = cm1(x,y), C = "cm")$distance
res0 <- idtw2vec(Q = cm1(x,y[1:20,]), newObs =  "cm")
idtw2vec(Q = cm1(x,y[21:30,]), newObs =  "cm",
                gcm_lc = res0$gcm_lc_new)$distance
```

`norm` *Time Series Normalisation*

Description

normalises a time series per dimension

Usage

```
norm(x, type = c('z', '01'),
     xmean = NULL, xsd = NULL, xmin = NULL, xmax = NULL)
```

Arguments

`x`	time series as vector or matrix
`type`	character, describes the method of to normalise the time series (per column if x is multivariate). The parameter type is either 'z' for z-normalisation or '01' for max-min normalisation.
`xmean`	mean used for z-normalization
`xsd`	standard deviation used for z-normalization
`xmin`	minimum used for 0-1 normalization
`xmax`	maximum used for 0-1 normalization

Details

For a vector `x` the z-normalisation subtracts the mean and devides by the standard deviation: of `(x-mean(x))/sd(x)`. The min-max normalisation performs `(x-min(x))/(max(x)-min(x))`.

Value

`x`	the normalised vector

Author(s)

Maximilian Leodolter

Examples

```
x <- cumsum(rnorm(100, 10, 5))
y <- norm(x, "01")
z <- norm(x, "z")
```

`plot_dba` *Plot the results from Dynamic Time Warping Barycenter Averaging*

Description

Plot function for objects of type dba, the output of `DBA()`.

Usage

```
## S3 method for class 'dba'
plot(x, type = c("barycenter", "m2m", "m2lot"), ...)
# an alias for plot_idtw
plot_dba(x, type = c("barycenter", "m2m", "m2lot"), ...)

plotBary(x, ...)

plotM2m(x, ...)

plotM2lot(x, ...)
```

Arguments

`x`	output from `DBA()`
`type`	character, one of c('barycenter', 'm2m', 'm2lot')
`...`	Other arguments passed on to methods. Not currently used.

Details

- `'barycenter'` plots the iterations of the barycenter per dimension.
- `'m2m'` plots the distances (distance method set by `iter_dist_method`, see DBA) of one barycenter-iteration to the former iteration step.
- `'m2lot'` plots the distances (if `step_pattern == 'symmetric2'` the normalized distances are plotted) of the barycenter to the list of time series per iteration.

Author(s)

Maximilian Leodolter

See Also

`DBA`

plot_idtw *Plot the results from Dynamic Time Warping*

Description

Plot function for objects of type `idtw`, the output of `dtw()` and `idtw()` respectively.

Usage

```
## S3 method for class 'idtw'
plot(x, type = c("QC", "warp"),
        partial = NULL,
        selDim = 1, ...)

# an alias for plot_idtw
plot_idtw(x, type = c("QC", "warp"),
             partial = NULL,
             selDim = 1, ...)

plotQC(x, Q, C, partial = NULL, selDim = 1, ...)

plotWarp(x, Q, C, partial = NULL, selDim = 1, ...)
```

Arguments

`x`	output from `dtw(Q, C)`
`Q`	one dimensional numeric vector
`C`	one dimensional numeric vector
`type`	character, one of c('QC', 'warp')
`partial`	list, the return value of `dtw_partial()`. Default = NULL, see `dtw_partial()` for details.
`selDim`	integer, gives the column index of the multivariate time series (matrices) to be plotted. (default = 1) If Q and C are univariate time series (vectors) then selDim is neglected.
`...`	Other arguments passed on to methods.

Details

The plot function visualizes the time warp and the alignment of the two time series. Also for partial alignmnets see `dtw_partial()`

Author(s)

Maximilian Leodolter

Examples

```
Q <- cumsum(rnorm(100))
C <- Q[11:100] + rnorm(90, 0, 0.5)
# the ordinary calculation
tmp <- dtw(Q = Q, C = C, ws = 15, return_wp = TRUE, return_QC = TRUE)

plotQC(tmp, Q, C)
plotWarp(tmp, Q, C)
plot(tmp, type = 'warp')
plot(tmp, Q = Q, C = C, type = 'QC')
# since return_QC = TRUE, this is also possible
plot(tmp, type = 'QC')
```

---

simulate_timewarp *Simulate time warp*

Description

Simulate a time warp for a given time series.

Usage

```
simulate_timewarp(x, stretch = 0, compress = 0,
                  stretch_method = insert_linear_interp,
                  p_index = "rnorm", p_number = "rlnorm",
                  p_index_list = NULL, p_number_list = NULL,
                  seed = NULL, ...)
```

Arguments

x	time series, vector or matrix
stretch	numeric parameter for stretching the time series. stretch >= 1, see details
compress	numeric parameter for compressing the time series. compress >= 0 & compress < 1, see details
stretch_method	function, either one of (insert_const, insert_linear_interp, insert_norm, insert_linear_norm), or any user defined function that needs the parameters x (univariate time series as vector), ix (index where to insert), N (number of observations to insert) and any other arguments required for that function. See Details.
p_index	string, distribution for simulating the indices where to insert simulated observations, e.g. "rnorm", "runif", etc.
p_number	string, distribution for simulating the number of obervations to insert per index, e.g. "rnorm", "runif", etc.
p_index_list	list of named parameters for the distribution p_index
p_number_list	list of named parameters for the distribution p_number

seed	set a seed for reproducible results
...	named parameters for the stretch_method

Details

The different distributions p_index and p_number also determine the behaviour of the warp. A uniform distribution for p_number more likely draws high number than e.g. log-normal distributions with appropriate parameters. So, a uniform distribution more likely simulates fewer, but longer warps, that is points of time where the algorithm inserts simulations.

The algorithm stretches by randomly selecting an index between 1 and the length of the time series. Then a number of observations to be inserted is drawn out of the range 1 to the remaining number of observations to be inserted. These observations are inserted. Then the algorithm starts again with drawing an index, drawing a number of observations to be inserted, and proceeds until the requested time series length is achieved.

The algorithm for compressing works analogous, except it simply omits observations instead of linear interpolation.

The parameter stretch describes the ratio how much the time series x is stretched: e.g. if compress = 0 and ...

- stretch = 0 then length(x_new) = length(x), or
- stretch = 0.1 then length(x_new) = length(x) * 1.1, or
- stretch = 1 then length(x_new) = length(x) * 2

The parameter compress describes the ratio how much the time series x is compressed: e.g. if stretch = 0 and ...

- compress = 0 then length(x_new) = length(x), or
- compress = 0.2 then length(x_new) = length(x) * 0.8

There are four functions to chose from to insert new simulated observations. You can also define yur own function and apply this one. The four functions to chose from are:

- insert a constant, either a constant defined by the user via the iniput parameter const, or if const = NULL, then the last observation of the time series where the insertion starts is set as const
- insert linear interpolated observations (default)
- insert a constant with gaussian noise
- insert linear interpolated observations and add gaussian noise.

For the methods with gaussian noise the parameters mean and sd for rnorm can be set at the function call of simulate_timewarp().

Value

time warped time series

Author(s)

Maximilian Leodolter

Examples

```
## Not run:
set.seed(123)
x <- cumsum(rnorm(100))
x_new <- simulate_timewarp(x, stretch = 0.1, compress = 0.2, seed = 123)
plot(x, type = "l")
lines(x_new, col = "red")

y <- matrix(cumsum(rnorm(10^3)), ncol = 2)


# insert NA with uniform distributions
y_warp <- simulate_timewarp(y, stretch = 0.2, p_number = "runif", p_index = "runif",
                            stretch_method = insert_const,
                            const = NA)
matplot(y_warp, type = "l")


# insert NA with log-normal distribution
y_warp <- simulate_timewarp(y, stretch = 0.2, p_number = "rlnorm",
                            p_number_list = list(meanlog = 0, sdlog = 1),
                            stretch_method = insert_const,
                            const = NA)
matplot(y_warp, type = "l")


# insert linear interpolation
y_warp <- simulate_timewarp(y, stretch = 0.2, p_number = "rlnorm",
                            stretch_method = insert_linear_interp)
matplot(y_warp, type = "l")


# insert random walk with gaussian noise
y_warp <- simulate_timewarp(y, stretch = 0.2, p_number = "rlnorm",
                            stretch_method = insert_norm,
                            sd = 1, mean = 0)
matplot(y_warp, type = "l")


# insert constant, only 1 observation per random index
y_warp <- simulate_timewarp(y, stretch = 0.2, p_number = "runif", p_index = "runif",
                              p_number_list = list(min=1, max=1),
                              stretch_method = insert_const)
matplot(y_warp, type = "l")


## End(Not run)
```

Index