Package 'dtwclust'

December 1, 2016

Type Package

Title Time Series Clustering Along with Optimizations for the Dynamic Time Warping Distance

Version 3.0.0

Date 2016-12-01

Depends R (>= 3.2.0), proxy (>= 0.4-16), clue, dtw, ggplot2

Imports methods, parallel, stats, utils, caTools, flexclust, foreach, Rcpp, reshape2, rngtools

Suggests TSdist, TSclust, cluster, doParallel, testthat, knitr

LinkingTo Rcpp

Author Alexis Sarda-Espinosa

Maintainer Alexis Sarda <alexis.sarda@gmail.com>

Description Time series clustering along with optimized techniques related to the Dynamic Time Warping distance and its corresponding lower bounds. Implementations of partitional, hierarchical, fuzzy, k-Shape and TADPole clustering are available. Functionality can be easily extended with custom distance measures and centroid definitions.

URL https://github.com/asardaes/dtwclust

BugReports https://github.com/asardaes/dtwclust/issues

License GPL-3

LazyData TRUE

NeedsCompilation yes

RoxygenNote 5.0.1

Collate 'DBA.R' 'GAK.R' 'NCCc.R' 'SBD.R' 'TADPole.R' 'all-cent.R' 'create-dtwclust.R' 'cvi.R' 'ddist.R' 'dtw-basic.R' 'dtw-lb.R' 'dtw2.R' 'dtwclust-classes.R' 'dtwclust-methods.R' 'dtwclust.R' 'fuzzy.R' 'lb-improved.R' 'lb-keogh.R' 'partitional.R' 'utils.R' 'pkg.R' 'reinterpolate.R' 'shape-extraction.R' 'uciCT.R' 'zscore.R'

VignetteBuilder knitr

Repository CRAN

Date/Publication 2016-12-01 11:28:36

R topics documented:

`dtwclust-package` *Time series clustering along with optimizations for the Dynamic Time Warping distance*

Description

Time series clustering with a wide variety of strategies and a series of optimizations specific to the Dynamic Time Warping (DTW) distance and its corresponding lower bounds (LBs). There are implementations of both traditional clustering algorithms, and more recent procedures such as k-Shape and TADPole clustering. Functionality can be easily extended with custom distance measures and centroid definitions.

Details

Many of the algorithms implemented in this package are specifically tailored to time series and DTW, hence its name. However, the main clustering function is flexible so that one can test many different clustering approaches, using either the time series directly, or by applying suitable transformations and then clustering in the resulting space.

DTW is a dynamic programming algorithm that tries to find the optimum warping path between two series. Over the years, several variations have appeared in order to make the procedure faster or more efficient. Please refer to the included references for more information, especially Giorgino (2009), which is a good practical introduction.

Most optimizations require equal dimensionality, which means time series should have equal length. DTW itself does not require this, but it is relatively expensive to compute. Other distance definitions may be used, or series could be reinterpolated to a matching length (Ratanamahatana and Keogh, 2004).

Other packages that are particularly leveraged here are the `proxy` package for distance matrix calculations and the `dtw` package for the core DTW calculations. The main clustering function and entry point for this package is `dtwclust`.

Please note the random number generator is set to L'Ecuyer-CMRG when `dtwclust` is attached in an attempt to preserve reproducibility. You are free to change this afterwards if you wish. See `RNGkind`.

For more information, please read the included package vignette, which can be accessed by typing `vignette("dtwclust")`.

Author(s)

Alexis Sarda-Espinosa

References

Please refer to the package vignette references.

See Also

`dtwclust`, `dist`, `dtw`

`as.matrix` *as.matrix*

Description

proxy exported a non-generic `as.matrix` function. This is to re-export the base version and add some coercion methods for `pairdist` and `crossdist`.

Usage

```
as.matrix(x, ...)
```

Arguments

`x, ...` See `as.matrix`.

See Also

`as.matrix`

---

clusterSim *Cluster Similarity Matrix*

Description

Returns a matrix of cluster similarities. Currently two methods for computing similarities of clusters are implemented. This generic is included in the **flexclust** package.

Usage

```
## S4 method for signature 'dtwclust'
clusterSim(object, data = NULL, method = c("shadow",
  "centers"), symmetric = FALSE, ...)
```

Arguments

`object, data, method, symmetric, ...`
See `clusterSim`.

See Also

`clusterSim`

---

create_dtwclust *Create formal* dtwclust *objects*

Description

Helper function to manually create formal `dtwclust-class` objects

Usage

```
create_dtwclust(..., override.family = TRUE)
```

Arguments

`...` Any valid slots of `dtwclust-class`.

`override.family`
Attempt to substitute the default family with one that conforms to the provided elements? See details.

Details

This function can calculate some of the slots if certain elements are provided by the user. In order to get a fully functional object at least the following slots should be provided:

- `type`: "partitional", "hierarchical", "fuzzy" or "tadpole".
- `datalist`: The data in one of the supported formats.
- `centroids`: The time series centroids in one of the supported formats.
- `cluster`: The cluster indices for each series in the `datalist`.
- `control*`: A `dtwclustControl` object (or a named list) with the desired parameters.
- `distance*`: A string indicating the distance that should be used.
- `centroid*`: A string indicating the centroid to use (only necessary for partitional clustering).

*Necessary when overriding the default family for the calculation of other slots, CVIs or prediction. Maybe not always needed, e.g. for plotting.

Value

A `dtwclust-class` object.

Examples

```
data(uciCT)

# Assuming this was generated by some clustering procedure
centroids <- CharTraj[seq(1L, 100L, 5L)]
control <- new("dtwclustControl", window.size = 8L, norm = "L2")
cluster <- unclass(CharTrajLabels)

pc_obj <- create_dtwclust(type = "partitional", datalist = CharTraj, centroids = centroids,
                          control = control, cluster = cluster,
                          distance = "sbd", centroid = "dba",
                          dots = list(step.pattern = symmetric1))

fc_obj <- create_dtwclust(type = "fuzzy", datalist = CharTraj, centroids = centroids,
                          control = control, cluster = cluster,
                          distance = "sbd", centroid = "fcm")

fc_obj
```

cvi *Cluster validity indices*

Description

Compute different cluster validity indices (CVIs) of a given cluster partition, using the clustering distance measure and centroid function if applicable.

Usage

```
cvi(a, b = NULL, type = "valid", ..., log.base = 10)

## S4 method for signature 'dtwclust'
cvi(a, b = NULL, type = "valid", ..., log.base = 10)
```

Arguments

a	An object returned by the dtwclust function, or a vector that can be coerced to integers which indicate the cluster memeberships.
b	If needed, a vector that can be coerced to integers which indicate the cluster memeberships. The ground truth (if known) should be provided here.
type	Character vector indicating which indices are to be computed. See supported values below.
...	Arguments to pass to and from other methods.
log.base	Base of the logarithm to be used in the calculation of VI.

Details

Clustering is commonly considered to be an unsupervised procedure, so evaluating its performance can be rather subjective. However, a great amount of effort has been invested in trying to standardize cluster evaluation metrics by using cluster validity indices (CVIs).

CVIs can be classified as internal, external or relative depending on how they are computed. Focusing on the first two, the crucial difference is that internal CVIs only consider the partitioned data and try to define a measure of cluster purity, whereas external CVIs compare the obtained partition to the correct one. Thus, external CVIs can only be used if the ground truth is known. Each index defines their range of values and whether they are to be minimized or maximized. In many cases, these CVIs can be used to evaluate the result of a clustering algorithm regardless of how the clustering works internally, or how the partition came to be.

Knowing which CVI will work best cannot be determined a priori, so they should be tested for each specific application. Usually, many CVIs are utilized and compared to each other, maybe using a majority vote to decide on a final result. Furthermore, it should be noted that many CVIs perform additional distance calculations when being computed, which can be very considerable if using DTW.

Note that, even though a fuzzy partition can be changed into a crisp one, making it compatible with many of the existing CVIs, there are also fuzzy CVIs tailored specifically to fuzzy clustering, and these may be more suitable in those situations, but have not been implemented here yet.

Value

The chosen CVIs

External CVIs

The first 4 CVIs are calculated via `comPart`, so please refer to that function.

- `"RI"`: Rand Index (to be maximized).
- `"ARI"`: Adjusted Rand Index (to be maximized).
- `"J"`: Jaccard Index (to be maximized).
- `"FM"`: Fowlkes-Mallows (to be maximized).
- `"VI"`: Variation of Information (Meila (2003); to be minimized).

Internal CVIs

The indices marked with an exclamation mark (!) calculate (or re-use if already available) the whole distance matrix between the series in the data. If you were trying to avoid this in the first place, then these CVIs might not be suitable for your application.

The indices marked with a question mark (?) depend on the extracted centroids, so bear that in mind if a hierarchical procedure was used and/or the centroid function has associated randomness (such as `shape_extraction` with series of different length).

The indices marked with a tilde (~) require the calculation of a global centroid. Since `DBA` and `shape_extraction` (for series of different length) have some randomness associated, these indices might not be appropriate for those centroids.

- `"Sil"` (!): Silhouette index (Arbelaitz et al. (2013); to be maximized).
- `"D"` (!): Dunn index (Arbelaitz et al. (2013); to be maximized).
- `"COP"` (!): COP index (Arbelaitz et al. (2013); to be minimized).
- `"DB"` (?): Davies-Bouldin index (Arbelaitz et al. (2013); to be minimized).
- `"DBstar"` (?): Modified Davies-Bouldin index (DB*) (Kim and Ramakrishna (2005); to be minimized).
- `"CH"` (~): Calinski-Harabasz index (Arbelaitz et al. (2013); to be maximized).
- `"SF"` (~): Score Function (Saitta et al. (2007); to be maximized).

Additionally

- `"valid"`: Returns all valid indices depending on the type of `a` and whether `b` was provided or not.
- `"internal"`: Returns all internal CVIs. Only supported for `dtwclust-class` objects.
- `"external"`: Returns all external CVIs. Requires `b` to be provided.

Note

In the original definition of many internal CVIs, the Euclidean distance and a mean centroid was used. The implementations here change this, making use of whatever distance/centroid was chosen during clustering.

Some internal indices require the original data for calculations, so the control flag `save.data` must be set to `TRUE` when running the clustering algorithm.

References

Arbelaitz, O., Gurrutxaga, I., Muguerza, J., Perez, J. M., & Perona, I. (2013). An extensive comparative study of cluster validity indices. Pattern Recognition, 46(1), 243-256.

Kim, M., & Ramakrishna, R. S. (2005). New indices for cluster validity assessment. Pattern Recognition Letters, 26(15), 2353-2363.

Meila, M. (2003). Comparing clusterings by the variation of information. In Learning theory and kernel machines (pp. 173-187). Springer Berlin Heidelberg.

Saitta, S., Raphael, B., & Smith, I. F. (2007). A bounded index for cluster validity. In International Workshop on Machine Learning and Data Mining in Pattern Recognition (pp. 174-187). Springer Berlin Heidelberg.

DBA *DTW Barycenter Averaging*

Description

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

Usage

```
DBA(X, centroid = NULL, ..., window.size = NULL, norm = "L1",
  max.iter = 20L, delta = 0.001, error.check = TRUE, trace = FALSE)
```

Arguments

`X`	A data matrix where each row is a time series, or a list where each element is a time series. Multivariate series should be provided as a list of matrices where time spans the rows and the variables span the columns.
`centroid`	Optionally, a time series to use as reference. Defaults to a random series of `X` if `NULL`. For multivariate series, this should be a matrix with the same characteristics as the matrices in `X`.
`...`	Currently ignored.
`window.size`	Window constraint for the DTW calculations. `NULL` means no constraint. A slanted band is used by default.
`norm`	Norm for the local cost matrix of DTW. Either "L1" for Manhattan distance or "L2" for Euclidean distance.

max.iter	Maximum number of iterations allowed.
delta	At iteration i, if all(abs(centroid_{i} - centroid_{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 centroid, the algorithm should always converge to the same result provided the elements of X keep the same values, although their order may change.

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 average time series.

Parallel Computing

Please note that running tasks in parallel does **not** guarantee faster computations. The overhead introduced is sometimes too large, and it's better to run tasks sequentially.

The user can register a parallel backend, e.g. with the doParallel package, in order to attempt to speed up the calculations (see the examples).

References

Petitjean F, Ketterlin A and Gancarski P (2011). "A global averaging method for dynamic time warping, with applications to clustering." *Pattern Recognition*, **44**(3), pp. 678 - 693. ISSN 0031-3203, http://dx.doi.org/10.1016/j.patcog.2010.09.013, http://www.sciencedirect.com/science/article/pii/S003132031000453X.

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
matplot(do.call(cbind, CharTraj[1:5]), type = "l")
points(dtw.avg)
```

```
# Change the provided order
dtw.avg2 <- DBA(CharTraj[5:1], CharTraj[[1]], trace = TRUE)

# Same result?
all(dtw.avg == dtw.avg2)

## Not run:
#### Running DBA with parallel support
# For such a small dataset, this is probably slower in parallel
require(doParallel)

# Create parallel workers
cl <- makeCluster(detectCores())
invisible(clusterEvalQ(cl, library(dtwclust)))
registerDoParallel(cl)

# DTW Average
cen <- DBA(CharTraj[1:5], CharTraj[[1]], trace = TRUE)

# Stop parallel workers
stopCluster(cl)

# Return to sequential computations
registerDoSEQ()

## End(Not run)
```

---

dtw2 *DTW distance with L2 norm*

Description

Wrapper for the dtw function using L2 norm for both the local cost matrix (LCM) creation as well as the final cost aggregation step.

Usage

```
dtw2(x, y, ...)
```

Arguments

x, y	A time series. A multivariate series should have time spanning the rows and variables spanning the columns.
...	Further arguments for dtw.

Details

The L-norms are used in two different steps by the DTW algorithm. First when creating the LCM, where the element (i, j) of the matrix is computed as the L-norm of $x_i^v - y_j^v$ for all variables v. Note that this means that, in case of multivariate series, they must have the same number of variables, and that univariate series will produce the same LCM regardless of the L-norm used. After the warping path is found by DTW, the final distance is calculated as the L-norm of all (i, j) elements of the LCM that fall on the warping path.

The `dtw` function allows changing the norm by means of its `dist.method` parameter, but it only uses it when creating the LCM, and not when calculating the final aggregated cost, i.e. the DTW distance.

This wrapper simply returns the appropriate DTW distance using L2 norm (Euclidean norm).

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

An object of class `dtw`.

`dtwclust` *Time series clustering*

Description

This is the main function to perform time series clustering. It supports partitional, hierarchical, fuzzy, k-Shape and TADPole clustering. See the details and the examples for more information, as well as the included package vignette (which can be loaded by typing `vignette("dtwclust")`).

Usage

```
dtwclust(data = NULL, type = "partitional", k = 2L, method = "average",
  distance = "dtw_basic", centroid = "pam", preproc = NULL, dc = NULL,
  control = NULL, seed = NULL, distmat = NULL, ...)
```

Arguments

`data`	A list of series, a numeric matrix or a data frame. Matrices and data frames are coerced row-wise.
`type`	What type of clustering method to use: `"partitional"`, `"hierarchical"`, `"tadpole"` or `"fuzzy"`.
`k`	Number of desired clusters. It may be a numeric vector with different values.
`method`	Character vector with one or more linkage methods to use in hierarchical procedures (see `hclust`) or a function that performs hierarchical clustering based on distance matrices (e.g. `diana`). See Hierarchical section for more details.

distance	A supported distance from proxy's dist (see Distance section). Ignored for type = "tadpole".
centroid	Either a supported string or an appropriate function to calculate centroids when using partitional or prototypes for hierarchical/tadpole 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.
dc	Cutoff distance for the TADPole algorithm.
control	Named list of parameters or dtwclustControl object for clustering algorithms. See dtwclustControl. NULL means defaults.
seed	Random seed for reproducibility.
distmat	If a cross-distance matrix is already available, it can be provided here so it's re-used. Only relevant if centroid = "pam" or type = "hierarchical". See examples.
...	Additional arguments to pass to dist or a custom function (preprocessing, centroid, etc.)

Details

Partitional and fuzzy clustering procedures use a custom implementation. Hierarchical clustering is done with hclust. TADPole clustering uses the TADPole function. Specifying type = "partitional", distance = "sbd" and centroid = "shape" is equivalent to the k-Shape algorithm (Paparrizos and Gravano, 2015).

The data may be a matrix, a data frame or a list. Matrices and data frames are coerced to a list, both row-wise. Only lists can have series with different lengths or multiple dimensions. Most of the optimizations require series to have the same length, so consider reinterpolating them to save some time (see Ratanamahatana and Keogh, 2004; reinterpolate). No missing values are allowed.

In the case of multivariate time series, they should be provided as a list of matrices, where time spans the rows of each matrix and the dimensions span the columns. At the moment, only DTW and DTW2 suppport such series, which means only partitional and hierarchical procedures using those distances will work. You can of course create your own custom distances. All included centroid functions should work with the aforementioned format, although shape is **not** recommended. Note that the plot method will simply append all dimensions (columns) one after the other.

Several parameters can be adjusted with the control argument. See dtwclustControl. In the following sections, elements marked with an asterisk (*) are those that can be adjutsed with this argument.

Value

An object with formal class dtwclust-class.

If control@nrep > 1 and a partitional procedure is used, length(method) > 1 and hierarchical procedures are used, or length(k) > 1, a list of objects is returned.

Partitional Clustering

Stochastic algorithm that creates a hard partition of the data into k clusters, where each cluster has a centroid. In case of time series clustering, the centroids are also time series.

The cluster centroids are first randomly initialized by selecting some of the series in the data. The distance between each series and each centroid is calculated, and the series are assigned to the cluster whose centroid is closest. The centroids are then updated by using a given rule, and the procedure is repeated until no series changes from one cluster to another, or until the maximum number of iterations* has been reached. The distance and centroid definitions can be specified through the corresponding parameters of this function. See their respective sections below.

Note that it is possible for a cluster to become empty, in which case a new cluster is reinitialized randomly. However, if you see that the algorithm doesn't converge or the overall distance sum increases, it could mean that the chosen value of k is too large, or the chosen distance measure is not able to assess similarity effectively. The random reinitialization attempts to enforce a certain number of clusters, but can result in instability in the aforementioned cases.

Fuzzy Clustering

This procedure is very similar to partitional clustering, except that each series no longer belongs exclusively to one cluster, but belongs to each cluster to a certain degree. For each series, the total degree of membership across clusters must sum to 1.

The default implementation uses the fuzzy c-means algorithm. In its definition, an objective function is to be minimized. The objective is defined in terms of a squared distance, which is usually the Euclidean (L2) distance, although the definition could be modified. The `distance` parameter of this function controls the one that is utilized. The fuzziness of the clustering can be controlled by means of the fuzziness exponent*. Bear in mind that the centroid definition of fuzzy c-means requires equal dimensions, which means that all series must have the same length. This problem can be circumvented by applying transformations to the series (see for example D'Urso and Maharaj, 2009).

Note that the fuzzy clustering could be transformed to a crisp one by finding the highest membership coefficient. Some of the slots of the object returned by this function assume this, so be careful with interpretation (see `dtwclust-class`).

Hierarchical Clustering

This is (by default) a deterministic algorithm that creates a hierarchy of groups by using different linkage methods (see `hclust`). The linkage method is controlled through the `method` parameter of this function, which can be a character vector with several methods, with the additional option "all" that uses all of the available methods in `hclust`. The distance to be used can be controlled with the `distance` parameter.

Optionally, `method` may be a **function** that performs the hierarchical clustering based on a distance matrix, such as the functions included in package **cluster**. The function will receive the `dist` object as first argument (see `as.dist`), followed by the elements in `...` that match the its formal arguments. The object it returns must support the `as.hclust` generic so that `cutree` can be used. See the examples.

The hierarchy does not imply a specific number of clusters, but one can be induced by cutting the resulting dendrogram (see `cutree`). This results in a crisp partition, and some of the slots of the returned object are calculated by cutting the dendrogram so that k clusters are created.

TADPole Clustering

TADPole clustering adopts a relatively new clustering framework and adapts it to time series clustering with DTW. Because of the way it works, it can be considered a kind of Partitioning Around Medoids (PAM). This means that the cluster centroids are always elements of the data. However, this algorithm is deterministic, depending on the value of the cutoff distance `dc`, which can be controlled with the corresponding parameter of this function.

The algorithm relies on the DTW lower bounds, which are only defined for time series of equal length. Additionally, it requires a window constraint* for DTW. See the Sakoe-Chiba constraint section below. Unlike the other algorithms, TADPole always uses DTW2 as distance (with a symmetric1 step pattern).

Centroid Calculation

In the case of partitional/fuzzy algorithms, a suitable function should calculate the cluster centroids at every iteration. In this case, the centroids are themselves time series. Fuzzy clustering uses the standard fuzzy c-means centroid by default.

In either case, a custom function can be provided. If one is provided, it will receive the following parameters with the shown names (examples for partitional clustering are shown in parenthesis):

- `"x"`: The *whole* data list (`list(ts1, ts2, ts3)`)
- `"cl_id"`: A numeric vector with length equal to the number of series in `data`, indicating which cluster a series belongs to (`c(1L, 2L, 2L)`)
- `"k"`: The desired number of total clusters (`2L`)
- `"cent"`: The current centroids in order, in a list (`list(centroid1, centroid2)`)
- `"cl_old"`: The membership vector of the *previous* iteration (`c(1L, 1L, 2L)`)
- The elements of `...` that match its formal arguments

In case of fuzzy clustering, the membership vectors (2nd and 5th elements above) are matrices with number of rows equal to amount of elements in the data, and number of columns equal to the number of desired clusters. Each row must sum to 1.

The other option is to provide a character string for the custom implementations. The following options are available:

- "mean": The average along each dimension. In other words, the average of all x_i^j 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. By default, all series are z-normalized in this case, since the resulting centroids will also have this normalization. See `shape_extraction` for more details.
- "dba": DTW Barycenter Averaging. See `DBA` for more details.
- "pam": Partition around medoids (PAM). This basically means that the cluster centroids are always one of the time series in the data. In this case, the distance matrix can be pre-computed once using all time series in the data and then re-used at each iteration. It usually saves overhead overall.
- "fcm": Fuzzy c-means. Only supported for fuzzy clustering and always used for that type of clustering if a string is provided in `centroid`.

These check for the special cases where parallelization might be desired. Note that only shape, dba and pam support series of different length. Also note that, for shape and dba, this support has a caveat: the final centroids' length will depend on the length of those series that were randomly chosen at the beginning of the clustering algorithm. For example, if the series in the dataset have a length of either 10 or 15, 2 clusters are desired, and the initial choice selects two series with length of 10, the final centroids will have this same length.

As special cases, if hierarchical or tadpole clustering is used, you can provide a centroid function that takes a list of series as only input and returns a single centroid series. These centroids are returned in the centroids slot. By default, a type of PAM centroid function is used.

Distance Measures

The distance measure to be used with partitional, hierarchical and fuzzy clustering can be modified with the distance parameter. The supported option is to provide a string, which must represent a compatible distance registered with proxy's dist. Registration is done via pr_DB, and extra parameters can be provided in ....

Note that you are free to create your own distance functions and register them. Optionally, you can use one of the following custom implementations (all registered with proxy):

- "dtw": DTW, optionally with a Sakoe-Chiba/Slanted-band constraint*.
- "dtw2": DTW with L2 norm and optionally a Sakoe-Chiba/Slanted-band constraint*. Read details below.
- "dtw_basic": A custom version of DTW with less functionality, but slightly faster. See dtw_basic.
- "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 dtw_lb. Not suitable for pam.precompute* = TRUE.
- "lbk": Keogh's lower bound with either L1 or L2 norm* for the Sakoe-Chiba constraint*.
- "lbi": Lemire's lower bound with either L1 or L2 norm* for the Sakoe-Chiba constraint*.
- "sbd": Shape-based distance. See SBD for more details.
- "gak": Global alignment kernels. See GAK for more details.

DTW2 is done with dtw, 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. See dtw2.

Only dtw, dtw2 and sbd support series of different length. The lower bounds are probably unsuitable for direct clustering unless series are very easily distinguishable.

If you create your own distance, register it with proxy, and it includes the ellipsis (...) in its definition, it will receive the following parameters*:

- window.type: Either "none" for a NULL window.size, or "slantedband" otherwise
- window.size: The provided window size
- norm: The provided desired norm

- . . . : Any additional parameters provided in the original call's ellipsis

Whether your function makes use of them or not, is up to you.

If you know that the distance function is symmetric, and you use a hierarchical algorithm, or a partitional algorithm with PAM centroids and `pam.precompute* = TRUE`, some time can be saved by calculating only half the distance matrix. Therefore, consider setting the symmetric* control parameter to `TRUE` if this is the case.

Sakoe-Chiba Constraint

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

The windowing constraint uses a centered window. The function expects 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.

The computations actually use a `slantedband` window, which is equivalent to the Sakoe-Chiba one if series have equal length, and stays along the diagonal of the local cost matrix if series have different length.

Preprocessing

It is strongly advised to use z-normalization in case of `centroid = "shape"`, because the resulting series have this normalization (see `shape_extraction`). Therefore, `zscore` is the default in this case. 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 list of time series.

Setting to `NULL` means no preprocessing (except for `centroid = "shape"`). A provided function will receive the data as first argument, followed by the contents of `...` that match its formal arguments.

It is convenient to provide this function if you're planning on using the `predict` generic.

Repetitions

Due to their stochastic nature, partitional clustering is usually repeated* several times with different random seeds to allow for different starting points. This function uses `RNGseq` to obtain different seed streams for each repetition, utilizing the `seed` parameter (if provided) to initialize it. If more than one repetition is made, the streams are returned in an attribute called `rng`.

Technically, you can also perform random repetitions for fuzzy clustering, although it might be difficult to evaluate the results, since they are usually evaluated relative to each other and not in an absolute way. Ideally, the groups wouldn't change too much once the algorithm converges.

Multiple values of `k` can also be provided to get different partitions using any `type` of clustering.

Repetitions are greatly optimized when PAM centroids are used and the whole distance matrix is precomputed*, since said matrix is reused for every repetition, and can be comptued in parallel (see Parallel section).

Parallel Computing

Please note that running tasks in parallel does **not** guarantee faster computations. The overhead introduced is sometimes too large, and it's better to run tasks sequentially.

The user can register a parallel backend, for eample with the `doParallel` package, in order to do the repetitions in parallel, as well as distance and some centroid calculations.

Unless each repetition requires a few seconds, parallel computing probably isn't worth it. As such, I would only use this feature with `shape` and `DBA` centroids, or an expensive distance function like `DTW`.

If you register a parallel backend, the function will also try to do the calculation of the distance matrices in parallel. This should work with any function registered with `dist` via `pr_DB` whose `loop` flag is set to `TRUE`. If the function requires special packages to be loaded, provide their names in the `packages*` slot of `control`. Note that "dtwclust" is always loaded in each parallel worker, so that doesn't need to be included. Alternatively, you may want to pre-load `dtwclust` in each worker with `clusterEvalQ`.

In case of multiple repetitions, each worker gets a repetition task. Otherwise, the tasks (which can be a distance matrix or a centroid calculation) are usually divided into chunks according to the number of workers available.

Notes

The lower bounds are defined only for time series of equal length. `DTW` and `DTW2` don't require this, but they are much slower to compute.

The lower bounds are **not** symmetric, and `DTW` is not symmetric in general.

Author(s)

Alexis Sarda-Espinosa

References

Please refer to the package vignette references.

See Also

`dtwclust-methods`, `dtwclust-class`, `dtwclustControl`, `dtwclustFamily`.

Examples

```
#' NOTE: More examples are available in the vignette. Here are just some miscellaneous
#' examples that might come in handy. They should all work, but some don't run
#' automatically.

# Load data
data(uciCT)

# ================================================================================
# Simple partitional clustering with Euclidean distance and PAM centroids
# ================================================================================
```

```
# Reinterpolate to same length
series <- reinterpolate(CharTraj, new.length = max(lengths(CharTraj)))

# Subset for speed
series <- series[1:20]
labels <- CharTrajLabels[1:20]

# Making many repetitions
pc.l2 <- dtwclust(series, k = 4L,
                  distance = "L2", centroid = "pam",
                  seed = 3247, control = list(trace = TRUE,
                                              nrep = 10L))

# Cluster validity indices
sapply(pc.l2, cvi, b = labels)

# ============================================================================
# Hierarchical clustering with Euclidean distance
# ============================================================================

# Re-use the distance matrix from the previous example (all matrices are the same)
# Use all available linkage methods for function hclust
hc.l2 <- dtwclust(series, type = "hierarchical",
                  k = 4L, method = "all",
                  control = list(trace = TRUE),
                  distmat = pc.l2[[1L]]@distmat)

# Plot the best dendrogram according to variation of information
plot(hc.l2[[which.min(sapply(hc.l2, cvi, b = labels, type = "VI"))]])

# ============================================================================
# Multivariate time series
# ============================================================================

# Multivariate series, provided as a list of matrices
mv <- CharTrajMV[1L:20L]

# Using GAK distance
mvc <- dtwclust(mv, k = 4L, distance = "gak", seed = 390)

# Note how the variables of each series are appended one after the other in the plot
plot(mvc)

## Not run:
# Common controls
ctrl <- new("dtwclustControl", trace = TRUE, window.size = 18L)

# ============================================================================
# Registering a custom distance with the 'proxy' package and using it
# ============================================================================

# Normalized asymmetric DTW distance
```

```
ndtw <- function(x, y, ...) {
    dtw::dtw(x, y, step.pattern = asymmetric,
             distance.only = TRUE, ...)$normalizedDistance
}

# Registering the function with 'proxy'
if (!pr_DB$entry_exists("nDTW"))
    proxy::pr_DB$set_entry(FUN = ndtw, names=c("nDTW"),
                           loop = TRUE, type = "metric", distance = TRUE,
                           description = "Normalized asymmetric DTW")

# Subset of (original) data for speed
pc.ndtw <- dtwclust(series[-1L], k = 4L,
                    distance = "nDTW",
                    seed = 8319,
                    control = ctrl)

# Which cluster would the first series belong to?
# Notice that newdata is provided as a list
predict(pc.ndtw, newdata = series[1L])

# ============================================================================
# Custom hierarchical clustering
# ============================================================================

require(cluster)

hc.diana <- dtwclust(series, type = "h", k = 4L,
                     distance = "L2", method = diana,
                     control = ctrl)

plot(hc.diana, type = "sc")

# ============================================================================
# TADPole clustering
# ============================================================================

pc.tadp <- dtwclust(series, type = "tadpole", k = 4L,
                    dc = 1.5, control = ctrl)

# Modify plot, show only clusters 3 and 4
plot(pc.tadp, clus = 3:4,
     labs.arg = list(title = "TADPole, clusters 3 and 4",
                     x = "time", y = "series"))

# Saving and modifying the ggplot object with custom time labels
require(scales)
t <- seq(Sys.Date(), len = length(series[[1L]]), by = "day")
gpc <- plot(pc.tadp, time = t, plot = FALSE)

gpc + scale_x_date(labels = date_format("%b-%Y"),
                   breaks = date_breaks("2 months"))
```

```
# ============================================================================================
# Specifying a centroid function for prototype extraction in hierarchical clustering
# ============================================================================================

# Seed is due to possible randomness in shape_extraction when selecting a basis series
hc.sbd <- dtwclust(CharTraj, type = "hierarchical",
                   k = 20L, distance = "sbd",
                   preproc = zscore, centroid = shape_extraction,
                   seed = 320L)

plot(hc.sbd, type = "sc")

# ============================================================================================
# Using parallel computation to optimize several random repetitions
# and distance matrix calculation
# ============================================================================================
require(doParallel)

# Create parallel workers
cl <- makeCluster(detectCores())
invisible(clusterEvalQ(cl, library(dtwclust)))
registerDoParallel(cl)

## Use constrained DTW and PAM (less than a second with 8 cores)
pc.dtw <- dtwclust(CharTraj, k = 20L, seed = 3251, control = ctrl)

## Use constrained DTW with DBA centroids (~3 seconds with 8 cores)
pc.dba <- dtwclust(CharTraj, k = 20L, centroid = "dba", seed = 3251, control = ctrl)

#' Using distance based on global alignment kernels
#' (~35 seconds with 8 cores for all repetitions)
pc.gak <- dtwclust(CharTraj, k = 20L,
                   distance = "gak",
                   centroid = "dba",
                   seed = 8319,
                   control = list(trace = TRUE,
                                  window.size = 18L,
                                  nrep = 8L))

# Stop parallel workers
stopCluster(cl)

# Return to sequential computations. This MUST be done after stopCluster()
registerDoSEQ()

## End(Not run)
```

---

dtwclust-class *Class definition for* dtwclust

Description

Formal S4 class.

Details

This class contains `hclust` as superclass and supports all its methods. Plot is a special case (see `dtwclust-methods`).

Please note that not all slots will contain valid information for all clustering types. In some cases, for example for fuzzy and hierarchical clustering, some results are computed assuming a hard partition is created based on the fuzzy memberships or dendrogram tree, and the provided value of k.

Slots

`call` The function call.

`control` An object of class `dtwclustControl`.

`family` An object of class `dtwclustFamily`.

`distmat` If computed, the cross-distance matrix.

`k` Integer indicating the number of desired clusters.

`cluster` Integer vector indicating which cluster a series belongs to (crisp partition).

`fcluster` Numeric matrix that contains membership of fuzzy clusters. It has one row for each series and one column for each cluster. The rows must sum to 1. Only relevant for fuzzy clustering.

`iter` The number of iterations used.

`converged` A logical indicating whether the function converged.

`clusinfo` A data frame with two columns: `size` indicates the number of series each cluster has, and `av_dist` indicates, for each cluster, the average distance between series and their respective centroids (crisp partition).

`centroids` A list with the centroid time series.

`cldist` A column vector with the distance between each series in the data and its corresponding centroid (crisp partition).

`type` A string indicating one of the supported clustering types of `dtwclust`.

`method` A string indicating which hierarchical method was used.

`distance` A string indicating the distance used.

`centroid` A string indicating the centroid used.

`preproc` A string indicating the preprocessing used.

`datalist` The provided data in the form of a list, where each element is a time series.

`proctime` Time during function execution, as measured with `proc.time`.

`dots` The contents of the original call's ellipsis (...).

dtwclust-methods *Methods for* dtwclust

Description

Methods associated with dtwclust-class objects.

Usage

```
## S4 method for signature 'dtwclust'
show(object)

## S3 method for class 'dtwclust'
update(object, ..., evaluate = TRUE)

## S4 method for signature 'dtwclust'
update(object, ..., evaluate = TRUE)

## S3 method for class 'dtwclust'
predict(object, newdata = NULL, ...)

## S4 method for signature 'dtwclust'
predict(object, newdata = NULL, ...)

## S3 method for class 'dtwclust'
plot(x, y, ..., clus = seq_len(x@k), labs.arg = NULL,
  data = NULL, time = NULL, plot = TRUE, type = NULL)

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

Arguments

object, x	An object of class dtwclust-class as returned by dtwclust.
...	For plot, further arguments to pass to geom_line for the plotting of the *cluster centroids*, or to plot.hclust. See details. For update, any supported argument. Otherwise ignored.
evaluate	Logical. Defaults to TRUE and evaluates the updated call, which will result in a new dtwclust object. Otherwise, it returns the unevaluated call.
newdata	New data to be assigned to a cluster. It can take any of the supported formats of dtwclust. Note that for multivariate series, this means that it **must** be a list of matrices, even if the list has only one element.
y	Ignored.
clus	A numeric vector indicating which clusters to plot.

labs.arg	Arguments to change the title and/or axis labels. See labs for more information
data	Optionally, 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	Logical flag. You can set this to FALSE in case you want to save the ggplot object without printing anything to screen
type	What to plot. NULL means default. See details.

Details

Supported generics from the flexclust package are: randIndex and clusterSim.

All generics from package **clue** are also supported in order to use its functions.

Show method displays basic information from the clustering results.

The update method takes the original function call, replaces any provided argument and optionally evaluates the call again. Use evaluate = FALSE if you want to get the unevaluated call.

The predict generic can take the usual newdata argument and it returns the cluster(s) to which the data belongs; if NULL, it simply returns the obtained cluster indices. It preprocesses the data with the corresponding function if available.

Value

The plot method returns a gg object (or NULL for dendrogram plot) invisibly.

Plotting

The plot method uses the ggplot2 plotting system (see ggplot).

The default depends on whether a hierarchical method was used or not. In those cases, the dendrogram is plotted by default; you can pass any extra parameters to plot.hclust via ....

Otherwise, the function plots the time series of each cluster along with the obtained centroid. The default values for cluster centroids are: linetype = "dashed", size = 1.5, colour = "black", alpha = 0.5. You can change this by means of ....

You can choose what to plot with the type parameter. Possible options are:

- "dendrogram": Only available for hierarchical clustering.
- "series": Plot the time series divided into clusters without including centroids.
- "centroids": Plot the obtained centroids only.
- "sc": Plot both series and centroids

The flag save.data should be set to TRUE when running dtwclust to be able to use this. Optionally, you can manually provide the data in the data parameter.

If created, 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.

If you want to free the scale of the X axis, you can do the following:

```
plot(object, plot = FALSE) + facet_wrap(~cl, scales = "free")
```

See Also

dtwclust-class, dtwclust, ggplot, cvi

`dtwclustControl-class` *Class definition for* `dtwclustControl`

Description

Formal S4 class with several control parameters used in dtwclust.

Details

Default values are shown at the end.

Slots

`window.size` Integer or `NULL`. Window constraint for GAK, DTW, DBA and LB calculations. `NULL` means no constraint.

`norm` Character. Pointwise distance for DTW, DBA and the LBs. Either `"L1"` for Manhattan distance or `"L2"` for Euclidean. Ignored for `distance = "DTW2"` (which always uses `"L2"`).

`delta` Numeric. Convergence criterion for DBA centroids and fuzzy clustering.

`trace` Logical flag. If `TRUE`, more output regarding the progress is printed to screen.

`save.data` Return a "copy" of the data in the returned object? Because of the way `R` handles things internally, all copies should point to the same memory address.

`symmetric` Logical flag. Is the distance function symmetric? In other words, is `dist(x,y) == dist(y,x)`? If `TRUE`, only half the distance matrix needs to be computed. Only relevant for PAM centroids and hierarchical clustering. Overridden if the function detects an invalid user-provided value.

`packages` Character vector with the names of any packages required for custom `proxy` functions. See Parallel Computing section in dtwclust. Since the distance entries are re-registered in each parallel worker if needed, this slot is probably useless, but just in case.

`dba.iter` Integer. Maximum number of iterations for DBA centroids.

`pam.precompute` Logical flag. Precompute the whole distance matrix once and reuse it on each iteration if using PAM centroids. Otherwise calculate distances at every iteration.

`fuzziness` Numeric. Exponent used for fuzzy clustering. Commonly termed `m` in the literature.

`iter.max` Integer. Maximum number of allowed iterations for partitional/fuzzy clustering.

`nrep` Integer. How many times to repeat clustering with different starting points. See section Repetitions in dtwclust.

Common parameters

- window.size = NULL
- norm = "L1"
- delta = 1e-3
- trace = FALSE
- save.data = TRUE
- symmetric = FALSE
- packages = character(0)

Only for partitional procedures

- dba.iter = 15L
- pam.precompute = TRUE

Only for fuzzy clustering

- fuzziness = 2

For both partitional and fuzzy

- iter.max = 100L
- nrep = 1L

---

dtwclustFamily-class *Class definition for* dtwclustFamily

Description

Formal S4 class with a family of functions used in dtwclust.

Details

The custom implementations also handle parallelization.

Since the distance function makes use of proxy, it also supports any extra dist parameters in ....

The prototype includes the cluster function for partitional methods, as well as a pass-through preproc function.

Slots

dist The function to calculate the distance matrices.

allcent The function to calculate centroids on each iteration.

cluster The function used to assign a series to a cluster.

preproc The function used to preprocess the data (relevant for predict).

Examples

```
# The dist() function in dtwclustFamily works like proxy::dist() but supports
# parallelization and optimized symmetric calculations. If you like, you can
# use the function more or less directly, but provide a control argument when
# creating the family.

## Not run:
data(uciCT)
ctrl <- new("dtwclustControl", window.size = 18L, symmetric = TRUE)
fam <- new("dtwclustFamily", dist = "gak",
           control = list(symmetric = TRUE, window.size = 18L))
fam@dist(CharTraj)

## End(Not run)

# If you want the fuzzy family, use fuzzy = TRUE
ffam <- new("dtwclustFamily", control = new("dtwclustControl"), fuzzy = TRUE)
```

---

dtw_basic *Basic DTW distance*

Description

This is a custom implementation of the DTW algorithm without all the functionality included in dtw. Because of that, it should be slightly faster, while still supporting the most common options.

Usage

```
dtw_basic(x, y, window.size = NULL, norm = "L1",
  step.pattern = symmetric2, backtrack = FALSE, normalize = FALSE, ...,
  gcm = NULL, error.check = TRUE)
```

Arguments

x, y	Time series. Multivariate series must have time spanning the rows and variables spanning the columns.
window.size	Size for slanted band window. NULL means no constraint.
norm	Norm for the DTW calculation, "L1" for Manhattan or "L2" for Euclidean.
step.pattern	Step pattern for DTW. Only symmetric1 or symmetric2 supported here. See stepPattern.
backtrack	Also compute the warping path between series? See details.
normalize	Should the distance be normalized? Only supported for symmetric2.
...	Currently ignored.

gcm	Optionally, a matrix with NROW(x)+1 rows and NROW(y)+1 columns to use for the global cost matrix calculations. Used internally for memory optimization. If provided, it **will** be modified *in place* by C code, except in the parallel version in proxy::dist which ignores it for thread-safe reasons.
error.check	Check data inconsistencies?

Details

If backtrack is TRUE, the mapping of indices between series is returned in a list.

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 DTW distance. For backtrack = TRUE, a list with:

- distance: The DTW distance.
- index1: x indices for the matched elements in the warping path.
- index2: y indices for the matched elements in the warping path.

Note

The DTW algorithm (and the functions that depend on it) might return different values in 32 bit installations compared to 64 bit ones.

---

dtw_lb *DTW distance matrix guided by Lemire's improved lower bound*

Description

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

Usage

```
dtw_lb(x, y = NULL, window.size = NULL, norm = "L1", error.check = TRUE,
  pairwise = FALSE, dtw.func = "dtw_basic", force.symmetry = FALSE, ...)
```

Arguments

x, y	A matrix where rows are time series, or a list of time series.
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?
pairwise	Calculate pairwise distances?
dtw.func	Which function to use for core DTW the calculations, either "dtw" or "dtw_basic". See dtw and dtw_basic.
force.symmetry	Force symmetry of the distance matrix. Only supported for y = NULL.
...	Further arguments for dtw.func or lb_improved.

Details

This function first calculates an initial estimate of a distance matrix between two sets of time series using LB_Improved. Afterwards, it uses the estimate to calculate the corresponding true DTW distance between *only* the nearest neighbors of each series in x found in y, and it continues iteratively until no changes in the nearest neighbors occur.

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 within 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.

Parallel Computing

Please note that running tasks in parallel does **not** guarantee faster computations. The overhead introduced is sometimes too large, and it's better to run tasks sequentially.

The user can register a parallel backend, e.g. with the doParallel package, in order to attempt to speed up the calculations (see the examples).

Note

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

Author(s)

Alexis Sarda-Espinosa

References

Lemire D (2009). "Faster retrieval with a two-pass dynamic-time-warping lower bound ." *Pattern Recognition*, **42**(9), pp. 2169 - 2180. ISSN 0031-3203, http://dx.doi.org/10.1016/j.patcog.2008.11.030, http://www.sciencedirect.com/science/article/pii/S0031320308004925.

See Also

lb_keogh, lb_improved

Examples

```
# Load data
data(uciCT)

# Reinterpolate to same length
data <- reinterpolate(CharTraj, new.length = max(lengths(CharTraj)))

# 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 three 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)

## Not run:
#### Running DTW_LB with parallel support
# For such a small dataset, this is probably slower in parallel
require(doParallel)

# Create parallel workers
cl <- makeCluster(detectCores())
invisible(clusterEvalQ(cl, library(dtwclust)))
registerDoParallel(cl)

# Distance matrix
D <- dtw_lb(data[1:50], data[51:100], window.size = 20)

# Stop parallel workers
stopCluster(cl)

# Return to sequential computations
registerDoSEQ()
```

```
# Nearest neighbors
NN <- apply(D, 1, which.min)
cbind(names(data[1:50]), names(data[51:100][NN]))

## End(Not run)
```

GAK *Fast global alignment kernels*

Description

Distance based on (triangular) global alignment kernels.

Usage

```
GAK(x, y, ..., sigma = NULL, window.size = NULL, normalize = TRUE,
  logs = NULL, error.check = TRUE)
```

Arguments

x, y	Time series. A multivariate series should have time spanning the rows and variables spanning the columns.
...	Currently ignored.
sigma	Parameter for the Gaussian kernel's width. See details for the interpretation of NULL.
window.size	Parameterization of the constraining band (T in Cuturi (2011)). See details.
normalize	Normalize the result by considering diagonal terms.
logs	Optionally, a matrix with max(NROW(x), NROW(y)) + 1 rows and 3 columns to use for the logarithm calculations. Used internally for memory optimization. If provided, it **will** be modified *in place* by C code, except in the parallel version in proxy::dist which ignores it for thread-safe reasons.
error.check	Check data inconsistencies?

Details

This function uses the Triangular Global Alignment Kernel (TGAK) described in Cuturi (2011). It supports series of different length and multivariate series, so long as the ratio of the series' lengths don't differ by more than 2 (or less than 0.5).

The window.size parameter is similar to the one used in DTW, so NULL signifies no constraint, and its value should be greater than 1 for series of different length.

The Gaussian kernel is parameterized by sigma. Providing NULL means that the value will be estimated by using the strategy mentioned in Cuturi (2011) with a constant of 1. This estimation is subject to **randomness**, so consider estimating the value once and re-using it (the estimate is returned as an attribute of the result). See the examples.

For more information, refer to the package vignette and the referenced article.

Value

The logarithm of the GAK if `normalize = FALSE`, otherwise 1 minus the normalized GAK. The value of `sigma` is assigned as an attribute of the result.

Note

If `normalize` is set to `FALSE`, the returned value is **not** a distance, rather a similarity. The `proxy::dist` version is thus always normalized.

A constrained unnormalized calculation (i.e. with `window.size > 0` and `normalize = FALSE`) will return negative infinity if `abs(NROW(x) - NROW(y)) > window.size`. Since the function won't perform calculations in that case, it might be faster, but if this behavior is not desired, consider reinterpolating the time series (see `reinterpolate`) or increasing the window size.

References

Cuturi, M. (2011). Fast global alignment kernels. In *Proceedings of the 28th international conference on machine learning (ICML-11)* (pp. 929-936).

Examples

```
## Not run:
data(uciCT)

set.seed(832)
GAKd <- proxy::dist(zscore(CharTraj), method = "gak",
                    pairwise = TRUE, window.size = 18L)

# Obtained estimate of sigma
sigma <- attr(GAKd, "sigma")

# Use value for clustering
dtwclust(CharTraj, k = 20L,
         distance = "gak", centroid = "shape",
         sigma = sigma,
         control = list(trace = TRUE,
                        window.size = 18L))

## End(Not run)
```

---

lb_improved *Lemire's improved DTW lower bound*

Description

This function calculates an improved 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, force.symmetry = FALSE, error.check = TRUE)
```

Arguments

x	A time series (reference).
y	A time series with the same length as x (query).
window.size	Window size for envelope calculation. See details.
norm	Vector norm. Either "L1" for Manhattan distance or "L2" for Euclidean.
lower.env	Optionally, a pre-computed lower envelope for y can be provided (non-proxy version only).
upper.env	Optionally, a pre-computed upper envelope for y can be provided (non-proxy version only).
force.symmetry	If TRUE, a second lower bound is calculated by swapping x and y, and whichever result has a *higher* distance value is returned. The proxy version can only work if a square matrix is obtained, but use carefully.
error.check	Check data inconsistencies?

Details

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.

The reference time series should go in `x`, whereas the query time series should go in `y`.

Value

The improved lower bound for the DTW distance.

Note

The lower bound is defined for time series of equal length only and is **not** symmetric.

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. The convention mentioned above for references and queries still holds. See the examples.

The proxy version of `force.symmetry` should only be used when only `x` is provided or both `x` and `y` are identical. It compares the lower and upper triangular of the resulting distance matrix and forces symmetry in such a way that the tightest lower bound is obtained.

References

Lemire D (2009). "Faster retrieval with a two-pass dynamic-time-warping lower bound ." *Pattern Recognition*, **42**(9), pp. 2169 - 2180. ISSN 0031-3203, http://dx.doi.org/10.1016/j.patcog.2008.11.030, http://www.sciencedirect.com/science/article/pii/S0031320308004925.

Examples

```
# Sample data
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 argments 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 documentation 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
```

---

lb_keogh *Keogh's 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_keogh(x, y, window.size = NULL, norm = "L1", lower.env = NULL,
  upper.env = NULL, force.symmetry = FALSE, error.check = TRUE)
```

Arguments

x	A time series (reference).
y	A time series with the same length as x (query).
window.size	Window size for envelope calculation. See details.
norm	Vector norm. Either "L1" for Manhattan distance or "L2" for Euclidean.
lower.env	Optionally, a pre-computed lower envelope for y can be provided (non-proxy version only).

`upper.env`	Optionally, a pre-computed upper envelope for `y` can be provided (non-proxy version only).
`force.symmetry`	If `TRUE`, a second lower bound is calculated by swapping `x` and `y`, and whichever result has a *higher* distance value is returned. The proxy version can only work if a square matrix is obtained, but use carefully.
`error.check`	Check data inconsistencies?

Details

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.

The reference time series should go in `x`, whereas the query time series should go in `y`.

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

The lower bound is defined for time series of equal length only and is **not** symmetric.

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. The convention mentioned above for references and queries still holds. See the examples.

The proxy version of `force.symmetry` should only be used when only `x` is provided or both `x` and `y` are identical. It compares the lower and upper triangular of the resulting distance matrix and forces symmetry in such a way that the tightest lower bound is obtained.

References

Keogh E and Ratanamahatana CA (2005). "Exact indexing of dynamic time warping." *Knowledge and information systems*, **7**(3), pp. 358-386.

Examples

```
# Sample data
data(uciCT)

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

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

d.lbk <= d.dtw

# Calculating the LB between several time series using the 'proxy' package
# (notice how both argments 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.dtw <- proxy::dist(CharTraj[1], CharTraj[2:5], method = "DTW2",
                     window.type = "slantedband", window.size = 20)

D.lbk <= D.dtw
```

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, y`	Univariate time series.

Value

The cross-correlation sequence with length `length(x) + length(y) - 1L`.

References

Paparrizos J and Gravano L (2015). "k-Shape: Efficient and Accurate Clustering of Time Series." In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, series SIGMOD '15, pp. 1855-1870. ISBN 978-1-4503-2758-9, http://doi.org/10.1145/2723372.2737793.

See Also

SBD

randIndex *Compare partitions*

Description

Compute the (adjusted) Rand, Jaccard and Fowlkes-Mallows index for agreement of two partitions. This generic is included in the `flexclust` package.

Usage

```
## S4 method for signature 'dtwclust,ANY'
randIndex(x, y, correct = TRUE,
  original = !correct)

## S4 method for signature 'ANY,dtwclust'
randIndex(x, y, correct = TRUE,
  original = !correct)

## S4 method for signature 'dtwclust,dtwclust'
randIndex(x, y, correct = TRUE,
  original = !correct)
```

Arguments

`x, y, correct, original`
: See `randIndex`.

See Also

`randIndex`

reinterpolate *Wrapper for simple linear reinterpolation*

Description

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

Usage

```
reinterpolate(x, new.length, multivariate = FALSE)
```

Arguments

x	Data to reinterpolate. Either a vector, a matrix/data.frame where each row is to be reinterpolated, or a list of vectors.
new.length	Desired length of the output series.
multivariate	Is x a multivariate time series? It will be detected automatically if a list is provided in x.

Value

Reinterpolated time series

Examples

```
data(uciCT)

# list of univariate series
series <- reinterpolate(CharTraj, 205L)

# list of multivariate series
series <- reinterpolate(CharTrajMV, 205L)

# single multivariate series
series <- reinterpolate(CharTrajMV[[1L]], 205L, TRUE)
```

SBD *Shape-based distance*

Description

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

Usage

```
SBD(x, y, znorm = FALSE, error.check = TRUE)
```

Arguments

x, y	A time series.
znorm	Logical. Should each series be z-normalized before calculating the distance?
error.check	Check data inconsistencies?

Details

This distance works best if the series are *z-normalized*. If not, at least they should have corresponding amplitudes, since the values of the signals **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`. After matching, the series may have to be truncated or extended and padded with zeros if needed.

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

Value

A list with:

- `dist`: The shape-based distance between `x` and `y`.
- `yshift`: A shifted version of `y` so that it optimally matches `x` (based on correlation).

Note

If you wish to calculate the distance 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.

This distance is calculated with help of the Fast Fourier Transform, so it can be sensitive to numerical precision. Thus, this function (and the functions that depend on it) might return different values in 32 bit installations compared to 64 bit ones.

References

Paparrizos J and Gravano L (2015). "k-Shape: Efficient and Accurate Clustering of Time Series." In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, series SIGMOD '15, pp. 1855-1870. ISBN 978-1-4503-2758-9, http://doi.org/10.1145/2723372.2737793.

See Also

NCCc, shape_extraction

Examples

```
# load data
data(uciCT)

# distance between series of different lengths
sbd <- SBD(CharTraj[[1]], CharTraj[[100]], 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 Paparrizos and Gravano, 2015, for the k-Shape clustering algorithm.

Usage

```
shape_extraction(X, centroid = NULL, znorm = FALSE)
```

Arguments

X	A data matrix where each row is a time series, or a list where each element is a time series. Multivariate series should be provided as a list of matrices where time spans the rows and the variables span the columns.
centroid	Optionally, a time series to use as reference. Defaults to a random series of X if NULL. For multivariate series, this should be a matrix with the same characteristics as the matrices in X. *It will be z-normalized.*
znorm	Logical flag. Should z-scores be calculated for X before processing?

Details

This works only if the series are *z-normalized*, since the output will also have this normalization.

The resulting centroid will have the same length as centroid if provided. Otherwise, there are two possibilities: if all series from X have the same length, all of them will be used as-is, and the output will have the same length as the series; if series have different lengths, a series will be chosen at random and used as reference. The output series will then have the same length as the chosen series.

This centroid computation is casted as an optimization problem called maximization of Rayleigh Quotient. It depends on the SBD algorithm. See the cited article for more details.

Value

Centroid time series (z-normalized).

References

Paparrizos J and Gravano L (2015). "k-Shape: Efficient and Accurate Clustering of Time Series." In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, series SIGMOD '15, pp. 1855-1870. ISBN 978-1-4503-2758-9, http://doi.org/10.1145/2723372.2737793.

See Also

SBD, zscore

Examples

```
# Sample data
data(uciCT)

# Normalize desired subset
X <- zscore(CharTraj[1:5])

# Obtain centroid series
C <- shape_extraction(X)

# Result
matplot(do.call(cbind, X),
        type = "l", col = 1:5)
points(C)
```

`TADPole` *TADPole clustering*

Description

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

Usage

```
TADPole(data, k = 2L, dc, window.size, error.check = TRUE, lb = "lbk")
```

Arguments

`data`	The data matrix where each row is a time series. Optionally a list with each time series.
`k`	The number of desired clusters.
`dc`	The cutoff distance.
`window.size`	Window size constraint for DTW. See details.
`error.check`	Should the data be checked for inconsistencies?
`lb`	Which lower bound to use, "lbk" for `lb_keogh` or "lbi" for `lb_improved`.

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 centroids are always elements of the data. However, this algorithm is deterministic, depending on the value of `dc`.

The algorithm first uses the DTW's upper and lower bounds (Euclidean and LB_Keogh respectively) 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 centroids.

The algorithm relies on the DTW bounds, which are only defined for time series of equal length.

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:

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

Parallel Computing

Please note that running tasks in parallel does **not** guarantee faster computations. The overhead introduced is sometimes too large, and it's better to run tasks sequentially.

The user can register a parallel backend, e.g. with the `doParallel` package, in order to attempt to speed up the calculations (see the examples).

References

Begum N, Ulanova L, Wang J and Keogh E (2015). "Accelerating Dynamic Time Warping Clustering with a Novel Admissible Pruning Strategy." In *Conference on Knowledge Discovery and Data Mining*, series KDD '15. ISBN 978-1-4503-3664-2/15/08, http://dx.doi.org/10.1145/2783258.2783286.

Examples

```
## Not run:
#### Running TADPole with parallel support
require(doParallel)

# Load data
data(uciCT)

# Reinterpolate to same length
data <- reinterpolate(CharTraj, new.length = max(lengths(CharTraj)))

# Create parallel workers
cl <- makeCluster(detectCores())
```

```
invisible(clusterEvalQ(cl, library(dtwclust)))
registerDoParallel(cl)

# Cluster
kc.tadp <- TADPole(data, k = 20, window.size = 20, dc = 1.5)

# Stop parallel workers
stopCluster(cl)

# Return to sequential computations
registerDoSEQ()

# Compute VI Index
cat("VI index for TADPole:", cvi(kc.tadp$cl, CharTrajLabels, "VI"), "\n\n")

## End(Not run)
```

`uciCT` *Subset of character trajectories data set*

Description

Subset: only 5 examples of each considered character. 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 in `CharTraj` has only 5 examples of the X velocity for each character. A vector with labels is also loaded in `CharTrajLabels`.

The subset included in `CharTrajMV` has 5 examples too, but includes tip force as well as X and Y velocity. Each element of the list is a multivariate series with 3 variables.

Source

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

`zscore` *Wrapper for z-normalization*

Description

Wrapper for function `scale` that returns zeros instead of `NaN`. It also supports a list of vectors and a matrix input.

Usage

```
zscore(x, ..., multivariate = FALSE, keep.attributes = FALSE)
```

Arguments

`x`	Data to normalize. Either a vector, a matrix/data.frame where each row is to be normalized, or a list of vectors.
`...`	Further arguments to pass to `scale`.
`multivariate`	Is `x` a multivariate time series? It will be detected automatically if a list is provided in `x`.
`keep.attributes`	Should the mean and standard deviation returned by `scale` be preserved?

Value

Normalized data in the same format as provided.

Index