Package: twdtw (via r-universe)

October 7, 2024

Title Time-Weighted Dynamic Time Warping

Version 1.0-1

Description Implements Time-Weighted Dynamic Time Warping (TWDTW), a measure for quantifying time series similarity. The TWDTW algorithm, described in Maus et al. (2016) <doi:10.1109/JSTARS.2016.2517118> and Maus et al. (2019) <doi:10.18637/jss.v088.i05>, is applicable to multi-dimensional time series of various resolutions. It is particularly suitable for comparing time series with seasonality for environmental and ecological data analysis, covering domains such as remote sensing imagery, climate data, hydrology, and animal movement. The 'twdtw' package offers a user-friendly 'R' interface, efficient 'Fortran' routines for TWDTW calculations, flexible time weighting definitions, as well as utilities for time series preprocessing and visualization.

License GPL (>= 3)

URL `https://github.com/vwmaus/twdtw/`

BugReports `https://github.com/vwmaus/twdtw/issues/`

Encoding UTF-8

RoxygenNote 7.2.3

Imports Rcpp, proxy

Suggests rbenchmark, testthat (>= 3.0.0)

LinkingTo Rcpp

Collate 'RcppExports.R' 'convert_date_to_numeric.R' 'init.R' 'plot_cost_matrix.R' 'twdtw.R' 'zzz.R'

Config/testthat/edition 3

NeedsCompilation yes

Author Victor Maus [aut, cre] (<`https://orcid.org/0000-0002-7385-4723`>)

Maintainer Victor Maus <`vwmaus1@gmail.com`>

Repository CRAN

Date/Publication 2023-08-08 07:20:02 UTC

Contents

`date_to_numeric_cycle` *Convert Date/POSIXct to a Numeric Cycle*

Description

This function takes a date or datetime and converts it to a numeric cycle. The cycle can be specified in units of years, months, days, hours, minutes, or seconds. When cycle_length is a string, time_scale only changes the unit in which the result is expressed. When cycle_length is numeric, time_scale and origin are used to compute the elapsed time.

Usage

```
date_to_numeric_cycle(x, cycle_length, time_scale, origin = NULL)
```

Arguments

`x`	A vector of dates or datetimes to convert. If not of type Date or POSIXct, the function attempts to convert it.
`cycle_length`	The length of the cycle. Can be a numeric value or a string specifying the units ('year', 'month', 'day', 'hour', 'minute', 'second'). When numeric, the cycle length is in the same units as time_scale. When a string, it specifies the time unit of the cycle.
`time_scale`	Specifies the time scale for the conversion. Must be one of 'year', 'month', 'day', 'hour', 'minute', 'second'. When cycle_length is a string, time_scale changes the unit in which the result is expressed. When cycle_length is numeric, time_scale is used to compute the elapsed time in seconds.
`origin`	For numeric cycle_length, the origin must be specified. This is the point from which the elapsed time is computed. Must be of the same class as x.

Value

The numeric cycle value(s) corresponding to x.

Examples

```
date_to_numeric_cycle(Sys.time(), "year", "day") # Returns the day of the year
date_to_numeric_cycle(Sys.time(), "day", "hour") # Returns the hour of the day
```

`max_cycle_length` *Calculate the Maximum Possible Value of a Time Cycle*

Description

This function returns the maximum possible value that a specific time component can take, given a cycle length and scale.

Usage

```
max_cycle_length(cycle_length, time_scale)
```

Arguments

`cycle_length`	A character string indicating the larger unit of time. It must be one of "year", "month", "day", "hour", "minute".
`time_scale`	A character string indicating the smaller unit of time, which is a division of the `cycle_length`. If `cycle_length` is "year", `time_scale` can be one of "month", "day", "hour", "minute", "second". If `cycle_length` is "month", `time_scale` can be "day", "hour", "minute", "second", and so on.

Value

The function returns the maximum possible value that the `time_scale` can take within one `cycle_length`.

Examples

```
max_cycle_length("year", "month")  # Maximum months is a year 12
max_cycle_length("day", "minute")  # Maximum minutes in a day 1440
max_cycle_length("year", "day")    # Maximum days in a year 366
```

`plot_cost_matrix` *Plot TWDTW cost matrix*

Description

This function visualizes the Time-Weighted Dynamic Time Warping cost matrix.

Usage

```
plot_cost_matrix(x, ...)
```

Arguments

`x`	An object of class 'twdtw' including internal data.
`...`	Additional arguments passed to `image`.

Value

An image plot of the TWDTW cost matrix. The x-axis represents the time series x, and the y-axis represents the time series y. The cost matrix is color-coded, with darker shades indicating higher costs and lighter shades indicating lower costs. No object is returned by this function; the plot is directly outputted to the active device.

Examples

```
# Create a time series
n <- 23
t <- seq(0, pi, length.out = n)
d <- seq(as.Date('2020-09-01'), length.out = n, by = "15 day")

x <- data.frame(time = d,      v1 = sin(t)*2 + runif(n))

# shift time by 30 days
y <- data.frame(time = d + 30, v1 = sin(t)*2 + runif(n))

plot(x, type = "l", xlim = range(c(d, d + 5)))
lines(y, col = "red")

# Call twdtw using "output = 'internals'
twdtw_obj <- twdtw(x, y,
       cycle_length = 'year',
       time_scale = 'day',
       time_weight = c(steepness = 0.1, midpoint = 50), output = 'internals')

plot_cost_matrix(twdtw_obj)
```

`print.twdtw` *Print method for twdtw class*

Description

Print method for twdtw class

Usage

```
## S3 method for class 'twdtw'
print(x, ...)
```

Arguments

x	An object of class `twdtw`.
...	Arguments passed to `print.default` or other methods.

Value

This function returns a textual representation of the object `twdtw`, which is printed directly to the console. If `x` is a list, the function will print a summary of matches and omit `twdtw`'s internal data, see `names(x)`. If `x` is not a list, it prints the content of `x`, i.e. either a matrix with all matches or the lowest `twdtw` distance.

`twdtw` *Calculate Time-Weighted Dynamic Time Warping (TWDTW) distance*

Description

This function calculates the Time-Weighted Dynamic Time Warping (TWDTW) distance between two time series.

Usage

```
twdtw(x, y, time_weight, cycle_length, time_scale, ...)

## S3 method for class 'data.frame'
twdtw(
  x,
  y,
  time_weight,
  cycle_length,
  time_scale,
  origin = NULL,
  index_column = "time",
  max_elapsed = Inf,
  output = "distance",
  version = "f90",
  ...
)

## S3 method for class 'matrix'
twdtw(
  x,
  y,
  time_weight,
  cycle_length,
  time_scale = NULL,
  index_column = 1,
  max_elapsed = Inf,
  output = "distance",
  version = "f90",
  ...
)
```

Arguments

`x`	A data.frame or matrix representing time series.
`y`	A data.frame or matrix representing a labeled time series (reference).
`time_weight`	A numeric vector with length two (steepness and midpoint of logistic weight) or a function. See details.
`cycle_length`	The length of the cycle. Can be a numeric value or a string specifying the units ('year', 'month', 'day', 'hour', 'minute', 'second'). When numeric, the cycle length is in the same units as time_scale. When a string, it specifies the time unit of the cycle.
`time_scale`	Specifies the time scale for the conversion. Must be one of 'year', 'month', 'day', 'hour', 'minute', 'second'. When cycle_length is a string, time_scale changes the unit in which the result is expressed. When cycle_length is numeric, time_scale is used to compute the elapsed time in seconds.
`...`	ignore
`origin`	For numeric cycle_length, the origin must be specified. This is the point from which the elapsed time is computed. Must be of the same class as x.
`index_column`	(optional) The column name of the time index for data.frame inputs. Defaults to "time". For matrix input, an integer indicating the column with the time index. Defaults to 1.
`max_elapsed`	Numeric value constraining the TWDTW calculation to the lower band given by a maximum elapsed time. Defaults to Inf.
`output`	A character string defining the output. It must be one of 'distance', 'matches', 'internals'. Defaults to 'distance'. 'distance' will return the lowest TWDTW distance between `x` and `y`. 'matches' will return all matches within the TWDTW matrix. 'internals' will return all TWDTW internal data.
`version`	A string identifying the version of TWDTW implementation. Options are 'f90' for Fortran 90, 'f90goto' for Fortran 90 with goto statements, or 'cpp' for C++ version. Defaults to 'f90'. See details.

Details

TWDTW calculates a time-weighted version of DTW by modifying each element of the DTW's local cost matrix (see details in Maus et al. (2016) and Maus et al. (2019)). The default time weight is calculated using a logistic function that adds a weight to each pair of observations in the time series `x` and `y` based on the time difference between observations, such that

$$tw(dist_{i,j}) = dist_{i,j} + \frac{1}{1 + e^{-\alpha(el_{i,j} - \beta)}}$$

Where:

- tw is the time-weight function
- $dist_{i,j}$ is the Euclidean distance between the i-th element of `x` and the j-th element of `y` in a multi-dimensional space
- $el_{i,j}$ is the time elapsed between the i-th element of `x` and the j-th element of `y`

- α and β are the steepness and midpoint of the logistic function, respectively

The logistic function is implemented as the default option in the C++ and Fortran versions of the code. To use the native implementation, α and β must be provided as a numeric vector of length two using the argument `time_weight`. This implementation provides high processing performance.

The `time_weight` argument also accepts a function defined in R, allowing the user to define a different weighting scheme. However, passing a function to `time_weight` can degrade the processing performance, i.e., it can be up to 3x slower than using the default logistic time-weight.

A time-weight function passed to `time_weight` must receive two numeric arguments and return a single numeric value. The first argument received is the Euclidean $dist_{i,j}$ and the second is the elapsed time $el_{i,j}$. For example, `time_weight = function(dist, el) dist + 0.1*el` defines a linear weighting scheme with a slope of 0.1.

The Fortran 90 versions of `twdtw` are usually faster than the C++ version. The '`f90goto`' version, which uses goto statements, is slightly quicker than the '`f90`' version that uses while and for loops. You can use the `max_elapsed` parameter to limit the TWDTW calculation to a maximum elapsed time. This means it will skip comparisons between pairs of observations in `x` and `y` that are far apart in time. Be careful, though: if `max_elapsed` is set too low, it could change the results. It important to try out different settings for your specific problem.

Value

An S3 object twdtw either: If output = 'distance', a numeric value representing the TWDTW distance between the two time series. If output = 'matches', a numeric matrix of all TWDTW matches. For each match the starting index, ending index, and distance are returned. If output = 'internals', a list of all TWDTW internal data is returned.

References

Maus, V., Camara, G., Cartaxo, R., Sanchez, A., Ramos, F. M., & de Moura, Y. M. (2016). A Time-Weighted Dynamic Time Warping Method for Land-Use and Land-Cover Mapping. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 9(8), 3729-3739. doi:10.1109/JSTARS.2016.2517118

Maus, V., Camara, G., Appel, M., & Pebesma, E. (2019). dtwSat: Time-Weighted Dynamic Time Warping for Satellite Image Time Series Analysis in R. Journal of Statistical Software, 88(5), 1-31. doi:10.18637/jss.v088.i05

Examples

```
# Create a time series
n <- 23
t <- seq(0, pi, length.out = n)
d <- seq(as.Date('2020-09-01'), length.out = n, by = "15 day")

x <- data.frame(time = d,      v1 = sin(t)*2 + runif(n))

# shift time by 30 days
y <- data.frame(time = d + 30, v1 = sin(t)*2 + runif(n))

plot(x, type = "l", xlim = range(c(d, d + 5)))
```

```
lines(y, col = "red")

# Calculate TWDTW distance between x and y using logistic weight
twdtw(x, y,
      cycle_length = 'year',
      time_scale = 'day',
      time_weight = c(steepness = 0.1, midpoint = 50))

# Pass a generic time-weight function
twdtw(x, y,
      cycle_length = 'year',
      time_scale = 'day',
      time_weight = function(x,y) x + 1.0 / (1.0 + exp(-0.1 * (y - 50))))

# Test other version
twdtw(x, y,
      cycle_length = 'year',
      time_scale = 'day',
      time_weight = c(steepness = 0.1, midpoint = 50),
      version = 'f90goto')

twdtw(x, y,
      cycle_length = 'year',
      time_scale = 'day',
      time_weight = c(steepness = 0.1, midpoint = 50),
      version = 'cpp')
```

Index