Package 'dtwSat'

May 25, 2018

Type Package

Title Time-Weighted Dynamic Time Warping for Satellite Image Time Series Analysis

Version 0.2.4

Date 2018-05-24

Description Provides an implementation of the Time-Weighted Dynamic Time Warping (TWDTW) method for land cover mapping using satellite image time series. TWDTW is based on the Dynamic Time Warping technique and has achieved high accuracy for land cover classification using satellite data. The method is based on comparing unclassified satellite image time series with a set of known temporal patterns (e.g. phenological cycles associated with the vegetation). Using 'dtwSat' the user can build temporal patterns for land cover types, apply the TWDTW analysis for satellite datasets, visualize the results of the time series analysis, produce land cover maps, create temporal plots for land cover change, and compute accuracy assessment metrics.

Depends R (>= 3.2.0), zoo, raster, snow, ggplot2

Imports methods, rgdal, dtw, proxy, scales, reshape2, grDevices, RColorBrewer, plyr, stats, sp, lubridate, caret, mgcv, xtable

Suggests knitr, rmarkdown, rticles, gridExtra, grid, png, Hmisc, tikzDevice

License GPL (>= 2) | file LICENSE

URL https://github.com/vwmaus/dtwSat/

BugReports https://github.com/vwmaus/dtwSat/issues

Author Victor Maus [aut, cre] (<https://orcid.org/0000-0002-7385-4723>), Marius Appel [ctb], Toni Giorgino [ctb]

Maintainer Victor Maus <vwmaus1@gmail.com>

LazyData true

RoxygenNote 6.0.1

Collate 'class-crossValidation.R' 'class-twdtwRaster.R'
'class-twdtwAssessment.R' 'class-twdtwTimeSeries.R'
'class-twdtwMatches.R' 'createPatterns.R' 'data.R' 'dtw.R'
'dwtSat.R' 'getInternals.R' 'getTimeSeries.R' 'linearWeight.R'
'logisticWeight.R' 'methods.R' 'miscellaneous.R' 'plot.R'
'plotAccuracy.R' 'plotAdjustedArea.R' 'plotAlignments.R'
'plotArea.R' 'plotChanges.R' 'plotClassification.R'
'plotCostMatrix.R' 'plotDistance.R' 'plotMapSamples.R'
'plotMaps.R' 'plotMatches.R' 'plotPaths.R' 'plotPatterns.R'
'plotTimeSeries.R' 'resampleTimeSeries.R' 'subset.R' 'twdtw.R'
'twdtwApply.R' 'twdtwApplyParallel.R' 'twdtwAssess.R'
'twdtwClassify.R' 'twdtwCrossValidate.R' 'twdtwDist.R'
'twdtwXtable.R' 'zzz.R'

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-05-25 07:45:22 UTC

R topics documented:

`createPatterns` *Create patterns*

Description

Create temporal patterns from objects of class twdtwTimeSeries.

Usage

```
createPatterns(x, from = NULL, to = NULL, freq = 1, attr = NULL,
  split = TRUE, formula, ...)

## S4 method for signature 'twdtwTimeSeries'
createPatterns(x, from = NULL, to = NULL,
  freq = 1, attr = NULL, split = TRUE, formula, ...)
```

Arguments

`x`	an object of class `twdtwTimeSeries`.
`from`	A character or `Dates` object in the format "yyyy-mm-dd". If not informed it is equal to the smallest date of the first element in x. See details.
`to`	A `character` or `Dates` object in the format "yyyy-mm-dd". If not informed it is equal to the greatest date of the first element in x. See details.
`freq`	An integer. The sampling frequency of the output patterns.
`attr`	A vector character or numeric. The attributes in x to be used. If not declared the function uses all attributes.
`split`	A logical. If TRUE the samples are split by label. If FALSE all samples are set to the same label.
`formula`	A formula. Argument to pass to `gam`.
`...`	other arguments to pass to the function `gam` in the packege **mgcv**.

Details

The hidden assumption is that the temporal pattern is a cycle the repeats itself within a given time interval. Therefore, all time series samples in x are aligned to each other keeping their respective sequence of days of the year. The function fits a Generalized Additive Model (GAM) to the aligned set of samples.

Value

an object of class twdtwTimeSeries

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwMatches-class, twdtwTimeSeries-class, getTimeSeries, and twdtwApply

Examples

```
# Creating patterns from objects of class twdtwTimeSeries
evi = brick(system.file("lucc_MT/data/evi.tif", package="dtwSat"))
ndvi = brick(system.file("lucc_MT/data/ndvi.tif", package="dtwSat"))
timeline = scan(system.file("lucc_MT/data/timeline", package="dtwSat"), what="date")
rts = twdtwRaster(evi, ndvi, timeline=timeline)

# Read field samples
## Not run:
field_samples = read.csv(system.file("lucc_MT/data/samples.csv", package="dtwSat"))
prj_string = scan(system.file("lucc_MT/data/samples_projection", package="dtwSat"),
                  what = "character")

# Extract time series
ts = getTimeSeries(rts, y = field_samples, proj4string = prj_string)

# Create temporal patterns
patt = createPatterns(x=ts, from="2005-09-01", to="2006-09-01", freq=8, formula = y~s(x))

# Plot patterns
autoplot(patt[[1]], facets = NULL) + xlab("Time") + ylab("Value")


## End(Not run)
```

dtwSat *Time-Weighted Dynamic Time Warping for Satellite Image Time Series*

Description

Provides an implementation of the Time-Weighted Dynamic Time Warping (TWDTW) method for land use and land cover mapping using satellite image time series [1]. TWDTW is based on the Dynamic Time Warping technique and has achieved high accuracy for land use and land cover classification using satellite data. The method is based on comparing unclassified satellite image time series with a set of known temporal patterns (e.g. phenological cycles associated with the vegetation). Using 'dtwSat' the user can build temporal patterns for land cover types, apply the TWDTW analysis for satellite datasets, visualize the results of the time series analysis, produce land cover maps, and create temporal plots for land cover change analysis.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

References

[1] Maus V, Camara G, Cartaxo R, Sanchez A, Ramos FM, de Queiroz, GR. (2016). A Time-Weighted Dynamic Time Warping method for land use and land cover mapping. Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of, vol.PP, no.99, pp.1-11.

See Also

twdtwApply

---

get *Get elements from twdtwMatches objects*

Description

Get elements from twdtwMatches-class objects.

Usage

```
## S4 method for signature 'twdtwMatches'
getAlignments(object, timeseries.labels = NULL,
  patterns.labels = NULL)

## S4 method for signature 'twdtwMatches'
getInternals(object, timeseries.labels = NULL,
  patterns.labels = NULL)

## S4 method for signature 'twdtwMatches'
getMatches(object, timeseries.labels = NULL,
  patterns.labels = NULL)
```

Arguments

`object`	an object of class twdtwMatches.
`timeseries.labels`	a vector with labels of the time series.
`patterns.labels`	a vector with labels of the patterns.

Value

a list with TWDTW results or an object `twdtwTimeSeries-class`.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

`twdtwMatches-class`, and `twdtwApply`

Examples

```
# Getting patterns from objects of class twdtwMatches
patt = twdtwTimeSeries(MOD13Q1.patterns.list)
ts = twdtwTimeSeries(MOD13Q1.ts.list)
mat = twdtwApply(x=ts, y=patt, weight.fun=logisticWeight(-0.1,100), keep=TRUE)
getPatterns(mat)
getTimeSeries(mat)
getAlignments(mat)
getMatches(mat)
getInternals(mat)
```

`getDatesFromDOY` *Get dates from year and day of the year*

Description

This function retrieves the date corresponding to the ginven year and day of the year.

Usage

```
getDatesFromDOY(year, doy)
```

Arguments

`year`	An vector with the years.
`doy`	An vector with the day of the year. It must have the same lenght as `year`.

Value

A Dates object.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

shiftDates

Examples

```
year = c(2000, 2001)
doy = c(366, 365)
dates = getDatesFromDOY(year, doy)
dates
```

---

getTimeSeries *Get time series from twdtw* objects*

Description

Get time series from objects of class twdtw*.

Usage

```
## S4 method for signature 'twdtwTimeSeries'
getTimeSeries(object, labels = NULL)

## S4 method for signature 'twdtwMatches'
getTimeSeries(object, labels = NULL)

## S4 method for signature 'twdtwMatches'
getPatterns(object, labels = NULL)

## S4 method for signature 'twdtwRaster'
getTimeSeries(object, y, labels = NULL,
  proj4string = NULL, id.labels = NULL)
```

Arguments

object	an object of class of class twdtw*.
labels	character vector with time series labels. For signature twdtwRaster this argument can be used to set the labels for each sample in y, or it can be combined with id.labels to select samples with a specific label.

y	a data.frame whose attributes are: longitude, latitude, the start "from" and the end "to" of the time interval for each sample. This can also be a SpatialPointsDataFrame whose attributes are the start "from" and the end "to" of the time interval. If missing "from" and/or "to", they are set to the time range of the object.
proj4string	projection string, see CRS-class. Used if y is a data.frame.
id.labels	a numeric or character with an column name from y to be used as samples labels. Optional.

Value

An object of class twdtwTimeSeries.

a list with TWDTW results or an object twdtwTimeSeries-class.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwRaster-class, twdtwTimeSeries-class, and twdtwMatches-class

Examples

```
# Getting time series from objects of class twdtwTimeSeries
ts = twdtwTimeSeries(MOD13Q1.ts.list)
getTimeSeries(ts, 2)
# Getting time series from objects of class twdtwTimeSeries
ts = twdtwTimeSeries(MOD13Q1.ts.list)
patt = twdtwTimeSeries(MOD13Q1.patterns.list)
mat = twdtwApply(x=ts, y=patt)
getTimeSeries(mat, 2)
## This example creates a twdtwRaster object and extract time series from it.

# Creating objects of class twdtwRaster with evi and ndvi time series
evi = brick(system.file("lucc_MT/data/evi.tif", package="dtwSat"))
ndvi = brick(system.file("lucc_MT/data/ndvi.tif", package="dtwSat"))
timeline = scan(system.file("lucc_MT/data/timeline", package="dtwSat"), what="date")
rts = twdtwRaster(evi, ndvi, timeline=timeline)

# Location and time range
ts_location = data.frame(longitude = -55.96957, latitude = -12.03864,
                         from = "2007-09-01", to = "2013-09-01")
prj_string = "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"

## Extract time series
ts = getTimeSeries(rts, y = ts_location, proj4string = prj_string)

autoplot(ts[[1]], facets = NULL) + xlab("Time") + ylab("Value")
```

`linearWeight` *Linear weight function*

Description

Builds a linear time weight function to compute the TWDTW local cost matrix [1].

Usage

```
linearWeight(a, b = 0)
```

Arguments

a	numeric. The slop of the line.
b	numeric. The intercept of the line.

Details

The linear `linearWeight` and `logisticWeight` weight functions can be passed to `twdtwApply` through the argument `weight.fun`. This will add a time-weight to the dynamic time warping analysis. The time weight creates a global constraint useful to analyse time series with phenological cycles of vegetation that are usually bound to seasons. In previous studies by [1] the logistic weight had better results than the linear for land cover classification. See [1] for details about the method.

Value

An `function` object.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

References

[1] Maus V, Camara G, Cartaxo R, Sanchez A, Ramos FM, de Queiroz, GR. (2016). A Time-Weighted Dynamic Time Warping method for land use and land cover mapping. Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of, vol.PP, no.99, pp.1-11.

See Also

`twdtwApply`

Examples

```
lin_fun = linearWeight(a=0.1)
lin_fun
```

`logisticWeight` *Logistic weight function*

Description

Builds a logistic time weight function to compute the TWDTW local cost matrix [1].

Usage

```
logisticWeight(alpha, beta)
```

Arguments

`alpha`	numeric. The steepness of logistic weight.
`beta`	numeric. The midpoint of logistic weight.

Details

The linear `linearWeight` and `logisticWeight` weight functions can be passed to `twdtwApply` through the argument `weight.fun`. This will add a time-weight to the dynamic time warping analysis. The time weight creates a global constraint useful to analyse time series with phenological cycles of vegetation that are usually bound to seasons. In previous studies by [1] the logistic weight had better results than the linear for land cover classification. See [1] for details about the method.

Value

An `function` object.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

References

[1] Maus V, Camara G, Cartaxo R, Sanchez A, Ramos FM, de Queiroz, GR. (2016). A Time-Weighted Dynamic Time Warping method for land use and land cover mapping. Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of, vol.PP, no.99, pp.1-11.

See Also

`twdtwApply`

Examples

```
log_fun = logisticWeight(alpha=-0.1, beta=100)
log_fun
```

MOD13Q1.MT.yearly.patterns *Data: patterns time series*

Description

This dataset has a list of patterns with the phenological cycle of: Water, Cotton-Fallow, Forest, Low vegetation, Pasture, Soybean-Cotton, Soybean-Maize, Soybean-Millet, Soybean-Sunflower, and Wetland. These time series are based on the MODIS product MOD13Q1 250 m 16 days [1]. The patterns were build from ground truth samples of each crop using Generalized Additive Models (GAM), see createPatterns.

Usage

```
MOD13Q1.MT.yearly.patterns
```

Format

A twdtwTimeSeries object.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

References

[1] Friedl MA, Sulla-Menashe D, Tan B, Schneider A, Ramankutty N, Sibley A, Huang X. (2010). MODIS Collection 5 global land cover: Algorithm refinements and characterization of new datasets. Remote Sensing of Environment, 114(1), 168 182.

See Also

MOD13Q1 documentation: See https://lpdaac.usgs.gov/dataset_discovery/modis/modis_products_table/mod13q1_v006.

MOD13Q1.patterns.list *Data: patterns time series*

Description

This dataset has a list of patterns with the phenological cycle of: Soybean, Cotton, and Maize. These time series are based on the MODIS product MOD13Q1 250 m 16 days [1]. The patterns were build from ground truth samples of each crop using Generalized Additive Models (GAM), see createPatterns.

Usage

```
MOD13Q1.patterns.list
```

Format

A named list of 3 zoo objects, "Soybean", "Cotton", and "Maize", whose indices are Dates in the format "yyyy-mm-dd". Each node has 6 attributes: "ndvi", "evi", "red", "nir", "blue", and "mir".

Author(s)

Victor Maus, <vwmaus1@gmail.com>

References

[1] Friedl MA, Sulla-Menashe D, Tan B, Schneider A, Ramankutty N, Sibley A, Huang X. (2010). MODIS Collection 5 global land cover: Algorithm refinements and characterization of new datasets. Remote Sensing of Environment, 114(1), 168 182.

See Also

MOD13Q1.ts, MOD13Q1.ts.list, and createPatterns.

MOD13Q1 documentation: See https://lpdaac.usgs.gov/dataset_discovery/modis/modis_products_table/mod13q1_v006.

MOD13Q1.ts *Data: An example of satellite time series*

Description

This dataset has a time series based on the MODIS product MOD13Q1 250 m 16 days [1]. It is an irregularly sampled time series using the real date of each pixel from "2009-08-05" to "2013-07-31".

Usage

```
MOD13Q1.ts
```

Format

A zoo object, whose indices are Dates in the format "yyyy-mm-dd". Each node has 6 attributes: "ndvi", "evi", "red", "nir", "blue", and "mir".

Author(s)

Victor Maus, <vwmaus1@gmail.com>

References

[1] Friedl MA, Sulla-Menashe D, Tan B, Schneider A, Ramankutty N, Sibley A, Huang X. (2010). MODIS Collection 5 global land cover: Algorithm refinements and characterization of new datasets. Remote Sensing of Environment, 114(1), 168 182.

See Also

MOD13Q1.ts.list, MOD13Q1.patterns.list.

MOD13Q1 documentation: https://lpdaac.usgs.gov/dataset_discovery/modis/modis_products_table/mod13q1_v006.

MOD13Q1.ts.labels *Data: Labels of the satellite time series in MOD13Q1.ts*

Description

This labels are based on field work.

Usage

```
MOD13Q1.ts.labels
```

Format

An object of class data.frame, whose attributas are: the label of the crop class "label", the start of the crop period "from", and the end of the crop period "to". The dates are in the format "yyyy-mm-dd".

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

MOD13Q1.ts.

MOD13Q1.ts.list *Data: A list of satellite time series*

Description

This dataset has a list of time series based on the MODIS product MOD13Q1 250 m 16 days [1]. It is an irregularly sampled time series using the real date of each pixel from "2009-08-05" to "2013-07-31".

Usage

```
MOD13Q1.ts.list
```

Format

A zoo object, whose indices are Dates in the format "yyyy-mm-dd". Each node has 6 attributes: "ndvi", "evi", "red", "nir", "blue", and "mir".

Author(s)

Victor Maus, <vwmaus1@gmail.com>

References

[1] Friedl MA, Sulla-Menashe D, Tan B, Schneider A, Ramankutty N, Sibley A, Huang X. (2010). MODIS Collection 5 global land cover: Algorithm refinements and characterization of new datasets. Remote Sensing of Environment, 114(1), 168 182.

See Also

MOD13Q1.ts, and MOD13Q1.patterns.list.

MOD13Q1 documentation: https://lpdaac.usgs.gov/dataset_discovery/modis/modis_products_table/mod13q1_v006.

plot *Plotting twdtw* objects*

Description

Methods for plotting objects of class twdtw*.

Usage

```
## S4 method for signature 'twdtwAssessment,ANY'
plot(x, type = "area", ...)

## S4 method for signature 'twdtwCrossValidation,ANY'
plot(x, type = "crossvalidation", ...)

## S4 method for signature 'twdtwTimeSeries,ANY'
plot(x, type = "timeseries", ...)

## S4 method for signature 'twdtwMatches,ANY'
plot(x, type = "alignments", ...)

## S4 method for signature 'twdtwRaster,ANY'
plot(x, type = "maps", ...)
```

Arguments

x	An object of class twdtw*.
type	A character for the plot type: "paths", "matches", "alignments", "classification", "cost", "patterns", "timeseries", "maps", "area", "changes", and "distance".
...	additional arguments to pass to plotting functions. plotPaths, plotCostMatrix, plotAlignments, plotMatches, plotClassification, plotPatterns, plotTimeSeries, plotMaps, plotArea, or plotChanges.

Details

Plot types :

paths: Method for plotting the minimum paths in the cost matrix of TWDTW.
matches: Method for plotting the matching points from TWDTW analysis.
alignments: Method for plotting the alignments and respective TWDTW dissimilarity measures.
classification: Method for plotting the classification of each subinterval of the time series based on TWDTW analysis.
cost: Method for plotting the internal matrices used during the TWDTW computation.
patterns: Method for plotting the temporal patterns.
timeseries: Method for plotting the temporal patterns.

Value

A ggplot object.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

plotAccuracy *Plotting accuracy assessment*

Description

Method for plotting accuracy assessment results.

Usage

```
plotAccuracy(x, perc = TRUE, conf.int = 0.95, time.labels = NULL,
  category.name = NULL, category.type = NULL)
```

Arguments

x	An object of class twdtwAssessment or twdtwCrossValidation.
perc	if TRUE shows the results in percent of area. Otherwise shows the area in the map units or km2 for no project raster. Default is TRUE.
conf.int	confidence level (0-1) for interval estimation of the population mean. for details see smean.cl.normal. Used if x is twdtwCrossValidation.
time.labels	a character or numeric for the time periods or NULL to aggregate all classified periods in the same plot. Default is NULL. Used if x is twdtwAssessment.
category.name	a character vector defining the class names. If NULL then use the classe names in the object x. Default is NULL.
category.type	a character defining the categories type "numeric" or "letter", if NULL then use the class names. Default is NULL.

Value

A ggplot object.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwAssessment and twdtwAssess

Examples

```
## Not run:

# See ?twdtwAssess and ?twdtwCrosValidate

plotAccuracy(x)

plotAccuracy(x, category.type="letter")
```

```
## End(Not run)
```

---

plotAdjustedArea *Plotting area and uncertainty*

Description

Method for plotting area and uncertainty.

Usage

```
plotAdjustedArea(x, perc = TRUE, time.labels = NULL, category.name = NULL,
  category.type = NULL)
```

Arguments

x	An object of class twdtwAssessment or twdtwCrossValidation.
perc	if TRUE shows the results in percent of area. Otherwise shows the area in the map units or km2 for no project raster. Default is TRUE.
time.labels	a character or numeric for the time periods or NULL to aggregate all classified periods in the same plot. Default is NULL. Used if x is twdtwAssessment.
category.name	a character vector defining the class names. If NULL then use the classe names in the object x. Default is NULL.
category.type	a character defining the categories type "numeric" or "letter", if NULL then use the class names. Default is NULL.

Value

A ggplot object.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwAssessment and twdtwAssess

Examples

```
## Not run:

# See ?twdtwAssess

plotAdjustedArea(twdtw_assess)

plotAdjustedArea(twdtw_assess, category.type="letter")


## End(Not run)
```

`plotAlignments` *Plotting alignments*

Description

Method for plotting the alignments and TWDTW dissimilarity measures.

Usage

```
plotAlignments(x, timeseries.labels = NULL, patterns.labels = NULL,
  attr = 1, threshold = Inf)
```

Arguments

`x`	An object of class `twdtwMatches`.
`timeseries.labels`	the label or index of the time series. Default is 1.
`patterns.labels`	a vector with labels of the patterns. If not declared the function will plot the alignments for all patterna in `x`.
`attr`	An integer or character vector indicating the attribute for plotting. Default is 1.
`threshold`	A number. The TWDTW dissimilarity threshold, *i.e.* the maximum TWDTW cost for consideration. Default is `Inf`.

Value

A ggplot object.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

`twdtwMatches-class`, `twdtwApply`, `plotPaths`, `plotCostMatrix`, `plotMatches`, and `plotClassification`.

Examples

```
log_fun = logisticWeight(-0.1, 100)
ts = twdtwTimeSeries(MOD13Q1.ts.list)
patt = twdtwTimeSeries(MOD13Q1.patterns.list)
mat1 = twdtwApply(x=ts, y=patt, weight.fun=log_fun)

plotAlignments(mat1)

plotAlignments(mat1, attr=c("evi","ndvi"))
```

`plotArea` *Plotting accumulated area*

Description

Method for plotting time series of accumulated area.

Usage

```
plotArea(x, time.levels = NULL, time.labels = NULL, class.levels = NULL,
  class.labels = NULL, class.colors = NULL, perc = TRUE)
```

Arguments

`x`	An object of class `twdtwRaster`.
`time.levels`	A character or numeric vector with the layers to plot. For plot type "change" the minimum length is two.
`time.labels`	A character or numeric vector with the labels of the layers. It must have the same length as time.levels. Default is NULL.
`class.levels`	A character or numeric vector with the levels of the raster values. Default is NULL.
`class.labels`	A character or numeric vector with the labels of the raster values. It must have the same length as class.levels. Default is NULL.
`class.colors`	a set of aesthetic values. It must have the same length as class.levels. Default is NULL. See scale_fill_manual for details.
`perc`	if TRUE shows the results in percent of area. Otherwise shows the area in the map units or km2 for no project raster. Default is TRUE.

Value

A ggplot object.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwRaster-class, twdtwApply, plotMaps, plotChanges, and plotDistance.

Examples

```
## Not run:

# Create raster time series
evi = brick(system.file("lucc_MT/data/evi.tif", package="dtwSat"))
ndvi = brick(system.file("lucc_MT/data/ndvi.tif", package="dtwSat"))
red = brick(system.file("lucc_MT/data/red.tif", package="dtwSat"))
blue = brick(system.file("lucc_MT/data/blue.tif", package="dtwSat"))
nir = brick(system.file("lucc_MT/data/nir.tif", package="dtwSat"))
mir = brick(system.file("lucc_MT/data/mir.tif", package="dtwSat"))
doy = brick(system.file("lucc_MT/data/doy.tif", package="dtwSat"))
timeline = scan(system.file("lucc_MT/data/timeline", package="dtwSat"), what="date")
rts = twdtwRaster(evi, ndvi, red, blue, nir, mir, timeline = timeline, doy = doy)

# Read fiels samples
field_samples = read.csv(system.file("lucc_MT/data/samples.csv", package="dtwSat"))
proj_str = scan(system.file("lucc_MT/data/samples_projection",
                package="dtwSat"), what = "character")

# Split samples for training (10%) and validation (90%) using stratified sampling
library(caret)
set.seed(1)
I = unlist(createDataPartition(field_samples$label, p = 0.1))
training_samples = field_samples[I,]
validation_samples = field_samples[-I,]

# Create temporal patterns
training_ts = getTimeSeries(rts, y = training_samples, proj4string = proj_str)
temporal_patterns = createPatterns(training_ts, freq = 8, formula = y ~ s(x))

# Run TWDTW analysis for raster time series
log_fun = weight.fun=logisticWeight(-0.1,50)
r_twdtw = twdtwApply(x=rts, y=temporal_patterns, weight.fun=log_fun, format="GTiff",
                     overwrite=TRUE)

# Classify raster based on the TWDTW analysis
r_lucc = twdtwClassify(r_twdtw, format="GTiff", overwrite=TRUE)

plotArea(r_lucc)

plotArea(r_lucc, perc=FALSE)


## End(Not run)
```

plotChanges *Plotting changes*

Description

Method for plotting changes over time.

Usage

```
plotChanges(x, time.levels = NULL, time.labels = NULL,
  class.levels = NULL, class.labels = NULL, class.colors = NULL)
```

Arguments

x	An object of class twdtwRaster.
time.levels	A character or numeric vector with the layers to plot. For plot type "change" the minimum length is two.
time.labels	A character or numeric vector with the labels of the layers. It must have the same length as time.levels. Default is NULL.
class.levels	A character or numeric vector with the levels of the raster values. Default is NULL.
class.labels	A character or numeric vector with the labels of the raster values. It must have the same length as class.levels. Default is NULL.
class.colors	a set of aesthetic values. It must have the same length as class.levels. Default is NULL. See scale_fill_manual for details.

Value

A ggplot object.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwRaster-class, twdtwApply, plotArea, plotMaps, and plotDistance.

Examples

```
## Not run:
# Run TWDTW analysis for raster time series
patt = MOD13Q1.MT.yearly.patterns
evi = brick(system.file("lucc_MT/data/evi.tif", package="dtwSat"))
ndvi = brick(system.file("lucc_MT/data/ndvi.tif", package="dtwSat"))
red = brick(system.file("lucc_MT/data/red.tif", package="dtwSat"))
blue = brick(system.file("lucc_MT/data/blue.tif", package="dtwSat"))
```

```
nir = brick(system.file("lucc_MT/data/nir.tif", package="dtwSat"))
mir = brick(system.file("lucc_MT/data/mir.tif", package="dtwSat"))
doy = brick(system.file("lucc_MT/data/doy.tif", package="dtwSat"))
timeline = scan(system.file("lucc_MT/data/timeline", package="dtwSat"), what="date")
rts = twdtwRaster(evi, ndvi, red, blue, nir, mir, timeline = timeline, doy = doy)

time_interval = seq(from=as.Date("2007-09-01"), to=as.Date("2013-09-01"),
                    by="12 month")
log_fun = weight.fun=logisticWeight(-0.1,50)

r_twdtw = twdtwApply(x=rts, y=patt, weight.fun=log_fun, breaks=time_interval,
          filepath="~/test_twdtw", overwrite=TRUE, format="GTiff")

r_lucc = twdtwClassify(r_twdtw, format="GTiff", overwrite=TRUE)

plotChanges(r_lucc)


## End(Not run)
```

`plotClassification` *Plotting subintervals classification*

Description

Method for plotting the classification of each subinterval of the time series based on TWDTW analysis.

Usage

```
plotClassification(x, timeseries.labels = NULL, patterns.labels = NULL,
  attr, ...)
```

Arguments

`x`	An object of class `twdtwMatches`.
`timeseries.labels`	the label or index of the time series. Default is 1.
`patterns.labels`	a vector with labels of the patterns. If not declared the function will plot one alignment for each pattern.
`attr`	An integer vector or character vector indicating the attribute for plotting. If not declared the function will plot all attributes.
`...`	additional arguments passed to `twdtwClassify`.

Value

A ggplot object.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwMatches-class, twdtwApply, twdtwClassify, plotAlignments, plotPaths, plotMatches, and plotCostMatrix.

Examples

```
log_fun = logisticWeight(-0.1, 100)
ts = twdtwTimeSeries(MOD13Q1.ts.list)
patt = twdtwTimeSeries(MOD13Q1.patterns.list)
mat1 = twdtwApply(x=ts, y=patt, weight.fun=log_fun)

# Classify interval
from = as.Date("2007-09-01")
to = as.Date("2013-09-01")
by = "6 month"
gp = plotClassification(x=mat1, from=from, to=to, by=by, overlap=.5)
gp
```

---

plotCostMatrix *Plotting paths*

Description

Method for plotting low cost paths in the TWDTW cost matrix.

Usage

```
plotCostMatrix(x, timeseries.labels = NULL, patterns.labels = NULL,
  matrix.name = "costMatrix")
```

Arguments

x	An object of class twdtwMatches.
timeseries.labels	the label or index of the time series. Default is 1.
patterns.labels	a vector with labels of the patterns. If not declared the function will plot one alignment for each pattern.
matrix.name	A character. The name of the matrix to plot, "costMatrix" for accumulated cost, "localMatrix" for local cost, or "timeWeight" for time-weight. Default is "costMatrix".

Value

A ggplot object.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwMatches-class, twdtwApply, plotAlignments, plotPaths, plotMatches, and plotClassification.

Examples

```
log_fun = logisticWeight(-0.1, 100)
ts = twdtwTimeSeries(MOD13Q1.ts.list)
patt = twdtwTimeSeries(MOD13Q1.patterns.list)
mat1 = twdtwApply(x=ts, y=patt, weight.fun=log_fun, keep=TRUE)

plotCostMatrix(mat1, matrix.name="costMatrix")

plotCostMatrix(mat1, matrix.name="localMatrix")

plotCostMatrix(mat1, matrix.name="timeWeight")
```

---

plotDistance *Plotting distance maps*

Description

Method for plotting TWDTW distance maps.

Usage

```
plotDistance(x, time.levels = 1, time.labels = 1, layers = NULL)
```

Arguments

x	An object of class twdtwRaster.
time.levels	A character or numeric vector with the layers to plot. For plot type "change" the minimum length is two.
time.labels	A character or numeric vector with the labels of the layers. It must have the same length as time.levels. Default is NULL.
layers	A character or numeric vector with the layers/bands of the raster time series.

Value

A ggplot object.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwRaster-class, twdtwApply, plotArea, plotChanges, and plotDistance.

Examples

```
## Not run:
# Run TWDTW analysis for raster time series
patt = MOD13Q1.MT.yearly.patterns
evi = brick(system.file("lucc_MT/data/evi.tif", package="dtwSat"))
ndvi = brick(system.file("lucc_MT/data/ndvi.tif", package="dtwSat"))
red = brick(system.file("lucc_MT/data/red.tif", package="dtwSat"))
blue = brick(system.file("lucc_MT/data/blue.tif", package="dtwSat"))
nir = brick(system.file("lucc_MT/data/nir.tif", package="dtwSat"))
mir = brick(system.file("lucc_MT/data/mir.tif", package="dtwSat"))
doy = brick(system.file("lucc_MT/data/doy.tif", package="dtwSat"))
timeline = scan(system.file("lucc_MT/data/timeline", package="dtwSat"), what="date")
rts = twdtwRaster(evi, ndvi, red, blue, nir, mir, timeline = timeline, doy = doy)

time_interval = seq(from=as.Date("2007-09-01"), to=as.Date("2013-09-01"),
                    by="12 month")
log_fun = weight.fun=logisticWeight(-0.1,50)

r_twdtw = twdtwApply(x=rts, y=patt, weight.fun=log_fun, breaks=time_interval,
          filepath="~/test_twdtw", overwrite=TRUE, format="GTiff", mc.cores=3)

plotDistance(r_twdtw)


## End(Not run)
```

---

plotMaps *Plotting maps*

Description

Method for plotting time series of maps.

Usage

```
plotMaps(x, time.levels = NULL, time.labels = NULL, class.levels = NULL,
  class.labels = NULL, class.colors = NULL)
```

Arguments

x	An object of class twdtwRaster.
time.levels	A character or numeric vector with the layers to plot. For plot type "change" the minimum length is two.
time.labels	A character or numeric vector with the labels of the layers. It must have the same length as time.levels. Default is NULL.
class.levels	A character or numeric vector with the levels of the raster values. Default is NULL.
class.labels	A character or numeric vector with the labels of the raster values. It must have the same length as class.levels. Default is NULL.
class.colors	a set of aesthetic values. It must have the same length as class.levels. Default is NULL. See scale_fill_manual for details.

Value

A ggplot object.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwRaster-class, twdtwApply, plotArea, plotChanges, and plotDistance.

Examples

```
## Not run:
# Run TWDTW analysis for raster time series
patt = MOD13Q1.MT.yearly.patterns
evi = brick(system.file("lucc_MT/data/evi.tif", package="dtwSat"))
ndvi = brick(system.file("lucc_MT/data/ndvi.tif", package="dtwSat"))
red = brick(system.file("lucc_MT/data/red.tif", package="dtwSat"))
blue = brick(system.file("lucc_MT/data/blue.tif", package="dtwSat"))
nir = brick(system.file("lucc_MT/data/nir.tif", package="dtwSat"))
mir = brick(system.file("lucc_MT/data/mir.tif", package="dtwSat"))
doy = brick(system.file("lucc_MT/data/doy.tif", package="dtwSat"))
timeline = scan(system.file("lucc_MT/data/timeline", package="dtwSat"), what="date")
rts = twdtwRaster(evi, ndvi, red, blue, nir, mir, timeline = timeline, doy = doy)

time_interval = seq(from=as.Date("2007-09-01"), to=as.Date("2013-09-01"),
                    by="12 month")
log_fun = weight.fun=logisticWeight(-0.1,50)

r_twdtw = twdtwApply(x=rts, y=patt, weight.fun=log_fun, breaks=time_interval,
          filepath="~/test_twdtw", overwrite=TRUE, format="GTiff", mc.cores=3)

r_lucc = twdtwClassify(r_twdtw, format="GTiff", overwrite=TRUE)
```

```
plotMaps(r_lucc)


## End(Not run)
```

`plotMapSamples` *Plotting maps*

Description

Method for plotting maps and samples.

Usage

```
plotMapSamples(x, samples = "all", ...)
```

Arguments

`x`	An object of class `twdtwAssessment`.
`samples`	a character defining the samples to plot "correct", "incorrect", "all". Default is "all".
`...`	other arguments to pass to `twdtwRaster`

Value

A ggplot object.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

`twdtwAssessment`, `plotAccuracy`, and `plotAdjustedArea`.

Examples

```
## Not run:

# Example of TWDTW analysis using raster files
library(dtwSat)
library(caret)

# Load raster data
evi  <- brick(system.file("lucc_MT/data/evi.tif",  package = "dtwSat"))
ndvi <- brick(system.file("lucc_MT/data/ndvi.tif", package = "dtwSat"))
red  <- brick(system.file("lucc_MT/data/red.tif",  package = "dtwSat"))
blue <- brick(system.file("lucc_MT/data/blue.tif", package = "dtwSat"))
nir  <- brick(system.file("lucc_MT/data/nir.tif",  package = "dtwSat"))
```

```
mir  <- brick(system.file("lucc_MT/data/mir.tif",  package = "dtwSat"))
doy  <- brick(system.file("lucc_MT/data/doy.tif",  package = "dtwSat"))
timeline <-
  scan(system.file("lucc_MT/data/timeline", package = "dtwSat"), what="date")

# Create raster time series
rts <- twdtwRaster(evi, ndvi, red, blue, nir, mir, timeline = timeline, doy = doy)

# Load field samples and projection
field_samples <-
  read.csv(system.file("lucc_MT/data/samples.csv", package = "dtwSat"))
proj_str <-
  scan(system.file("lucc_MT/data/samples_projection", package = "dtwSat"),
       what = "character")

# Split samples for training (10%) and validation (90%) using stratified sampling
set.seed(1)
I <- unlist(createDataPartition(field_samples$label, p = 0.1))
training_samples <- field_samples[I, ]
validation_samples <- field_samples[-I, ]

# Get time series form raster
training_ts <- getTimeSeries(rts, y = training_samples, proj4string = proj_str)
validation_ts <- getTimeSeries(rts, y = validation_samples, proj4string = proj_str)

# Create temporal patterns
temporal_patterns <- createPatterns(training_ts, freq = 8, formula = y ~ s(x))

# Set TWDTW weight function
log_fun <- logisticWeight(-0.1, 50)

# Run serial TWDTW analysis
r_twdtw <-
  twdtwApply(x = rts, y = temporal_patterns, weight.fun = log_fun, progress = 'text')

# or Run parallel TWDTW analysis
beginCluster()
r_twdtw <-
  twdtwApplyParallel(x = rts, y = temporal_patterns, weight.fun = log_fun, progress = 'text')
endCluster()

# Plot TWDTW distances for the first year
plot(r_twdtw, type = "distance", time.levels = 1)

# Classify raster based on the TWDTW analysis
r_lucc <- twdtwClassify(r_twdtw, progress = 'text')

# Plot TWDTW classification results
plot(r_lucc, type = "map")

# Assess classification
twdtw_assess <-
  twdtwAssess(object = r_lucc, y = validation_samples,
```

```
                proj4string = proj_str, conf.int = .95, rm.nosample = TRUE)

# Plot map accuracy
plot(twdtw_assess, type = "accuracy")

# Plot area uncertainty
plot(twdtw_assess, type = "area")

# Plot misclassified samples
plot(twdtw_assess, type = "map", samples = "incorrect")

# Get latex table with error matrix
twdtwXtable(twdtw_assess, table.type = "matrix")

# Get latex table with error accuracy
twdtwXtable(twdtw_assess, table.type = "accuracy")

# Get latex table with area uncertainty
twdtwXtable(twdtw_assess, table.type = "area")


## End(Not run)
```

`plotMatches` *Plotting matching points*

Description

Method for plotting the matching points from TWDTW analysis.

Usage

```
plotMatches(x, timeseries.labels = 1, patterns.labels = NULL, k = 1,
  attr = 1, shift = 0.5, show.dist = FALSE)
```

Arguments

`x`	An object of class `twdtwMatches`.
`timeseries.labels`	the label or index of the time series. Default is 1.
`patterns.labels`	a vector with labels of the patterns. If not declared the function will plot one alignment for each pattern.
`k`	A positive integer. The index of the last alignment to include in the plot. If not declared the function will plot the best match for each pattern.
`attr`	An integer or character vector indicating the attribute for plotting. Default is 1.
`shift`	A number, it shifts the pattern position in the x direction. Default is 0.5.
`show.dist`	show the distance for each alignment. Default is FALSE.

Value

A ggplot object.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwMatches-class, twdtwApply, plotPaths, plotCostMatrix, plotAlignments, and plotClassification.

Examples

```
log_fun = logisticWeight(-0.1, 100)
ts = twdtwTimeSeries(MOD13Q1.ts.list)
patt = twdtwTimeSeries(MOD13Q1.patterns.list)
mat1 = twdtwApply(x=ts, y=patt, weight.fun=log_fun, keep=TRUE)

plotMatches(mat1)

plotMatches(mat1, patterns.labels="Soybean", k=4)

plotMatches(mat1, patterns.labels=c("Soybean","Maize"), k=4)

plotMatches(mat1, patterns.labels=c("Soybean","Cotton"), k=c(3,1))
```

---

plotPaths *Plotting paths*

Description

Method for plotting low cost paths in the TWDTW cost matrix.

Usage

```
plotPaths(x, timeseries.labels = NULL, patterns.labels = NULL, k = NULL)
```

Arguments

x	An object of class twdtwMatches.
timeseries.labels	the label or index of the time series. Default is 1.
patterns.labels	a vector with labels of the patterns. If not declared the function will plot one alignment for each pattern.
k	A positive integer. The index of the last alignment to include in the plot. If not declared the function will plot all low cost paths.

Value

A ggplot object.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwMatches-class, twdtwApply, plotAlignments, plotCostMatrix, plotMatches, and plotClassification.

Examples

```
log_fun = logisticWeight(-0.1, 100)
ts = twdtwTimeSeries(MOD13Q1.ts.list)
patt = twdtwTimeSeries(MOD13Q1.patterns.list)
mat1 = twdtwApply(x=ts, y=patt, weight.fun=log_fun, keep=TRUE)

plotPaths(mat1)

plotPaths(mat1, patterns.labels="Soybean", k=1:2)

plotPaths(mat1, patterns.labels=c("Maize","Cotton"), k=2)
```

---

plotPatterns *Plotting temporal patterns*

Description

Method for plotting the temporal patterns.

Usage

```
plotPatterns(x, labels = NULL, attr, year = 2005)
```

Arguments

x	An object of class twdtwTimeSeries, zoo, or list of zoo.
labels	a vector with labels of the time series. If not declared the function will plot all time series.
attr	An integer vector or character vector indicating the attribute for plotting. If not declared the function will plot all attributes.
year	An integer. The base year to shift the dates of the time series to. If NULL then it does not shif the time series. Default is 2005.

Value

A ggplot object.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwTimeSeries-class and plotTimeSeries

Examples

```
patt = twdtwTimeSeries(MOD13Q1.patterns.list)
plotPatterns(patt)
plotPatterns(patt, attr="evi")
```

---

plotTimeSeries *Plotting time series*

Description

Method for plotting the temporal patterns.

Usage

```
plotTimeSeries(x, labels = NULL, attr)
```

Arguments

x	An object of class twdtwTimeSeries, zoo, or list of zoo.
labels	a vector with labels of the time series. If missing, all elements in the list will be plotted (up to a maximum of 16).
attr	An integer vector or character vector indicating the attribute for plotting. If not declared the function will plot all attributes.

Value

A ggplot object.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwTimeSeries-class and plotPatterns

Examples

```
ts = twdtwTimeSeries(MOD13Q1.ts.list)
plotTimeSeries(ts)
plotTimeSeries(ts, attr="evi")
```

`resampleTimeSeries` *Resample time series*

Description

resample time series in the same object to have the same the length.

Usage

```
resampleTimeSeries(object, length = NULL)

## S4 method for signature 'twdtwTimeSeries'
resampleTimeSeries(object, length = NULL)
```

Arguments

`object`	an object of class twdtwTimeSeries.
`length`	An integer. The number of samples to resample the time series. If not declared the length is set to the length of the longest time series.

Value

An object of class `twdtwTimeSeries` whose time series have the same number of samples (points).

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

`twdtwTimeSeries-class`, and `twdtwApply`

Examples

```
# Resampling time series from objects of class twdtwTimeSeries
patt = twdtwTimeSeries(MOD13Q1.patterns.list)
npatt = resampleTimeSeries(patt, length=46)
nrow(patt)
nrow(npatt)
```

shiftDates *Shift dates*

Description

This function shifts the dates of the time series to a given base year.

Usage

```
shiftDates(object, year = NULL)

## S4 method for signature 'twdtwTimeSeries'
shiftDates(object, year = NULL)

## S4 method for signature 'list'
shiftDates(object, year = NULL)

## S4 method for signature 'zoo'
shiftDates(object, year = NULL)
```

Arguments

object	twdtwTimeSeries objects, zoo objects or a list of zoo objects.
year	the base year to shit the time series.

Value

An object of the same class as the input object.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwTimeSeries-class

Examples

```
patt = twdtwTimeSeries(MOD13Q1.patterns.list)
npatt = shiftDates(patt, year=2005)
index(patt)
index(npatt)
```

subset *Subset time series*

Description

Get subsets from objects of class twdtw*.

Usage

```
## S4 method for signature 'twdtwTimeSeries'
subset(x, labels = NULL)

## S4 method for signature 'twdtwMatches'
subset(x, timeseries.labels = NULL,
  patterns.labels = NULL, k = NULL)

## S4 method for signature 'twdtwRaster'
subset(x, e = NULL, layers = NULL)
```

Arguments

x	An objects of class twdtw*.
labels	character vector with time series labels.
timeseries.labels	a vector with labels of the time series.
patterns.labels	a vector with labels of the patterns.
k	A positive integer. The index of the last alignment to include in the subset.
e	An extent object, or any object from which an Extent object can be extracted. See crop for details.
layers	a vector with the names of the twdtwRaster object to include in the subset.

Value

an object of class twdtw*.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwRaster-class, twdtwTimeSeries-class, and twdtwMatches-class

Examples

```
# Getting time series from objects of class twdtwTimeSeries
ts = twdtwTimeSeries(MOD13Q1.ts.list)
ts = subset(ts, 2)
ts
# Getting time series from objects of class twdtwTimeSeries
patt = twdtwTimeSeries(MOD13Q1.patterns.list)
mat = twdtwApply(x=ts, y=patt, weight.fun=logisticWeight(-0.1,100))
mat = subset(mat, k=4)

## This example creates a twdtwRaster object and extract time series from it.

# Creating objects of class twdtwRaster with evi and ndvi time series
evi = brick(system.file("lucc_MT/data/evi.tif", package="dtwSat"))
ndvi = brick(system.file("lucc_MT/data/ndvi.tif", package="dtwSat"))
timeline = scan(system.file("lucc_MT/data/timeline", package="dtwSat"), what="date")
rts = twdtwRaster(evi, ndvi, timeline=timeline)

rts_evi = subset(rts, layers="evi")

field_samples = read.csv(system.file("lucc_MT/data/samples.csv", package="dtwSat"))
prj_string = scan(system.file("lucc_MT/data/samples_projection", package="dtwSat"),
                  what = "character")

# Extract time series
ts_evi = getTimeSeries(rts_evi, y = field_samples, proj4string = prj_string)

# subset all labels = "Forest"
ts_forest = subset(ts_evi, labels="Forest")
```

---

twdtwApply *Apply TWDTW analysis*

Description

This function performs a multidimensional Time-Weighted DTW analysis and retrieves the matches between the temporal patterns and a set of time series [1].

Usage

```
twdtwApply(x, y, resample = TRUE, length = NULL, weight.fun = NULL,
  dist.method = "Euclidean", step.matrix = symmetric1, n = NULL,
  span = NULL, min.length = 0, theta = 0.5, ...)

## S4 method for signature 'twdtwTimeSeries'
twdtwApply(x, y, resample, length, weight.fun,
  dist.method, step.matrix, n, span, min.length, theta, keep = FALSE, ...)
```

```
## S4 method for signature 'twdtwRaster'
twdtwApply(x, y, resample, length, weight.fun,
  dist.method, step.matrix, n, span, min.length, theta, breaks = NULL,
  from = NULL, to = NULL, by = NULL, overlap = 0.5, filepath = "",
  ...)
```

Arguments

x	an object of class twdtw*. This is the target time series. Usually, it is a set of unclassified time series.
y	an object of class twdtwTimeSeries. The temporal patterns.
resample	resample the patterns to have the same length. Default is TRUE. See resample-TimeSeries for details.
length	An integer. Patterns length used with patterns.length. If not declared the length of the output patterns will be the length of the longest pattern.
weight.fun	A function. Any function that receive and performs a computation on a matrix. The function receives a matrix of time differences in days and returns a matrix of time-weights. If not declared the time-weight is zero. In this case the function runs the standard version of the dynamic time warping. See details.
dist.method	A character. Method to derive the local cost matrix. Default is "Euclidean" see dist in package **proxy**.
step.matrix	see stepPattern in package **dtw** [2].
n	An integer. The maximun number of matches to perform. NULL will return all matches.
span	A number. Span between two matches, *i.e.* the minimum interval between two matches, for details see [3]. If not declared it removes all overlapping matches of the same pattern. To include overlapping matches of the same pattern use span=0.
min.length	A number between 0 an 1. This argument removes the over fittings. Minimum length after warping. Percentage of the original pattern length. Default is 0.5, meaning that the matching cannot be shorter than half of the pattern length.
theta	numeric between 0 and 1. The weight of the time for the TWDTW computation. Use theta=0 to cancel the time-weight, *i.e.* to run the original DTW algorithm. Default is 0.5, meaning that the time has the same weight as the curve shape in the TWDTW analysis.
...	arguments to pass to writeRaster and pbCreate
keep	preserves the cost matrix, inputs, and other internal structures. Default is FALSE. For plot methods use keep=TRUE.
breaks	A vector of class Dates. This replaces the arguments from, to, and by.
from	A character or Dates object in the format "yyyy-mm-dd".
to	A character or Dates object in the format "yyyy-mm-dd".
by	A character with the intevals size, *e.g.* "6 month".
overlap	A number between 0 and 1. The minimum overlapping between one match and the interval of classification. Default is 0.5, *i.e.* an overlap minimum of 50%.

`filepath` A character. The path to save the raster with results. If not informed the function saves in the current work directory.

Details

The linear `linearWeight` and `logisticWeight` weight functions can be passed to `twdtwApply` through the argument `weight.fun`. This will add a time-weight to the dynamic time warping analysis. The time weight creates a global constraint useful to analyse time series with phenological cycles of vegetation that are usually bound to seasons. In previous studies by [1] the logistic weight had better results than the linear for land cover classification. See [1] for details about the method.

Value

An object of class twdtw*.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

References

[1] Maus V, Camara G, Cartaxo R, Sanchez A, Ramos FM, de Queiroz, GR. (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, vol.9, no.8, pp.3729-3739.

[2] Giorgino, T. (2009). Computing and Visualizing Dynamic Time Warping Alignments in R: The dtw Package. Journal of Statistical Software, 31, 1-24.

[3] Muller, M. (2007). Dynamic Time Warping. In Information Retrieval for Music and Motion (pp. 79-84). London: Springer London, Limited.

See Also

`twdtwMatches-class`, `twdtwTimeSeries-class`, `twdtwRaster-class`, `getTimeSeries`, and `createPatterns`

Examples

```
# Applying TWDTW analysis to objects of class twdtwTimeSeries
log_fun = logisticWeight(-0.1, 100)
ts = twdtwTimeSeries(MOD13Q1.ts.list)
patt = twdtwTimeSeries(MOD13Q1.patterns.list)
mat1 = twdtwApply(x=ts, y=patt, weight.fun=log_fun)
mat1

## Not run:
# Parallel processin
require(parallel)
mat_list = mclapply(as.list(ts), mc.cores=2, FUN=twdtwApply, y=patt, weight.fun=log_fun)
mat2 = twdtwMatches(alignments=mat_list)

## End(Not run)
## Not run:
```

```
# Example of TWDTW analysis using raster files
library(dtwSat)
library(caret)

# Load raster data
evi  <- brick(system.file("lucc_MT/data/evi.tif",  package = "dtwSat"))
ndvi <- brick(system.file("lucc_MT/data/ndvi.tif", package = "dtwSat"))
red  <- brick(system.file("lucc_MT/data/red.tif",  package = "dtwSat"))
blue <- brick(system.file("lucc_MT/data/blue.tif", package = "dtwSat"))
nir  <- brick(system.file("lucc_MT/data/nir.tif",  package = "dtwSat"))
mir  <- brick(system.file("lucc_MT/data/mir.tif",  package = "dtwSat"))
doy  <- brick(system.file("lucc_MT/data/doy.tif",  package = "dtwSat"))
timeline <-
  scan(system.file("lucc_MT/data/timeline", package = "dtwSat"), what="date")

# Create raster time series
rts <- twdtwRaster(evi, ndvi, red, blue, nir, mir, timeline = timeline, doy = doy)

# Load field samples and projection
field_samples <-
  read.csv(system.file("lucc_MT/data/samples.csv", package = "dtwSat"))
proj_str <-
  scan(system.file("lucc_MT/data/samples_projection", package = "dtwSat"),
       what = "character")

# Split samples for training (10%) and validation (90%) using stratified sampling
set.seed(1)
I <- unlist(createDataPartition(field_samples$label, p = 0.1))
training_samples <- field_samples[I, ]
validation_samples <- field_samples[-I, ]

# Get time series form raster
training_ts <- getTimeSeries(rts, y = training_samples, proj4string = proj_str)
validation_ts <- getTimeSeries(rts, y = validation_samples, proj4string = proj_str)

# Create temporal patterns
temporal_patterns <- createPatterns(training_ts, freq = 8, formula = y ~ s(x))

# Set TWDTW weight function
log_fun <- logisticWeight(-0.1, 50)

# Run serial TWDTW analysis
r_twdtw <-
  twdtwApply(x = rts, y = temporal_patterns, weight.fun = log_fun, progress = 'text')

# or Run parallel TWDTW analysis
beginCluster()
r_twdtw <-
  twdtwApplyParallel(x = rts, y = temporal_patterns, weight.fun = log_fun, progress = 'text')
endCluster()

# Plot TWDTW distances for the first year
```

```
plot(r_twdtw, type = "distance", time.levels = 1)

# Classify raster based on the TWDTW analysis
r_lucc <- twdtwClassify(r_twdtw, progress = 'text')

# Plot TWDTW classification results
plot(r_lucc, type = "map")

# Assess classification
twdtw_assess <-
  twdtwAssess(object = r_lucc, y = validation_samples,
              proj4string = proj_str, conf.int = .95, rm.nosample = TRUE)

# Plot map accuracy
plot(twdtw_assess, type = "accuracy")

# Plot area uncertainty
plot(twdtw_assess, type = "area")

# Plot misclassified samples
plot(twdtw_assess, type = "map", samples = "incorrect")

# Get latex table with error matrix
twdtwXtable(twdtw_assess, table.type = "matrix")

# Get latex table with error accuracy
twdtwXtable(twdtw_assess, table.type = "accuracy")

# Get latex table with area uncertainty
twdtwXtable(twdtw_assess, table.type = "area")


## End(Not run)
```

`twdtwApplyParallel` *Apply TWDTW analysis to twdtwRaster using parallel processing*

Description

This function performs a multidimensional Time-Weighted DTW analysis and retrieves the matches between the temporal patterns and a set of time series [1].

Usage

```
twdtwApplyParallel(x, y, resample = TRUE, length = NULL,
  weight.fun = NULL, dist.method = "Euclidean", step.matrix = symmetric1,
  n = NULL, span = NULL, min.length = 0, theta = 0.5, ...)

## S4 method for signature 'twdtwRaster'
```

```
twdtwApplyParallel(x, y, resample, length, weight.fun,
  dist.method, step.matrix, n, span, min.length, theta, breaks = NULL,
  from = NULL, to = NULL, by = NULL, overlap = 0.5, filepath = "",
  ...)
```

Arguments

x	an object of class twdtw*. This is the target time series. Usually, it is a set of unclassified time series.
y	an object of class twdtwTimeSeries. The temporal patterns.
resample	resample the patterns to have the same length. Default is TRUE. See resample-TimeSeries for details.
length	An integer. Patterns length used with patterns.length. If not declared the length of the output patterns will be the length of the longest pattern.
weight.fun	A function. Any function that receive and performs a computation on a matrix. The function receives a matrix of time differences in days and returns a matrix of time-weights. If not declared the time-weight is zero. In this case the function runs the standard version of the dynamic time warping. See details.
dist.method	A character. Method to derive the local cost matrix. Default is "Euclidean" see dist in package **proxy**.
step.matrix	see stepPattern in package **dtw** [2].
n	An integer. The maximun number of matches to perform. NULL will return all matches.
span	A number. Span between two matches, *i.e.* the minimum interval between two matches, for details see [3]. If not declared it removes all overlapping matches of the same pattern. To include overlapping matches of the same pattern use span=0.
min.length	A number between 0 an 1. This argument removes the over fittings. Minimum length after warping. Percentage of the original pattern length. Default is 0.5, meaning that the matching cannot be shorter than half of the pattern length.
theta	numeric between 0 and 1. The weight of the time for the TWDTW computation. Use theta=0 to cancel the time-weight, *i.e.* to run the original DTW algorithm. Default is 0.5, meaning that the time has the same weight as the curve shape in the TWDTW analysis.
...	arguments to pass to writeRaster and pbCreate
breaks	A vector of class Dates. This replaces the arguments from, to, and by.
from	A character or Dates object in the format "yyyy-mm-dd".
to	A character or Dates object in the format "yyyy-mm-dd".
by	A character with the intevals size, *e.g.* "6 month".
overlap	A number between 0 and 1. The minimum overlapping between one match and the interval of classification. Default is 0.5, *i.e.* an overlap minimum of 50%.
filepath	A character. The path to save the raster with results. If not informed the function saves in the current work directory.

Details

The linear `linearWeight` and `logisticWeight` weight functions can be passed to `twdtwApply` through the argument `weight.fun`. This will add a time-weight to the dynamic time warping analysis. The time weight creates a global constraint useful to analyse time series with phenological cycles of vegetation that are usually bound to seasons. In previous studies by [1] the logistic weight had better results than the linear for land cover classification. See [1] for details about the method.

Value

An object of class twdtwRaster.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

References

[1] Maus V, Camara G, Cartaxo R, Sanchez A, Ramos FM, de Queiroz, GR. (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, vol.9, no.8, pp.3729-3739.

[2] Giorgino, T. (2009). Computing and Visualizing Dynamic Time Warping Alignments in R: The dtw Package. Journal of Statistical Software, 31, 1-24.

[3] Muller, M. (2007). Dynamic Time Warping. In Information Retrieval for Music and Motion (pp. 79-84). London: Springer London, Limited.

See Also

`twdtwRaster-class`, and `createPatterns`

Examples

```
## Not run:

# Example of TWDTW analysis using raster files
library(dtwSat)
library(caret)

# Load raster data
evi  <- brick(system.file("lucc_MT/data/evi.tif",  package = "dtwSat"))
ndvi <- brick(system.file("lucc_MT/data/ndvi.tif", package = "dtwSat"))
red  <- brick(system.file("lucc_MT/data/red.tif",  package = "dtwSat"))
blue <- brick(system.file("lucc_MT/data/blue.tif", package = "dtwSat"))
nir  <- brick(system.file("lucc_MT/data/nir.tif",  package = "dtwSat"))
mir  <- brick(system.file("lucc_MT/data/mir.tif",  package = "dtwSat"))
doy  <- brick(system.file("lucc_MT/data/doy.tif",  package = "dtwSat"))
timeline <-
  scan(system.file("lucc_MT/data/timeline", package = "dtwSat"), what="date")

# Create raster time series
rts <- twdtwRaster(evi, ndvi, red, blue, nir, mir, timeline = timeline, doy = doy)
```

```
# Load field samples and projection
field_samples <-
  read.csv(system.file("lucc_MT/data/samples.csv", package = "dtwSat"))
proj_str <-
  scan(system.file("lucc_MT/data/samples_projection", package = "dtwSat"),
       what = "character")

# Split samples for training (10%) and validation (90%) using stratified sampling
set.seed(1)
I <- unlist(createDataPartition(field_samples$label, p = 0.1))
training_samples <- field_samples[I, ]
validation_samples <- field_samples[-I, ]

# Get time series form raster
training_ts <- getTimeSeries(rts, y = training_samples, proj4string = proj_str)
validation_ts <- getTimeSeries(rts, y = validation_samples, proj4string = proj_str)

# Create temporal patterns
temporal_patterns <- createPatterns(training_ts, freq = 8, formula = y ~ s(x))

# Set TWDTW weight function
log_fun <- logisticWeight(-0.1, 50)

# Run serial TWDTW analysis
r_twdtw <-
  twdtwApply(x = rts, y = temporal_patterns, weight.fun = log_fun, progress = 'text')

# or Run parallel TWDTW analysis
beginCluster()
r_twdtw <-
  twdtwApplyParallel(x = rts, y = temporal_patterns, weight.fun = log_fun, progress = 'text')
endCluster()

# Plot TWDTW distances for the first year
plot(r_twdtw, type = "distance", time.levels = 1)

# Classify raster based on the TWDTW analysis
r_lucc <- twdtwClassify(r_twdtw, progress = 'text')

# Plot TWDTW classification results
plot(r_lucc, type = "map")

# Assess classification
twdtw_assess <-
  twdtwAssess(object = r_lucc, y = validation_samples,
              proj4string = proj_str, conf.int = .95, rm.nosample = TRUE)

# Plot map accuracy
plot(twdtw_assess, type = "accuracy")

# Plot area uncertainty
plot(twdtw_assess, type = "area")
```

```
# Plot misclassified samples
plot(twdtw_assess, type = "map", samples = "incorrect")

# Get latex table with error matrix
twdtwXtable(twdtw_assess, table.type = "matrix")

# Get latex table with error accuracy
twdtwXtable(twdtw_assess, table.type = "accuracy")

# Get latex table with area uncertainty
twdtwXtable(twdtw_assess, table.type = "area")


## End(Not run)
```

`twdtwAssess` *Assess TWDTW classification*

Description

Performs an accuracy assessment of the classified maps. The function returns Overall Accuracy, User's Accuracy, Produce's Accuracy, error matrix (confusion matrix), and estimated area according to [1-2]. The function returns the metrics for each time interval and a summary considering all classified intervals.

Usage

```
## S4 method for signature 'twdtwRaster'
twdtwAssess(object, y, labels = NULL,
  id.labels = NULL, proj4string = NULL, conf.int = 0.95,
  rm.nosample = TRUE)

## S4 method for signature 'data.frame'
twdtwAssess(object, area, conf.int = 0.95,
  rm.nosample = TRUE)

## S4 method for signature 'table'
twdtwAssess(object, area, conf.int = 0.95,
  rm.nosample = TRUE)

## S4 method for signature 'matrix'
twdtwAssess(object, area, conf.int = 0.95,
  rm.nosample = TRUE)

## S4 method for signature 'twdtwMatches'
twdtwAssess(object, area, conf.int = 0.95,
  rm.nosample = TRUE)
```

Arguments

object	an object of class twdtwRaster resulting from the classification, i.e. twdtwClassify. The argument can also receive an error matrix (confusion matrix) in using the classes data.frame or table. In this case the user must inform the area for each class to the argument area.
y	a data.frame whose attributes are: longitude, latitude, the start "from" and the end "to" of the time interval for each sample. This can also be a SpatialPointsDataFrame whose attributes are the start "from" and the end "to" of the time interval. If missing "from" and/or "to", they are set to the time range of the object.
labels	character vector with time series labels. For signature twdtwRaster this argument can be used to set the labels for each sample in y, or it can be combined with id.labels to select samples with a specific label.
id.labels	a numeric or character with an column name from y to be used as samples labels. Optional.
proj4string	projection string, see CRS-class. Used if y is a data.frame.
conf.int	specifies the confidence level (0-1).
rm.nosample	if sum of columns and sum of rows of the error matrix are zero then remove class. Default is TRUE.
area	a numeric vector with the area for each class if the argument object is an error matrix (confusion matrix). If object is twdtwMatches area can be either a vector with the area of each classified object, or a single number if the objects are single pixels.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

References

[1] Olofsson, P., Foody, G.M., Stehman, S.V., Woodcock, C.E. (2013). Making better use of accuracy data in land change studies: Estimating accuracy and area and quantifying uncertainty using stratified estimation. Remote Sensing of Environment, 129, pp.122-131.

[2] Olofsson, P., Foody G.M., Herold M., Stehman, S.V., Woodcock, C.E., Wulder, M.A. (2014) Good practices for estimating area and assessing accuracy of land change. Remote Sensing of Environment, 148, pp. 42-57.

See Also

twdtwClassify, twdtwAssessment, and twdtwXtable.

Examples

```
## Not run:

# Example of TWDTW analysis using raster files
library(dtwSat)
library(caret)
```

```
# Load raster data
evi  <- brick(system.file("lucc_MT/data/evi.tif",  package = "dtwSat"))
ndvi <- brick(system.file("lucc_MT/data/ndvi.tif", package = "dtwSat"))
red  <- brick(system.file("lucc_MT/data/red.tif",  package = "dtwSat"))
blue <- brick(system.file("lucc_MT/data/blue.tif", package = "dtwSat"))
nir  <- brick(system.file("lucc_MT/data/nir.tif",  package = "dtwSat"))
mir  <- brick(system.file("lucc_MT/data/mir.tif",  package = "dtwSat"))
doy  <- brick(system.file("lucc_MT/data/doy.tif",  package = "dtwSat"))
timeline <-
  scan(system.file("lucc_MT/data/timeline", package = "dtwSat"), what="date")

# Create raster time series
rts <- twdtwRaster(evi, ndvi, red, blue, nir, mir, timeline = timeline, doy = doy)

# Load field samples and projection
field_samples <-
  read.csv(system.file("lucc_MT/data/samples.csv", package = "dtwSat"))
proj_str <-
  scan(system.file("lucc_MT/data/samples_projection", package = "dtwSat"),
       what = "character")

# Split samples for training (10%) and validation (90%) using stratified sampling
set.seed(1)
I <- unlist(createDataPartition(field_samples$label, p = 0.1))
training_samples <- field_samples[I, ]
validation_samples <- field_samples[-I, ]

# Get time series form raster
training_ts <- getTimeSeries(rts, y = training_samples, proj4string = proj_str)
validation_ts <- getTimeSeries(rts, y = validation_samples, proj4string = proj_str)

# Create temporal patterns
temporal_patterns <- createPatterns(training_ts, freq = 8, formula = y ~ s(x))

# Set TWDTW weight function
log_fun <- logisticWeight(-0.1, 50)

# Run serial TWDTW analysis
r_twdtw <-
  twdtwApply(x = rts, y = temporal_patterns, weight.fun = log_fun, progress = 'text')

# or Run parallel TWDTW analysis
beginCluster()
r_twdtw <-
  twdtwApplyParallel(x = rts, y = temporal_patterns, weight.fun = log_fun, progress = 'text')
endCluster()

# Plot TWDTW distances for the first year
plot(r_twdtw, type = "distance", time.levels = 1)

# Classify raster based on the TWDTW analysis
r_lucc <- twdtwClassify(r_twdtw, progress = 'text')
```

```
# Plot TWDTW classification results
plot(r_lucc, type = "map")

# Assess classification
twdtw_assess <-
  twdtwAssess(object = r_lucc, y = validation_samples,
              proj4string = proj_str, conf.int = .95, rm.nosample = TRUE)

# Plot map accuracy
plot(twdtw_assess, type = "accuracy")

# Plot area uncertainty
plot(twdtw_assess, type = "area")

# Plot misclassified samples
plot(twdtw_assess, type = "map", samples = "incorrect")

# Get latex table with error matrix
twdtwXtable(twdtw_assess, table.type = "matrix")

# Get latex table with error accuracy
twdtwXtable(twdtw_assess, table.type = "accuracy")

# Get latex table with area uncertainty
twdtwXtable(twdtw_assess, table.type = "area")


## End(Not run)


# Total mapped area by class. Data from [1]
area = c(A = 22353, B = 1122543, C = 610228)

# Error matrix, columns (Reference) rows (Map)
x =
    rbind(
         c( 97, 0,   3),
         c(  3, 279,  18),
         c(  2,   1,  97)
   )

table_assess = twdtwAssess(x, area, conf.int = .95)

table_assess

plot(table_assess, type="area", perc=FALSE)

plot(table_assess, type="accuracy")

## Not run:

# Example of TWDTW analysis using raster files
```

```
library(dtwSat)
library(caret)

# Load raster data
evi  <- brick(system.file("lucc_MT/data/evi.tif",  package = "dtwSat"))
ndvi <- brick(system.file("lucc_MT/data/ndvi.tif", package = "dtwSat"))
red  <- brick(system.file("lucc_MT/data/red.tif",  package = "dtwSat"))
blue <- brick(system.file("lucc_MT/data/blue.tif", package = "dtwSat"))
nir  <- brick(system.file("lucc_MT/data/nir.tif",  package = "dtwSat"))
mir  <- brick(system.file("lucc_MT/data/mir.tif",  package = "dtwSat"))
doy  <- brick(system.file("lucc_MT/data/doy.tif",  package = "dtwSat"))
timeline <-
  scan(system.file("lucc_MT/data/timeline", package = "dtwSat"), what="date")

# Create raster time series
rts <- twdtwRaster(evi, ndvi, red, blue, nir, mir, timeline = timeline, doy = doy)

# Load field samples and projection
field_samples <-
  read.csv(system.file("lucc_MT/data/samples.csv", package = "dtwSat"))
proj_str <-
  scan(system.file("lucc_MT/data/samples_projection", package = "dtwSat"),
       what = "character")

# Split samples for training (10%) and validation (90%) using stratified sampling
set.seed(1)
I <- unlist(createDataPartition(field_samples$label, p = 0.1))
training_samples <- field_samples[I, ]
validation_samples <- field_samples[-I, ]

# Get time series form raster
training_ts <- getTimeSeries(rts, y = training_samples, proj4string = proj_str)
validation_ts <- getTimeSeries(rts, y = validation_samples, proj4string = proj_str)

# Create temporal patterns
temporal_patterns = createPatterns(training_ts, freq = 8, formula = y ~ s(x))

# Set TWDTW weight function
log_fun <- logisticWeight(-0.1, 50)

# Run serial TWDTW analysis
r_twdtw <- twdtwApply(x = validation_ts,
                      y = temporal_patterns, weight.fun = log_fun)

# Classify raster based on the TWDTW analysis
r_lucc <- twdtwClassify(x = r_twdtw, overlap = 0.5)

# Accuracy assessment
twdtw_assess <- twdtwAssess(r_lucc, area = 53664.67, conf.int = .95)

# Plot map accuracy
plot(twdtw_assess, type = "accuracy")
```

```
# Plot area uncertainty
plot(twdtw_assess, type = "area")

# Get latex table with error matrix
twdtwXtable(twdtw_assess, table.type = "matrix")

# Get latex table with error accuracy
twdtwXtable(twdtw_assess, table.type = "accuracy")

# Get latex table with area uncertainty
twdtwXtable(twdtw_assess, table.type = "area")


## End(Not run)
```

`twdtwAssessment-class` *class "twdtwAssessment"*

Description

This class stores the map assessment metrics.

Usage

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

Arguments

`object`	an object of class twdtwAssessment.

Details

If the twdtwRaster is unprojected (longitude/latitude) the estimated area is sum of the approximate surface area in km2 of each cell (pixel). If the twdtwRaster is projected the estimated area is calculated using the the pixel resolution in the map unit.

Slots

`accuracySummary`**:** Overall Accuracy, User's Accuracy, Produce's Accuracy, Error Matrix (confusion matrix), and Estimated Area, considering all time periods.

`accuracyByPeriod`**:** Overall Accuracy, User's Accuracy, Produce's Accuracy, Error Matrix (confusion matrix), and Estimated Area, for each time periods independently from each other.

`data`**:** A `SpatialPointsDataFrame` with sample ID, period, date from, date to, reference labels, predicted labels, and TWDTW distance.

`map`**:** A `twdtwRaster` with the raster maps.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwClassify, twdtwAssess, and twdtwXtable.

---

twdtwClassify *Classify time series*

Description

This function classifies the intervals of a time series based on the TWDTW results.

Usage

```
twdtwClassify(x, ...)

## S4 method for signature 'twdtwMatches'
twdtwClassify(x, patterns.labels = NULL,
  from = NULL, to = NULL, by = NULL, breaks = NULL, overlap = 0.5,
  thresholds = Inf, fill = "unclassified")

## S4 method for signature 'twdtwRaster'
twdtwClassify(x, patterns.labels = NULL,
  thresholds = Inf, fill = 255, filepath = "", ...)
```

Arguments

x	an object of class twdtw*. This is the target time series. Usually, it is a set of unclassified time series.
...	arguments to pass to specifique methods for each twdtw* class and other arguments to pass to writeRaster and pbCreate.
patterns.labels	a vector with labels of the patterns.
from	A character or Dates object in the format "yyyy-mm-dd".
to	A character or Dates object in the format "yyyy-mm-dd".
by	A character with the intevals size, *e.g.* "6 month".
breaks	A vector of class Dates. This replaces the arguments from, to, and by.
overlap	A number between 0 and 1. The minimum overlapping between one match and the interval of classification. Default is 0.5, *i.e.* an overlap minimum of 50%.
thresholds	A numeric vector the same length as patterns.labels. The TWDTW dissimilarity thresholds, i.e. the maximum TWDTW cost for consideration in the classification. Default is Inf for all patterns.labels.

fill	a character or value to fill the classification gaps. For signature twdtwTimeSeries the default is fill="unclassified", and for signature twdtwRaster the default is fill="unclassified".
filepath	A character. The path to save the raster with results. If not informed the function saves in the same directory as the input time series raster.

Value

An object of class twdtw*.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwApply, twdtwMatches-class, twdtwTimeSeries-class, and twdtwRaster-class,

Examples

```
## Not run:

# Example of TWDTW analysis using raster files
library(dtwSat)
library(caret)

# Load raster data
evi  <- brick(system.file("lucc_MT/data/evi.tif",  package = "dtwSat"))
ndvi <- brick(system.file("lucc_MT/data/ndvi.tif", package = "dtwSat"))
red  <- brick(system.file("lucc_MT/data/red.tif",  package = "dtwSat"))
blue <- brick(system.file("lucc_MT/data/blue.tif", package = "dtwSat"))
nir  <- brick(system.file("lucc_MT/data/nir.tif",  package = "dtwSat"))
mir  <- brick(system.file("lucc_MT/data/mir.tif",  package = "dtwSat"))
doy  <- brick(system.file("lucc_MT/data/doy.tif",  package = "dtwSat"))
timeline <-
  scan(system.file("lucc_MT/data/timeline", package = "dtwSat"), what="date")

# Create raster time series
rts <- twdtwRaster(evi, ndvi, red, blue, nir, mir, timeline = timeline, doy = doy)

# Load field samples and projection
field_samples <-
  read.csv(system.file("lucc_MT/data/samples.csv", package = "dtwSat"))
proj_str <-
  scan(system.file("lucc_MT/data/samples_projection", package = "dtwSat"),
       what = "character")

# Split samples for training (10%) and validation (90%) using stratified sampling
set.seed(1)
I <- unlist(createDataPartition(field_samples$label, p = 0.1))
training_samples <- field_samples[I, ]
validation_samples <- field_samples[-I, ]
```

```
# Get time series form raster
training_ts <- getTimeSeries(rts, y = training_samples, proj4string = proj_str)
validation_ts <- getTimeSeries(rts, y = validation_samples, proj4string = proj_str)

# Create temporal patterns
temporal_patterns = createPatterns(training_ts, freq = 8, formula = y ~ s(x))

# Set TWDTW weight function
log_fun <- logisticWeight(-0.1, 50)

# Run serial TWDTW analysis
r_twdtw <- twdtwApply(x = validation_ts,
                      y = temporal_patterns, weight.fun = log_fun)

# Classify raster based on the TWDTW analysis
r_lucc <- twdtwClassify(x = r_twdtw, overlap = 0.5)

# Accuracy assessment
twdtw_assess <- twdtwAssess(r_lucc, area = 53664.67, conf.int = .95)

# Plot map accuracy
plot(twdtw_assess, type = "accuracy")

# Plot area uncertainty
plot(twdtw_assess, type = "area")

# Get latex table with error matrix
twdtwXtable(twdtw_assess, table.type = "matrix")

# Get latex table with error accuracy
twdtwXtable(twdtw_assess, table.type = "accuracy")

# Get latex table with area uncertainty
twdtwXtable(twdtw_assess, table.type = "area")


## End(Not run)
## Not run:

# Example of TWDTW analysis using raster files
library(dtwSat)
library(caret)

# Load raster data
evi  <- brick(system.file("lucc_MT/data/evi.tif",  package = "dtwSat"))
ndvi <- brick(system.file("lucc_MT/data/ndvi.tif", package = "dtwSat"))
red  <- brick(system.file("lucc_MT/data/red.tif",  package = "dtwSat"))
blue <- brick(system.file("lucc_MT/data/blue.tif", package = "dtwSat"))
nir  <- brick(system.file("lucc_MT/data/nir.tif",  package = "dtwSat"))
mir  <- brick(system.file("lucc_MT/data/mir.tif",  package = "dtwSat"))
doy  <- brick(system.file("lucc_MT/data/doy.tif",  package = "dtwSat"))
timeline <-
```

```
  scan(system.file("lucc_MT/data/timeline", package = "dtwSat"), what="date")

# Create raster time series
rts <- twdtwRaster(evi, ndvi, red, blue, nir, mir, timeline = timeline, doy = doy)

# Load field samples and projection
field_samples <-
  read.csv(system.file("lucc_MT/data/samples.csv", package = "dtwSat"))
proj_str <-
  scan(system.file("lucc_MT/data/samples_projection", package = "dtwSat"),
       what = "character")

# Split samples for training (10%) and validation (90%) using stratified sampling
set.seed(1)
I <- unlist(createDataPartition(field_samples$label, p = 0.1))
training_samples <- field_samples[I, ]
validation_samples <- field_samples[-I, ]

# Get time series form raster
training_ts <- getTimeSeries(rts, y = training_samples, proj4string = proj_str)
validation_ts <- getTimeSeries(rts, y = validation_samples, proj4string = proj_str)

# Create temporal patterns
temporal_patterns <- createPatterns(training_ts, freq = 8, formula = y ~ s(x))

# Set TWDTW weight function
log_fun <- logisticWeight(-0.1, 50)

# Run serial TWDTW analysis
r_twdtw <-
  twdtwApply(x = rts, y = temporal_patterns, weight.fun = log_fun, progress = 'text')

# or Run parallel TWDTW analysis
beginCluster()
r_twdtw <-
  twdtwApplyParallel(x = rts, y = temporal_patterns, weight.fun = log_fun, progress = 'text')
endCluster()

# Plot TWDTW distances for the first year
plot(r_twdtw, type = "distance", time.levels = 1)

# Classify raster based on the TWDTW analysis
r_lucc <- twdtwClassify(r_twdtw, progress = 'text')

# Plot TWDTW classification results
plot(r_lucc, type = "map")

# Assess classification
twdtw_assess <-
  twdtwAssess(object = r_lucc, y = validation_samples,
              proj4string = proj_str, conf.int = .95, rm.nosample = TRUE)

# Plot map accuracy
```

```
plot(twdtw_assess, type = "accuracy")

# Plot area uncertainty
plot(twdtw_assess, type = "area")

# Plot misclassified samples
plot(twdtw_assess, type = "map", samples = "incorrect")

# Get latex table with error matrix
twdtwXtable(twdtw_assess, table.type = "matrix")

# Get latex table with error accuracy
twdtwXtable(twdtw_assess, table.type = "accuracy")

# Get latex table with area uncertainty
twdtwXtable(twdtw_assess, table.type = "area")


## End(Not run)
```

`twdtwCrossValidate` *Cross Validate temporal patterns*

Description

Splits the set of time series into training and validation and compute accuracy metrics. The function uses stratified sampling and a simple random sampling for each stratum. For each data partition this function performs a TWDTW analysis and returns the Overall Accuracy, User's Accuracy, Produce's Accuracy, error matrix (confusion matrix), and a `data.frame` with the classification (Predicted), the reference classes (Reference), and the results of the TWDTW analysis.

Usage

```
## S4 method for signature 'twdtwTimeSeries'
twdtwCrossValidate(object, times, p, ...)
```

Arguments

`object`	an object of class `twdtwTimeSeries`.
`times`	Number of partitions to create.
`p`	the percentage of data that goes to training. See `createDataPartition` for details.
`...`	Other arguments to be passed to `createPatterns` and to `twdtwApply`.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

Examples

```
## Not run:
# Data folder
data_folder = system.file("lucc_MT/data", package = "dtwSat")

# Read dates
dates = scan(paste(data_folder,"timeline", sep = "/"), what = "dates")

# Read raster time series
evi = brick(paste(data_folder,"evi.tif", sep = "/"))
raster_timeseries = twdtwRaster(evi, timeline = dates)

# Read field samples
field_samples = read.csv(paste(data_folder,"samples.csv", sep = "/"))
table(field_samples[["label"]])

# Read field samples projection
proj_str = scan(paste(data_folder,"samples_projection", sep = "/"),
     what = "character")

# Get sample time series from raster time series
field_samples_ts = getTimeSeries(raster_timeseries,
     y = field_samples, proj4string = proj_str)
field_samples_ts

# Run cross validation
set.seed(1)
# Define TWDTW weight function
log_fun = logisticWeight(alpha=-0.1, beta=50)
cross_validation = twdtwCrossValidate(field_samples_ts, times=3, p=0.1,
                          freq = 8, formula = y ~ s(x, bs="cc"), weight.fun = log_fun)
cross_validation

summary(cross_validation)

plot(cross_validation)

twdtwXtable(cross_validation)

twdtwXtable(cross_validation, show.overall=FALSE)


## End(Not run)
```

---

twdtwCrossValidation-class

class "twdtwCrossValidation"

Description

This class stores the results of the cross-validation.

Usage

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

## S4 method for signature 'twdtwCrossValidation'
summary(object, conf.int = 0.95, ...)
```

Arguments

object	an object of class twdtwCrossValidation.
conf.int	specifies the confidence level (0-1) for interval estimation of the population mean. For more details see mean_cl_boot.
...	Other arguments. Not used.

Slots

partitions: A list with the indices of time series used for training.

accuracy: A list with the accuracy and other TWDTW information for each data partitions.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwMatches-class, createPatterns, and twdtwApply.

Examples

```
## Not run:
# Data folder
data_folder = system.file("lucc_MT/data", package = "dtwSat")

# Read dates
dates = scan(paste(data_folder,"timeline", sep = "/"), what = "dates")

# Read raster time series
evi = brick(paste(data_folder,"evi.tif", sep = "/"))
raster_timeseries = twdtwRaster(evi, timeline = dates)

# Read field samples
field_samples = read.csv(paste(data_folder,"samples.csv", sep = "/"))
table(field_samples[["label"]])

# Read field samples projection
proj_str = scan(paste(data_folder,"samples_projection", sep = "/"),
     what = "character")

# Get sample time series from raster time series
field_samples_ts = getTimeSeries(raster_timeseries,
```

```
     y = field_samples, proj4string = proj_str)
field_samples_ts

# Run cross validation
set.seed(1)
# Define TWDTW weight function
log_fun = logisticWeight(alpha=-0.1, beta=50)
cross_validation = twdtwCrossValidate(field_samples_ts, times=3, p=0.1,
                          freq = 8, formula = y ~ s(x, bs="cc"), weight.fun = log_fun)
cross_validation

summary(cross_validation)

plot(cross_validation)


## End(Not run)
```

---

twdtwMatches-class *class "twdtwMatches"*

Description

Class for Time-Weighted Dynamic Time Warping results.

Usage

```
## S4 method for signature 'ANY'
twdtwMatches(timeseries = NULL, patterns = NULL,
  alignments = NULL)

## S4 method for signature 'twdtwMatches'
index(x)

## S4 method for signature 'twdtwMatches'
length(x)

## S4 method for signature 'twdtwMatches'
as.list(x)

## S4 method for signature 'twdtwRaster'
as.list(x)

## S4 method for signature 'twdtwMatches,ANY,ANY,ANY'
x[i, j, drop = TRUE]

## S4 method for signature 'twdtwMatches,numeric,ANY'
x[[i, j, drop = TRUE]]
```

```
## S4 method for signature 'twdtwMatches'
labels(object)

## S4 method for signature 'twdtwMatches'
show(object)

## S4 method for signature 'ANY'
is.twdtwMatches(x)
```

Arguments

timeseries	a twdtwTimeSeries object.
patterns	a twdtwTimeSeries object.
alignments	an object of class list with the TWDTW results with the same length as timeseries or a list of twdtwMatches.
x	an object of class twdtwMatches.
i	indices of the time series.
j	indices of the pattern.
drop	if TRUE returns a data.frame, if FALSE returns a list. Default is TRUE.
object	an object of class twdtwMatches.
labels	a vector with labels of the time series.
...	objects of class twdtwMatches.

Methods (by generic)

- twdtwMatches: Create object of class twdtwMatches.
- is.twdtwMatches: Check if the object belongs to the class twdtwMatches.

Slots

timeseries: An object of class twdtwTimeSeries-class with the satellite time series.

pattern: An object of class twdtwTimeSeries-class with the temporal patterns.

alignments: A list of TWDTW results with the same length as the timeseries. Each element in this list has the following results for each temporal pattern in patterns:
from: a vector with the starting dates of each match in the format "YYYY-MM-DD",
to: a vector with the ending dates of each match in the format "YYYY-MM-DD",
distance: a vector with TWDTW dissimilarity measure, and
K: the number of matches of the pattern.

This list might have additional elements: if keep=TRUE in the twdtwApply call the list is extended to include internal structures used during the TWDTW computation:
costMatrix: cumulative cost matrix,
directionMatrix: directions of steps that would be taken from each element of matrix,
startingMatrix: the starting points of each element of the matrix,
stepPattern: stepPattern used for the computation, see package dtw,

N: the length of the pattern,
M: the length of the time series timeseries,
timeWeight: time weight matrix,
localMatrix: local cost matrix,
matching: A list whose elements have the matching points for each match between pattern the time series, such that:
–index1: a vector with matching points of the pattern, and
–index2: a vector with matching points of the time series.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwApply, twdtwTimeSeries-class, and twdtwRaster-class

Examples

```
ts = twdtwTimeSeries(timeseries=MOD13Q1.ts.list)
patterns = twdtwTimeSeries(timeseries=MOD13Q1.patterns.list)
matches = twdtwApply(x = ts, y = patterns)
class(matches)
length(matches)
matches
# Creating objects of class twdtwMatches
ts  = twdtwTimeSeries(MOD13Q1.ts.list)
patt = twdtwTimeSeries(MOD13Q1.patterns.list)
mat = twdtwApply(ts, patt, weight.fun = logisticWeight(-0.1, 100))
mat = twdtwMatches(ts, patterns=patt, alignments=mat)
mat
```

---

twdtwRaster-class *class "twdtwRaster"*

Description

Class for set of satellite time series.

Usage

```
## S4 method for signature 'ANY'
twdtwRaster(..., timeline, doy = NULL, layers = NULL,
  labels = NULL, levels = NULL, filepath = NULL)

## S4 method for signature 'twdtwRaster'
dim(x)
```

```
## S4 method for signature 'twdtwRaster'
res(x)

## S4 method for signature 'twdtwRaster'
extent(x, y, ...)

## S4 method for signature 'twdtwRaster,ANY'
writeRaster(x, filepath = ".", ...)

## S4 method for signature 'twdtwRaster'
projection(x)

## S4 method for signature 'twdtwRaster'
ncol(x)

## S4 method for signature 'twdtwRaster'
nrow(x)

## S4 method for signature 'twdtwRaster'
nlayers(x)

## S4 method for signature 'twdtwRaster'
levels(x)

## S4 method for signature 'twdtwRaster'
layers(x)

## S4 method for signature 'twdtwRaster'
coverages(x)

## S4 method for signature 'twdtwRaster'
bands(x)

## S4 method for signature 'twdtwRaster'
names(x)

## S4 method for signature 'twdtwRaster'
index(x)

## S4 method for signature 'twdtwRaster'
length(x)

## S4 method for signature 'twdtwRaster,ANY,ANY,ANY'
x[i]

## S4 method for signature 'twdtwRaster,ANY,ANY'
x[[i]]
```

```
## S4 method for signature 'twdtwRaster'
labels(object)

## S4 method for signature 'twdtwRaster'
crop(x, y, ...)

## S4 method for signature 'twdtwRaster'
coordinates(obj, ...)

## S4 method for signature 'twdtwRaster'
extent(x, y, ...)

## S4 method for signature 'twdtwRaster'
show(object)

## S4 method for signature 'ANY'
is.twdtwRaster(x)

## S4 method for signature 'twdtwRaster'
projecttwdtwRaster(x, crs, ...)
```

Arguments

...	objects of class RasterBrick-class or RasterStack-class.
timeline	a vector with the dates of the satellite images in the format of "YYYY-MM-DD".
doy	A RasterBrick-class or RasterStack-class with a sequence of days of the year for each pixel. doy must have the same spatial and temporal extents as the Raster* objects passed to .... If doy is not informed then at least one Raster* object must be passed through ....
layers	a vector with the names of the Raster* objects passed to "...". If not informed the layers are set to the names of objects in "...".
labels	a vector of class character with labels of the values in the Raster* objects. This is useful for categorical Raster* values of land use classes.
levels	a vector of class numeric with levels of the values in the Raster* objects. This is useful for categorical Raster* values of land use classes.
filepath	A character. The path to save the raster time series. If informed the function saves a raster file for each Raster* object in the list, *i.e* one file for each time series. This way the function retrieves an list of RasterBrick-class. It is useful when the time series are originally stores in separated files. See details.
x	an object of class twdtwRaster.
y	Extent object, or any object from which an Extent object can be extracted.
i	indices of the time series.
object	an object of class twdtwRaster.
obj	object of cless twdtwRaster
crs	character or object of class 'CRS'. PROJ.4 description of the coordinate reference system. For other arguments and more details see projectRaster.

Details

The performance the functions twdtwApply and getTimeSeries is improved if the Raster* objects are connected to files with the whole time series for each attribute.

Methods (by generic)

- twdtwRaster: Create object of class twdtwRaster.
- is.twdtwRaster: Check if the object belongs to the class twdtwRaster.
- projecttwdtwRaster: project twdtwRaster object.

Slots

timeseries: A list of multi-layers Raster* objects with the satellite image time series.

timeline: A vector of class date with dates of the satellite images in timeseries.

layers: A vector of class character with the names of the Raster* objects.

labels: A vector of class factor with levels and labels of the values in the Raster* objects. This is useful for categorical Raster* values of land use classes.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwApply, getTimeSeries, twdtwMatches-class, and twdtwTimeSeries-class

Examples

```
# Creating new object of class twdtwTimeSeries
evi = brick(system.file("lucc_MT/data/evi.tif", package="dtwSat"))
timeline = scan(system.file("lucc_MT/data/timeline", package="dtwSat"), what="date")
rts = new("twdtwRaster", timeseries = evi, timeline = timeline)

## Not run:
# Creating objects of class twdtwRaster
evi = brick(system.file("lucc_MT/data/evi.tif", package="dtwSat"))
timeline = scan(system.file("lucc_MT/data/timeline", package="dtwSat"), what="date")
ts_evi = twdtwRaster(evi, timeline=timeline)

ndvi = brick(system.file("lucc_MT/data/ndvi.tif", package="dtwSat"))
blue = brick(system.file("lucc_MT/data/blue.tif", package="dtwSat"))
red = brick(system.file("lucc_MT/data/red.tif", package="dtwSat"))
nir = brick(system.file("lucc_MT/data/nir.tif", package="dtwSat"))
mir = brick(system.file("lucc_MT/data/mir.tif", package="dtwSat"))
doy = brick(system.file("lucc_MT/data/doy.tif", package="dtwSat"))
rts = twdtwRaster(doy, evi, ndvi, blue, red, nir, mir, timeline = timeline)

## End(Not run)
```

twdtwTimeSeries-class *class "twdtwTimeSeries"*

Description

Class for set of irregular time series.

Usage

```
## S4 method for signature 'ANY'
twdtwTimeSeries(..., labels = NULL)

## S4 method for signature 'twdtwTimeSeries'
dim(x)

## S4 method for signature 'twdtwTimeSeries'
index(x)

## S4 method for signature 'twdtwTimeSeries'
nrow(x)

## S4 method for signature 'twdtwTimeSeries'
ncol(x)

## S4 method for signature 'twdtwTimeSeries'
length(x)

## S4 method for signature 'twdtwTimeSeries'
as.list(x)

## S4 method for signature 'twdtwTimeSeries,ANY,ANY,ANY'
x[i]

## S4 method for signature 'twdtwTimeSeries,ANY,ANY'
x[[i]]

## S4 method for signature 'twdtwTimeSeries'
labels(object)

## S4 method for signature 'twdtwTimeSeries'
levels(x)

## S4 method for signature 'twdtwTimeSeries'
show(object)

## S4 method for signature 'ANY'
is.twdtwTimeSeries(x)
```

Arguments

...	twdtwTimeSeries objects, zoo objects or a list of zoo objects.
labels	a vector with labels of the time series.
x	an object of class twdtwTimeSeries.
i	indices of the time series.
object	an object of class twdtwTimeSeries.

Methods (by generic)

- twdtwTimeSeries: Create object of class twdtwTimeSeries.
- is.twdtwTimeSeries: Check if the object belongs to the class twdtwTimeSeries.

Slots

timeseries: A list of zoo objects.

labels: A vector of class factor with time series labels.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwMatches-class, twdtwRaster-class, getTimeSeries, and twdtwApply

Examples

```
# Creating new object of class twdtwTimeSeries
ptt = new("twdtwTimeSeries", timeseries = MOD13Q1.patterns.list,
          labels = names(MOD13Q1.patterns.list))
class(ptt)
labels(ptt)
levels(ptt)
length(ptt)
nrow(ptt)
ncol(ptt)
dim(ptt)
# Creating objects of class twdtwTimeSeries from zoo objects
ts = twdtwTimeSeries(MOD13Q1.ts)
ts

# Creating objects of class twdtwTimeSeries from list of zoo objects
patt = twdtwTimeSeries(MOD13Q1.patterns.list)
patt

# Joining objects of class twdtwTimeSeries
tsA = twdtwTimeSeries(MOD13Q1.ts.list[[1]], labels = "A")
tsB = twdtwTimeSeries(B = MOD13Q1.ts.list[[2]])
ts = twdtwTimeSeries(tsA, tsB, C=MOD13Q1.ts)
```

ts

`twdtwXtable` *Latex table from accuracy metrics*

Description

Creates Latex table from accuracy metrics

Usage

```
## S4 method for signature 'twdtwAssessment'
twdtwXtable(object, table.type = "accuracy",
  show.prop = TRUE, category.name = NULL, category.type = NULL,
  rotate.col = FALSE, time.labels = NULL, caption = NULL, digits = 2,
  show.footnote = TRUE, ...)

## S4 method for signature 'twdtwCrossValidation'
twdtwXtable(object, conf.int = 0.95,
  show.overall = TRUE, category.name = NULL, category.type = NULL,
  caption = NULL, digits = 2, show.footnote = TRUE, ...)
```

Arguments

`object`	an object of class twdtwAssessment.
`table.type`	table type, 'accuracy' for User's and Producer's Accuracy, 'errormatrix' for error matrix, and 'area' for area and uncertainty. Default is 'accuracy'.
`show.prop`	if TRUE shows the estimated proportion of area. Used with `table.type='accuracy'`. Default is TRUE.
`category.name`	a character vector defining the class names. If NULL then use the classe names in the object `x`. Default is NULL.
`category.type`	a character defining the categories type "numeric" or "letter", if NULL then use the class names. Default is NULL.
`rotate.col`	rotate class column names in latex table. Default is FALSE.
`time.labels`	a character or numeric for the time period or NULL to include all classified periods. Default is NULL.
`caption`	the table caption.
`digits`	number of digits to show.
`show.footnote`	show confidence interval in the footnote.
`...`	other arguments to pass to and `print.xtable`.
`conf.int`	specifies the confidence level (0-1).
`show.overall`	if TRUE shows the overall accuracy of the cross-validation. Default is TRUE.

Author(s)

Victor Maus, <vwmaus1@gmail.com>

See Also

twdtwAssess and twdtwAssessment.

Examples

```
## Not run:

# Create raster time series
evi = brick(system.file("lucc_MT/data/evi.tif", package="dtwSat"))
ndvi = brick(system.file("lucc_MT/data/ndvi.tif", package="dtwSat"))
red = brick(system.file("lucc_MT/data/red.tif", package="dtwSat"))
blue = brick(system.file("lucc_MT/data/blue.tif", package="dtwSat"))
nir = brick(system.file("lucc_MT/data/nir.tif", package="dtwSat"))
mir = brick(system.file("lucc_MT/data/mir.tif", package="dtwSat"))
doy = brick(system.file("lucc_MT/data/doy.tif", package="dtwSat"))
timeline = scan(system.file("lucc_MT/data/timeline", package="dtwSat"), what="date")
rts = twdtwRaster(evi, ndvi, red, blue, nir, mir, timeline = timeline, doy = doy)

# Read fiels samples
field_samples = read.csv(system.file("lucc_MT/data/samples.csv", package="dtwSat"))
proj_str = scan(system.file("lucc_MT/data/samples_projection",
                package="dtwSat"), what = "character")

# Split samples for training (10%) and validation (90%) using stratified sampling
library(caret)
set.seed(1)
I = unlist(createDataPartition(field_samples$label, p = 0.1))
training_samples = field_samples[I,]
validation_samples = field_samples[-I,]

# Create temporal patterns
training_ts = getTimeSeries(rts, y = training_samples, proj4string = proj_str)
temporal_patterns = createPatterns(training_ts, freq = 8, formula = y ~ s(x))

# Run TWDTW analysis for raster time series
log_fun = weight.fun=logisticWeight(-0.1,50)
r_twdtw = twdtwApply(x=rts, y=temporal_patterns, weight.fun=log_fun, format="GTiff",
                     overwrite=TRUE)

# Classify raster based on the TWDTW analysis
r_lucc = twdtwClassify(r_twdtw, format="GTiff", overwrite=TRUE)
plot(r_lucc)

# Assess classification
twdtw_assess = twdtwAssess(object = r_lucc, y = validation_samples,
                           proj4string = proj_str, conf.int=.95)
twdtw_assess
```

```
# Create latex tables
twdtwXtable(twdtw_assess, table.type="errormatrix", rotate.col=TRUE,
  caption="Error matrix", digits=2, comment=FALSE)
twdtwXtable(twdtw_assess, table.type="accuracy", category.type="letter",
  caption="Accuracy metrics.")
twdtwXtable(twdtw_assess, table.type="area", category.type="letter",
  digits = 0, caption="Area and uncertainty")


## End(Not run)
```

Index