Package 'rsMove'

August 19, 2017

Type Package

Title Remote Sensing for Movement Ecology

Version 0.2.1

Date 2017-08-19

URL https://github.com/RRemelgado/rsMove/tree/master/

BugReports https://github.com/RRemelgado/rsMove/issues/

Maintainer Ruben Remelgado <ruben.remelgado@uni-wuerzburg.de>

Description Tools to analyze animal tracking data with remote sensing.

LazyData TRUE

Imports raster, sp, caret, rgdal, gdalUtils, ggplot2, grDevices, ncdf4, RCurl, XML, httr

RoxygenNote 6.0.1

License GPL (>= 3)

NeedsCompilation no

Author Ruben Remelgado [aut, cre]

Repository CRAN

Date/Publication 2017-08-19 17:07:56 UTC

R topics documented:

backSample 2
dataQuery 3
getEnv 5
hotMove 6
hotMoveStats 7
imgInt 9
labelSample 10
modelApply 11
moveCloud 12
moveModel 14

moveReduce 16
moveSeg 17
plotMove 18
poly2sample 20
proSat 21
rsComposite 22
sampleMove 24
satTime 25
segRaster 26
sMoveRes 27
spaceDir 28
specVar 30
timeDir 31
tMoveRes 33

Index **35**

`backSample` *backSample*

Description

Background sample selection.

Usage

```
backSample(xy = xy, rid = rid, method = method, img = img, nb = NULL)
```

Arguments

`xy`	Object of class *SpatialPoints* of *SpatialPointsDataFrame*.
`rid`	Vector of region identifyers for each sample.
`method`	One of *random* or *pca*. Default is *random*.
`img`	Object of class *RasterLayer*, *RasterStack* or *RasterBrick*.
`nb`	Number of random background samples.

Details

The function selects a set of n background samples where n is determined by *nb*. If *nb* is not provided all background pixels are considered. If the method is *random* these samples are returned in the form of a *SpatialPoints* object. If *pca* is used, a PCA analysis is performed over all available samples (provided and random). For each unique region ID (*rid*) the median and the Median Absolute Deviation (MAD) is estimated for a given Principal Component (PC). Background samples where the difference between their variance and the variance of the region samples exceeds the MAD are selected. Only the samples that are selected by all unique regions are kept. This process is repeated for all PC's with eigenvalues greater than 1 (Kaiser rule). The final set of samples corresponds to the unique records selected by the different PC's. These samples are returned as a SpatialPointsDataFrame containing the variables extracted from *img*.

Value

A *SpatialPoints* or a *SpatialPointsDataFrame*.

See Also

labelSample hotMove dataQuery

Examples

```
{

 require(raster)

 # read raster data
 file <- list.files(system.file('extdata', '', package="rsMove"), 'tc.*tif', full.names=TRUE)
 rsStk <- stack(file)

 # read movement data
 moveData <- read.csv(system.file('extdata', 'konstanz_20130805-20130811.csv', package="rsMove"))
 moveData <- SpatialPointsDataFrame(moveData[,1:2], moveData, proj4string=crs(rsStk))

 # find sample regions
 label <- labelSample(xy=moveData, rad=500, npx=2, pxr=30)

 # select background samples
 ind <- which(label>0) # selected samples
 bSamples <- backSample(xy=moveData[ind,], rid=label[ind], img=rsStk, method='random')

}
```

---

dataQuery *dataQuery*

Description

Query environmental data for coordinate pairs.

Usage

```
dataQuery(xy = xy, st = NULL, img = img, rt = NULL, tb = NULL,
  bs = NULL, fun = NULL)
```

Arguments

xy	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
st	Object of class *Date* with *xy* observation dates.
img	Object of class *RasterLayer*, *RasterStack* or *RasterBrick*.
rt	Object of class *Date* with *img* observation dates.

tb	Two element vector with temporal search buffer, expressed in days.
bs	Buffer size (unit depends on the raster projection).
fun	Passes an external function.

Details

Returns environmental variables from a raster object for a given set of x and y coordinates. A buffer size (*bs*) and a user defined function (*fun*) can be specified to sample within an area. The defaut is to estimate a weighted mean. If raster acquisition times are provided (*rt*) and the date of sampling (*st*). In this case, the function will treat the raster data as a time series and search for clear pixel in time within the contraints of a temporal buffer defined by *tb*. *tb* passes two values which represent the size of the buffer in two directions: before and after the target date. This allows for bacward and forward sampling.

Value

A n object of class *vector* or *data.frame*.

See Also

sampleMove backSample

Examples

```
{

 require(raster)

 # read raster data
 file <- list.files(system.file('extdata', '', package="rsMove"), 'tc.*tif', full.names=TRUE)
 rsStk <- stack(file)
 rsStk <- stack(rsStk, rsStk, rsStk) # dummy files for the example

 # read movement data
 moveData <- read.csv(system.file('extdata', 'konstanz_20130805-20130811.csv', package="rsMove"))
 moveData <- SpatialPointsDataFrame(moveData[1:10,1:2], moveData[1:10,], proj4string=crs(rsStk))

 # raster dates
 r.date <- seq.Date(as.Date("2013-08-01"), as.Date("2013-08-09"), 1)

 # sample dates
 o.date <- as.Date(moveData@data$date)

 # retrieve remote sensing data for samples
 rsQuery <- dataQuery(xy=moveData, st=o.date, img=rsStk, rt=r.date, tb=c(30,30))

}
```

getEnv *getEnv*

Description

Interface to download ecologically relevant data.

Usage

```
getEnv(d.path = NULL, d.source = NULL, t.var = NULL, ref = NULL,
  p.raster = FALSE, p.res = NULL, pad = 10)
```

Arguments

d.path	Output data path for downloaded data.
d.source	Data source. One of "EarthEnv", "GFC", "GSW", "CCI" or "HSM".
t.var	Target variable.
ref	Projected spatial object from which an extent can be derived.
p.raster	Logical. Should the output be reprojected?
p.res	Pixel resolution (used if p.raster is TRUE).
pad	Mi,ner pf cells used to pad the output raster when resampling.

Details

Downloads data from earthenv.org. To check which variables can be downloaded, run the function without specifying *t.var* and specifying *d.source*. This will return a data frame listing the existing variables for a given data source. Here, the user can refer to the column *"code"* to retrieve the keywords that can be passed to the function through *t.var*. The keywords recognized by *d.source* are:

- *"EarthEnv"* - EarthEnv project.
- *"GFC"* - Maryland University Global Forest Change.
- *"GSW"* - JRC Global Surface Water.
- *"CCI"* - ESA CCI Global land cover.
- *"HSM"* - JRC Human Settlement map.
- *"NEO"* - NASA Earth Observations.

If *t.var* contains *"DEM90"* from "EarthEv" or any variable from *"GFC"* or *"GSW"*, The function might require more time as it will mosaic the tiles with which the rerence spatial object (*ref*) overlap with. In any circunstance, the output will be cropped to the extent of *ref* and, if prompted, use it as a reference to reproject the output.

Value

One or multiple raster objects.

References

http://www.earthenv.org/ https://earthenginepartners.appspot.com/science-2013-global-forest https://global-surface-water.appspot.com/ http://ghsl.jrc.ec.europa.eu/ http://maps.elie.ucl.ac.be/CCI/viewer/ https://neo.sci.gsfc.nasa.gov/

See Also

dataQuery proSat

Examples

```
{

 # return list of variables
 ee.var <- getEnv(d.source="EarthEnv")
 gfc.var <- getEnv(d.source="GFC")
 gsw.var <- getEnv(d.source="GSW")
 cci.var <- getEnv(d.source="CCI")
 hsm.var <- getEnv(d.source="HSM")
 neo.var <- getEnv(d.source="NEO")

}
```

---

hotMove *hotMove*

Description

Identifies hotspots of samples using a pixel based approach.

Usage

```
hotMove(xy = xy, pxr = pxr, shp = FALSE)
```

Arguments

xy	Object of class *SpatialPoints* of *SpatialPointsDataFrame*.
pxr	Grid resolution. Unit depends on *xy* projection.
shp	Logical. Should the function provide polygons? Default is FALSE.

Details

The function builds a matrix for a given resolution and a spatial extent derived from a set of samples (*xy*). First, these samples are converted into unique pixel coordinates. Then, the spatial connectivity of these pixels is evaluated using a 8-neighboor connected component labelling algorithm. The function returns a vector with the region ID per sample (*$indices*). If *shp* is TRUE the function also returns a shapefile for each region defined by the convex hull of the samples within them (*$polygons*).

Value

A list object.

See Also

sampleMove hotMoveStats

Examples

```
{

require(raster)

# reference data
sprj <- crs("+proj=longlat +ellps=WGS84 +no_defs")
moveData <- read.csv(system.file('extdata', 'latlon_example.csv', package="rsMove"))
moveData <- SpatialPointsDataFrame(moveData[,2:3], moveData, proj4string=sprj)

# extract regions
hm <- hotMove(xy=moveData, pxr=0.1, shp=TRUE)

# plot shapefile (color by region)
plot(hm$polygons, col=hm$indices)

# add new information to original shapefile
moveData@data$indices <- hm$indices

}
```

---

hotMoveStats *hotMoveStats*

Description

Provides statistics for the output of hotMove.

Usage

```
hotMoveStats(rid = NULL, o.time = NULL, tUnit = NULL, aid = NULL,
  method = "deg")
```

Arguments

rid	List object as provided by *hotMove()*.
o.time	Object of class *Date*, *POSIXlt* or *POSIXct*.
tUnit	Time unit for stats. Default is *days*. See difftime for additional keywords.
aid	Optional. Unique identifiers.
method	Method used to estimate polygon area.

Details

This functions analysis the attributes of sample regions define by hotMove(). Alternatively, the user can keep *rid* as NULL. This case, all information will be assumed as part of one region. For each sample region, the function returns the amount of samples (*tns*. If a vector of unique identifiers is provided (*aid*) the number of unique identifiers observed within each region is also reported. If temporal information is provided (*time*) the function identifies unique temporal segment corresponding to periods of consecutive days with observations. For each segment, the function reports on the amount of segments (*nts*) as well as the minimum (*mnt*), maximum (*mxt*) and mean (*avt*) of the time segments and the total amount of time that they amount to (*tts*). For each region, the function will also report on the start and end of each temporal segment (*$temporal.segments*) and will provide the sample indices for associated to each segment (*$segment.indices*). If *rid* contains polygons for each region, the function also reports on the area of convex polygons. In this case, the user can use the *method* keyword to specify how the polygons should be handled. If the polygons are in lat-lon, method *deg* can be used to re-project each polygon to its corresponding UTZ zone before retrieving the area. This is the default, if the polygons are in a cartesian coordinate system use *m*.

Value

A *data frame*.

See Also

hotMove

Examples

```
{

require(raster)

# reference data
sprj <- CRS("+proj=longlat +ellps=WGS84 +no_defs")
moveData <- read.csv(system.file('extdata', 'latlon_example.csv', package="rsMove"))
moveData <- SpatialPointsDataFrame(moveData[,2:3], moveData, proj4string=sprj)

# extract regions
hm <- hotMove(xy=moveData, pxr=0.1, shp=TRUE)

# plot shapefile (color by region)
plot(hm$polygons, col=hm$indices)

# add new information to original shapefile
moveData@data <- cbind(moveData@data, hm$indices)

# derive statistics
hm.region.stats <- hotMoveStats(rid=hm, o.time=as.Date(moveData@data$timestamp))
hm.time.stats <- hotMoveStats(o.time=as.Date(moveData@data$timestamp))

}
```

imgInt *imgInt*

Description

Temporal linear interpolation of raster data.

Usage

```
imgInt(img = NULL, rd = rd, td = td, bs = NULL, xy = NULL,
  edata = NULL)
```

Arguments

img	Object of class *RasterStack* or *RasterBrick*.
rd	Raster dates. Object of class *Date*.
td	Target dates. Object of class *Date*.
bs	wo element vector with temporal search buffer (expressed in days).
xy	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
edata	Object of class *data frame* or *matrix* with remote sensing data.

Details

Performs a pixel-wise linear interpolation over a raster for a given set of dates (*td*). A teporal buffer (*bs*) is required to limit the search for reference data points (*rd*). *bs* is defined by a two element vector which limits the search of images in the past and future. If *xy* is provided the function only considers the pixels that overlap with the these sample points. Otherwise, a RasterBrick is provided. However, if *edata* is provided, *xy* and *img* are ignored. and a data frame is provided.

Value

A *RasterBrick* or a *data frame*.

See Also

@seealso dataQuery timeDir spaceDir moveSeg

Examples

```
{

 require(raster)

 #'  # read raster data
file <- list.files(system.file('extdata', '', package="rsMove"), 'tc.*tif', full.names=TRUE)
rsStk <- stack(file)
rsStk <- stack(rsStk, rsStk, rsStk) # dummy files for the example
```

```
# read movement data
moveData <- read.csv(system.file('extdata', 'konstanz_20130805-20130811.csv', package="rsMove"))
moveData <- SpatialPointsDataFrame(moveData[1:10,1:2], moveData[1:10,], proj4string=crs(rsStk))

# raster dates
rd = seq.Date(as.Date("2012-01-01"), as.Date("2012-12-31"), 45)

# target dates
td = as.Date("2012-04-01")

# interpolate raster data to target dates
i.img <- imgInt(img=rsStk, rd=rd, td=td, bs=c(60,60), xy=moveData)

}
```

`labelSample` *labelSample*

Description

Region labeling of samples based on their spatial connectivity.

Usage

```
labelSample(xy = xy, rad = rad, npt = NULL, npx = NULL, pxr = r)
```

Arguments

`xy`	Object of class *SpatialPoints* of *SpatialPointsDataFrame*.
`rad`	Minimum radius. Unit depends on the projection of the data.
`npt`	Minimum pixel count per pixel.
`npx`	Minimum number of pixels.
`pxr`	Pixel resolution os a valid raster layer.

Details

First, the samples are converted to pixel coordinates and then one of two occur: 1) if *npt* is set, the function removes pixels with a pixel count smaller than the one specified; 2) If *npx* is set, the connectivity between neighboring samples is evaluated and regions with a count small than the specified value are filtered. Only one option may be set at a time. Then, the remaining pixels are dilated and the samples are again labeled accounting for regions that are not connected but are near to each other. Regions within a given distance of each other (defined by *rad*) are aggregated. The grid used for this analysis is built from the spatial extent of *xy* and a given pixel resolution (*pxr*). If *pxr* is a raster, this will be used to define the dimensions of the grid. Doing so can be of use when the user has pre-select environmental predictors that will be used for modeling. Note that the finer the resolution the more independent regions are likely to be returned. The output is a vector with ID's assigning each sample to its region. Samples filtered by *npt* or *npx* will be returned as zeros.

Value

A *vector*.

See Also

sampleMove hotMove

Examples

```
{

 require(raster)

 # read raster data
 r <- raster(system.file('extdata', 'tcb_1.tif', package="rsMove"))

 # read movement data
 moveData <- read.csv(system.file('extdata', 'konstanz_20130804.csv', package="rsMove"))
 moveData <- SpatialPointsDataFrame(moveData[,1:2], moveData, proj4string=crs(r))

 # derive region labels
 labels <- labelSample(xy=moveData, rad=90, npx=2, pxr=30)

}
```

---

modelApply *modelApply*

Description

Apply a model or an ensemble of models to raster data.

Usage

```
modelApply(model, img, fun = NULL)
```

Arguments

model	List object as provided by *moveModel()*.
img	Object of class *RasterLayer*, *RasterStack* or *RasterBrick*.
fun	Function that specifies how the model should be applied.

Details

The function uses the output of *moveModel()*. If this contains a list of models from multiple runs, the function creates a stack of predictions and summarizes it on a pixel-by-pixel basis using a weighted mean. The weights are defined by the average performance for *presence* and *background* samples in each iteration.

Value

A *Raster*.

See Also

segRaster moveModel

Examples

```
## Not run:

 require(raster)

 # read remote sensing data
 file <- list.files(system.file('extdata', '', package="rsMove"), 'tc.*tif', full.names=TRUE)
 rsStk <- stack(file)

 # read movement data
 moveData <- read.csv(system.file('extdata', 'konstanz_20130805-20130811.csv', package="rsMove"))
 moveData <- SpatialPointsDataFrame(moveData[,1:2], moveData, proj4string=crs(rsStk))

 # retrieve remote sensing data for samples
 rsQuery <- dataQuery(xy=moveData,img=rsStk, remove.dup=TRUE)

 # identify unique sample regions
 label <- labelSample(xy=rsQuery, rad=90, npx=1, pxr=rsStk)

 # select background samples
 ind <- which(label>0) # selected samples
 bSamples <- backSample(xy=moveData[ind,], rid=label[ind], img=rsStk, nb=4000, method='pca')

 # derive model predictions
 fun <- function(x,y) {train(x, y, method="rf", trControl=trainControl(method='oob'))}
 p.model <- moveModel(p.data=rsQuery@data, a.data=bSamples@data, label=label, fun=fun, nruns=1)

 # derive prediction from model ensemble
 fun <- function(x,y) {predict(x, y, type='prob')[[1]]$`1`}
 prob <- modelApply(p.model, rsStk, fun=fun)

 # see output
 plot(prob)


## End(Not run)
```

---

moveCloud *moveCloud*

Description

Provides historical information on cloud cover.

Usage

```
moveCloud(xy = xy, o.time = o.time, d.path = NULL, b.size = NULL,
  remove.file = TRUE, p.res = T)
```

Arguments

xy	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
o.time	Object of class *Date*.
d.path	Output data path for downloaded data.
b.size	Two element vector with temporal buffer size (expressed in days).
remove.file	Logical. Should the files be deleted after usage?
p.res	Should the output be ploted on screen? Default is TRUE.

Details

This function makes uses daily cloud fraction data from NASA's NEO service. For each observation date (*o.time*), the function downloads the correspondent image and extracts the percent of cloud cover for the samples acquired at the target date. If *d.path* is specified, the function will look within the provided directory for the required files. If so, they won't be downloaded. If *d.buffer* is specified, for each date, the function will consider images before and after within a temporal buffer. *d.buffer* requires two elements which specify the buffer size before and after the target date. These new images will be used to report on the closest time step with the lowest possible cloud cover. The final report provides information on:

- *day.cover*: cloud cover for the observation date
- *p.day.before*: date before the obsertation date with the lowest cloud cover
- *p.cover.before*: cloud cover for *p.day.before*
- *p.day.after*: date after the obsertation date with the lowest cloud cover
- *p.cover.after*: cloud cover for *p.day.after*

The output will also contain a two plots with information on the distance between the observation dates and the closest date with the lowest cloud cover. The plots show the amount of samples that are covered in each of the target dates for the best dates before (*$plot.before*) and after (*$plot.after*) the observation dates.

Value

A *list*.

References

https://cneos.jpl.nasa.gov/

See Also

sMoveRes tMoveRes

Examples

```
## Not run:

 require(raster)

 # read raster data
 r <- raster(system.file('extdata', 'tcb_1.tif', package="rsMove"))

 # read movement data
 moveData <- read.csv(system.file('extdata', 'konstanz_20130804.csv', package="rsMove"))
 moveData <- SpatialPointsDataFrame(moveData[,1:2], moveData, proj4string=crs(r))

 # test function for 5, 10 20 and 30 m
 od <- as.Date(moveData@data$date)
 c.cover <- moveCloud(xy=moveData, o.time=od, d.path=".", b.size=30)


## End(Not run)
```

---

moveModel *moveModel*

Description

Spatially stratified predictive modeling.

Usage

```
moveModel(p.data = p.data, a.data = a.data, label = NULL, fun = fun,
  nruns = 1)
```

Arguments

p.data	Object of class *data.frame* with environmental variables for presence samples.
a.data	Object of class *data.frame* with environmental variables for background samples.
label	Region labels. If missing, "p.data" is assumed as one region.
fun	A function with the modeling algorithm to use.
nruns	Number of runs. Default is 1.

Details

For n iterations, where n is the number of unique sample regions, the function uses one sample region for validation while the remaining ones are used for training. The background samples are split randomly at each iteration. The final accuracy, provided as a F1-score for both presence and background sampels, is derived from the total of true and false positives (*$f1*). Additionaly, for each run, the function returns a model (*$model*) which is trained using all the samples. This output can be passed to modelApply(). By default, the function uses a Random Forest classifier. However, the a user specified function can be passed using *fun*.

Value

A *list*.

See Also

sampleMove labelSample backSample modelApply train

Examples

```
## Not run:

 require(rgdal)
 require(raster)
 require(sp)

 # read remote sensing data
 file <- list.files(system.file('extdata', '', package="rsMove"), 'tc.*tif', full.names=TRUE)
 rsStk <- stack(file)

 # read movement data
 moveData <- read.csv(system.file('extdata', 'konstanz_20130805-20130811.csv', package="rsMove"))
 moveData <- SpatialPointsDataFrame(moveData[,1:2], moveData, proj4string=crs(rsStk))

 # retrieve remote sensing data for samples
 rsQuery <- dataQuery(xy=moveData,img=rsStk, remove.dup=TRUE)

 # identify unique sample regions
 label <- labelSample(xy=rsQuery, rad=3000, pxr=rsStk)

 # select background samples
 ind <- which(label>0) # selected samples
 bSamples <- backSample(xy=moveData[ind,], rid=label[ind], img=rsStk, method='pca')

 # derive model predictions
 fun <- function(x,y) {train(x, y, method="rf", trControl=trainControl(method='oob'))}
 out <- moveModel(p.data=rsQuery@data, a.data=bSamples@data, label=label, fun=fun, nruns=1)


## End(Not run)
```

moveReduce *moveReduce*

Description

Remote sensing based point segmentation.

Usage

```
moveReduce(xy = xy, ot = ot, img = img)
```

Arguments

xy	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
ot	Object of class *Date*, *POSIXlt* or *POSIXct* with *xy* observation dates.
img	Object of class *RasterLayer*, *RasterStack* or *RasterBrick*.

Details

SReduces a set of input samples (*xy*) based on their assignment to unique pixels within a reference raster (*img*). The function looks at consecutive points ordered by time (*ot*) and aggregates samples if they remain within the same pixel. If the same pixel is revisited on a later time, that observation is kept as a separate occurrence. For each temporal segment, the function returns mean x and y coordinates, the start and end timestamps, the mean timestamp and the elapsed time.

Value

A *list*.

See Also

sampleMove moveSeg

Examples

```
{

 require(raster)

 # read raster data
 r <- raster(system.file('extdata', 'tcb_1.tif', package="rsMove"))

 # read movement data
 moveData <- read.csv(system.file('extdata', 'konstanz_20130804.csv', package="rsMove"))
 moveData <- SpatialPointsDataFrame(moveData[,1:2], moveData, proj4string=crs(r))

 # observation time
 o.time <- strptime(paste0(moveData@data$date, ' ', moveData@data$time), format="%Y/%m/%d %H:%M:%S")
```

```
 # reduce amount of samples
 move.reduce <- moveReduce(xy=moveData, ot=o.time, img=r)

}
```

---

moveSeg *moveSeg*

Description

Remote sensing based point segmentation

Usage

```
moveSeg(xy = xy, edata = edata, type = "cont", ot = NULL,
  b.size = NULL, threshold = NULL, s.fun = NULL)
```

Arguments

xy	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
edata	Object of class *RasterLayer* or *data.frame*.
type	Raster data type. One of *cont* (continues) or *cat* (for categorical).
ot	Object of class *Date*, *POSIXlt* or *POSIXct* with *xy* observation dates.
b.size	Buffer size expressed in the map units.
threshold	Change threshold.
s.fun	Output summary function. Default is mean.

Details

Segmentation of a point shapefile based on the spatial variability of a raster dataset. When the *method* is set to *'cont'*, the raster data is assumed to be continuous. Then, the function determines the percentual change between each pair of two consecutive coordinate pairs. If this change is above a predifined *threshold*, a new pointer is added and the previous sequence of samples is labeled as a unique segment. If *method* is set as *'cat'*, the function assumes the raster data is categorical ignoring the *theshold* and *s.fun* keywords. In this case, a new segment is identified if the any change is observed between two consecutife points. The output consists of a list containing a *SpatialPointsDataFrame* (*$points*) reporting on the segment ID (*sid*) associated to each sample and a data frame (*$report*) with the amount of points in each region and the value returned by *s.fun*. If *ot* is provided, the function also provides the elapsed time within each segment. If *fun* is set by the user, the provided function will be used to summarize the raster values at each segment. Also, if *edata* is a *RasterStack* or a *RasterBrick*, *r.fun* is used to reduce the multi-layered object to a single layer. he user can either use a Principal Component Analysis (PCA) analysis by setting *r.fun* to *pca* or provide a function. If *pca* is selected, the first Principal Conponent (PC) *threshold* will be set automatically to the standard deviation of the first PC. This is the default. If *method* is *cat* the function will assume the data is categorical and wil define segments whenever a change occurres.

Value

A *list*.

See Also

dataQuery imgInt

Examples

```
{

 require(raster)

 # read raster data
 r <- raster(system.file('extdata', 'tcb_1.tif', package="rsMove"))

 # read movement data
 moveData <- read.csv(system.file('extdata', 'konstanz_20130804.csv', package="rsMove"))
 moveData <- SpatialPointsDataFrame(moveData[,1:2], moveData, proj4string=crs(r))

 # observation time
 o.time <- strptime(paste0(moveData@data$date, ' ', moveData@data$time), format="%Y/%m/%d %H:%M:%S")

 # perform directional sampling
 seg <- moveSeg(xy=moveData, ot=o.time, edata=r, type="cont", threshold=0.1)

}
```

---

plotMove *plotMove*

Description

Plots per pixel time and value.

Usage

```
plotMove(x = x, y = y, o.time = NULL, value = NULL, type = NULL)
```

Arguments

x	Vector of x coordinates.
y	Vector of y coordinates.
o.time	Vector with time length.
value	Vector with environmental data.edata Object of class *RasterLayer* or *data.frame*.
type	One of 'cont' or 'cat'. Defines the type of *value*.

Details

This function plots a provided set of x and y coordinates adding information on the elapsed time at each coordinate (*e.time*) and/or a given environmental variable (*value*). If only *time* or *value* are set, the function builds a scatterplot where the size of the points is defined by the input variables. If both are provided, the size of the points is defined by the elapsed time and the raster value is used to build a color scheme for the points. When *value* is provided, the keyword *type* is required. It will influence how the plots are built. The breaks a limits for the point size and colors is define automatically.

Value

A *ggplot* object.

See Also

dataQuery moveReduce

Examples

```
{

 require(raster)

 # read raster data
 r <- raster(system.file('extdata', 'tcb_1.tif', package="rsMove"))

 # read movement data
 moveData <- read.csv(system.file('extdata', 'konstanz_20130804.csv', package="rsMove"))
 moveData <- SpatialPointsDataFrame(moveData[,1:2], moveData, proj4string=crs(r))

 # observation time
 o.time <- strptime(paste0(moveData@data$date, ' ', moveData@data$time), format="%Y/%m/%d %H:%M:%S")

 # reduce amount of samples
 move.reduce <- moveReduce(xy=moveData, ot=o.time, img=r)

 # query data
 ov <- dataQuery(xy=move.reduce$points, img=r)

 # plot output
 x <- move.reduce$points@coords[,1]
 y <- move.reduce$points@coords[,2]
 et <- move.reduce$points@data$elapsed.time
 op <- plotMove(x=x, y=y, o.time=et, value=ov[,1], type="cont")

}
```

poly2sample *poly2sample*

Description

Convert spatial polygons into point samples.

Usage

```
poly2sample(pol = pol, re = NULL, mpc = NULL, pr = NULL)
```

Arguments

pol	Object of class *SpatialPolygons* or *SpatialPolygonDataFrame*.
re	Object of class *Extent* or a raster object from which an extent can be derived.
mpc	Minimum pixel cover (0-100). Default is 100.
pr	Pixel resolution.

Details

Determines coordinates of pixels within a given extent. If *re* is missing the target extent corresponds to the extent of the polygon layer. In this case, *pr* is required. If *re* is an *Extent* object or *re* is missing *rp* is required. *mpc* can be used to filter pixels with low purity, i.e. pixels where the percentage of area cover by a polygon is below the defined threshold. The output provides the coordinates (*x* and *y*) the pixel percent cover (*cover*) for each selected pixel.

Value

A *SpatialPointsDataFrame*.

See Also

dataQuery imgInt

Examples

```
{

 require(raster)

 # load example probability image
 file <- system.file('extdata', 'konstanz_probabilities.tif', package="rsMove")
 img <- raster(file)

 # load area of interest
 file <- system.file('extdata', 'konstanz_roi.shp', package="rsMove")
 roi <- shapefile(file)
```

```
 # segment probabilities
 samples <- poly2sample(pol=roi, re=img)

}
```

---

proSat *proSat*

Description

Interface to download and process satelite data.

Usage

```
proSat(t.var = NULL, xy = NULL, o.time = NULL, d.path = NULL,
  p.raster = FALSE, p.res = NULL, user.cred = NULL, pad = 10)
```

Arguments

t.var	Target variable.
xy	Object from which an extent can be derived.
o.time	Object of class *Date*.
d.path	Output data path for downloaded data.
p.raster	Logical. Should the output be re-projected?
p.res	Target pixel resolution (if p.raster is TRUE).
user.cred	Two element character vector containing username and password.
pad	Padding expressed in number of pixels. Used when re-projecting and resampling.

Details

Downloads and pre-processes pre-selected satellite datasets That provide ecologically meaningful variables. *xy* is required to define a reference extent for which the data will be downloaded and croped. If *p.raster* is TRUE, the function will also re-project the output to the projection of *xy*. Note that *xy* does not required reprojection a priori. The function will find the approapriate projection to use when deriving a reference extent. If *p.raster* is TRUE *p.res* is also required to determine the output pixel resolution. The outputs are presented in GTiff format and are croped and masked by default. Note that the download files might not correspond to the dates supplied through *o.time*. The function will consider the temporal resolution of the target dataset and compare the possible and the requested dates. If a requested date is not found, the function will instead provide the closest image time. As a consequence, the number of downloaded files might be lesser than the number of dates. The function will inform the user on this by providing a list object which contains the downloaded dates (*$date*) as well as the path for the correspondent image (*$path*).

Value

One or multiple raster objects.

Note

To consult the list of provided variables, run the function without specifying any input. This will provide information on the variables, variable codes (used in *t.var*), temporal and spatial resolution and the sensor of origin. Some variables might require login credentials. Check the table to know which credentials to use and assign them through *user.cred*.

See Also

`getEnv` `imgInt` `dataQuery`

Examples

```
{

 # return list of variables
 modis.var <- proSat()

}
```

`rsComposite` *rsComposite*

Description

Compositing of remote sensing data based on GPS tracking dates.

Usage

```
rsComposite(img, rd, ot, cm = 1, type = "norm", d.buffer = NULL)
```

Arguments

`img`	Object of class *RasterSpack* or *RasterBrick*.
`rd`	Object of class *Date* with *img* observation dates.
`ot`	Object of class *Date* with reference dates.
`cm`	Number of deviations from the target date. Default is 1.
`type`	One of "norm" or "pheno" or "pheno2".
`d.buffer`	Search buffer (expressed in days).

Details

The function uses a multi-layer raster object to build a composite. It looks at a ginve set of dates (e.g. GPS tracking dates) and estimates a reference date to build the composite for defined by the median of *ot*. The median is then used to estimate Median Absolute Deviation (MAD) which specifies the size of the buffer set aroung the target date within which bands will be considered. Here, *cm* is used as a multiplier to enlarge the temporal buffer. Alternatively, a user define temporal buffer is allowed by using the keyword *d.buffer*. If *ot* countains only one element, the function will use it as a reference date. In this case, if *d.buffer* is NULL the function will set it to 30 by default. The way how the function handles temporal information depends on the *type* keyword. If set to *norm*, the function will search for the nearest possible dates within the temporal buffer. However, if *pheno* is set, then the day of the year will be given priority. Thus, if multi-year raster data is provided, older data with a DOY closer to the target that will be used when possible. The output provides: #'

- *value* - composite of target images
- *dates* - per pixel date code
- *count* - pixel count of *dates*
- *na.count* - count of NA values
- *target* - target date
- *mad* - temporal buffer

If *pheno2* is used, for each pixel, the function wilhin estimate a weighted mean of the clear pixels within the temporal buffer. The weights represent the inverse time difference between the target and the available dates giver higher weights to small differences.

Value

A *list*.

See Also

`imgInt` `dataQuery` `proSat`

Examples

```
## Not run:

 require(raster)

 # read raster data
 file <- list.files(system.file('extdata', '', package="rsMove"), 'tc.*tif', full.names=TRUE)
 rsStk <- stack(file)
 rsStk <- stack(rsStk, rsStk, rsStk) # dummy files for the example

 # raster dates
 rd = seq.Date(as.Date("2013-01-01"), as.Date("2013-12-31"), 45)

 # target date
 ot = as.Date("2013-06-01")
```

```
 # build composite
 r.comp <- rsComposite(rsStk, rd, ot, d.buffer=90)


## End(Not run)
```

---

sampleMove *sampleMove*

Description

Sampling of possible stops along a movement track.

Usage

```
sampleMove(xy = xy, ot = ot, error = error, method = "m",
  tUnit = NULL)
```

Arguments

xy	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
ot	Object of class *Date*, *POSIXlt* or *POSIXct* with the same length as *xy*.
error	Distance (in meters).
method	How should the disntance be estimated? One of 'm' or 'deg'. Default is 'm'.
tUnit	Time unit to estimate elapsed time. See difftime for keywords. Default is *mins*.

Details

This function offers a simple approach to sample from locati where an animal showed little or no movement based on GPS tracking data. It looks at the distance among consecutive samples (*error*) and estimates mean coordinates for the temporal segments where the animal moved less than the defined distance from the first location of the segment. The user should selected *method* in accordance with the projection system associated to the data. If 'm' it estimates the ecludian distance. If 'deg' it uses the haversine formula. The output reports on the mean sample coordinates for the sample locations ('x' and 'y'), the start, end and total time spent per sample ('time' expressed in minutes) and the total number of observations per sample ('count').

Value

A *SpatialPointsDataFrame*.

See Also

labelSample backSample dataQuery

Examples

```
{

 require(raster)

# reference data
sprj <- crs("+proj=longlat +ellps=WGS84 +no_defs")
moveData <- read.csv(system.file('extdata', 'latlon_example.csv', package="rsMove"))
moveData <- SpatialPointsDataFrame(moveData[,2:3], moveData, proj4string=sprj)

 # sampling without reference grid
 ot = strptime(moveData$timestamp, "%Y-%m-%d %H:%M:%S")
 output <- sampleMove(xy=moveData, ot=ot, error=7, method='deg')

 # compare original vs new samples
 plot(moveData, col="black", pch=16)
 points(output$x, output$y, col="red", pch=15)

}
```

`satTime` *satTime*

Description

Finds available satellite dates for tracking data.

Usage

```
satTime(o.time = o.time, t.var = t.var)
```

Arguments

`o.time`	Object of class *Date*.
`t.var`	Target variable.

Details

This function compares a set of input dates (*o.time*) to the possible dates for which satellite data can be downloaded. This analysis is performed for the set of variables provided through the function proSat(). The function provides a report with the closest dates available for each of the samples and a faceted plot for each unique year showing the observed dates, the dates covered by the satellite data and the dates covered by both.

Value

One or multiple plots.

See Also

moveCloud proSat

Examples

```
{

 # return list of variables
 var.ls <- satTime(o.time=as.Date("2013-08-04"), t.var="ndvi")

}
```

---

segRaster *segRaster*

Description

Connencte-region based raster segmentation.

Usage

```
segRaster(prob, pt = 0.1, mp = 0.5)
```

Arguments

prob	Object of class *RasterLayer*.
pt	Difference threshold. Default is 0.05.
mp	Minimum value. Default is 0.5.

Details

The function segments an input layer using a connected component region labeling approach. For each pixel, the function estimates the difference between it and its imediate 8 neighbors. The pixels where the difference is below the defined threshold (*ct*) are aggregated into a single region. The user can define a minimum pixel value using *mp* which will limit the range of pixels under evaluation. The result contains a raster with unique values for each segment region region (*$segment*) as well as a data frame (*$stats*) with statistics for each region. The data frame report on the minimum (*min*), maximum (*max*), mean (*mean*) and standard deviation (*sd*) of the pixels contained in each region.

Value

A list object.

See Also

moveModel modelApply

Examples

```
{

 require(raster)

 # load example probability image
 file <- system.file('extdata', 'konstanz_probabilities.tif', package="rsMove")
 probImg <- raster(file)

 # segment probabilities
 rs <- segRaster(probImg)

}
```

---

sMoveRes *sMoveRes*

Description

Evaluates how a change in raster resolution impacst the availability of data points.

Usage

```
sMoveRes(xy = xy, pxr = pxr)
```

Arguments

xy	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
pxr	vector of target resolutions.

Details

Given a vector of pixel resolutions (*pxr*), the function determines the number of unique pixels and unique pixel groups. Additionaly, for each pixel, the function returns the corresponding pixel indices per resolution showing which samples would be grouped. The function returns a data frame (*$stats*) and a plot (*$plot*) with the statistics per resolution as well as a data frame with the pixel indices per resolution (*$indices*).

Value

A *list*.

See Also

tMoveRes specVar

Examples

```
{

 require(raster)

 # read movement data
 moveData <- read.csv(system.file('extdata', 'konstanz_20130804.csv', package="rsMove"))
 moveData <- SpatialPointsDataFrame(moveData[,1:2], moveData)

 # test function for 5, 10 20 and 30 m
 a.res <- sMoveRes(xy=moveData, pxr=c(5, 10, 20, 30))

}
```

---

spaceDir *spaceDir*

Description

Spatial directional raster analysis.

Usage

```
spaceDir(xy = xy, ot = NULL, img = img, dir = dir, type = type,
  dm = "m", b.size = NULL, fun = NULL, npx = 2)
```

Arguments

xy	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
ot	Object of class *Date*, *POSIXlt* or *POSIXct* with *xy* observation dates.
img	Object of class *RasterLayer*.
dir	One of *fwd*, *bwd* or *both*. Default is *both*.
type	One of 'cont' or 'cat'. Defines which type of variable is in use.
dm	One of 'm' or 'deg' specifying the projection unit. Default is 'm'.
b.size	Buffer size expressed in the map units.
fun	Summary function.
npx	Minimum number of pixels used in *fun*. Default is 2.

Details

This function evaluates how do environmental conditions change in space along a movement track. First, it looks at consectuve observations to define spatial segments. All the pixels between the two endpoints of the segment are considered in this process. The user can use the argument *dir* to prompt the function to focus on previous (*bwd*) or following (*fwd*) observations or to look in both directions (*both*) when defining the segments. The output consists of a list containing two

shapefiles, one *SpatialPointsDataFrame* (*$endpoints*) with the endpoints of each segment and a *SpatialLinesDataFrame* (*$segments*) representing each segment. The data frames provided with these shapefiles reports on:

- *x* - mean x coordinates
- *y* - mean y coordinates
- *timestamp* - mean observation time
- *pixel.time* - elapsed time within a pixel for a given segment
- *travel.distance* - distance between a segment endpoints
- *travel.time* - elapsed time between a segment endpoints

The additionaly information added to the data frame depends on the type of data. If *type* is set to *'cont'*, the function will assume the input raster is a continuous variable as estimate a statistical metric for each segment. If none is provided throught the keyword *fun*, the slope will be returned by default and assigned to the column *'stat'* within the output data frame. If *type* is set to *''cat'*, the function will assume the data is categorical. As a result, the keyword *fun* is ignored. Instead, the function will return the proportion of pixels occupied by each class for each segment. In addition, the dominant class (*'main'*) and the shannon index (*'shannon'*) will be returned for each segment within the output data frame. In order to include the area surrounding the sample points in this analysis, *d.buffer* can be used. This keyword dilates the interpolated sample points by a given distance transforming the initial line into a polygon. @note If *d.buffer* is used, the default *fun* (the slope) is not adequate given that spatial order of the samples is not preserved. This should be considered when using these keywords simulaneously,

Value

A *list*.

See Also

timeDir dataQuery imgInt

Examples

```
{

 require(raster)

 # read raster data
 r <- raster(system.file('extdata', 'tcb_1.tif', package="rsMove"))

 # read movement data
 moveData <- read.csv(system.file('extdata', 'konstanz_20130804.csv', package="rsMove"))
 moveData <- SpatialPointsDataFrame(moveData[,1:2], moveData, proj4string=crs(r))

 # observation time
 ot <- strptime(paste0(moveData@data$date, ' ',moveData@data$time), format="%Y/%m/%d %H:%M:%S")

 # perform directional sampling
 of <- function(x) {lm(x~c(1:length(x)))$coefficients[2]}
```

```
 s.sample <- spaceDir(xy=moveData, ot=ot, img=r, dir="bwd", type='cont', fun=of)

}
```

---

specVar *specVar*

Description

Quantifies how changes in the resolution of a raster affects the perception of spectral complexity.

Usage

```
specVar(img = img, xy = NULL, pxr = pxr)
```

Arguments

img	Object of class *RasterLayer*.
xy	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
pxr	vector of target resolutions.

Details

Given a raster object, the function determines how reducing the resolution of the raster impacts our ability to perceive the complexity of the landscape. The function aggregates *img* to each of the given pixel resolutions (*pxr*) and estimates the Mean Absolute Error (MAE) for each pixel with the aggregated layer where the differences are estimated from the pixel within the original raster that overlap with the target pixel. If a point shapefile is provided (*xy*), the function will only report on the values that overlap with the points. The output of the function consists of:

- *mae* - MAE, either for all pixels or for the points within *xy*
- *pixel.optimal* - raster reporting on the resolution with the lowest MAE for each pixel
- *pixel.optimal.stats* - Proportion of samples per resolution derived from *pixel.optimal*
- *plot* - boxplots of the variability of the MAE per resolution

If *xy* is set, the output will contain a vector with the optimal resolution per sample (*$sample.optimal*).

Value

A *list*.

See Also

tMoveRes sMoveRes

Examples

```
## Not run:

 require(raster)

 # read raster data
 r <- raster(system.file('extdata', 'tcb_1.tif', package="rsMove"))

 # read movement data
 moveData <- read.csv(system.file('extdata', 'konstanz_20130804.csv', package="rsMove"))
 moveData <- SpatialPointsDataFrame(moveData[,1:2], moveData, proj4string=crs(r))

 # apply function
 s.var <- specVar(img=r, xy=moveData, pxr=60)


## End(Not run)
```

---

timeDir *timeDir*

Description

Temporal directional raster analysis.

Usage

```
timeDir(xy = NULL, ot = ot, img = NULL, edata = NULL, rt = rt,
  mws = NULL, dir = NULL, fun = NULL)
```

Arguments

xy	Object of class "SpatialPoints" or "SpatialPointsDataFrame".
ot	Object of class *Date*, *POSIXlt* or *POSIXct* with *xy* observation dates.
img	Object of class *RasterStack* or *RasterBrick*.
edata	Object of class *data frame*.
rt	Object of class *Date*, *POSIXlt* or *POSIXct* with *img* observation dates.
mws	Moving window size (expressed in days).
dir	One of *fwd*, *bwd* or *both*. Default is *both*.
fun	Summary function.

Details

This function evaluates how do environmental conditions change in time along a movement track. First, it compares the observation times (*ot*) of *xy* against the acquisition times (*rt*) of *img* to search for relevant information within a pre-defined temporal window (*mws*). The user can chose to only consider time steps before (*bwd*) or after (*fwd* the target observation time or look at both directios (*both*). If the latest is chosen, the function applies *mws* equally to both directions. After selecting adequate temporal information, a statistical metric (*fun*) is used to summarize the selected time steps. By default, the slope will be used. The slope is estimated from a linear regression estimated between the acquisition times of *img* and their corresponding values. When providing a new function, set x for time and y for the raster values. The output reports on:

- *x* - mean x coordinates
- *y* - mean y coordinates
- *timestamp* - mean observation time
- *pixel.time* - elapsed time within a pixel for a given segment
- *stat*: statistical metric

If *edata* is provided, *img* will be ignores as *edata* will contain the environmental data with each column representing a different variable. Otherwise, this data will be retrieved from *img*. Also, if *edata* is provided, *xy* is not required. However, it can be provided to built a spatial plot with the results of the analysis.

Value

A *vector*.

See Also

spaceDir dataQuery imgInt

Examples

```
{

 require(raster)

 # read raster data
 file <- list.files(system.file('extdata', '', package="rsMove"), 'tc.*tif', full.names=TRUE)
 rsStk <- stack(file)
 rsStk <- stack(rsStk, rsStk, rsStk) # dummy files for the example

 # read movement data
 moveData <- read.csv(system.file('extdata', 'konstanz_20130804.csv', package="rsMove"))
 moveData <- SpatialPointsDataFrame(moveData[,1:2], moveData, proj4string=crs(rsStk))

 # raster dates
 rd <- seq.Date(as.Date("2013-08-01"), as.Date("2013-08-09"), 1)

 # sample dates
 ot <- strptime(paste0(moveData@data$date, ' ',moveData@data$time), format="%Y/%m/%d %H:%M:%S")
```

```
 # perform directional sampling
 of <- function(x,y) {lm(y~x)$coefficients[2]}
 time.env <- timeDir(xy=moveData, ot=ot, img=rsStk, rt=rd, mws=10, dir="bwd", fun=of)

}
```

---

tMoveRes *tMoveRes*

Description

Provides historical information on cloud cover.

Usage

```
tMoveRes(xy = xy, o.time = o.time, t.res = t.res, s.res = s.res)
```

Arguments

xy	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
o.time	Object of class *Date*, *POSIXlt* or *POSIXct* with *xy* observation dates.
t.res	Temporal resolution.
s.res	Spatial resolution.

Details

Given a vector of temporal resolutions (*t.res*), the function determines the number of unique pixels and unique pixel groups after their temporal agggation. The function returns the corresponding pixel indices per resolution showing which samples would be grouped (*$indices*). The function returns a data frame (*$stats*) and a plot (*$plot*) with the statistics per temporal resolution.

Value

A *list*.

See Also

sMoveRes specVar

Examples

```
## Not run:

 require(raster)

 # read raster data
 r <- raster(system.file('extdata', 'tcb_1.tif', package="rsMove"))

 # read movement data
 moveData <- read.csv(system.file('extdata', 'konstanz_20130804.csv', package="rsMove"))
 moveData <- SpatialPointsDataFrame(moveData[,1:2], moveData, proj4string=crs(r))

 # test function for 5, 10 20 and 30 m
 a.res <- tMoveRes(xy=moveData, dpath='.')


## End(Not run)
```

Index

backSample, 2, *4*, *15*, *24*

dataQuery, *3*, 3, *6*, *9*, *18–20*, *22–24*, *29*, *32*
difftime, *7*, *24*

getEnv, 5, *22*

hotMove, *3*, 6, *8*, *11*
hotMoveStats, *7*, 7

imgInt, 9, *18*, *20*, *22, 23*, *29*, *32*

labelSample, *3*, 10, *15*, *24*

modelApply, 11, *15*, *26*
moveCloud, 12, *26*
moveModel, *12*, 14, *26*
moveReduce, 16, *19*
moveSeg, *9*, *16*, 17

plotMove, 18
poly2sample, 20
proSat, *6*, 21, *23*, *26*

rsComposite, 22

sampleMove, *4*, *7*, *11*, *15, 16*, 24
satTime, 25
segRaster, *12*, 26
sMoveRes, *14*, 27, *30*, *33*
spaceDir, *9*, 28, *32*
specVar, *27*, 30, *33*

timeDir, *9*, *29*, 31
tMoveRes, *14*, *27*, *30*, 33
train, *15*