Package 'rsMove'

January 19, 2018

Type Package

Title Remote Sensing for Movement Ecology

Version 0.2.3

Date 2018-01-19

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 that support the combined use of animal movement and remote sensing data.

LazyData TRUE

Imports raster, sp, caret, rgdal, gdalUtils, ggplot2, grDevices, RCurl

RoxygenNote 6.0.1

License GPL (>= 3)

Suggests knitr, rmarkdown, kableExtra, imager

VignetteBuilder knitr

NeedsCompilation no

Author Ruben Remelgado [aut, cre]

Repository CRAN

Date/Publication 2018-01-19 19:46:44 UTC

R topics documented:

`backSample` *backSample*

Description

Background sample selection.

Usage

```
backSample(xy = xy, region.id = region.id,
  sampling.method = sampling.method, env.data = env.data,
  nr.samples = NULL)
```

Arguments

`xy`	Object of class *SpatialPoints* of *SpatialPointsDataFrame*.
`region.id`	Vector of region identifiers for each sample.
`sampling.method`	One of *random* or *pca*. Default is *random*.
`env.data`	Object of class *RasterLayer*, *RasterStack* or *RasterBrick*.
`nr.samples`	Number of random background samples.

Details

First, the function determines the unique pixel coordinates for *xy* based on the dimensions of *env.data* and retrieves n background samples where n is determined by *nr.samples*. Then, the selection of samples is dependent on the method chosen by the user. If *sampling.method* is set to *random*, the function will select samples randomly. However, if *pca* is used, the function will use a Principal Components Analysis (PCA) over *env.data* to evaluate the similarity between the samples associated to *xy* and the initial set of random samples First, based on this PCA, the function selects the most important Principal Components (PC's) using the kaiser rule (i.e. PC's with eigenvalues greater than 1). Then, for each PC, the function estimates the median and the Median Absolute Deviation (MAD) for each unique identifier in *region.id*) and selects background samples where the difference between their variance and the variance of the region samples exceeds the MAD. Then, the algorithm removes the background samples that were not selected by all sample regions.

If *nr.samples* is not provided all background pixels are returned

object.

Value

A *SpatialPoints* or a *SpatialPointsDataFrame* of background samples for unique pixels in *env.data*.

References

Remelgado, R., Leutner, B., Safi, K., Sonnenschein, R., Kuebert, C. and Wegmann, M. (2017), Linking animal movement and remote sensing - mapping resource suitability from a remote sensing perspective. Remote Sens Ecol Conserv.

See Also

labelSample hotMove dataQuery

Examples

```
{

 require(raster)

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

 # read movement data
 data(shortMove)

 # find sample regions
 label <- labelSample(xy=shortMove, agg.radius=500, nr.pixels=2, pixel.res=30)

 # select background samples
 ind <- which(label>0) # selected samples
 bSamples <- backSample(xy=shortMove[ind,], region.id=label[ind],
 env.data=r.stk, sampling.method='random')
```

```
}
```

`dataQuery` *dataQuery*

Description

Query environmental data for coordinate pairs using the nearest non NA value in time.

Usage

```
dataQuery(xy = xy, obs.dates = obs.dates, env.data = env.data,
  env.dates = env.dates, time.buffer = NULL, spatial.buffer = NULL,
  smooth.fun = NULL)
```

Arguments

`xy`	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
`obs.dates`	Object of class *Date* with *xy* observation dates.
`env.data`	Object of class *RasterStack*, *RasterBrick* or *data.frame*.
`env.dates`	Object of class *Date* with *env.data* observation dates.
`time.buffer`	Two element vector with temporal search buffer (expressed in days).
`spatial.buffer`	Spatial buffer size used to smooth the returned values. The unit depends on the spatial projection.
`smooth.fun`	Smoothing function applied with *spatial.buffer*.

Details

Returns environmental variables from a raster object for a given set of x and y coordinates depending on the temporal distance between the sample observation date (*obs.dates*) and the date on which the environmental data was collected (*env.dates*). Within the buffer specified by *time.buffer*, the function will search for the nearest non *NA* value with the shortest temporal distance. The user can adjust *time.buffer* to control which pixels are considered in this analysis. For example, *time.buffer* can be set to c(30,0) prompting the function to ignore environmental information acquired after the sample observation date and limit the search to -30 days. If *time.buffer* is set to null all acquisitions are considered. The user may also provide *spatial.buffer* to spatially smooth the selected environmental information. In this case, for each sample, the function will consider the neighboring pixels within the selected acquisition and apply a smoothing function defined by *smooth.fun*. If *smooth.fun* is not specified, a weighted mean will be returned by default. If *env.data* is a *data.frame* *spatial.buffer* and *smooth.fun* are ignored and *env.dates* should refer to each column.

Value

An object of class *data.frame* with the selected values and their corresponding dates.

See Also

sampleMove backSample

Examples

```
{

 require(raster)

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

 # read movement data
 data(shortMove)

 # raster dates
 file.name <- names(r.stk)
 r.dates <- as.Date(paste0(substr(file.name, 2, 5), '-',
 substr(file.name, 7, 8), '-', substr(file.name, 10, 11)))

 # sample dates
 obs.dates <- as.Date(shortMove@data$date)

 # retrieve remote sensing data for samples
 rsQuery <- dataQuery(xy=shortMove, obs.dates=obs.dates,
 env.data=r.stk, env.dates=r.dates, time.buffer=c(30,30))

}
```

---

hotMove *hotMove*

Description

Detection of geographic regions of samples using a pixel based approach.

Usage

```
hotMove(xy = xy, pixel.res = pixel.res, return.shp = FALSE)
```

Arguments

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

Details

First, the function builds a raster with a resolution equal to *pixel.res* and the spatial extent of *xy*. Then, each point in *xy* is converted into pixel coordinates. Based on the unique pixel coordinates, the function then evaluates the spatial connectivity of these pixels using a 8-neighbor connected component labeling algorithm to detect regions. Finally, the ID's are related back to each individual data point in *xy* based on their pixel coordinates and - if *return.shp* is TRUE - a polygon is derived from the convex hull of the points within each region.

Value

List containing a vector of region ID's per data point (*$indices*) and region polygons (*$polygons*).

See Also

sampleMove hotMoveStats

Examples

```
{

require(raster)

# reference data
data(longMove)

# extract regions
hm <- hotMove(xy=longMove, pixel.res=0.1, return.shp=TRUE)

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

}
```

---

hotMoveStats *hotMoveStats*

Description

Segmentation and statistical analysis of the time spent by an animal within a geographical region.

Usage

```
hotMoveStats(region.id = region.id, obs.time = obs.time,
  individual.id = NULL)
```

Arguments

region.id	Region unique identifiers. Vector of class *numeric*.
obs.time	Observation time. Object of class *Date*.
individual.id	Individual identifier. Vector of class *character*.

Details

For each unique region defined by *region.id*, the function identifies unique temporal segments defined as periods of consecutive days with observations. Then, for each region, the function uses the identified segments to report on the minimum, maximum and mean time spent as well as the total amount of time spent within the region. Moreover, the function provides a detailed report of each segment and informs on the corresponding sample indices. If *individual.id* is specified, the function will in addition count the number of individuals found within each region and within each temporal segment.

Value

A list containing statistical information for each region (*region.stats*) and for each temporal segment (*temporal.segment.stats*) and sample indices for each segment (temporal.segment.indices)

See Also

hotMove

Examples

```
{

require(raster)

# reference data
data(longMove)

# extract regions
hm <- hotMove(xy=longMove, pixel.res=0.1, return.shp=TRUE)

# plot shapefile (color by region)
plot(hm$polygons)

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

# derive statistics
hm.region.stats <- hotMoveStats(region.id=hm$indices,
obs.time=as.Date(longMove@data$timestamp))

}
```

---

imgInt *imgInt*

Description

Temporal linear interpolation of environmental data using a *raster*, *SpatialPointsDataFrames* or *data frames*.

Usage

```
imgInt(env.data = env.data, env.dates = env.dates,
  target.dates = target.dates, time.buffer = time.buffer, xy = NULL)
```

Arguments

env.data	Object of class *RasterStack*, *RasterBrick* or *data.frame*.
env.dates	Object of class *Date* with dates of *env.data*.
target.dates	Object of class *Date* with target dates.
time.buffer	A two-element vector with temporal search buffer (expressed in days).
xy	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.

Details

Performs a pixel-wise linear interpolation over a raster for a given set of dates (*target.dates*). A temporal buffer (*time.buffer*) is required to limit the search for reference data points (*time.buffer*). This is defined by a two element vector which limits the search in the past and future. If *xy* is provided and *env.data* is a *raster* object the function only considers the pixels that overlap with the shapefile. Otherwise, all pixels are considered providing a *RasterBrick*. However, if *env.data* is a *data.frame*, *xy* is ignored.

Value

A *RasterBrick* or a *data frame*. If a *RasterBrick*, each layer represents a date. If a *data.frame*, columns represent dates and rows represent samples.

See Also

dataQuery timeDir spaceDir moveSeg

Examples

```
{

 require(raster)

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

 # read movement data
 data(shortMove)

 # raster dates
 file.name <- names(r.stk)
 env.dates <- as.Date(paste0(substr(file.name, 2, 5), '-',
 substr(file.name, 7, 8), '-', substr(file.name, 10, 11)))
```

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

 # interpolate raster data to target dates
 i.env.data <- imgInt(env.data=r.stk, env.dates=env.dates,
 target.dates=target.dates, time.buffer=c(60,60), xy=shortMove)

}
```

`labelSample` *labelSample*

Description

Pixel-based labeling of spatially connected groups of samples for splitting them between training and validation.

Usage

```
labelSample(xy = xy, agg.radius = agg.radius, nr.points = NULL,
  nr.pixels = NULL, pixel.res = pixel.res)
```

Arguments

`xy`	Object of class *SpatialPoints* of *SpatialPointsDataFrame*.
`agg.radius`	Minimum radius for pixel aggregation. Unit depends on the projection of the data.
`nr.points`	Minimum number of samples per pixel.
`nr.pixels`	Minimum number of pixels per region.
`pixel.res`	Pixel resolution or a valid raster layer.

Details

First, the samples are converted to pixel coordinates and removes pixels with a corresponding number of points greater than *nr.points*. Then, if *nr.pixels* is set, the connectivity between neighboring samples is evaluated. Internally, the function will label groups of pixels based on their connectivity and regions with a pixel count smaller than the one specified by *nr.pixels* are excluded. Then, the algorithm aggregates nearby regions using a dilation algorithm within the radius specified by *agg.radius* and proceeds to reliable the pixels covered by samples. Finally, this information is used to label the original samples provided by *xy* based on their corresponding pixel coordinates. This analysis is based on the spatial extent of *xy* and a given pixel resolution (*pixel.res*). Alternatively, the user may assign a raster object to *pixel.res*.

Value

A *vector* of unique identifiers assigning each point in *xy* to their correspondent pixel region. Filtered observations are returned as *NA*.

References

Remelgado, R., Leutner, B., Safi, K., Sonnenschein, R., Kuebert, C. and Wegmann, M. (2017), Linking animal movement and remote sensing - mapping resource suitability from a remote sensing perspective. Remote Sens Ecol Conserv.

See Also

sampleMove hotMove

Examples

```
{

 require(raster)

 # read raster data
 r <- raster(system.file('extdata', '2013-07-16_ndvi.tif', package="rsMove"))

 # read movement data
 data(shortMove)

 # derive region labels
 labels <- labelSample(xy=shortMove, agg.radius=90, nr.pixels=2, pixel.res=30)

}
```

---

longMove *Example data of animal movements during a migration.*

Description

Movement data for one White Stork collected during it migration between Germany and Spain.

Usage

```
data(longMove)
```

Format

A SpatialPointsDataFrame

Details

- timestampobservation timestamp.
- longlongitude.
- latlatitude.

moveCloud *moveCloud*

Description

Provides historical information on cloud cover for a set of coordinate pairs. The temporal information is adjusted to the sample observation date.

Usage

```
moveCloud(xy = xy, obs.dates = obs.dates, data.path = NULL,
  buffer.size = NULL, remove.file = FALSE)
```

Arguments

xy	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
obs.dates	Object of class *Date* with observation dates of *xy*.
data.path	Output data path for downloaded data.
buffer.size	Two element vector with temporal buffer size (expressed in days).
remove.file	Logical. Should the files be deleted after usage?

Details

This function uses daily cloud fraction data from NASA's NEO service. For each observation date in *obs.dates*, the function downloads the correspondent image and extracts the percent of cloud cover for the corresponding samples in *xy*. If *data.path* is specified, the function will look within the provided directory for the cloud cover images. If they exist, they won't be downloaded reducing the amount of time required by the function. Moreover, if *buffer.size* is specified, for each date, the function will only consider images that are within the specified temporal buffer. *buffer.size* requires a two element vector which specifies the buffer size before and after the target dates. These additional images will be used to report on the closest time step with the lowest possible cloud cover. The final output provides a *data.frame* ($report) with information on:

- *cloud cover % (day)*: cloud cover for the observation dates.
- *best date (after)*: dates before the observation dates with the lowest cloud cover.
- *best date cloud cover % (before)*: cloud cover for best before dates.
- *best date (after)*: dates after the observation dates with the lowest cloud cover.
- *best date cloud cover % (after)*: cloud cover best after dates.

Finally, the function generates a plot ($plot) reporting on the variability of cloud cover within the dates provided by *obs.dates* and the number of samples registered within them.

Value

A *list* object reporting on the variability of cloud cover within and around each observation dates.

References

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

See Also

sMoveRes tMoveRes

Examples

```
## Not run:

 require(raster)

 # read movement data
 data(shortMove)

 # test function for 30 day buffer
 od <- as.Date(shortMove@data$date)
 c.cover <- moveCloud(xy=shortMove, obs.dates=od, data.path=".", buffer.size=c(30,30))


## End(Not run)
```

---

moveReduce *moveReduce*

Description

Pixel based summary of movement data that preserves periodic movements.

Usage

```
moveReduce(xy = xy, obs.time = obs.time, img = img)
```

Arguments

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

Details

Reduces a set of input samples (*xy*) based on their corresponding pixel coordinates within a reference raster (*img*). Using this data, the function identifies temporal segments corresponding to groups of consecutive samples found within the same pixel. In this process, revisits to recorded pixels are preserved. Once the segments are identified, the function derives mean x and y coordinates for each of them and evaluates the time spent within each pixel. The function reports on the start and end timestamps, the mean timestamp and the elapsed time. The output of the function consists of:

- *r.shp* - Shapefile with reduced sample set and its corresponding temporal information.
- *total.time* - Raster showing the total time spent at each pixel.
- *indices* - Indices for each sample in *xy* showing which samples were aggregated.

Value

A *list* object.

See Also

sampleMove moveSeg

Examples

```
{

 require(raster)

 # read raster data
 r <- raster(system.file('extdata', '2013-07-16_ndvi.tif', package="rsMove"))

 # read movement data
 data(shortMove)

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

 # reduce amount of samples
 move.reduce <- moveReduce(xy=shortMove, obs.time=obs.time, img=r)

}
```

---

moveSeg *moveSeg*

Description

Pixel based segmentation of movement data using environmental data.

Usage

```
moveSeg(xy = xy, env.data = env.data, data.type = "cont",
  threshold = threshold, obs.time = NULL, summary.fun = NULL,
  buffer.size = NULL, smooth.fun = NULL)
```

Arguments

xy	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
env.data	Object of class *RasterLayer* or *data.frame*.
data.type	Raster data data.type. One of *cont* (continuous) or *cat* (for categorical).
threshold	Change threshold. Required if *data.type* is set to *cat*.
obs.time	Object of class *Date*, *POSIXlt* or *POSIXct* with *xy* observation dates.
summary.fun	Summary function used to summarize the values within each segment when *method* is *cont*. Default is mean.
buffer.size	Spatial buffer size applied around each segment (unit depends on spatial projection).
smooth.fun	Smoothing function applied with *buffer.size* when *method* is *cont*. Default is mean.

Details

This function identifies segments of comparable environmental conditions along the movement track given by *xy*. Looking at consecutive data points, the function queries *env.data* and proceeds to identify a new segment if *threshold* is exceeded. Then, for each segment, the function summarizes *env.data* using *summary.fun* and reports on the amount of points found within it. Moreover, if *obs.time* is set, the function reports on the start and end timestamps and the elapsed time. If *method* is set as *'cont'*, the function assumes the raster data is a continuous variable. This will require the user to define *threshold* which indicates when the difference between consecutive points should be considered a change. In order to smooth the extracted values the user can specify *buffer.size*. This will prompt the function to summarize the values around each sample in *xy* using a metric define by*smooth.fun*. However, if *data.type* is set to *cat smooth.fun* is ignored. In this case, the function will report on the majority value within the buffer. The output of this function consists of:

- *indices* - Vector reporting on the segment identifiers associated to each sample in *xy*.
- *stats* - Statistical information for each segment reporting on the corresponding environmental and temporal information.
- *plot* - plot of *stats* showing the variability of environmental conditions and time spent per segment.

Value

A *list*.

See Also

dataQuery imgInt timeDir spaceDir

Examples

```
{

 require(raster)
```

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

# read movement data
data(shortMove)

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

# perform directional sampling
seg <- moveSeg(xy=shortMove, obs.time=obs.time, env.data=r, data.type="cat")

}
```

---

plotMove *plotMove*

Description

Standardized plotting of environmental and temporal information for a set of coordinate pairs.

Usage

```
plotMove(x = x, y = y, size.var = NULL, fill.var = NULL,
  var.type = NULL)
```

Arguments

x	Vector of x coordinates.
y	Vector of y coordinates.
size.var	Optional. Vector with elapsed time as report by moveReduce, sampleMove or timeDir. Controls the point size.
fill.var	Optional. Vector with environmental information. Controls the fill color.
var.type	One of 'cont' or 'cat'. Defines the type of *fill.var*.

Details

This function was designed to extent on other functions such as dataQuery, which provides environmental information, and moveReduce, which provides information on the time spent per sample. Using these two functions as an example, *plotMove* can represent the relation between the elapsed time and the change in environmental conditions.

Value

A *ggplot* object.

See Also

dataQuery moveReduce

Examples

```
{

 require(raster)

 # read raster data
 r <- raster(system.file('extdata', '2013-07-16_ndvi.tif', package="rsMove"))

 # read movement data
 data(shortMove)

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

 # reduce amount of samples
 move.reduce <- moveReduce(xy=shortMove, obs.time=time, img=r)

 # query data
 ov <- extract(r, move.reduce$points)

 # plot output
 x <- move.reduce$points@data$x
 y <- move.reduce$points@data$y
 et <- move.reduce$points@data$`Elapsed time (minutes)`
 op <- plotMove(x=x, y=y, size.var=et, fill.var=ov, var.type="cont")

}
```

---

poly2sample *poly2sample*

Description

Converts a raster grid to points depending on how much each pixel is covered by a polygon.

Usage

```
poly2sample(pol.shp = pol.shp, ref.ext = ref.ext, min.cover = NULL,
  pixel.res = NULL)
```

Arguments

pol.shp	Object of class *SpatialPolygons* or *SpatialPolygonDataFrame*.
ref.ext	Object of class *Extent* or a *RasterLayer* from which an extent can be derived.

`min.cover`	Minimum percent a pixel should be covered by a polygon for sampling (0-100). Default is 100.
`pixel.res`	Pixel resolution. Required if *ref.ext* is an *Extent* object. Unit depends on spatial projection.

Details

poly2Sample extends on the `rasterize` function from the raster package making it more efficient over large areas and converting its output into point samples rather than a raster object. For each polygon in (*"pol.shp"*), *poly2sample* extracts the overlapping pixels derived from *ref.ext*. Then, for each pixel, the function estimates the percentage of it that is covered by the reference polygon. Finally, the function extracts coordinate pairs for pixels that has a percent coverage equal to or greater than *min.cover*.

Value

A *SpatialPointsDataFrame* with sampled pixels reporting on polygon percent coverage.

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.shp=roi, ref.ext=img)

}
```

`predictResources` *predictResources*

Description

Spatially stratified predictive modeling of resource suitability based on presence/absence samples.

Usage

```
predictResources(presence.data = presence.data, absence.data = absence.data,
  sample.label = NULL, env.data = NULL)
```

Arguments

`presence.data`	Object of class *data.frame* with environmental variables for presence samples.
`absence.data`	Object of class *data.frame* with environmental variables for background samples.
`sample.label`	Numeric or character vector with sample region labels. If missing, "presence.data" is assumed as one region.
`env.data`	Object of class *RasterStack* or *RasterBrick* with environmental variables in *presence.data* and *absence.data*.

Details

Modeling of resource suitability using animal movement data following a recent paper (Remelgado et al, 2017). For each unique label in *sample.label*, the function keeps it for validation and uses the remaining samples for training. Then, the function evaluates the performance of this model reporting (internally) on the number of true positives, false positives and the number of validation and predicted cases for both presences and absences. Once all sample regions are used for validation, the reported values are summed and used to derive the F1-measure. The F1-measure is estimated as *2 * (P * R) / (P + R)* where *P* is the Precision (ratio of true positives within the number of predicted values) and *R* is the Recall (ratio of true positives within the number of validation samples). As a consequence, rather than reporting on an average performance, the final performance assessment reported by *predictResources* depicts an objective picture on how the model performed among the different sets sample regions. This metric is provided for presences (*presence.data*) and absences (*absence.data*) separately offering an overview on the stability of the model. This analysis is performed using a Random Forest model as provided within the `train` function of the caret package. The final predictive model is then derived with all samples. The output of *predictResources* is a list object consisting of:

- *f1* - *data.frame* with final F1-measure for presences and absences.
- validation - *data.frame* with region identifiers and validation sample count at each iteration.
- *iteration.models* - List of models estimated at each iteration.
- *final.model* - Final predictive model based on all samples.
- *probabilities* - Predicted probability image. Given if *env.data* is set.

Value

A *list*.

References

Remelgado, R., Leutner, B., Safi, K., Sonnenschein, R., Kuebert, C. and Wegmann, M. (2017), Linking animal movement and remote sensing - mapping resource suitability from a remote sensing perspective. Remote Sens Ecol Conserv.

See Also

sampleMove labelSample backSample train

Examples

```
## Not run:

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

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

 # read movement data
 data(shortMove)

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

 # remove redundant samples
 shortMove <- moveReduce(xy=shortMove, obs.time=obs.time, img=rsStk)$points

 # retrieve remote sensing data for samples
 rsQuery <- extract(rsStk, shortMove)

 # identify unique sample regions
 label <- labelSample(xy=shortMove, agg.radius=90, pixel.res=rsStk)

 # select background samples
 ind <- which(!is.na(label)) # selected samples
 bSamples <- backSample(xy=shortMove[ind,], region.id=label[ind],
 img=rsStk, sampling.method='pca')

 # derive model predictions
 out <- predictResources(presence.data=rsQuery,
 absence.data=bSamples@data, sample.label=label, env.data=rsStk)


## End(Not run)
```

---

rsComposite *rsComposite*

Description

Phenological and date driven Pixel Based Compositing (PBC) of remote sensing data supported by GPS tracking date information.

Usage

```
rsComposite(img, img.dates, obs.dates, comp.method = "closest",
  temporal.buffer = NULL)
```

Arguments

`img`	Object of class *RasterStack* or *RasterBrick*.
`img.dates`	Object of class *Date* with *img* observation dates.
`obs.dates`	Object of class *Date* with reference dates.
`comp.method`	One of "closest" or "phenological".
`temporal.buffer`	Search buffer (expressed in days). The default is 30.

Details

The function uses a multi-layer raster object to build a composite for a reference date which corresponds to the median of *obs.dates*. Moreover, the function determines the Median Absolute Deviation (MAD) of *obs.dates* which determines the temporal buffer that is used to search for usable images. As an alternative, *temporal.buffer* can be specified manually and will be required if *obs.dates* consists of a single value. The user can also specify how the compositing should be done. *comp.method* can be set to: #'

- *closest* - Uses layer with the closest possible date in relation to the reference date.
- *phenological* - Uses the layer with the Day of the Year (DoY) in relation to the reference date.

The final output of *rsComposite* is a list consisting of: #'

- *composite* - Final image composite
- *dates* - Temporal composition of the composite reporting on the julian day
- *pixel.count* - pixel count of unique values in *dates*. Additionally, it reports on NA values.
- *target.date* - Reference date used during compositing.
- *temporal.buffer* - Temporal buffer used during compositing.

If *pheno2* is used, for each pixel, the function will 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`

Examples

```
## Not run:

 require(raster)

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

 # raster dates
 file.name <- names(r.stk)
 img.dates <- as.Date(paste0(substr(file.name, 2, 5), '-',
 substr(file.name, 7, 8), '-', substr(file.name, 10, 11)))

 # target date
 obs.dates = as.Date("2013-06-01")

 # build composite
 r.comp <- rsComposite(r.stk, img.dates, obs.dates, comp.method="closest", temporal.buffer=90)


## End(Not run)
```

---

rsMove *rsMove.*

Description

rsMove.

---

sampleMove *sampleMove*

Description

Remote sensing oriented sampling of stops along a movement track.

Usage

```
sampleMove(xy = xy, obs.time = obs.time, search.radius = search.radius,
  method = "m", tUnit = NULL)
```

Arguments

xy	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
obs.time	Object of class *Date*, *POSIXlt* or *POSIXct* with the same length as *xy*.
search.radius	Sample search radius (in meters).
method	How should the distance 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 finds location where an animal showed little or no movement based on GPS tracking data. It looks at the distance among consecutive samples and places pointer when the distance is bellow the defined threshold (*search.radius*). When a pointer is found, the function looks at the distance between the pointer and the following samples. While this is below the distance threshold, the samples are assigned to the same segment. Then, for each segment, the function summarizes the corresponding samples deriving mean coordinates, the start, end and total time spent and the total number of samples per segment ('count'). The user should selected *method* in accordance with the projection system associated to the data. If 'm' it bases this analysis on the the ecludian distance. However, if 'deg' it set, the function uses the haversine formula.

Value

A *SpatialPointsDataFrame* with a reduced sample set.

See Also

labelSample backSample dataQuery

Examples

```
{

 require(raster)

# reference data
data(longMove)

 # sampling without reference grid
 obs.time = strptime(longMove$timestamp, "%Y-%m-%d %H:%M:%S")
 output <- sampleMove(xy=longMove, obs.time=obs.time, search.radius=7, method='deg')

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

}
```

segRaster *segRaster*

Description

Connected-region based raster segmentation that preserves spatial gradients.

Usage

```
segRaster(img, break.point = 0.1, min.value = 0.5)
```

Arguments

img	Object of class *RasterLayer*.
break.point	Difference threshold. Default is 0.05.
min.value	Minimum value. Default is 0.5.

Details

The function segments an input raster layer (*img*) using a connected component region labeling approach. For each pixel, the function estimates the difference between it and its immediate neighbors. If the difference is below the threshold defined by *break.point* these are aggregated into a single region. Moreover, the user can define a minimum pixel value using *min.value* which will ignore all pixels below that value. The output of this function consists of:

- *regions* - Region raster image.
- *stats* - Basic statistics for each pixel region.

Value

A list object.

See Also

predictResources

Examples

```
{

 require(raster)

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

 # segment probabilities
 rs <- segRaster(r)
```

```
}
```

`shortMove` *Example data of animal movements during the nesting period.*

Description

Movement data for one White Stork collected within its nesting site.

Usage

```
data(shortMove)
```

Format

A SpatialPointsDataFrame

Details

- xx coordinate.
- yy coordinate.
- dateobservation date.
- timeobservation time.

`sMoveRes` *sMoveRes*

Description

Tool to support the selection of an adequate satellite spatial resolution. Evaluates how the change in spatial resolution changes the amount of samples and sample regions based on a set of coordinate pairs.

Usage

```
sMoveRes(xy = xy, pixel.res = pixel.res)
```

Arguments

`xy`	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
`pixel.res`	vector of spatial resolutions (unit depends on spatial projection).

Details

Given a vector of pixel resolutions (*pixel.res*), the function determines the number of unique pixels and unique pixel regions after their temporal aggregation. For each spatial resolution, the function starts by converting *xy* to unique pixel coordinates and labels them based on their spatial aggregation. Then, the function counts the number of samples and sample regions. The output of the function consists of:

- *stats* - Summary statistics reporting on the number of unique samples and sample regions per spatial resolution.
- *plot* - Plot representing the change in number of samples and sample regions per spatial resolution.
- *indices* - Indices for each sample in *xy* based on their spatial aggregation within each spatial resolution.

Value

A *list*.

See Also

tMoveRes specVar

Examples

```
{

 require(raster)

 # read movement data
 data(shortMove)

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

}
```

---

spaceDir *spaceDir*

Description

Analysis of environmental change in space along a set of coordinate pairs.

Usage

```
spaceDir(xy = xy, obs.time = NULL, img = img,
  sample.direction = sample.direction, data.type = data.type,
  distance.method = "m", buffer.size = NULL, stat.fun = NULL,
  min.count = 2)
```

Arguments

xy	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
obs.time	Object of class *Date*, *POSIXlt* or *POSIXct* with *xy* observation dates.
img	Object of class *RasterLayer*.
sample.direction	One of *forward*, *backward* or *both*. Default is *both*.
data.type	One of 'cont' or 'cat'. Defines which type of variable is in use.
distance.method	One of 'm' or 'deg' specifying the projection unit. Default is 'm'.
buffer.size	Spatial buffer size expressed in the map units.
stat.fun	Output statistical metric.
min.count	Minimum number of pixels required by *stat.fun*. Default is 2.

Details

This function evaluates how do environmental conditions change in space along a movement track. For each set of consecutive points, the function applies a spatial moving window which boundaries depend on the definition of *sample.direction*. Then, within each segment, the function extracts all pixels within it. If *buffer.size* is defined, the function will consider a buffer when performing this extraction. Finally, the the extracted *NA* values are summarized into a given metric. If *data.type* is *cont*, a statistical function can be provided through *stat.fun*. However, if *data.type* is *cat*, the function will report on the dominant class and on the shannon index for each segment. Note that the function will work with the raster value associated to each class. On top of this, *spaceDir* will also report on the linear distance traveled between endpoints (in meters) and the travel time (in minutes). The output of the function is a list consisting of:

- *endpoints* - Point shapefile with endpoints of each spatial segment. Reports on a given statistical metric, traveled distance, travel time and the mean timestamp.
- *segments* - Line shapefile with spatial segments. Reports on the same information as *endpoints*. *plot* - Plotting of *segments* where each segment is colored according to its corresponding statistical value.

Value

A *list* containing shapefiles with information on environmental change and travel distance/time and a plot of the results.

See Also

timeDir dataQuery imgInt

Examples

```
{

 require(raster)
```

```
# read raster data
r <- raster(system.file('extdata', '2013-07-16_ndvi.tif', package="rsMove"))

# read movement data
data(shortMove)

# observation time
obs.time <- strptime(paste0(shortMove@data$date, ' ',shortMove@data$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=shortMove, obs.time=obs.time, img=r,
sample.direction="backward", data.type='cont', stat.fun=of)

}
```

---

specVar *specVar*

Description

Tool to support the selection of adequate satellite spatial resolution. Evaluates how the spectral variability within a pixel change with the change in spatial resolution.

Usage

```
specVar(img = img, pixel.res = pixel.res)
```

Arguments

img	Object of class *RasterLayer*.
pixel.res	Spatial resolution (unit depends on the spatial projection).

Details

Given a raster object (*img*), the function determines how degrading its spatial resolution impacts our ability to perceive the complexity of the landscape. For the pixel resolution given by *pixel.res*, The function resamples *img* and estimates the Mean Absolute Percentage Error (MAPE) for each pixel. The MAPE is estimated as $100/n * sum(abs(O - A/O)$ where *O* are the original value in *img*, *A* the aggregated value in the aggregated image and *n* the number of non-NA pixels in the original image The output of the function consists of:

- *mape* - MAPE raster.
- *plot* - Histogram of *mape*.

Value

A *list*.

See Also

tMoveRes sMoveRes

Examples

```
## Not run:

 require(raster)

 # read raster data
 r <- raster(system.file('extdata', '2013-07-16_ndvi.tif', package="rsMove"))

 # apply function
 s.var <- specVar(img=r, pixel.res=60)


## End(Not run)
```

---

timeDir *timeDir*

Description

Analysis of environmental change in time for a set of coordinate pairs.

Usage

```
timeDir(xy = NULL, obs.dates = obs.dates, img = NULL, env.data = NULL,
  env.dates = env.dates, temporal.buffer = temporal.buffer,
  stat.fun = NULL, min.count = 2)
```

Arguments

xy	Object of class "SpatialPoints" or "SpatialPointsDataFrame".
obs.dates	Object of class *Date* with *xy* observation dates.
img	Object of class
env.data	Object of class *RasterStack* or *RasterBrick* or *data.frame*.
env.dates	Object of class *Date* with *env.data* observation dates.
temporal.buffer	two element vector with temporal window size (expressed in days).
stat.fun	Output statistical metric.
min.count	Minimum number of samples required by *stat.fun*. Default is 2.

Details

This function evaluates how environmental conditions change in time along a movement track. First, for each point in *xy*, the function compares its observation date (*obs.dates*) against the acquisition dates (*env.dates*) of *env.data* to select non *NA* timesteps within a predefined temporal window (*temporal.buffer*). The user can adjust this window to determine which images are the most important. For example, if one wishes to know how the landscape evolved up to the observation date of the target sample and daily satellite data is available, *temporal.buffer* can be define as, e.g., c(30,0) forcing the function to use all images to only use pixels recorded within the previous 30 days. After selecting adequate temporal information for each data point, a statistical metric is estimated. The statistical metric is provided by (*stat.fun*). By default, the slope is reported from a linear regression between the acquisition times of *env.data* and their corresponding values. When providing a new function, set x for *env.dates* and y for *env.data*.

Value

A *vector* with a requested statistical metric for each point in *xy*.

See Also

spaceDir dataQuery imgInt

Examples

```
{

 require(raster)

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

 # read movement data
 data(shortMove)

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

 # sample dates
 obs.dates <- as.Date(shortMove@data$date)

 # perform directional sampling
 of <- function(x,y) {lm(y~x)$coefficients[2]}
 time.env <- timeDir(xy=shortMove, obs.dates=obs.dates, env.data=r.stk,
 env.dates=r.dates, temporal.buffer=c(30,30), stat.fun=of)

}
```

tMoveRes *tMoveRes*

Description

Tool to support the selection of an adequate satellite temporal resolution. It evaluates how the change in temporal resolution changes the amount of samples and sample regions based on a set of coordinate pairs and their observation dates.

Usage

```
tMoveRes(xy = xy, obs.date = obs.date, time.res = time.res,
  pixel.res = pixel.res)
```

Arguments

xy	Object of class *SpatialPoints* or *SpatialPointsDataFrame*.
obs.date	Object of class *Date* with *xy* observation dates.
time.res	Vector of temporal resolutions (expressed in days).
pixel.res	Spatial resolution (unit depends on spatial projection).

Details

Given a base spatial resolution (*pixel.res* and a vector of temporal resolutions (*time.res*), the function determines the number of unique pixels and unique pixel regions after their temporal aggregation. For each temporal resolution, the function starts by converting *xy* to unique pixel coordinates and labels them based on their spatial aggregation. Then, the function counts the number of samples and sample regions. The output of the function consists of:

- *stats* - Summary statistics reporting on the number of temporal widows, unique samples and unique sample regions per temporal resolution.
- *plot* - Plot representing the change in number of samples and sample regions per temporal resolution.
- *indices* - Indices for each sample in *xy* based on their spatial aggregation within each temporal resolution.

Value

A *list* object reporting on the amount and distribution of unique pixels and connected pixel regions per temporal resolution.

See Also

sMoveRes specVar

Examples

```
{

 require(raster)

 # reference data
 data(longMove)

 # test function for intervals of 1, 8 and 16 days (e.g. of MODIS data)
 obs.date <- as.Date(longMove@data$timestamp)
 a.res <- tMoveRes(xy=longMove, obs.date=obs.date, time.res=c(1,8,16), pixel.res=0.01)

}
```

Index