Package 'foster'

November 5, 2020

Type Package

Title Forest Structure Extrapolation with R

Version 0.1.0

Description Set of tools to streamline the modeling of the relationship between satellite imagery time series or any other environmental information, such as terrain elevation, with forest structural attributes derived from 3D point cloud data and their subsequent imputation over the broader landscape.

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Imports raster, reshape2, dplyr, stats, RStoolbox, yaImpute, sp, tools, spatstat, randomForest, rgdal, caret, trend, data.table

Suggests ggplot2, knitr, rmarkdown

VignetteBuilder knitr

License GPL-3

Depends R (>= 3.5.0)

BugReports https://github.com/mqueinnec/foster/issues

NeedsCompilation no

Author Martin Queinnec [cre, aut],
Piotr Tompalski [aut],
Douglas Bolton [aut],
Nicholas Coops [aut]

Maintainer Martin Queinnec <queinnec@mail.ubc.ca>

Repository CRAN

Date/Publication 2020-11-05 07:50:02 UTC

R topics documented:

`accuracy` *Calculate accuracy metrics*

Description

Calculate coefficient of determination (R2), root-mean square error (RMSE) and bias between predictions and observations of continuous variables.

Usage

```
accuracy(obs, preds, vars = NULL, folds = NULL)
```

Arguments

`obs`	A vector of observed values
`preds`	A vector of predicted values
`vars`	Optional vector indicating different variables
`folds`	Optional vector indicating the folds

Details

R2 is calculated with the following formula:

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y}_i)^2}$$

RMSE is calculated with the following formula:

$$RMSE = \sqrt{\frac{1}{n}\sum(\hat{y}_i - y_i)^2}$$

Bias is calculated with the following formula:

$$Bias = \frac{\sum(\hat{y}_i - y_i)}{n}$$

Relative RMSE and bias are also calculated by dividing their value by the mean of observations.

If accuracy assessment was performed using k-fold cross-validation the accuracy metrics are calculated for each fold separately. The mean value of the accuracy metrics across all folds is also returned.

Value

Data frame with following columns:

`vars` Response variable

`R2` R2

`RMSE` RMSE

`RMSE_rel` Relative RMSE

`bias` bias

`bias_rel` Relative bias

`count` Number of observations

Examples

```
# kNN_preds is a data frame obtained from foster::trainNN
# It contains predictions and observations of the trained kNN model
load(system.file("extdata/examples/kNN_preds.RData",package="foster"))

accuracy(obs = kNN_preds$obs,
         preds = kNN_preds$preds,
         vars = kNN_preds$variable,
         folds = kNN_preds$Fold)
```

`calcIndices` *Calculate spectral indices from multispectral data*

Description

Calculate spectral indices (e.g. NDVI, tasseled cap coefficients etc.) from multispectral data. Calculations are based on the functions `spectralIndices` and `tasseledCap`. Refer to the documentation of these functions for more details.

Usage

```
calcIndices(
  x,
  indices = "NDVI",
  sat = NULL,
  blue = NULL,
  green = NULL,
  red = NULL,
  nir = NULL,
  swir1 = NULL,
  swir2 = NULL,
  swir3 = NULL,
  coefs = list(L = 0.5, G = 2.5, L_evi = 1, C1 = 6, C2 = 7.5, s = 1, swir2ccc = NULL,
    swir2coc = NULL),
  filename = "",
  par = FALSE,
  threads = 2,
  m = 2,
  progress = TRUE,
  ...
)
```

Arguments

`x`	Raster* or SpatialPointsDataFrame object or list of Raster* or SpatialPointsDataFrame objects.
`indices`	Character vector indicating Which indices are calculated. Tasseled Cap indices are abbreviated as `TCB`, `TCW`, `TCG`, `TCA`, `TCD`. For a list of other supported indices see `spectralIndices`
`sat`	Character. If calculating tasseled cap indices, name of the sensor needs to be provided. One of: c("Landsat4TM", "Landsat5TM", "Landsat7ETM", "Landsat8OLI", "MODIS", "QuickBird", "Spot5", "RapidEye"). See `tasseledCap`.
`blue`	Integer. Blue band.
`green`	Integer. Green band.
`red`	Integer. Red band.

nir	Integer. Near infrared band (700-1100 nm).
swir1	temporarily deprecated
swir2	Integer. Shortwave infrared band (1400-1800 nm)
swir3	Integer. Shortwave infrared band (2000-2500 nm)
coefs	Coefficients necessary to calculate some of the spectral indices (e.g. EVI). See spectralIndices.
filename	Character. Output file name including path to directory and eventually extension. If x is a list, filename must be a vector of characters with one file name for each element of x. Default is "" (output not written to disk).
par	Logical. Should the function be executed on parallel threads
threads	Number of parallel threads used if par = TRUE
m	tuning parameter to determine how many blocks will be used (m blocks will be processed by each cluster)
progress	Logical. If TRUE (default) a progress bar is displayed when using parallel processing.
...	Other arguments passed to writeRaster or writeOGR.

Details

If x is a Raster* or list of Raster* objects, each layer should be one of the spectral bands used to calculate the indices. If x is a SpatialPointsDataFrame or list of spatialPointsDataFrame, each column should be a spectral band. When calculating tasseledCap indices, bands should be provided in a specific order specified in tasseledCap.

Tasseled Cap Angle (TCA) and Distance (TCD) are calculated from greenness (TCG) and brightness (TCB) as follows:

$$TCA = \arctan(\frac{TCG}{TCB})$$

$$TCD = \sqrt{TCB^2 + TCG^2}$$

If x is a list of Raster* objects, the processing can be parallelized using cluster. In that case the user has to set par = TRUE and provide the number of parallel threads threads. You can control how many blocks will be processed by each thread by setting m (see cluster).

Value

Raster* or SpatialPointsDataFrame object or list of Raster* or SpatialPointsDataFrame objects.

See Also

spectralIndices, tasseledCap, cluster

Examples

```
library(raster)

# Open Landsat BAP image
BAP_2006 <- stack(system.file("extdata/examples/Landsat_BAP_2006.tif",package =
                              "foster"))

# Calculate NDVI
VI_2006 <- calcIndices(BAP_2006,
                       indices = "NDVI",
                       red=3,
                       nir=4)
```

`defaultTemporalSummary` *Default temporal summary*

Description

Calculates median, IQR and Theil Sen slope (`sens.slope`). This function is usually called within `temporalMetrics`

Usage

```
defaultTemporalSummary(x)
```

Arguments

x	Vector of numeric values

Value

Named vector with median, IQR and slope

See Also

`temporalMetrics`, `sens.slope`

Examples

```
x <- rnorm(100)
defaultTemporalSummary(x)
```

edges *Assign NA values to the neighborhood of a boundary cell*

Description

Assigns NA value to all cells having a NA values within their w x w neighborhood.

Usage

```
edges(x, w, filename = "", ...)
```

Arguments

x	A Raster* object
w	Numeric. Size of the window around each cell. Must be an odd number.
filename	Character. Output file name including path to directory and eventually extension. Default is "" (output not written to disk).
...	Additional arguments passed to writeRaster

Value

Raster* object

See Also

focal

Examples

```
# Load raster package
library(raster)

# Open and stack ALS metrics
elev_p95 <- raster(system.file("extdata/examples/ALS_metrics_p95.tif",package="foster"))
cover <- raster(system.file("extdata/examples/ALS_metrics_cov_mean.tif",package="foster"))
Y_vars <- stack(elev_p95,cover)

# Remove edges in a 3 x 3 neighborhood
Y_vars_edges <- edges(Y_vars, w=3)
```

`focalMultiBand` *Apply a spatial filter to a Raster* object*

Description

Apply a spatial filter to a RasterLayer or all layers of a RasterStack or RasterBrick object. The mathematical operation applied within the neighborhood can be done by using a function (`fun`) or by setting the weights of the matrix `w`.

Usage

```
focalMultiBand(
  x,
  w,
  fun,
  filename = "",
  na.rm = FALSE,
  pad = FALSE,
  padValue = NA,
  NAonly = FALSE,
  keepNA = TRUE,
  ...
)
```

Arguments

`x`	Raster* object or list of Raster* objects.
`w`	Matrix of weights (moving window). A 3x3 windows with weights of 1 would be `w=matrix(1,nr=3,nc=3)` for example.
`fun`	Function (optional). The function should accept a vector of values and return a single number (e.g. mean). It should also accept a `na.rm` argument.
`filename`	Character. Output file name including path to directory and eventually extension. If `x` is a list, `filename` must be a vector of characters with one file name for each element of `x`. Default is `""` (output not written to disk).
`na.rm`	Logical. If TRUE (default), NAs are removed from computation
`pad`	Logical. IF TRUE, rows and columns are added around `x` to avoid removing border cells.
`padValue`	Numeric. Value of pad cells. Usually set to NA and used in combination with `na.rm=TRUE`
`NAonly`	Logical. If TRUE only cell values that are NA are replaced with the computed focal values.
`keepNA`	Logical. If TRUE (default), NA cells of `x` are unchanged
`...`	Additional arguments passed to `writeRaster`

Details

If x contains NA values and `na.rm = TRUE` is used , using `fun` or `w` with weights adjusted to apply equivalent mathematical operation might not produce the same outputs (in that case using weights would give wrong results). See the documentation of `focal` for more information.

Also, cells of x with NA values might get a non-NA value assigned when located in the neighborhood of non-NA cells and `na.rm = TRUE` is used. In that case, setting `keepNA = TRUE` (default) ensures that NA cells of x still have NA values in the output raster.

Value

Raster* object or list of Raster* objects.

See Also

`focal`

Examples

```
# Load raster package
library(raster)

# Open and stack ALS metrics
elev_p95 <- raster(system.file("extdata/examples/ALS_metrics_p95.tif",package="foster"))
cover <- raster(system.file("extdata/examples/ALS_metrics_cov_mean.tif",package="foster"))
Y_vars <- stack(elev_p95,cover)

#Define 3x3 filter with weights of 1
filt <- matrix(1, nrow = 3, ncol = 3)

# Smoothing
Y_vars_smooth <- focalMultiBand(x = Y_vars,
                                w=filt,
                                fun=mean,
                                pad=TRUE,
                                padValue=NA,
                                na.rm=TRUE,
                                keepNA = TRUE)
```

---

getSample *Stratified random sampling*

Description

Performs kmeans clustering to stratify x and randomly samples within the strata until n samples are selected. The number of samples selected in each strata is proportional to the occurrence of those strata across the classified raster.

Usage

```
getSample(
  x,
  strata = 5,
  layers,
  norm = TRUE,
  n,
  mindist = 0,
  maxIter = 30,
  xy = TRUE,
  filename_cluster = "",
  filename_sample = "",
  ...
)
```

Arguments

x	A Raster* object used to generate random sample
strata	Number of strata (kmeans clusters). Default is 5.
layers	Vector indicating the bands of x used in stratification (as integer or names). By default, all layers of x are used.
norm	Logical. If TRUE (default), x is normalized before k-means clustering. This is useful if layers have different scales.
n	Sample size
mindist	Minimum distance between samples (in units of x). Default is 0.
maxIter	Numeric. This number is multiplied to the number of samples to select per strata. If the number of iterations to select samples exceeds maxIter x the number of samples to select then the loop will break and a warning message be returned. Default is 30.
xy	Logical indicating if X and Y coordinates of samples should be included in the fields of the returned SpatialPoints object.
filename_cluster	Character. Output filename of the clustered x raster including path to directory and eventually extension
filename_sample	Character. Output filename of the sample points including path to directory. File will be automatically saved as an ESRI Shapefile and any extension in filename_sample will be overwritten
...	Further arguments passed to unsuperClass, writeRaster or writeOGR to control the kmeans algorithm or writing parameters

Details

x is stratified using kmeans clustering from unsuperClass. By default, clustering is performed on a random subset of x (10000 cells) and run with multiple starting configurations in order to find a convergent solution from the multiple starts. The parameters controlling the number of random

samples used to perform kmeans clustering and the number of starting configurations can be provided as additional ... arguments. More information on the behavior of the kmeans clustering can be found in unsuperClass. The default kmeans clustering method is Hartigan-Wong algorithm. The algorithm might not converge and output "Quick Transfer" warning. If this is the case, we suggest decreasing strata. Also, if mindist is too large, it might not be possible to select enough samples per strata. In that case, the warning "Exceeded maximum number of runs for strata" is displayed. In that case you can decrease the number of samples n or increase maxIter to control the number of maximum iterations allowed until the required number of samples are selected.

Value

A list with the following objects:

sample A SpatialPoints object containing sampled points

clusterMap The clustered x raster, output of unsuperClass

model The kmeans model, output of unsuperClass

See Also

unsuperClass

Examples

```
# Load raster package
library(raster)

# Open and stack ALS metrics
elev_p95 <- raster(system.file("extdata/examples/ALS_metrics_p95.tif",package="foster"))
cover <- raster(system.file("extdata/examples/ALS_metrics_cov_mean.tif",package="foster"))
Y_vars <- stack(elev_p95,cover)
names(Y_vars) <- c("p95","cover")

# Sample 5 cells in 3 strata (kmeans clusters). Sampled points should be at least 30 m apart.
set.seed(1234) #for example reproducibility
sample_strata <- getSample(Y_vars,
                           n = 5,
                           strata = 3,
                           mindist = 30)
```

---

getSampleValues *Extract raster values at sample points*

Description

Given a Raster* object and a SpatialPointsDataFrame object, the functions returns a SpatialPointsDataFrame objects with the values of the raster at sample points.

Usage

```
getSampleValues(x, s, keepCols = FALSE, filename = "", ...)
```

Arguments

x	A Raster* object
s	Location of the sample points. Object of class SpatialPointsDataFrame generated with getSample
keepCols	Should the columns of s be retained? Default is FALSE
filename	Character. Output filename including path to directory. File will be automatically saved as an ESRI Shapefile and any extension in filename will be overwritten
...	Additional arguments passed to writeOGR

Value

SpatialPointsDataFrame object

See Also

extract

Examples

```
# Load raster package
library(raster)

# Open and stack ALS metrics
elev_p95 <- raster(system.file("extdata/examples/ALS_metrics_p95.tif",package="foster"))
cover <- raster(system.file("extdata/examples/ALS_metrics_cov_mean.tif",package="foster"))
Y_vars <- stack(elev_p95,cover)
names(Y_vars) <- c("p95","cover")

# sample_points is a SpatialPointsDataFrame calculated and saved from getSample
# Load it into memory
load(system.file("extdata/examples/sample_points.RData",package="foster"))

getSampleValues(Y_vars, sample_points)
```

---

matchExtent *Match the extent of a reference raster*

Description

This function crops or extends the extent of a raster to the extent of a reference. Some cells of the reference raster can optionally be masked based on their values.

Usage

```
matchExtent(
  x,
  ref,
  mask = FALSE,
  inverse = FALSE,
  maskValue = NA,
  filename = "",
  ...
)
```

Arguments

x	Raster* object or list of Raster* objects.
ref	Raster* object. x extent will be matched to ref extent.
mask	Logical. Should x be masked by ref cells that have the value maskValue
inverse	Logical. If TRUE, cells of ref that are not maskvalue are masked
maskValue	Value of ref cells that should be masked in x. Default is NA.
filename	Character. Output file name including path to directory and eventually extension. If x is a list, filename must be a vector of characters with one file name for each element of x. Default is "" (output not written to disk).
...	Other arguments passed to writeRaster

Details

x and ref need to have the same CRS, spatial resolution and origin. If this is not the case, you can use matchResolution before matchExtent.

Value

Raster* object or list of Raster* objects.

See Also

crop, extend, mask

Examples

```
# Load raster package
library(raster)

# Open ALS p95 and mask of forested areas as Raster objects
BAP_2006 <- stack(system.file("extdata/examples/Landsat_BAP_2006.tif",package="foster"))
mask_forest <- raster(system.file("extdata/examples/VLCE_forest_2008.tif",package="foster"))

matchExtent(BAP_2006, mask_forest, mask = TRUE)
```

`matchResolution` *Match the resolution of two Raster* objects*

Description

Successively projects (if necessary) and resamples a raster coordinate system and spatial resolution to the reference

Usage

```
matchResolution(x, ref, method = "bilinear", filename = "", ...)
```

Arguments

`x`	Raster* object or list of Raster* objects.
`ref`	Reference Raster* object with parameters that `x` should be resampled to.
`method`	Character. Method used to compute values for the resampled raster. Can be `'bilinear'` for bilinear interpolation or `'ngb'` for nearest neighbor interpolation. See `resample`.
`filename`	Character. Output file name including path to directory and eventually extension. If `x` is a list, `filename` must be a vector of characters with one file name for each element of `x`. Default is `""` (output not written to disk).
`...`	Other arguments passed to `writeRaster`

Details

`x` and `ref` must have defined CRS (can be assigned using `projection`). If the CRS don't match, `x` is projected to `ref` CRS prior to resampling. `x` doesn't inherit the extent of `ref`.

Value

Raster* object or list of Raster* objects.

See Also

`resample`, `projectRaster`, `projection`

Examples

```
# Load raster package
library(raster)

# Open ALS metric and Landsat BAP imagery
elev_p95 <- raster(system.file("extdata/examples/ALS_metrics_p95.tif",package="foster"))
BAP_2006 <- stack(system.file("extdata/examples/Landsat_BAP_2006.tif",package="foster"))

matchResolution(x = elev_p95,ref = BAP_2006,method='bilinear')
```

partition *Split data into training and testing sets*

Description

Returns the row indices of x that should go to training or validation.

Usage

```
partition(
  x,
  type = "group holdout",
  p = 0.75,
  kfold = 5,
  groups = min(5, length(x)),
  returnTrain = TRUE
)
```

Arguments

x	A vector used for splitting data
type	Character. Type of partition. Valid values are "random holdout", "group holdout" or "kfold"
p	percentage of data that goes to training set (holdout). Only relevant if type = "random holdout" or type = "group holdout"
kfold	Number of folds for cross-validation. Only relevant if type = "kfold".
groups	For "group holdout" and when x is numeric, this is the number of breaks in the quantiles
returnTrain	Logical indicating whether training or validation indices should be returned. Default is TRUE.

Details

Three types of splits are currently implemented. "random holdout" randomly selects p percents of x for the training set. "group holdout" first groups x into groups quantiles and randomly samples within them (see createDataPartition) . "kfold" creates k folds where p percent of the data is used for training in each fold (see createFolds). This function is a wrapper around two functions of caret package: createDataPartition and createFolds

Value

List containing training or validation indices

See Also

createDataPartition

Examples

```
# sample_points is a SpatialPointsDataFrame calculated and saved from getSample
# Load it into memory
load(system.file("extdata/examples/sample_points.RData",package="foster"))

partition(sample_points$cluster, type = "kfold", kfold = 5)
```

`predictTrgs` *Impute response variables across the landscape*

Description

This function finds the k-NN of target observations and imputes response variables. `X` is a raster object where each layer correspond to one of the predictor variable used to train the k-NN model `model` obtained from `trainNN`.

Usage

```
predictTrgs(
  model = NULL,
  x = NULL,
  nrows = 200,
  nnID = TRUE,
  nnDist = TRUE,
  filename = "",
  par = FALSE,
  threads = 2,
  progress = TRUE,
  ...
)
```

Arguments

`model`	A trained kNN model obtained from `trainNN`
`x`	Raster object where each layer corresponds to a predictor variable calculated at targets
`nrows`	number of rows processed at a time. Default is 200 .
`nnID`	Logical. Should the ID of each target's nearest neighbor used for imputation be returned?
`nnDist`	Logical. Should the distance to each target's nearest neighbor used for imputation be returned?
`filename`	Character. Output file name including path to directory and eventually extension.Default is `""` (output not written to disk).
`par`	Logical. Should imputation be performed on parallel threads?
`threads`	Integer. Number of parallel threads (relevant only if par=TRUE)
`progress`	Logical. If TRUE (default) a progress bar is displayed.
`...`	Other arguments passed to `writeRaster`

Details

The method used to impute the NN is set from the kNN model trained by `trainNN`. If `k=1` the value of the single closest NN is imputed. If `k>1`, the closest, mean, median or weighted distance mean (default) of all k NN values is imputed. This is set using the `impute.cont` and `impute.fac` arguments of `trainNN`.

The raster `x` is processed as blocks of `nrows` to avoid creating very large objects (several Gb) that couldn't be stored in memory. However, low values of `nrows` slow down processing. Depending on the amount of RAM available on your computer and on the size of the area where k-NN need to be calculated, it is possible to process more rows at the same time and considerably reduce processing time.

Value

A RasterStack object where the first layers correspond to the imputed response variables and the remaining layers to the nearest neighbor(s) ID (if `nnID = TRUE`) and nearest neighbor(s) distance (if `nnDist = TRUE`)

See Also

`newtargets`, `impute.yai`

Examples

```
# Load data
# kNN_model: trained kNN model (from trainNN)
# X_vars: RasterStack of predictor variables
load(system.file("extdata/examples/example_predictTrgs.RData", package =
"foster"))

Y_imputed <- predictTrgs(model=kNN_model, x = X_vars, nnID = TRUE,
nnDist = TRUE)
```

---

scatter *Scatterplot with information on the errors between x and y.*

Description

Scatterplot between a vector of observed data and a vector of predicted data with information on the errors between them.

Usage

```
scatter(obs, preds, vars, info = TRUE)
```

Arguments

obs	A vector of observed values
preds	A vector of predicted values
vars	Optional vector indicating different variables
info	A logical value indicating whether information on count, R2, bias and RMSE should be added to the plot

Details

Accuracy metrics are calculated from accuracy

Value

A ggplot2 object or a list of ggplot2 objects (one per variable)

See Also

accuracy

Examples

```
# kNN_preds is a data frame obtained from foster::trainNN
# It contains predictions and observations of the trained kNN model
load(system.file("extdata/examples/kNN_preds.RData",package="foster"))

scatter(obs = kNN_preds$obs,
        preds = kNN_preds$preds,
        vars = kNN_preds$variable)
```

---

temporalMetrics *Calculate temporal summary metrics*

Description

This function calculates a set of user-defined or default statistics from spectral indices time series.

Usage

```
temporalMetrics(
  x,
  metrics = "defaultTemporalSummary",
  filename = "",
  stack = TRUE,
  par = FALSE,
  threads = 2,
  progress = TRUE,
  m = 2,
  ...
)
```

Arguments

x	List of Raster* or SpatialPointsDataFrame objects.Input Raster or SpatialPoints-DataFrame object containing a time series (may be generated with calcIndices)
metrics	Name of a function used to process the time series provided as a character.
filename	Character. Single output filename including path to directory and eventually extension. Each spectral index is written separately and the name of the spectral index is automatically appended to the file name.
stack	Logical. Should the output be returned as a single RasterStack (TRUE) or as a list containing one Raster per vegetation index (FALSE)
par	Logical. Should the function be executed in parallel threads
threads	Number of parallel threads used if par = TRUE
progress	Logical. If TRUE (default) a progress bar is displayed.
m	tuning parameter to determine how many blocks will be used (m blocks will be processed by each cluster)
...	Other arguments passed to writeRaster or writeOGR.

Details

Spectral indices can be calculated with calcIndices. The input to TemporalMetrics is a list where each element is a Raster* or a SpatialPointsDataFrame object with layers or columns being spectral indices. Each element should be one step in the time series and elements should be ordered in the time series ascending order. The argument fun defines which metrics will be calculated. It has to be the name of a function that takes a vector as input and returns a named vector corresponding to the summary metrics. The function defaultTemporalSummary is used by default and returns the median, IQR and Theil-Sen slope of the time series.

If x is a list of Raster* objects, the processing can be parallelized using cluster. In that case the user has to set par = TRUE and provide the number of parallel threads threads. You can control how many blocks will be processed by each thread by setting m (see cluster).

See Also

calc, cluster

Examples

```
# VI_ts is a list of Raster* calculated and saved from calcIndices
# Load it into memory
load(system.file("extdata/examples/VI_ts.RData",package="foster"))

temporalMetrics(VI_ts, metrics = "defaultTemporalSummary")

# User-defined temporal summary metrics can also be used
funSummary <- function(x) {
  c(
    mean = mean(x, na.rm = TRUE),
    median = median(x, na.rm = TRUE),
    std = sd(x, na.rm = TRUE)
```

```
  )
}
```

`theilSen` *Theil-Sen slope*

Description

Calculate the Theil-Sen slope from a time series. This is a wrapper around `sens.slope`

Usage

```
theilSen(x)
```

Arguments

`x`	A numeric vector

Value

numeric; Theil-Sen slope

See Also

`sens.slope`

Examples

```
x <- rnorm(100)
theilSen(x)
```

`tile` *Split a raster into tiles*

Description

This function is used to split a raster into smaller tiles. The raster is split in a grid pattern with `nx` columns and `ny` rows.

Usage

```
tile(x, nx, ny, filename = "", suffix = NULL, ...)
```

Arguments

x	Raster* object to split
nx	Number of horizontal cells in the splitting grid
ny	Number of vertical cells in the splitting grid
filename	Character. Output file name including path to directory and eventually extension.Default is "" (output not written to disk).
suffix	Character appended to filename to differentiate tiles (must have length nx x ny). If left NULL, tiles will be numbered by columns and rows
...	Additional parameters passed to writeRaster

Value

A list of Raster* objects

See Also

crop

Examples

```
# Load raster package
library(raster)

elev_p95 <- stack(system.file("extdata/examples/ALS_metrics_p95.tif",package="foster"))

# Split elev_p95 into a 1 x 2 grid
tile(elev_p95, nx = 1, ny = 2)
```

---

trainNN *Train and assess accuracy of a k-NN model*

Description

This function trains a k-NN model from response variables (Y) and predictors (X) at reference observations using the package yaImpute (see yai). By default, the distance between observations is obtained from the proximity matrix of random forest regression or classification trees. Optionally, training and testing sets can be provided to return the accuracy of the trained k-NN model.

Usage

```
trainNN(
  x,
  y,
  inTrain = NULL,
  inTest = NULL,
  k = 1,
```

```
  method = "randomForest",
  impute.cont = NULL,
  impute.fac = NULL,
  ntree = 500,
  mtry = NULL,
  rfMode = "",
  ...
)
```

Arguments

x	A dataframe or SpatialPointsDataFrame of predictors variables X for reference observations. Row names of X are used as identification of reference observations.
y	A dataframe or SpatialPointsDataFrame of response variables Y for the reference observations. Row names of Y are used as identification of reference observations.
inTrain	Optional. A list obtained from partitionindicating which rows of x and y go to training.
inTest	Optional list indicating which rows of x and y go to validation. If left NULL, all rows that are not in inTrain are used for validation.
k	Integer. Number of nearest neighbors
method	Character. Which nearness metrics is used to compute the nearest neighbors. Default is "randomForest". Other methods are listed in yai
impute.cont	Character. The method used to compute the imputed continuous variables. Can be "closest", "mean", "median" or "dstWeighted". Default is "closest" if k = 1 and "dstWeighted" if k > 1. See impute.yai for more details.
impute.fac	Character. The method used to compute the imputed values for factors. Default value is the same as impute.cont. See impute.yai for more details.
ntree	Number of classification or regression trees drawn for each response variable. Default is 500
mtry	Number of X variables picked randomly to split each node. Default is sqrt(number of X variables)
rfMode	By default, rfMode is set to "" which forces yai to create random forest regression trees instead of classification trees for continuous variables. Can be set to "buildClasses" if wanting continuous variables to be converted to classes and forcing random forest to build classification trees. (See yai)
...	Other arguments passed to yai (e.g. "rfXsubsets")

Details

If performing model validation, the function trains a kNN model from the training set, finds the k NN of the validation set and imputes the response variables from the k NN. If k = 1, only the closest NN value is imputed. If k > 1, the imputed value can be either the closest NN value, the mean, median or distance weighted mean of the k NN values.This is controlled by the arguments impute.cont or impute.fac.

If inTest = NULL, all rows that are not in inTrain will be used for model testing. If inTrain = NULL, all rows that are not in inTest will be used for model training. If both inTrain and inTest are NULL, all rows of x and y will be used for training and no testing is performed.

The final model returned by `findNN` is trained from all observations of `x` and `y`.

Value

A list containing the following objects:

`model` A `yai` object, the trained k-NN model

`preds` A data.frame with observed and predicted values of the testing set for each response variables

See Also

`yai`, `newtargets`, `impute.yai`, `accuracy`

Examples

```
# Load data in memory
# X_vars_sample: Predictor variables at sample (from getSample)
# Y_vars_sample: Response variables at sample (from getSample)
# train_idx: Rows of X_vars_sample and Y_vars_sample that are used for
# training (from (partition))
load(system.file("extdata/examples/example_trainNN.RData",package="foster"))

set.seed(1234) #for example reproducibility
kNN <- trainNN(x = X_vars_sample,
               y=Y_vars_sample,
               inTrain = train_idx,
               k = 1,
               method = "randomForest",
               ntree = 200)
```

---

varImp *Returns variable importance*

Description

When RF is used to find nearest neighbors, the importance of each variable in the RF trees is calculated. This function returns the importance of each variable and a `ggplot2` object

Usage

```
varImp(model, scaled = TRUE, plot = TRUE, plotType = "boxplot")
```

Arguments

model	A yai object
scaled	Logical. Should importance values be centered and scaled?
plot	Logical. If TRUE, returns a ggplot2 object based on plotType value
plotType	Either of "boxplot" or "grid"

Details

If scaled = TRUE, importance values are centered by subtracting their mean and scaled by dividing the centered importance by their standard deviation.

Value

A list containing the following objects:

importance A data.frame object containing the importance of each response variable and the mean importance of all variables combined

plot A ggplot object showing a plot of the importance values according to plotType

See Also

importance, yaiVarImp

Examples

```
# Load data
# kNN_model: trained kNN model (from trainNN)
load(system.file("extdata/examples/example_predictTrgs.RData", package = "foster"))

varImp(kNN_model,scaled=FALSE,plot=TRUE,plotType="boxplot")
```

Index