Package: RStoolbox (via r-universe)

October 29, 2026

Type Package

Title Remote Sensing Data Analysis

Version 1.0.2.3

Date 2026-07-20

Description Toolbox for remote sensing image processing and analysis such as calculating spectral indexes, principal component transformation, unsupervised and supervised classification or fractional cover analyses.

URL https://bleutner.github.io/RStoolbox/, https://github.com/bleutner/RStoolbox

BugReports https://github.com/bleutner/RStoolbox/issues

Encoding UTF-8

Depends R (>= 3.5.0)

Imports caret (>= 6.0-79), sf, terra (>= 1.6-41), XML, dplyr, ggplot2, tidyr, reshape2, lifecycle, exactextractr, Rcpp, methods, magrittr

Suggests randomForest, lattice, kernlab, e1071, gridExtra, pls, testthat, themis, ROSE

LinkingTo Rcpp, RcppArmadillo

License GPL (>= 3)

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Benjamin Leutner [aut] (ORCID: <https://orcid.org/0000-0002-6893-2002>), Ned Horning [aut], Jakob Schwalb-Willmann [aut] (ORCID: <https://orcid.org/0000-0003-2665-1509>), Robert J. Hijmans [ctb] (ORCID: <https://orcid.org/0000-0001-5872-2872>), Konstantin Mueller [aut, cre] (ORCID: <https://orcid.org/0000-0001-6540-3124>)

Maintainer Konstantin Mueller <konstantinfinn.mueller@gmx.de>

Repository https://cran.r-universe.dev

Date/Publication 2026-07-22 12:09:12 UTC

RemoteUrl https://github.com/cran/RStoolbox

RemoteRef HEAD

RemoteSha cb24c2c423b429a10045499cc15d3b865d47512d

Contents

classifyQA . . . 3
cloudMask . . . 4
cloudShadowMask . . . 6
coregisterImages . . . 8
decodeQA . . . 10
encodeQA . . . 11
estimateHaze . . . 12
fCover . . . 14
fortifySpatRaster . . . 16
getMeta . . . 17
getValidation . . . 18
ggR . . . 19
ggRGB . . . 22
histMatch . . . 24
ImageMetaData . . . 26
lsat . . . 28
mesma . . . 29
normImage . . . 31
oneHotEncode . . . 32
panSharpen . . . 33
pifMatch . . . 34
predict.unsuperClass . . . 36
radCor . . . 37
rasterCVA . . . 39
rasterEntropy . . . 41
rasterPCA . . . 41
readEE . . . 43
readMeta . . . 44
readSLI . . . 44
rescaleImage . . . 45
rlogo . . . 47
rsOpts . . . 47
RStoolbox . . . 48
sam . . . 49
saveRSTBX . . . 50
sen2 . . . 52
spectralIndices . . . 52
srtm . . . 54

srtm_sen2 55
stackMeta 55
superClass 56
tasseledCap 59
topCor 61
unsuperClass 62
validateMap 64
writeSLI 66

Index **68**

`classifyQA` *Classify Landsat QA bands*

Description

extracts five classes from QA band: background, cloud, cirrus, snow and water.

Usage

```
classifyQA(
  img,
  type = c("background", "cloud", "cirrus", "snow", "water"),
  confLayers = FALSE,
  sensor = "OLI",
  legacy = "collection1",
  ...
)
```

Arguments

`img`	SpatRaster. Landsat 8 OLI QA band.
`type`	Character. Classes which should be returned. One or more of c("background", "cloud", "cirrus","snow", "water").
`confLayers`	Logical. Return one layer per class classified by confidence levels, i.e. cloud:low, cloud:med, cloud:high.
`sensor`	Sensor to encode. Options: `c("OLI", "TIRS", "ETM+", "TM", "MSS")`.
`legacy`	Encoding systematic Options: `c("collection1", "pre_collection")`. Default is "collection1" for the Landsat Collection 1 8-bit quality designations. Use "pre_collection" for imagery downloaded before the Collection 1 quality designations were introduced
`...`	further arguments passed to writeRaster

Details

By default each class is queried for *high* confidence. See encodeQA for details. To return the different confidence levels per condition use `confLayers=TRUE`. This approach corresponds to the way LandsatLook Quality Images are produced by the USGS.

Value

Returns a SpatRaster with maximal five classes:

class	value
background	1L
cloud	2L
cirrus	3L
snow	4L
water	5L

Values outside of these classes are returned as NA. If `confLayers = TRUE` then a RasterStack with one layer per condition (except 'background') is returned, whereby each layer contains the confidence level of the condition.

Confidence	value
low	1L
med	2L
high	3L

See Also

encodeQA decodeQA

Examples

```
library(terra)
qa <- rast(ncol = 100, nrow=100, val = sample(1:2^14,  10000))

## QA classes
qacs <- classifyQA(img = qa)
## Confidence levels
qacs_conf <- classifyQA(img = qa, confLayers = TRUE)
```

`cloudMask` *Simple Cloud Detection*

Description

Developed for use with Landsat data `cloudMask` relies on the distinctive difference between the blue (or any other short-wave band) and thermal band for semi-automated creation of a cloud mask. Since it relies on thermal information it doesn't work well for sensors without thermal bands.

Usage

```
cloudMask(
  x,
  threshold = 0.2,
  blue = "B1_sre",
  tir = "B6_sre",
  buffer = NULL,
  plot = FALSE,
  verbose
)
```

Arguments

x	SpatRaster with reflectance and brightness temperature OR the mask of a previous run of cloudMask with returnDiffLayer=TRUE.
threshold	Numeric. cloud detection threshold. If not provided it will be guessed. Everything *below* this threshold will be considered a cloud pixel (unless it is removed by filtering afterwards).
blue	Character or integer. Bandname or number for the blue band
tir	Character or integer. Bandname or number for the thermal band
buffer	Integer. Number of pixels to use as a buffer that will be added to the identified cloud centers.
plot	Logical. Plots of the cloud mask for all sub-steps (sanitizing etc.) Helpful to find proper parametrization.
verbose	Logical. Print messages or suppress.

Value

Returns a SpatRaster with two layers: CMASK contains the binary cloud mask (1 = cloud, NA = not-cloud) and NDTCI contains the cloud index.

Note

Typically clouds are cold in the thermal region and have high reflectance in short wavelengths (blue). By calculating a normalized difference index between the two bands and thresholding a rough cloud mask can be obtained. Before calculating the spectral cloud index (let's call it Normalized Difference Thermal Cloud Index (NDTCI)) the thermal band will be matched to the same value range as the blue band. Therefore, it doesn't matter whether you provide DN, radiance or brightness temperature.

This approach to cloud masking is very simplistic. And aims only at rough removal of potentially clouded areas. Nevertheless, it is able to provide satisfactory results. More sophisticated approaches, including cloud cast shadow detection can be found elsewhere, e.g. fmask.

It can make sense to find a suitable threshold on a cropped version of the scene. Also make sure you make use of the returnDiffLayer argument to save yourself one processing step. Buffering should be seen as final polishing, i.e. as long as the pure cloud centers are not detected properly, you might want to turn it off. since it takes some time to calculate. Once your mask detects obvious cloud

pixels properly re-enable buffering for fine tuning if desired. Finally, once a suitable threshold is established re-run cloudMask on the whole scene with this threshold and go get a coffee.

See Also

cloudShadowMask

Examples

```
library(ggplot2)
## Import Landsat example subset
## We have two tiny clouds in the east
ggRGB(lsat, stretch = "lin")

## Calculate cloud index
cldmsk    <- cloudMask(lsat, blue = 1, tir = 6)
ggR(cldmsk, 2, geom_raster = TRUE)

## Define threshold (re-use the previously calculated index)
## Everything above the threshold is masked
## In addition we apply a region-growing around the core cloud pixels
cldmsk_final <- cloudMask(cldmsk, threshold = 0.1, buffer = 5)

## Plot cloudmask
ggRGB(lsat, stretch = "lin") +
   ggR(cldmsk_final[[1]], ggLayer = TRUE, forceCat = TRUE, geom_raster = TRUE) +
   scale_fill_manual(values = c("red"), na.value = NA)

#' ## Estimate cloud shadow displacement
## Interactively (click on cloud pixels and the corresponding shadow pixels)
## Not run:  shadow <- cloudShadowMask(lsat, cldmsk_final, nc = 2)

## Non-interactively. Pre-defined shadow displacement estimate (shiftEstimate)
shadow <- cloudShadowMask(lsat, cldmsk_final, shiftEstimate = c(-16,-6))

## Plot
csmask <- terra::merge(cldmsk_final[[1]], shadow)
ggRGB(lsat, stretch = "lin") +
        ggR(csmask, ggLayer = TRUE, forceCat = TRUE, geom_raster = TRUE) +
        scale_fill_manual(values = c("blue", "yellow"),
        labels = c("shadow", "cloud"), na.value = NA)
```

---

cloudShadowMask *Cloud Shadow Masking for Flat Terrain*

Description

Intended for interactive use, cloudShadowMask will ask the user to select a few corresponding cloud/cloudShadow pixels which will be used to estimate coordinates for a linear cloudmask shift.

Usage

```
cloudShadowMask(
  img,
  cm,
  nc = 5,
  shiftEstimate = NULL,
  preciseShift = NULL,
  quantile = 0.2,
  returnShift = FALSE
)
```

Arguments

`img`	SpatRaster containing the scene
`cm`	SpatRaster. Cloud mask (typically the result of `cloudMask`)
`nc`	Integer. Number of control points. A few points (default) are fine because the final shift is estimated by coregisterImages.
`shiftEstimate`	NULL or numeric vector of length two (x,y). Estimated displacement of shadows in map units. If `NULL`, the user will be asked to select control points interactively.
`preciseShift`	NULL or numeric vector of length two (x,y). Use this if cloud/cloud-shadow displacement is already known, e.g. from a previous run of `cloudShadowMask`.
`quantile`	Numeric (between 0 and 1). Quantile threshold used for image co-registration. By default the 20% quantile of the total intensity (sum) of the image is used as potential shadow mask.
`returnShift`	Logical. Return a numeric vector containing the shift parameters. Useful if you estimate parameters on a subset of the image.

Details

This is a very simplistic approach to cloud shadow masking (simple shift of the cloud mask). It is not image based and accuracy will suffer from clouds at different altitudes. However, just as cloudMask this is a quick and easy to use tool for Landsat data if you're just working on a few scenes and don't have fMask or CDR data at hand. Although for some test scenes it does perform surprisingly well.

Value

Returns a RasterLayer with the cloud shadow mask (0 = shadow, NA = not-shadow).

See Also

cloudMask

Examples

```
library(ggplot2)
## Import Landsat example subset
## We have two tiny clouds in the east
ggRGB(lsat, stretch = "lin")

## Calculate cloud index
cldmsk    <- cloudMask(lsat, blue = 1, tir = 6)
ggR(cldmsk, 2, geom_raster = TRUE)

## Define threshold (re-use the previously calculated index)
## Everything above the threshold is masked
## In addition we apply a region-growing around the core cloud pixels
cldmsk_final <- cloudMask(cldmsk, threshold = 0.1, buffer = 5)

## Plot cloudmask
ggRGB(lsat, stretch = "lin") +
   ggR(cldmsk_final[[1]], ggLayer = TRUE, forceCat = TRUE, geom_raster = TRUE) +
   scale_fill_manual(values = c("red"), na.value = NA)

#' ## Estimate cloud shadow displacement
## Interactively (click on cloud pixels and the corresponding shadow pixels)
## Not run:  shadow <- cloudShadowMask(lsat, cldmsk_final, nc = 2)

## Non-interactively. Pre-defined shadow displacement estimate (shiftEstimate)
shadow <- cloudShadowMask(lsat, cldmsk_final, shiftEstimate = c(-16,-6))

## Plot
csmask <- terra::merge(cldmsk_final[[1]], shadow)
ggRGB(lsat, stretch = "lin") +
        ggR(csmask, ggLayer = TRUE, forceCat = TRUE, geom_raster = TRUE) +
        scale_fill_manual(values = c("blue", "yellow"),
        labels = c("shadow", "cloud"), na.value = NA)
```

`coregisterImages` *Image to Image Co-Registration based on Mutual Information*

Description

Shifts an image to match a reference image. Matching is based on maximum mutual information.

Usage

```
coregisterImages(
  img,
  ref,
  shift = 3,
  shiftInc = 1,
```

```
  nSamples = 100,
  reportStats = FALSE,
  verbose,
  nBins = 100,
  master = deprecated(),
  slave = deprecated(),
  ...
)
```

Arguments

img	SpatRaster. Image to shift to match reference image. img and ref must have equal numbers of bands.
ref	SpatRaster. Reference image. img and ref must have equal numbers of bands.
shift	Numeric or matrix. If numeric, then shift is the maximal absolute radius (in pixels of img resolution) which img is shifted (seq(-shift, shift, by=shiftInc)). If shift is a matrix it must have two columns (x shift and y shift), then only these shift values will be tested.
shiftInc	Numeric. Shift increment (in pixels, but not restricted to integer). Ignored if shift is a matrix.
nSamples	Integer. Number of samples to calculate mutual information.
reportStats	Logical. If FALSE it will return only the shifted images. Otherwise it will return the shifted image in a list containing stats such as mutual information per shift and joint histograms.
verbose	Logical. Print status messages. Overrides global RStoolbox.verbose option.
nBins	Integer. Number of bins to calculate joint histogram.
master	DEPRECATED! Argument was renamed. Please use ref from now on.
slave	DEPRECATED! Argument was renamed. Please use img from now on.
...	further arguments passed to writeRaster.

Details

Currently only a simple linear x - y shift is considered and tested. No higher order shifts (e.g. rotation, non-linear transformation) are performed. This means that your imagery should already be properly geometrically corrected.

Mutual information is a similarity metric originating from information theory. Roughly speaking, the higher the mutual information of two data-sets, the higher is their shared information content, i.e. their similarity. When two images are exactly co-registered their mutual information is maximal. By trying different image shifts, we aim to find the best overlap which maximises the mutual information.

Value

reportStats=FALSE returns a SpatRaster (x-y shifted image). reportStats=TRUE returns a list containing a data.frame with mutual information per shift ($MI), the shift of maximum MI ($bestShift), the joint histograms per shift in a list ($jointHist) and the shifted image ($coregImg).

Examples

```
library(terra)
library(ggplot2)
library(reshape2)
reference <- rlogo
## Shift reference 2 pixels to the right and 3 up
missreg <- shift(reference,  2,  3)

## Compare shift
p <- ggR(reference, sat = 1, alpha = .5)
p + ggR(missreg, sat = 1, hue = .5, alpha = 0.5, ggLayer=TRUE)

## Coregister images (and report statistics)
coreg <- coregisterImages(missreg, ref = reference,
                          nSamples = 500, reportStats = TRUE)

## Plot mutual information per shift
ggplot(coreg$MI) + geom_raster(aes(x,y,fill=mi))

## Plot joint histograms per shift (x/y shift in facet labels)

df <- melt(coreg$jointHist)
df$L1 <- factor(df$L1, levels = names(coreg$jointHist))
df[df$value == 0, "value"] <- NA ## don't display p = 0
ggplot(df) + geom_raster(aes(x = Var2, y = Var1,fill=value)) + facet_wrap(~L1) +
       scale_fill_gradientn(name = "p", colours =  heat.colors(10), na.value = NA)

## Compare correction
ggR(reference, sat = 1, alpha = .5) +
  ggR(coreg$coregImg, sat = 1, hue = .5, alpha = 0.5, ggLayer=TRUE)
```

---

decodeQA *Decode QA flags to bit-words*

Description

Intended for use with Landsat 16-bit QA bands. Decodes pixel quality flags from integer to bit-words.

Usage

```
decodeQA(x)
```

Arguments

x	Integer (16bit)

Value

Returns the decoded QA values from an integer

See Also

encodeQA

Examples

```
decodeQA(53248)
```

---

encodeQA *Encode QA Conditions to Integers*

Description

Intended for use with Landsat 16-bit QA bands. Converts pixel quality flags from human readable to integer, which can then be used to subset a QA image. Please be aware of the default settings which differ for different parameters. Depending on, which `sensor` and `legacy` is selected, some quality parameters are not used, since the sequences of available bitwise quality designations differ per sensor and collection.

Usage

```
encodeQA(
  fill = "no",
  terrainOcclusion = "no",
  radSaturation = "na",
  cloudMask = "all",
  cloud = "all",
  cloudShadow = "all",
  snow = "all",
  cirrus = "all",
  droppedPixel = "no",
  water = "all",
  droppedFrame = "no",
  sensor = "OLI",
  legacy = "collection1"
)
```

Arguments

`fill`	Designated fill. Options: `c("yes", "no", "all")`.
`terrainOcclusion`	Terrain induced occlusion. Options: `c("yes", "no", "all")`.
`radSaturation`	Number of bands that contain radiometric saturation. Options: `c("na", "low", "med", "high", "all")` for no bands, 1-2 bands, 3-4 bands, 5 or more bands contain saturation.
`cloudMask`	Cloud mask. Options: `c("yes", "no", "all")`.
`cloud`	Cloud confidence. Options: `c("na", "low", "med", "high", "all")`.

cloudShadow	Cloud shadow confidence. Options: c("yes", "no", "all").
snow	Snow / ice confidence. Options: c("na", "low", "med", "high", "all").
cirrus	Cirrus confidence. Options: c("na", "low", "med", "high", "all").
droppedPixel	Dropped pixel. Options: c("yes", "no", "all").
water	Water confidence. Options: c("na", "low", "med", "high", "all").
droppedFrame	Dropped frame. Options: c("yes", "no", "all").
sensor	Sensor to encode. Options: c("OLI", "TIRS", "ETM+", "TM", "MSS").
legacy	Encoding systematic Options: c("collection1", "pre_collection"). Default is "collection1" for the Landsat Collection 1 8-bit quality designations. Use "pre_collection" for imagery downloaded before the Collection 1 quality designations were introduced

Value

Returns the Integer value for the QA values

Note

Only currently populated bits are available as arguments.

References

https://www.usgs.gov/landsat-missions/landsat-collection-1-level-1-quality-assessment-band for Collection 1 quality designations (legacy = "collection1")

Examples

```
encodeQA(snow = "low", cirrus = c("med", "high"), cloud = "high")
```

---

estimateHaze *Estimate Image Haze for Dark Object Subtraction (DOS)*

Description

estimates the digital number (DN) pixel value of *dark* objects for the visible wavelength range.

Usage

```
estimateHaze(
  x,
  hazeBands,
  darkProp = 0.01,
  maxSlope = TRUE,
  plot = FALSE,
  returnTables = FALSE
)
```

Arguments

x	SpatRaster or a previous result from estimateHaze with returnTables = TRUE from which to estimate haze
hazeBands	Integer or Character. Band number or bandname from which to estimate atmospheric haze (optional if x contains only one layer)
darkProp	Numeric. Proportion of pixels estimated to be dark.
maxSlope	Logical. Use darkProp only as an upper boundary and search for the DN of maximum slope in the histogram below this value.
plot	Logical. Option to display histograms and haze values
returnTables	Logical. Option to return the frequency table per layer. Only takes effect if x is a SpatRaster. If x is a result of estimateHaze tables will always be returned.

Details

It is assumed that any radiation originating from *dark* pixels is due to atmospheric haze and not the reflectance of the surface itself (the surface is dark, i.e. it has a reflectance close to zero). Hence, the haze values are estimates of path radiance, which can be subtracted in a dark object subtraction (DOS) procedure (see radCor)

Atmospheric haze affects almost exclusively the visible wavelength range. Therefore, typically, you'd only want to estimate haze in blue, green and red bands, occasionally also in the nir band.

Value

If returnTables is FALSE (default). Then a vector of length(hazeBands) containing the estimated haze DNs will be returned. If returnTables is TRUE a list with two components will be returned. The list element 'SHV' contains the haze values, while 'table' contains another list with the sampled frequency tables. The latter can be re-used to try different darkProp thresholds without having to sample the raster again.

Examples

```
## Estimate haze for blue, green and red band
haze <- estimateHaze(lsat, hazeBands = 1:3, plot = FALSE)
haze

## Find threshold interactively
#### Return the frequency tables for re-use
#### avoids having to sample the Raster again and again
haze <- estimateHaze(lsat, hazeBands = 1:3, returnTables = TRUE)
## Use frequency table instead of lsat and fiddle with
haze <- estimateHaze(haze, hazeBands = 1:3, darkProp = .1, plot = FALSE)
haze$SHV
```

fCover *Fractional Cover Analysis*

Description

fCover takes a classified high resolution image, e.g. vegetation and non-vegetation based on Landsat and calculates cover fractions for pixels of a coarser resolution, e.g. MODIS.

Usage

```
fCover(
  classImage,
  predImage,
  nSamples = 1000,
  classes = 1,
  maxNA = 0,
  clamp = TRUE,
  model = "rf",
  tuneLength = 3,
  trControl = trainControl(method = "cv"),
  method = deprecated(),
  filename = NULL,
  overwrite = FALSE,
  verbose,
  ...
)
```

Arguments

classImage	high resolution SpatRaster containing a landcover classification, e.g. as obtained by superClass.
predImage	coarse resolution SpatRaster for which fractional cover will be estimated.
nSamples	Integer. Number of pixels to sample from predImage to train the regression model
classes	Integer. Classes for which fractional cover should be estimated (one or more).
maxNA	Numeric. Maximal proportion of NAs allowed in training pixels.
clamp	Logical. Enforce results to stay within [0,1] interval. Values <0 are reset to 0 and values >1 to 1.
model	Character. Which model to fit for image regression. See train for options. Defaults to randomForest ('rf')
tuneLength	Integer. Number of levels for each tuning parameters that should be generated by train. See Details and train.
trControl	trainControl object, specifying resampling, validation etc.

method	DEPREACTED! in favor of trControl=trainControl(method="cv") Resampling method for parameter tuning. Defaults to 10 fold cross-validation. See trainControl for options.
filename	Character. Filename of the output raster file (optional).
overwrite	Logical. if TRUE, filename will be overwritten.
verbose	Logical. Print progress information.
...	further arguments to be passed to writeRaster

Details

fCover gets the pixel values in a high resolution classified image that correspond to randomly selected moderate resolution pixels and then calculates the percent of the classified image pixels that represent your cover type of interest. In other words, if your high resolution image has a pixel size of 1m and your moderate resolution image has a pixel size of 30m the sampling process would take a block of 900 of the 1m resolution pixels that correspond to a single 30m pixel and calculate the percentage of the 1m pixels that are forest. For example, if there were 600 forest pixels and 300 non-forest pixels the value given for the output pixel would be 0.67 since 67

fCover relies on the train() function from the caret package which provides access to a huge number of classifiers. Please see the available options at train. The default classifier (randomForest) we chose has been shown to provide very good results in image regression and hence it is recomended you start with this one. If you choose a different classifier, make sure it can run in regression mode.

Many models require tuning of certain parameters. Again, this is handled by train from the caret package. With the tuneLength argument you can specify how many different values of each tuning parameter should be tested. The Random Forest algorithm for example can be tuned by varying the mtry parameter. Hence, by specifying tuneLength = 10, ten different levels of mtry will be tested in a cross-validation scheme and the best-performing value will be chosen for the final model.

If the total no-data values for a block of high resolution pixels is greater than maxNA then it will not be included in the training data set since there is too much missing data to provide a reliable cover percentage. If the no-data proporton is less then maxNA the no-data pixels are removed from the total number of pixels when calculating the percent cover.

Value

Returns a list with two elements: models contains the fitted models evaluated in tenfold cross-validation (caret train objects); fCover contains a SpatRaster with a fractional cover layer for each requested class.

See Also

superClass

Examples

```
library(terra)
library(caret)
## Create fake input images
agg.level <- 9
modis <- terra::aggregate(rlogo, agg.level)
```

```
## Perform an exemplary classification
lc      <- unsuperClass(rlogo, nClass=2)

## Calculate the true cover, which is of course only possible in this example,
## because the fake corse resolution imagery is exactly res(rlogo)*9
trueCover <- terra::aggregate(lc$map, agg.level,
                  fun = function(x, ...){sum(x == 1, ...)/sum(!is.na(x))})

## Run with randomForest and support vector machine (radial basis kernel)
## Of course the SVM is handicapped in this example,
## due to poor tuning (tuneLength)
par(mfrow=c(2,3))
for(model in c("rf", "svmRadial")){
   fc <- fCover(
           classImage = lc$map ,
           predImage = modis,
           classes=1,
           trControl = trainControl(method = "cv", number = 3),
           model=model,
           nSample = 50,
           tuneLength=2
   )

   ## How close is it to the truth?
   compare.rf <- trueCover - fc$map
   plot(fc$map, main = paste("Fractional Cover: Class 1\nModel:", model))
   plot(compare.rf, main = "Diffence\n true vs. predicted")
   plot(trueCover[],fc$map[],  xlim = c(0,1), ylim =c(0,1),
           xlab = "True Cover", ylab = "Predicted Cover" )
   abline(coef=c(0,1))
   rmse <- sqrt(global(compare.rf^2, "sum", na.rm = TRUE))/ncell(compare.rf)
   r2 <- cor(trueCover[], fc$map[], "complete.obs")
   text(0.9,0.1, adj=1, labels =
        paste(c("RMSE:","\nR2:"), round(unlist(c(rmse, r2)),3), collapse=""))
}

## Reset par
par(mfrow=c(1,1))
```

`fortifySpatRaster` *Fortify method for classes from the terra package.*

Description

Fortify method for classes from the terra package.

Usage

```
fortifySpatRaster(x, maxpixels = 50000)
```

Arguments

x	SpatRaster object to convert into a dataframe.
maxpixels	Integer. Maximum number of pixels to sample

Value

Returns a data.frame with coordinates (x,y) and corresponding raster values.

Examples

```
r_df <- fortifySpatRaster(rlogo)
head(r_df)
```

---

getMeta *Extract bandwise information from ImageMetaData*

Description

This is an accessor function to quickly access information stored in ImageMetaData, e.g. scale factor per band. Intended for use with imagery which was imported using stackMeta. Will return parameters using the actual band order in img.

Usage

```
getMeta(img, metaData, what)
```

Arguments

img	SpatRaster or character vector with band names.
metaData	ImageMetaData or path to meta data file.
what	Character. Parameter to extract. Either data descriptors, or conversion parameters (see Details for options)

Details

Possible metadata parameters (what argument):

Data descriptors

'FILES'
'QUANTITY'
'CATEGORY'
'NA_VALUE'
'SATURATE_VALUE'
'SCALE_FACTOR'
'DATA_TYPE'
'SPATIAL_RESOLUTION'

Conversion parameters

'CALRAD'	Conversion parameters from DN to radiance
'CALBT'	Conversion parameters from radiance to brightness temperature
'CALREF'	Conversion parameters from DN to reflectance (Landsat 8 only)

Value

If `what` is one of `c('CALRAD', 'CALBT', 'CALREF')` a data.frame is returned with bands in rows (order corresponding to `img` band order). Otherwise a named numeric vector with the corresponding parameter is returned (layernames as names).

Examples

```
## Import example data
mtlFile <- system.file("external/landsat/LT52240631988227CUB02_MTL.txt", package="RStoolbox")
meta <- readMeta(mtlFile)
lsat_t <- stackMeta(mtlFile)

## Get integer scale factors
getMeta(lsat_t, metaData = meta, what = "SCALE_FACTOR")

## Conversion factors for brightness temperature
getMeta("B6_dn", metaData = meta, what = "CALBT")

## Conversion factors to top-of-atmosphere radiance
## Band order not corresponding to metaData order
getMeta(lsat_t[[5:1]], metaData = meta, what = "CALRAD")

## Get integer scale factors
getMeta(lsat_t, metaData = meta, what = "SCALE_FACTOR")

## Get file basenames
getMeta(lsat_t, metaData = meta, what = "FILES")
```

---

getValidation *Extract validation results from superClass objects*

Description

Extract validation results from superClass objects

Usage

```
getValidation(x, from = "testset", metrics = "overall")
```

Arguments

x	superClass object or caret::confusionMatrix
from	Character. 'testset' extracts the results from independent validation with testset. 'cv' extracts cross-validation results.
metrics	Character. Only relevant in classification mode (ignored for regression models). Select 'overall' for overall accuracy metrics, 'classwise' for classwise metrics, 'confmat' for the confusion matrix itself and 'caret' to return the whole caret::confusionMatrix object.

Value

Returns a data.frame with validation results. If metrics = 'confmat' or 'caret' will return a table or the full caret::confusionMatrix object, respectively.

Examples

```
library(pls)
## Fit classifier (splitting training into 70\% training data, 30\% validation data)
train <- readRDS(system.file("external/trainingPoints_rlogo.rds", package="RStoolbox"))
SC   <- superClass(rlogo, trainData = train, responseCol = "class",
                   model="pls", trainPartition = 0.7)
## Independent testset-validation
getValidation(SC)
getValidation(SC, metrics = "classwise")
## Cross-validation based
getValidation(SC, from = "cv")
```

---

ggR *Plot RasterLayers in ggplot with greyscale*

Description

Plot single layer imagery in grey-scale. Can be used with a SpatRaster.

Usage

```
ggR(
  img,
  layer = 1,
  maxpixels = 5e+05,
  alpha = 1,
  hue = 1,
  sat = 0,
  stretch = "none",
  quantiles = c(0.02, 0.98),
  ext = NULL,
  coord_equal = TRUE,
```

```
  ggLayer = FALSE,
  ggObj = TRUE,
  geom_raster = FALSE,
  forceCat = FALSE
)
```

Arguments

img	SpatRaster
layer	Character or numeric. Layername or number. Can be more than one layer, in which case each layer is plotted in a subplot.
maxpixels	Integer. Maximal number of pixels to sample.
alpha	Numeric. Transparency (0-1).
hue	Numeric. Hue value for color calculation [0,1] (see hsv). Change if you need anything else than greyscale. Only effective if sat > 0.
sat	Numeric. Saturation value for color calculation [0,1] (see hsv). Change if you need anything else than greyscale.
stretch	Character. Either 'none', 'lin', 'hist', 'sqrt' or 'log' for no stretch, linear, histogram, square-root or logarithmic stretch.
quantiles	Numeric vector with two elements. Min and max quantiles to stretch to. Defaults to 2% stretch, i.e. c(0.02,0.98).
ext	Extent object to crop the image
coord_equal	Logical. Force addition of coord_equal, i.e. aspect ratio of 1:1. Typically useful for remote sensing data (depending on your projection), hence it defaults to TRUE. Note however, that this does not apply if (ggLayer=FALSE).
ggLayer	Logical. Return only a ggplot layer which must be added to an existing ggplot. If FALSE s stand-alone ggplot will be returned.
ggObj	Logical. Return a stand-alone ggplot object (TRUE) or just the data.frame with values and colors
geom_raster	Logical. If FALSE uses annotation_raster (good to keep aestetic mappings free). If TRUE uses geom_raster (and aes(fill)). See Details.
forceCat	Logical. If TRUE the raster values will be forced to be categorical (will be converted to factor if needed).

Details

When img contains factor values and annotation=TRUE, the raster values will automatically be converted to numeric in order to proceed with the brightness calculation.

The geom_raster argument switches from the default use of annotation_raster to geom_raster. The difference between the two is that geom_raster performs a meaningful mapping from pixel values to fill colour, while annotation_raster is simply adding a picture to your plot. In practice this means that whenever you need a legend for your raster you should use geom_raster = TRUE. This also allows you to specify and modify the fill scale manually. The advantage of using annotation_raster (geom_raster = TRUE) is that you can still use the scale_fill* for another variable. For example you could add polygons and map a value to their fill colour. For more details on the theory behind aestetic mapping have a look at the ggplot2 manuals.

Value

ggObj = TRUE:	ggplot2 plot
ggLayer = TRUE:	ggplot2 layer to be combined with an existing ggplot2
ggObj = FALSE:	data.frame in long format suitable for plotting with ggplot2, includes the pixel values and the calculated co

See Also

ggRGB, fortifySpatRaster

Examples

```
library(ggplot2)
library(terra)

## Simple grey scale annotation
ggR(rlogo)

## With linear stretch contrast enhancement
ggR(rlogo, stretch = "lin", quantiles = c(0.1,0.9))

## ggplot with geom_raster instead of annotation_raster
## and default scale_fill*
ggR(rlogo, geom_raster = TRUE)

## with different scale
ggR(rlogo, geom_raster = TRUE) +
        scale_fill_gradientn(name = "mojo", colours = rainbow(10)) +
        ggtitle("**Funkadelic**")

## Plot multiple layers

ggR(lsat, 1:6, geom_raster=TRUE, stretch = "lin") +
    scale_fill_gradientn(colors=grey.colors(100), guide = "none") +
    theme(axis.text = element_text(size=5),
          axis.text.y = element_text(angle=90),
          axis.title=element_blank())

## Don't plot, just return a data.frame
df <- ggR(rlogo, ggObj = FALSE)
head(df, n = 3)

## Layermode (ggLayer=TRUE)
data <- data.frame(x = c(0, 0:100,100), y = c(0,sin(seq(0,2*pi,pi/50))*10+20, 0))
ggplot(data, aes(x, y)) +  ggR(rlogo, geom_raster= FALSE, ggLayer = TRUE) +
       geom_polygon(aes(x, y), fill = "blue", alpha = 0.4) +
       coord_equal(ylim=c(0,75))

## Categorical data
```

```
## In this case you probably want to use geom_raster=TRUE
## in order to perform aestetic mapping (i.e. a meaningful legend)
rc   <- rlogo
rc[] <- cut(rlogo[[1]][], seq(0,300, 50))
ggR(rc, geom_raster = TRUE)

## Legend cusomization etc. ...
ggR(rc, geom_raster = TRUE) + scale_fill_continuous(labels=paste("Class", 1:6))

## Creating a nicely looking DEM with hillshade background
terr <- terrain(srtm, c("slope", "aspect"))
hill <- shade(terr[["slope"]], terr[["aspect"]])
ggR(hill)

ggR(hill) +
   ggR(srtm, geom_raster = TRUE, ggLayer = TRUE, alpha = 0.3) +
   scale_fill_gradientn(colours = terrain.colors(100), name = "elevation")
```

---

ggRGB *Create ggplot2 Raster Plots with RGB from 3 RasterLayers*

Description

Calculates RGB color composite raster for plotting with ggplot2. Optional values for clipping and and stretching can be used to enhance the imagery.

Usage

```
ggRGB(
  img,
  r = 3,
  g = 2,
  b = 1,
  scale,
  maxpixels = 5e+05,
  stretch = "none",
  ext = NULL,
  limits = NULL,
  clipValues = "limits",
  quantiles = c(0.02, 0.98),
  ggObj = TRUE,
  ggLayer = FALSE,
  alpha = 1,
  coord_equal = TRUE,
  geom_raster = FALSE,
  nullValue = 0
)
```

Arguments

img	SpatRaster
r	Integer or character. Red layer in x. Can be set to NULL, in which case the red channel will be set to zero.
g	Integer or character. Green layer in x. Can be set to NULL, in which case the green channel will be set to zero.
b	Integer or character. Blue layer in x. Can be set to NULL, in which case the blue channel will be set to zero.
scale	Numeric. Maximum possible pixel value (optional). Defaults to 255 or to the maximum value of x if that is larger than 255
maxpixels	Integer. Maximal number of pixels used for plotting.
stretch	Character. Either 'none', 'lin', 'hist', 'sqrt' or 'log' for no stretch, linear, histogram, square-root or logarithmic stretch.
ext	Extent or SpatExtent object to crop the image
limits	Vector or matrix. Can be used to reduce the range of values. Either a vector of two values for all bands (c(min, max)) or a 3x2 matrix with min and max values (columns) for each layer (rows).
clipValues	Matrix, numeric vector, string or NA. Values to reset out of range (out of limits) values to. By default ('limits') values are reset to limits. A single value (e.g. NA) will be recycled to all lower/higher clippings, A vector of length two (c(min,max)) can be used to specify lower and higher replace values, applied to all bands. A two column matrix (typically with three rows) can be used to fully control lower and upper clipping values differently for each band.
quantiles	Numeric vector with two elements. Min and max quantiles to stretch. Defaults to 2% stretch, i.e. c(0.02,0.98).
ggObj	Logical. If TRUE a ggplot2 object is returned. If FALSE a data.frame with coordinates and color will be returned.
ggLayer	Logical. If TRUE a ggplot2 layer is returned. This is useful if you want to add it to an existing ggplot2 object. Note that if TRUE & annotate = FALSE you have to add a scale_fill_identity() manually in your call to ggplot().
alpha	Numeric. Transparency (0-1).
coord_equal	Logical. Force addition of coord_equal, i.e. aspect ratio of 1:1. Typically useful for remote sensing data (depending on your projection), hence it defaults to TRUE. Note howver, that this does not apply if (ggLayer=FALSE).
geom_raster	Logical. If FALSE annotation_raster is used, otherwise geom_raster()+scale_fill_identity is used. Note that you can't use scale_fill* in addition to the latter, because it already requires scale_fill_identity().
nullValue	Numeric. Intensity value used for NULL layers in color compositing. E.g. set g=NULL and fix green value at 0.5 (defaults to 0).

Details

Functionality is based on plotRGB from the raster package.

Value

ggObj = TRUE:	ggplot2 plot
ggLayer = TRUE:	ggplot2 layer to be combined with an existing ggplot2
ggObj = FALSE:	data.frame in long format suitable for plotting with ggplot2, includes the pixel values and the calculated cc

See Also

ggR, fortifySpatRaster

Examples

```
library(ggplot2)

ggRGB(rlogo, r=1, g=2, b=3)

## Define minMax ranges
ggRGB(rlogo, r=1,g=2, b=3, limits = matrix(c(100,150,10,200,50,255),  ncol = 2, by = TRUE))

## Perform stong linear contrast stretch
ggRGB(rlogo, r = 1, g = 2, b = 3,stretch = "lin", quantiles = c(0.2, 0.8))

## Use only two layers for color calculation
ggRGB(rlogo, r = 1, g = 2, b = NULL)

## Return only data.frame
df <- ggRGB(rlogo, ggObj = FALSE)
head(df)

## Use in layer-mode, e.g. to add to another plot
wave <- data.frame(x = c(0, 0:100,100), y = c(0,sin(seq(0,2*pi,pi/50))*10+20, 0))
p <- ggplot(wave, aes(x, y))
p + ggRGB(rlogo, ggLayer = TRUE) +
       geom_polygon(aes(x, y), fill = "blue", alpha = 0.4) +
       coord_equal(ylim=c(0,75))
```

---

histMatch *Image to Image Contrast Matching*

Description

Performs image to image contrast adjustments based on histogram matching using empirical cumulative distribution functions from both images.

Usage

```
histMatch(
  x,
  ref,
  xmask = NULL,
  refmask = NULL,
  nSamples = 1e+05,
  intersectOnly = TRUE,
  paired = TRUE,
  forceInteger = FALSE,
  returnFunctions = FALSE,
  ...
)
```

Arguments

x	SpatRaster. Source raster which is to be modified.
ref	SpatRaster. Reference raster, to which x will be matched.
xmask	RasterLayer or SpatRaster. Mask layer for x to exclude pixels which might distort the histogram, i.e. are not present in ref. Any NA pixel in xmask will be ignored (maskvalue = NA).
refmask	RasterLayer or SpatRaster. Mask layer for ref. Any NA pixel in refmask will be ignored (maskvalue = NA).
nSamples	Integer. Number of random samples from each image to build the histograms.
intersectOnly	Logical. If TRUE sampling will only take place in the overlap extent of the two rasters. Otherwise the full rasters will be used for sampling.
paired	Logical. If TRUE the corresponding pixels will be used in the overlap.
forceInteger	Logical. Force integer output.
returnFunctions	Logical. If TRUE the matching functions will be returned instead of applying them to x.
...	Further arguments to be passed to writeRaster.

Value

A SpatRaster of x adjusted to the histogram of ref. If returnFunctions = TRUE a list of functions (one for each layer) will be returned instead.

Note

x and ref must have the same number of layers.

References

Richards and Jia: Remote Sensing Digital Image Analysis. Springer, Berlin, Heidelberg, Germany, 439pp.

Examples

```
library(ggplot2)
library(terra)
## Original image a (+1 to prevent log(0))
img_a <-  rlogo + 1
## Degraded image b
img_b <- log(img_a)
## Cut-off half the image (just for better display)
img_b[, 1:50] <- NA

## Compare Images before histMatching
ggRGB(img_a,1,2,3)+
        ggRGB(img_b, 1,2,3, ggLayer = TRUE, stretch = "lin", q = 0:1) +
        geom_vline(aes(xintercept = 50))+
        ggtitle("Img_a vs. Img_b")

## Do histogram matching
img_b_matched <- histMatch(img_b, img_a)

## Compare Images after histMatching
ggRGB(img_a, 1, 2, 3)+
        ggRGB(img_b_matched, 1, 2, 3, ggLayer = TRUE, stretch = "lin", q = 0:1) +
        geom_vline(aes(xintercept = 50))+
        ggtitle("Img_a vs. Img_b_matched")

## Histogram comparison
opar <- par(mfrow = c(1, 3), no.readonly = TRUE)
img_a[,1:50] <- NA
redLayers <- c(img_a, img_b, img_b_matched)[[c(1,4,7)]]
names(redLayers) <- c("img_a", "img_b", "img_b_matched")

hist(redLayers)
## Reset par
par(opar)
```

`ImageMetaData` *ImageMetaData Class*

Description

ImageMetaData Class

Usage

```
ImageMetaData(
  file = NA,
  format = NA,
  sat = NA,
  sen = NA,
```

```
  scene = NA,
  colNum = NA,
  colTier = NA,
  proj = NA,
  date = NA,
  pdate = NA,
  path = NA,
  row = NA,
  az = NA,
  selv = NA,
  esd = NA,
  files = NA,
  bands = NA,
  quant = NA,
  cat = NA,
  na = NA,
  vsat = NA,
  scal = NA,
  dtyp = NA,
  calrad = NA,
  calref = NA,
  calbt = NA,
  radRes = NA,
  spatRes = NA
)
```

Arguments

file	Character. Metadata file
format	Character. Metadata format, e.g. xml, mtl
sat	Character. Satellite platform
sen	Character. Sensor
scene	Character. Scene_ID
colNum	Character Collection number
colTier	Character Collection tier
proj	CRS. Projection.
date	POSIXct. Aquisition date.
pdate	POSIXct. Processing date.
path	Integer. Path.
row	Integer. Row.
az	Numeric. Sun azimuth
selv	Numeric. Sun elevation
esd	Numeric. Earth-sun distance
files	Character vector. Files containing the data, e.g. tiff files

bands	Character vector. Band names
quant	Character vector. Quantity, one of c("dn", "tra", "tre", "sre", "bt", "idx", "angle")
cat	Character vector. Category, e.g. c("image", "pan", "index", "qa", "aux")
na	Numeric vector. No-data value per band
vsat	Numeric vector. Saturation value per band
scal	Numeric vector. Scale factor per band. e.g. if data was scaled to 1000*reflectance for integer conversion.
dtyp	Character vector. Data type per band.
calrad	data.frame. Calibration coefficients for dn->radiance conversion. Must have columns 'gain' and 'offset'. Rows named according to bands.
calref	data.frame. Calibration coefficients for dn->reflectance conversion. Must have columns 'gain' and 'offset'. Rows named according to bands.
calbt	data.frame. Calibration coefficients for dn->brightness temperature conversion. Must have columns 'K1' and 'K2'. Rows named according to bands.
radRes	Numeric vector. Radiometric resolution per band.
spatRes	Numeric vector. Spatial resolution per band.

Value

Returns a structured, fully customizable meta-data table of a file

---

lsat *Landsat 5TM Example Data*

Description

Subset of Landsat 5 TM Scene: LT52240631988227CUB02 Contains all seven bands in DN format.

Usage

```
lsat
```

Examples

```
ggRGB(lsat, stretch = "sqrt")
```

mesma *Multiple Endmember Spectral Mixture Analysis (Spectral Unmixing)*

Description

mesma performs a spectral mixture anlylsis (SMA) or multiple endmember spectral mixture analysis (MESMA) on a multiband raster image.

Usage

```
mesma(
  img,
  em,
  method = "NNLS",
  iterate = 400,
  tolerance = 1e-08,
  n_models = NULL,
  sum_to_one = TRUE,
  ...,
  verbose
)
```

Arguments

img	SpatRaster. Remote sensing imagery (usually hyperspectral).
em	Matrix or data.frame with spectral endmembers. Columns represent the spectral bands (i.e. columns correspond to number of bands in img). Rows represent either a single endmember per class (SMA) or multiple endmembers per class (MESMA), if a column with name class is present, containing the class name each endmember belongs to, e.g. "water" or "land". See details below. Number of rows needs to be > 1.
method	Character. Select an unmixing method. Currently, only "NNLS" is implemented. Default is "NNLS". • NNLS: applies a non-negative least squares (NNLS) regression which is using a sequential coordinate-wise algorithm (SCA) based on Franc et al. (2005).
iterate	Integer. Set maximum iteration per pixel. Processing time could increase the more iterations are made possible. Default is 400.
tolerance	Numeric. Tolerance limit representing a nearly zero minimal number. Default is 1e-8.
n_models	Logical. Only applies if em contains column class. Defines how many endmember combinations should be picked. Maximum is the number of possible endmember combinations (see details). Defaults to the maximum possible endmember combinations calculated from em.

sum_to_one	Logical. Defines whether a sum-to-one constraint should be applied so that probabilities of endmember classes sum to one (a constraint not covered by NNLS) to be interpretable as fractions. Defaults to TRUE. To get actual NNLS results, change to FALSE.
...	further arguments passed to writeRaster.
verbose	Logical. Prints progress messages during execution.

Details

Argument em determines whether an SMA (each row representing a single endmember per class) or a MESMA (multiple endmembers per class differentiate using the class column) is computed. If multiple endmembers per class are provided, mesma will compute a number of SMA (determined by argument n_models) for multiple selected endmember combinations drawn from em and will select the best fit per pixel based on the lowest RMSE, based on the MESMA approach proposed by Roberts et al. (1998). For this, mesma computes all possible combinations of endmembers in em, than uses all of them unless a lower number of endmember combinations is set through argument n_models. n_models cannot be larger than the maximum number of possible endmember combinations in em. mesma always uses those combinations where the number of unique endmembers used is maximal first.

Value

SpatRaster. The object will contain one band per class, with each value representing the estimated probability of the respective endmember class per pixel, and an RMSE band. If sum_to_one is TRUE (default), values of the class bands can be interpreted as fractions per endmember class (0 to 1).

Note

Depending on iterate and tolerance settings and the selected endmembers, the sum of estimated probabilities per pixel varies around 1. NNLS does not account for a sum-to-one constraint. Use sum_to_one to sum to one post-NNLS.

To achieve best results, it is recommended to adjust n_models in accordance to the number of endemembers per class provided through em so that as many endmember combinations as possible (with each endmember being used once) are computed. The more models are being calculated, the more processing and memory recourses are needed.

Author(s)

Jakob Schwalb-Willmann

References

Franc, V., Hlaváč, V., & Navara, M. (2005). Sequential coordinate-wise algorithm for the non-negative least squares problem. In: International Conference on Computer Analysis of Images and Patterns (pp. 407-414). Berlin, Heidelberg.

Roberts, D. A., Gardner, M., Church, R., Ustin, S., Scheer, G., & Green, R. O. (1998). Mapping chaparral in the Santa Monica Mountains using multiple endmember spectral mixture models. Remote sensing of environment, 65(3), 267-279.

Examples

```
library(RStoolbox)
library(terra)

#  to perform a SMA, use a single endmember per class, row by row:
em <- data.frame(lsat[c(5294, 47916)])
rownames(em) <- c("forest", "water")

# umix the lsat image
probs <- mesma(img = lsat, em = em)
plot(probs)

# to perform a MESMA, use multiple endmembers per class, differntiating them
# by a column named 'class':
## Not run:
em <- rbind(
  data.frame(lsat[c(4155, 17018, 53134, 69487, 83704)], class = "forest"),
  data.frame(lsat[c(22742, 25946, 38617, 59632, 67313)], class = "water")
)

# unmix the lsat image
probs <- mesma(img = lsat, em = em)
plot(probs)

# MESMA can also be performed on more than two endmember classes:
em <- rbind(
  data.frame(lsat[c(4155, 17018, 53134, 69487, 83704)], class = "forest"),
  data.frame(lsat[c(22742, 25946, 38617, 59632, 67313)], class = "water"),
  data.frame(lsat[c(4330, 1762, 1278, 1357, 17414)], class = "shortgrown")
)

# unmix the lsat image
probs <- mesma(img = lsat, em = em)
plot(probs)

## End(Not run)
```

---

normImage *Normalize Raster Images: Center and Scale*

Description

For each pixel subtracts the mean of the raster layer and optionally divide by its standard deviation.

Usage

```
normImage(img, norm = TRUE, ...)
```

Arguments

img	SpatRaster. Image to transform. Transformation will be performed separately for each layer.
norm	Logical. Perform normalization (scaling) in addition to centering, i.e. divide by standard deviation.
...	further arguments passed to writeRaster.

Value

Returns a SpatRaster with the same number layers as input layers with each layer being centered and optionally normalized.

Examples

```
library(terra)
## Load example data

## Normalization: Center and Scale
rlogo_center_norm <- normImage(rlogo)
hist(rlogo_center_norm)

## Centering
rlogo_center <- normImage(rlogo, norm = FALSE)
```

---

oneHotEncode *One-hot encode a raster or vector*

Description

Splits a categorical raster layer (or a vector) into a multilayer raster (or matrix).

Usage

```
oneHotEncode(img, classes, background = 0, foreground = 1, na.rm = FALSE, ...)
```

Arguments

img	SpatRaster or integer/numeric vector containing multiple classes
classes	integer: vector of classes which should be extracted
background	integer: background value (default = 0)
foreground	integer: foreground value (default = 1)
na.rm	logical: if TRUE, NAs will be coerced to the background value.
...	further arguments passed to writeRaster. Ignored if img is not a SpatRaster, but a numeric/integer vector or matrix

Value

A SpatRaster with as many layers as there are classes. Pixels matching the class of interest are set to 1, backround values by default are set to 0 (see background argument)

Examples

```
sc <- unsuperClass(rlogo, nClasses = 3)

## one-hot encode
sc_oneHot <- oneHotEncode(sc$map, classes = c(1,2,3))

## check results
sc_oneHot
```

---

panSharpen *Pan Sharpen Imagery / Image Fusion*

Description

provides different methods for pan sharpening a coarse resolution (typically multispectral) image with a higher reolution panchromatic image. Values of the pan-chromatic and multispectral images must be of the same scale, (e.g. from 0:1, or all DNs from 0:255)

Usage

```
panSharpen(img, pan, r, g, b, pc = 1, method = "brovey", norm = TRUE)
```

Arguments

img	SpatRaster. Coarse resolution multispectral image
pan	SpatRaster. High resolution image, typically panchromatic.
r	Character or Integer. Red band in img. Only relevant if method!='pca'
g	Character or Integer. Green band in img. Only relevant if method!='pca'
b	Character or Integer. Blue band in img. Only relevant if method!='pca'
pc	Integer. Only relevant if method = 'pca'. Which principal component to replace. Usually this should be the first component (default). Only if the first component is dominated by something else than brightness it might be worth a try to use the second component.
method	Character. Choose method from c("pca", "ihs", "brovey").
norm	Logical. Rescale pan image to match the 1st PC component. Only relevant if method = 'pca'. If TRUE only min and max are matched to the 1st PC. If FALSE pan will be histogram matched to the 1st PC.

Details

Pan sharpening options:

- `method='pca'`: Performs a pca using rasterPCA. The first component is then swapped for the pan band an the PCA is rotated backwards.
- `method='ihs'`: Performs a color space transform to Intensity-Hue-Saturation space, swaps intensity for the histogram matched pan and does the backwards transformation.
- `method='brovey'`: Performs Brovey reweighting. Pan and img must be at the same value scale (e.g. 0:1, or 0:255) otherwise you'll end up with psychodelic colors.

Value

pan-sharpened SpatRaster

Examples

```
library(terra)
library(ggplot2)

## Fake panchromatic image (30m resolution covering
## the visible range (integral from blue to red))
pan       <- sum(lsat[[1:3]])
ggR(pan, stretch = "lin")

## Fake coarse resolution image (150m spatial resolution)
lowResImg <- aggregate(lsat, 5)


## Brovey pan sharpening
lowResImg_pan <- panSharpen(lowResImg, pan, r = 3, g = 2, b = 1, method = "brovey")
lowResImg_pan
## Plot
ggRGB(lowResImg, stretch = "lin") + ggtitle("Original")
ggRGB(lowResImg_pan, stretch="lin") + ggtitle("Pansharpened (Brovey)")
```

---

pifMatch *Pseudo-Invariant Features based Image Matching*

Description

Match one scene to another based on linear regression of pseudo-invariant features (PIF).

Usage

```
pifMatch(
  img,
  ref,
  method = "cor",
  quantile = 0.95,
  returnPifMap = TRUE,
  returnSimMap = TRUE,
  returnModels = FALSE
)
```

Arguments

img	SpatRaster. Image to be adjusted.
ref	SpatRaster. Reference image.
method	Method to calculate pixel similarity. Options: euclidean distance ('ed'), spectral angle ('sam') or pearson correlation coefficient ('cor').
quantile	Numeric. Threshold quantile used to identify PIFs
returnPifMap	Logical. Return a binary raster map ot pixels which were identified as pesudo-invariant features.
returnSimMap	Logical. Return the similarity map as well
returnModels	Logical. Return the linear models along with the adjusted image.

Details

The function consists of three main steps: First, it calculates pixel-wise similarity between the two rasters and identifies pseudo-invariant pixels based on a similarity threshold. In the second step the values of the pseudo-invariant pixels are regressed against each other in a linear model for each layer. Finally the linear models are applied to all pixels in the `img`, thereby matching it to the reference scene.

Pixel-wise similarity can be calculated using one of three methods: euclidean distance (`method = "ed"`), spectral angle (`"sam"`) or pearsons correlation coefficient (`"cor"`). The threshold is defined as a similarity quantile. Setting `quantile=0.95` will select all pixels with a similarity above the 95% quantile as pseudo-invariant features.

Model fitting is performed with simple linear models (`lm`); fitting one model per layer.

Value

Returns a List with the adjusted image and intermediate products (if requested).

- `img`: the adjusted image
- `simMap`: pixel-wise similarity map (if `returnSimMap = TRUE`)
- `pifMap`: binary map of pixels selected as pseudo-invariant features (if `returnPifMap = TRUE`)
- `models`: list of linear models; one per layer (if `returnModels = TRUE`)

Examples

```
library(terra)


## Create fake example data
## In practice this would be an image from another acquisition date
lsat_b <- log(lsat)

## Run pifMatch and return similarity layer, invariant features mask and models
lsat_b_adj <- pifMatch(lsat_b, lsat, returnPifMap = TRUE,
                         returnSimMap = TRUE, returnModels = TRUE)

## Pixelwise similarity
ggR(lsat_b_adj$simMap, geom_raster = TRUE)

## Pesudo invariant feature mask
ggR(lsat_b_adj$pifMap)

## Histograms of changes
par(mfrow=c(1,3))
hist(lsat_b[[1]], main = "lsat_b")
hist(lsat[[1]], main = "reference")
hist(lsat_b_adj$img[[1]], main = "lsat_b adjusted")

## Model summary for first band
summary(lsat_b_adj$models[[1]])
```

`predict.unsuperClass` *Predict a raster map based on a unsuperClass model fit.*

Description

applies a kmeans cluster model to all pixels of a raster. Useful if you want to apply a kmeans model of scene A to scene B.

Usage

```
## S3 method for class 'unsuperClass'
predict(object, img, output = "classes", ...)
```

Arguments

`object`	unsuperClass object
`img`	Raster object. Layernames must correspond to layernames used to train the superClass model, i.e. layernames in the original raster image.
`output`	Character. Either 'classes' (kmeans class; default) or 'distances' (euclidean distance to each cluster center).
`...`	further arguments to be passed to writeRaster, e.g. filename

Value

Returns a raster with the K-means distances base on your object passed in the arguments.

Examples

```
## Load training data

## Perform unsupervised classification
uc  <- unsuperClass(rlogo, nClasses = 10)

## Apply the model to another raster
map <- predict(uc, rlogo)
```

---

radCor *Radiometric Calibration and Correction*

Description

Implements several different methods for radiometric calibration and correction of Landsat data. You can either specify a metadata file, or supply all neccesary values manually. With proper parametrization apref and sdos should work for other sensors as well.

Usage

```
radCor(
  img,
  metaData,
  method = "apref",
  bandSet = "full",
  hazeValues,
  hazeBands,
  atmosphere,
  darkProp = 0.01,
  clamp = TRUE,
  verbose
)
```

Arguments

img	SpatRaster
metaData	object of class ImageMetaData or a path to the meta data (MTL) file.
method	Radiometric conversion/correction method to be used. There are currently four methods available (see Details): "rad", "apref", "sdos", "dos", "costz".
bandSet	Numeric or character. original Landsat band numbers or names in the form of ("B1", "B2" etc). If set to 'full' all bands in the solar (optical) region will be processed.

hazeValues	Numeric. Either a vector with dark DNs per hazeBand (method = 'sdos'); possibly estimated using estimateHaze. Or the 'starting haze value' (DN) for the relative scattering models in method = 'dos' or 'costz'. If not provided, hazeValues will be estimated in an automated fashion for all hazeBands. Argument only applies to methods 'sdos', 'dos' and 'costz'.
hazeBands	Character or integer. Bands corresponding to hazeValues (method = 'sdos') or band to select starting haze value from ('dos' or 'costz').
atmosphere	Character. Atmospheric characteristics. Will be estimated if not expicilty provided. Must be one of "veryClear", "clear", "moderate", "hazy" or "veryHazy".
darkProp	Numeric. Estimated proportion of dark pixels in the scene. Used only for automatic guessing of hazeValues (typically one would choose 1 or 2%).
clamp	Logical. Enforce valid value range. By default reflectance will be forced to stay within [0,1] and radiance >= 0 by replacing invalid values with the correspinding boundary, e.g. -0.1 will become 0.
verbose	Logical. Print status information.

Details

The atmospheric correction methods (sdos, dos and costz) apply to the optical (solar) region of the spectrum and do not affect the thermal band.

Dark object subtraction approaches rely on the estimation of atmospheric haze based on *dark* pixels. Dark pixels are assumed to have zero reflectance, hence the name. It is then assumed further that any radiation originating from such *dark* pixels is due to atmospheric haze and not the reflectance of the surface itself.

The folloiwing methods are available:

rad	Radiance
apref	Apparent reflectance (top-of-atmosphere reflectance)
dos	Dark object subtratction following Chavez (1989)
costz	Dark object subtraction following Chavez (1996)
sdos	Simple dark object subtraction. Classical DOS, Lhaze must be estimated for each band separately.

If either "dos" or "costz" are selected, radCor will use the atmospheric haze decay model described by Chavez (1989). Depending on the atmosphere the following coefficients are used:

veryClear	$\lambda^{-4.0}$
clear	$\lambda^{-2.0}$
moderate	$\lambda^{-1.0}$
hazy	$\lambda^{-0.7}$
veryHazy	$\lambda^{-0.5}$

For Landsat 8, no values for extra-terrestrial irradiation (esun) are provided by NASA. These are, however, neccessary for DOS-based approaches. Therefore, these values were derived from a standard reference spectrum published by Thuillier et al. (2003) using the Landsat 8 OLI spectral response functions

The implemented sun-earth distances neglect the earth's eccentricity. Instead we use a 100 year daily average (1979-2070).

Value

SpatRaster with top-of-atmosphere radiance ($W/(m^2 * srad * \mu m)$), at-satellite brightness temperature (K), top-of-atmosphere reflectance (unitless) corrected for the sun angle or at-surface reflectance (unitless).

Note

This was originally a fork of randcorr() function in the landsat package. This version works on SpatRasters and hence is suitable for large rasters.

References

S. Goslee (2011): Analyzing Remote Sensing Data in R: The landsat Package. Journal of Statistical Software 43(4).

G. Thuillier et al. (2003) THE SOLAR SPECTRAL IRRADIANCE FROM 200 TO 2400 nm AS MEASURED BY THE SOLSPEC SPECTROMETER FROM THE ATLAS AND EURECA MISSIONS. Solar Physics 214(1): 1-22 (

Examples

```
library(terra)
## Import meta-data and bands based on MTL file
mtlFile <- system.file("external/landsat/LT52240631988227CUB02_MTL.txt", package="RStoolbox")
metaData <- readMeta(mtlFile)
lsat_t <- stackMeta(mtlFile)


## Convert DN to top of atmosphere reflectance and brightness temperature
lsat_ref <- radCor(lsat_t, metaData = metaData, method = "apref")

## Correct DN to at-surface-reflecatance with DOS (Chavez decay model)
lsat_sref <- radCor(lsat_t, metaData = metaData)

## Correct DN to at-surface-reflecatance with simple DOS
## Automatic haze estimation
hazeDN    <- estimateHaze(lsat_t, hazeBands = 1:4, darkProp = 0.01, plot = FALSE)
lsat_sref <- radCor(lsat_t, metaData = metaData, method = "sdos",
                    hazeValues = hazeDN, hazeBands = 1:4)
```

---

rasterCVA *Change Vector Analysis*

Description

Calculates angle and magnitude of change vectors. Dimensionality is limited to two bands per image.

Usage

```
rasterCVA(x, y, tmf = NULL, nct = NULL, ...)
```

Arguments

x	SpatRaster with two layers. This will be the reference/origin for the change calculations. Both rasters (y and y) need to correspond to each other, i.e. same resolution, extent and origin.
y	SpatRaster with two layers. Both rasters (y and y) need to correspond to each other, i.e. same resolution, extent and origin.
tmf	Numeric. Threshold median factor (optional). Used to calculate a threshold magnitude for which pixels are considered stable, i.e. no change. Calculated as `tmf * mean(magnitude[magnitude > 0])`.
nct	Numeric. No-change threshold (optional). Alternative to `tmf`. Sets an absolute threshold. Change magnitudes below `nct` are considered stable and set to NA.
...	further arguments passed to writeRaster

Details

Change Vector Analysis (CVA) is used to identify spectral changes between two identical scenes which were acquired at different times. CVA is limited to two bands per image. For each pixel it calculates the change vector in the two-dimensional spectral space. For example for a given pixel in image A and B for the red and nir band the change vector is calculated for the coordinate pairs: (red_A | nir_A) and (red_B | nir_B).

The coordinate system is defined by the order of the input bands: the first band defines the x-axis and the second band the y-axis, respectively. Angles are returned *in degree* beginning with 0 degrees pointing 'north', i.e. the y-axis, i.e. the second band.

Value

Returns a SpatRaster with two layers: change vector angle and change vector magnitude

Examples

```
library(terra)
pca <- rasterPCA(lsat)$map

## Do change vector analysis
cva <- rasterCVA(pca[[1:2]], pca[[3:4]])
cva
```

rasterEntropy *Multi-layer Pixel Entropy*

Description

Shannon entropy is calculated for each pixel based on it's layer values. To be used with categorical / integer valued rasters.

Usage

```
rasterEntropy(img, ...)
```

Arguments

img	SpatRaster
...	additional arguments passed to writeRaster

Details

Entropy is calculated as -sum(p log(p)); p being the class frequency per pixel.

Value

SpatRaster "entropy"

Examples

```
re <- rasterEntropy(rlogo)
ggR(re, geom_raster = TRUE)
```

rasterPCA *Principal Component Analysis for Rasters*

Description

Calculates R-mode PCA for SpatRasters and returns a SpatRaster with multiple layers of PCA scores.

Usage

```
rasterPCA(
  img,
  nSamples = NULL,
  nComp = nlyr(img),
  spca = FALSE,
  maskCheck = TRUE,
  ...
)
```

Arguments

img	SpatRaster.
nSamples	Integer or NULL. Number of pixels to sample for PCA fitting. If NULL, all pixels will be used.
nComp	Integer. Number of PCA components to return.
spca	Logical. If TRUE, perform standardized PCA. Corresponds to centered and scaled input image. This is usually beneficial for equal weighting of all layers. (FALSE by default)
maskCheck	Logical. Masks all pixels which have at least one NA (default TRUE is recomended but introduces a slow-down, see Details when it is wise to disable maskCheck). Takes effect only if nSamples is NULL.
...	further arguments to be passed to writeRaster, e.g. filename.

Details

Internally rasterPCA relies on the use of princomp (R-mode PCA). If nSamples is given the PCA will be calculated based on a random sample of pixels and then predicted for the full raster. If nSamples is NULL then the covariance matrix will be calculated first and will then be used to calculate princomp and predict the full raster. The latter is more precise, since it considers all pixels, however, it may be slower than calculating the PCA only on a subset of pixels.

Pixels with missing values in one or more bands will be set to NA. The built-in check for such pixels can lead to a slow-down of rasterPCA. However, if you make sure or know beforehand that all pixels have either only valid values or only NAs throughout all layers you can disable this check by setting maskCheck=FALSE which speeds up the computation.

Standardised PCA (SPCA) can be useful if imagery or bands of different dynamic ranges are combined. SPC uses the correlation matrix instead of the covariance matrix, which has the same effect as using normalised bands of unit variance.

Value

Returns a named list containing the PCA model object ($model) and a SpatRaster with the principal component layers ($object).

Examples

```
library(ggplot2)
library(reshape2)
ggRGB(rlogo, 1,2,3)

## Run PCA
set.seed(25)
rpc <- rasterPCA(rlogo)
rpc

## Model parameters:
summary(rpc$model)
loadings(rpc$model)
```

```
ggRGB(rpc$map,1,2,3, stretch="lin", q=0)
if(require(gridExtra)){
  plots <- lapply(1:3, function(x) ggR(rpc$map, x, geom_raster = TRUE))
  grid.arrange(plots[[1]],plots[[2]], plots[[3]], ncol=2)
}
```

---

readEE *Tidy import tool for EarthExplorer .csv export files*

Description

Imports and tidies CSV files exported from EarthExplorer into data.frames and annotates missing fields.

Usage

```
readEE(x)
```

Arguments

x	Character, Character or list. One or more paths to EarthExplorer export files.

Details

The EarthExplorer CSV file can be produced from the search results page. Above the results click on 'export results' and select 'comma (,) delimited'.

Note that only a subset of columns is imported which was deemed interesting. Please contact the maintainer if you think an omited column should be included.

Value

data.frame

Examples

```
library(ggplot2)
ee <- readEE(system.file("external/EarthExplorer_LS8.txt", package = "RStoolbox"))

## Scenes with cloud cover < 20%
ee[ee$Cloud.Cover < 20,]

## Available time-series
ggplot(ee) +
     geom_segment(aes(x = Date, xend = Date, y = 0, yend = 100 - Cloud.Cover,
     col = as.factor(Year))) +
        scale_y_continuous(name = "Scene quality (% clear sky)")
```

readMeta *Read Landsat MTL metadata files*

Description

Reads metadata and deals with legacy versions of Landsat metadata files and where possible adds missing information (radiometric gain and offset, earth-sun distance).

Usage

```
readMeta(file, raw = FALSE)
```

Arguments

file	path to Landsat MTL file (...MTL.txt)
raw	Logical. If TRUE the full raw metadata will be returned as a list. if FALSE (the default) all important metadata are homogenized into a standard format (ImageMetaData) and some information is added.

Value

Object of class ImageMetaData

Examples

```
## Example metadata file (MTL)
mtlFile <- system.file("external/landsat/LT52240631988227CUB02_MTL.txt", package="RStoolbox")

## Read metadata
metaData <- readMeta(mtlFile)

## Summary
summary(metaData)
```

readSLI *Read ENVI spectral libraries*

Description

read/write support for ENVI spectral libraries

Usage

```
readSLI(path)
```

Arguments

path	Path to spectral library file with ending .sli.

Details

ENVI spectral libraries consist of a binary data file (.sli) and a corresponding header (.hdr, or .sli.hdr) file.

Value

The spectral libraries are read into a data.frame. The first column contains the wavelengths and the remaining columns contain the spectra.

See Also

writeSLI

Examples

```
## Example data
sliFile <- system.file("external/vegSpec.sli", package="RStoolbox")
sliTmpFile <- paste0(tempdir(),"/vegetationSpectra.sli")

## Read spectral library
sli <- readSLI(sliFile)
head(sli)
plot(sli[,1:2], col = "orange", type = "l")
lines(sli[,c(1,3)], col = "green")

## Write to binary spectral library
writeSLI(sli, path = sliTmpFile)
```

---

rescaleImage *Linear Image Rescaling*

Description

performs linear shifts of value ranges either to match min/max of another image (y) or to any other min and max value (ymin and ymax).

Usage

```
rescaleImage(x, y, xmin, xmax, ymin, ymax, forceMinMax = FALSE, ...)
```

Arguments

x	SpatRaster or numeric vector. Image to normalise.
y	SpatRaster or numeric vector. Reference image. Optional. Used to extract min and max values if ymin or ymax are missing.
xmin	Numeric. Min value of x. Either a single value or one value per layer in x. If xmin is not provided it will be extracted from x.
xmax	Numeric. Max value of x. Either a single value or one value per layer in x. If xmax is not provided it will be extracted from x.
ymin	Numeric. Min value of y. Either a single value or one value per layer in x. If ymin is not provided it will be extracted from y.
ymax	Numeric. Max value of y. Either a single value or one value per layer in x. If ymax is not provided it will be extracted from y.
forceMinMax	Logical. Forces update of min and max data slots in x or y.
...	additional arguments passed to terra::writeRaster()

Details

Providing xmin and xmax values manually can be useful if the raster contains a variable of a known, fixed value range, e.g. NDVI from -1 to 1 but the actual pixel values don't encompass this entire range. By providing xmin = -1 and xmax = 1 the values can be rescaled to any other range, e.g. 1 to 100 while comparability to other rescaled NDVI scenes is retained.

Value

Returns a SpatRaster of the same dimensions as the input raster x but shifted and stretched to the new limits.

See Also

histMatch

Examples

```
lsat2 <- lsat - 1000
lsat2

## Rescale lsat2 to match original lsat value range
lsat2_rescaled <- rescaleImage(lsat2, lsat)
lsat2_rescaled

## Rescale lsat to value range [0,1]
lsat2_unity <- rescaleImage(lsat2, ymin = 0, ymax = 1)
lsat2_unity
```

rlogo *Rlogo as SpatRaster*

Description

Tiny example of raster data used to run examples.

Usage

```
rlogo
```

Examples

```
ggRGB(rlogo,r = 1,g = 2,b = 3)
```

rsOpts *Set global options for RStoolbox*

Description

shortcut to options(RStoolbox.*)

Usage

```
rsOpts(verbose = NULL, idxdb = NULL)
```

Arguments

verbose	Logical. If TRUE many functions will print status messages about the current processing step. By default verbose mode is disabled.
idxdb	List. The list conatins the formal calculation of spectral indices. Modify this list to pipe your own spectral index through the internal C++ calculation of RStoolbox.

Value

No return, just a setter for the verbosiness and the index-database of the RStoolbox package. For latter, see the example of Rstoolbox::spectralIndices()

Examples

```
rsOpts(verbose=TRUE)
```

RStoolbox *RStoolbox: A Collection of Remote Sensing Tools*

Description

The RStoolbox package provides a set of functions which simplify performing standard remote sensing tasks in R.

Data Import and Export

- readMeta: import Landsat metadata from MTL or XML files
- stackMeta, getMeta: load Landsat bands based on metadata
- readSLI & writeSLI: read and write ENVI spectral libraries
- saveRSTBX & readRSTBX: save and re-import RStoolbox classification objects (model and map)
- readEE: import and tidy EarthExplorer search results

Data Pre-Processing

- radCor: radiometric conversions and corrections. Primarily, yet not exclusively, intended for Landsat data processing. DN to radiance to reflectance conversion as well as DOS approaches
- topCor: topographic illumination correction
- cloudMask & cloudShadowMask: mask clouds and cloud shadows in Landsat or other imagery which comes with a thermal band
- classifyQA: extract layers from Landsat 8 QA bands, e.g. cloud confidence
- encodeQA & decodeQA: encode/decode Landsat 16-bit QA bands.
- rescaleImage: rescale image to match min/max from another image or a specified min/max range
- normImage: normalize imagery by centering and scaling
- oneHotEncode: one-hot encode a raster or vector
- histMatch: matches the histograms of two scenes
- pifMatch: matches one scene to another based on linear regression of Pseudo-Invariant Features (PIF)
- coregisterImages: co-register images based on mutual information
- panSharpen: sharpen a coarse resolution image with a high resolution image (typically panchromatic)
- estimateHaze: estimate image haze for Dark Object Subtraction (DOS)

Data Analysis

- spectralIndices: calculate a set of predefined multispectral indices like NDVI
- tasseledCap: tasseled cap transformation
- sam: spectral angle mapper
- rasterPCA: principal components transform for raster data
- rasterCVA: change vector analysis
- rasterEntropy: calculates shannon entropy
- unsuperClass: unsupervised classification
- superClass, validateMap, getValidation: supervised classification and validation
- fCover: fractional cover of coarse resolution imagery based on high resolution classification
- mesma: spectral unmixing using Multiple Endmember Spectral Mixture Analysis (MESMA)

Data Display

- ggR: single raster layer plotting with ggplot2
- ggRGB: efficient plotting of remote sensing imagery in RGB with ggplot2

`sam` *Spectral Angle Mapper*

Description

Calculates the angle in spectral space between pixels and a set of reference spectra (endmembers) for image classification based on spectral similarity.

Usage

```
sam(img, em, angles = FALSE, ...)
```

Arguments

`img`	SpatRaster. Remote sensing imagery.
`em`	Matrix or data.frame with endmembers. Each row should contain the endmember spectrum of a class, i.e. columns correspond to bands in `img`. It is reccomended to set the rownames to class names.
`angles`	Logical. If `TRUE` a RasterBrick containing each one layer per endmember will be returned containing the spectral angles.
`...`	further arguments to be passed to `writeRaster`

Details

For each pixel the spectral angle mapper calculates the angle between the vector defined by the pixel values and each endmember vector. The result of this is one raster layer for each endmember containing the spectral angle. The smaller the spectral angle the more similar a pixel is to a given endmember class. In a second step one can the go ahead an enforce thresholds of maximum angles or simply classify each pixel to the most similar endmember.

Value

SpatRaster If `angles = FALSE` a single Layer will be returned in which each pixel is assigned to the closest endmember class (integer pixel values correspond to row order of `em`.

Examples

```
library(terra)
library(ggplot2)

## Sample endmember spectra
## First location is water, second is open agricultural vegetation
pts <- data.frame(x = c(624720, 627480), y = c(-414690, -411090))
endmembers <- extract(lsat, pts)
rownames(endmembers) <- c("water", "vegetation")

## Calculate spectral angles
lsat_sam <- sam(lsat, endmembers, angles = TRUE)
plot(lsat_sam)

## Classify based on minimum angle
lsat_sam <- sam(lsat, endmembers, angles = FALSE)

ggR(lsat_sam, forceCat = TRUE, geom_raster=TRUE) +
         scale_fill_manual(values = c("blue", "green"), labels = c("water", "vegetation"))
```

---

saveRSTBX *Save and Read RStoolbox Classification Results*

Description

Saves objects of classes unsuperClass, superClass, rasterPCA and fCover to file. Useful to archive the fitted models.

Usage

```
saveRSTBX(x, filename, format = "raster", ...)

readRSTBX(filename)
```

Arguments

x	RStoolbox object of classes c("fCover", "rasterPCA", "superClass", "unsuperClass")
filename	Character. Path and filename. Any file extension will be ignored.
format	Character. Driver to use for the raster file
...	further arguments passed to writeRaster

Value

The output of writeRSTBX will be at least two files written to disk: a) an .rds file containing the object itself and b) the raster file (depending on the driver you choose this can be more than two files).

Functions

- `saveRSTBX()`: Save RStoolbox object to file
- `readRSTBX()`: Read files saved with saveRSTBX

Note

All files must be kept in the same directory to read the full object back into R by means of readRSTBX. You can move them to another location but you'll have to move *all* of them (just like you would with Shapefiles). In case the raster file(s) is missing, readRSTBX will still return the object but the raster will be missing.

writeRSTBX and readRSTBX are convenience wrappers around saveRDS, readRDS. This means you can read all files created this way also with base functionality as long as you don't move your files. This is because x$map is a SpatRaster object and hence contains only a static link to the file on disk.

Examples

```
## Not run:
input <- rlogo
## Create filename
file  <- paste0(tempdir(), "/test", runif(1))
## Run PCA
rpc   <- rasterPCA(input, nSample = 100)
## Save object
saveRSTBX(rpc, filename=file)
## Which files were written?
list.files(tempdir(), pattern = basename(file))
## Re-read files
re_rpc <- readRSTBX(file)
## Remove files
file.remove(list.files(tempdir(), pattern = basename(file), full = TRUE))

## End(Not run)
```

sen2 *Sentinel 2 MSI L2A Scene*

Description

Contains all 13 bands in already converted spectral reflectances

Usage

```
sen2
```

Examples

```
ggRGB(sen2, r=4, g=3, b=2, stretch = "lin")
```

spectralIndices *Spectral Indices*

Description

Calculate a suite of multispectral indices such as NDVI, SAVI etc. in an efficient way via C++.

Usage

```
spectralIndices(
  img,
  blue = NULL,
  green = NULL,
  red = NULL,
  nir = NULL,
  redEdge1 = NULL,
  redEdge2 = NULL,
  redEdge3 = NULL,
  swir1 = NULL,
  swir2 = NULL,
  swir3 = NULL,
  scaleFactor = 1,
  skipRefCheck = FALSE,
  indices = NULL,
  index = NULL,
  maskLayer = NULL,
  maskValue = 1,
  coefs = list(L = 0.5, G = 2.5, L_evi = 1, C1 = 6, C2 = 7.5, s = 1, swir2ccc = NULL,
    swir2coc = NULL),
  ...
)
```

Arguments

`img`	SpatRaster. Typically remote sensing imagery, which is to be classified.
`blue`	Character or integer. Blue band.
`green`	Character or integer. Green band.
`red`	Character or integer. Red band.
`nir`	Character or integer. Near-infrared band (700-1100nm).
`redEdge1`	Character or integer. Red-edge band (705nm)
`redEdge2`	Character or integer. Red-edge band (740nm)
`redEdge3`	Character or integer. Red-edge band (783nm)
`swir1`	not used
`swir2`	Character or integer. Short-wave-infrared band (1400-1800nm).
`swir3`	Character or integer. Short-wave-infrared band (2000-2500nm).
`scaleFactor`	Numeric. Scale factor for the conversion of scaled reflectances to [0,1] value range (applied as reflectance/scaleFactor) Neccesary for calculating EVI/EVI2 with scaled reflectance values.
`skipRefCheck`	Logical. When EVI/EVI2 is to be calculated there is a rough heuristic check, whether the data are inside [0,1]+/-0.5 (after applying a potential `scaleFactor`). If there are invalid reflectances, e.g. clouds with reflectance > 1 this check will result in a false positive and skip EVI calculation. Use this argument to skip this check in such cases *iff* you are sure the data and scaleFactor are valid.
`indices`	Character. One or more spectral indices to calculate (see Details). By default (NULL) all implemented indices given the spectral bands which are provided will be calculated.
`index`	Character. Alias for `indices`.
`maskLayer`	RasterLayer or SpatRaster containing a mask, e.g. clouds, for which pixels are set to NA. Alternatively a layername or -number can be provided if the mask is part of `img`.
`maskValue`	Integer. Pixel value in `maskLayer` which should be masked in output, i.e. will be set to `NA` in all calculated indices.
`coefs`	List of coefficients (see Details).
`...`	further arguments such as filename etc. passed to writeRaster

Details

`spectralIndices` calculates all indices in one go in C++, which is more efficient than calculating each index separately (for large rasters). By default all indices which can be calculated given the specified indices will be calculated. If you don't want all indices, use the `indices` argument to specify exactly which indices are to be calculated. See the table bellow for index names and required bands.

Index values outside the valid value ranges (if such a range exists) will be set to NA. For example a pixel with NDVI > 1 will be set to NA.

Value

SpatRaster

Examples

```
library(ggplot2)
library(terra)

## Calculate NDVI
ndvi <- spectralIndices(lsat, red = "B3_dn", nir = "B4_dn", indices = "NDVI")
ndvi
ggR(ndvi, geom_raster = TRUE) +
        scale_fill_gradientn(colours = c("black", "white"))


## Calculate all possible indices, given the provided bands
## Convert DNs to reflectance (required to calculate EVI and EVI2)
mtlFile <- system.file("external/landsat/LT52240631988227CUB02_MTL.txt", package="RStoolbox")
lsat_ref <- radCor(lsat, mtlFile, method = "apref")

SI <- spectralIndices(lsat_ref, red = "B3_tre", nir = "B4_tre")
plot(SI)

## Custom Spectral Index Calculation (beta) (supports only bands right now...)
# Get all indices
# Supports: Parentheses (), Addition +, Subtraction -, Multiplication *, Division /
idxdb <- getOption("RStoolbox.idxdb")

# Customize the RStoolbox index-database and overwrite the option
cdb <- c(idxdb, CUSTOM = list( list(c("Mueller2024", "Super custom index"),
        function(blue, red) {blue + red})))
rsOpts(idxdb = cdb)

# Calculate the custom index, (also together with the provided ones)
custom_ind <- spectralIndices(lsat, blue = 1, red = 3, nir = 4, indices = c("NDVI", "CUSTOM"))
```

---

srtm *SRTM Digital Elevation Model*

Description

DEM for the Landsat example area taken from SRTM v3 tile: s04_w050_1arc_v3.tif

Usage

```
srtm
```

Examples

```
ggR(srtm)
```

`srtm_sen2` *SRTM scene for the sen2 exemplary scene*

Description

DEM for the Sentinel 2 example area taken from SRTM v4

Usage

```
srtm_sen2
```

Examples

```
ggR(srtm_sen2)
```

`stackMeta` *Import separate Landsat files into single stack*

Description

Reads Landsat MTL or XML metadata files and loads single Landsat Tiffs into a rasterStack. Be aware that by default stackMeta() does NOT import panchromatic bands nor thermal bands with resolutions != 30m.

Usage

```
stackMeta(file, quantity = "all", category = "image", allResolutions = FALSE)
```

Arguments

`file`	Character. Path to Landsat MTL metadata (*_MTL.txt) file or an Landsat CDR xml metadata file (*.xml).
`quantity`	Character vector. Which quantity should be returned. Options: digital numbers ('dn'), top of atmosphere reflectance ('tre'), at surface reflectance ('sre'), brightness temperature ('bt'), spectral index ('index'), all quantities ('all').
`category`	Character vector. Which category of data to return. Options 'image': image data, 'pan': panchromatic image, 'index': multiband indices, 'qa' quality flag bands, 'all': all categories.
`allResolutions`	Logical. if `TRUE` a list will be returned with length = unique spatial resolutions. This argument was introduced to maintain backward compatibility and will be switched to TRUE in an upcoming release. Please base all new code on terra.

Value

Returns one single SpatRaster comprising all requested bands. If `allResolutions = TRUE` *and* there are different resolution layers (e.g. a 15m panchromatic band along wit 30m imagery) a list of RasterStacks will be returned.

Note

Be aware that by default stackMeta() does NOT import panchromatic bands nor thermal bands with resolutions != 30m. Use the allResolutions argument to import all layers. Note that nowadays the USGS uses cubic convolution to resample the TIR bands to 30m resolution.

Examples

```
## Example metadata file (MTL)
mtlFile <- system.file("external/landsat/LT52240631988227CUB02_MTL.txt", package="RStoolbox")

## Read metadata
metaData <- readMeta(mtlFile)
summary(metaData)

## Load rasters based on metadata file
lsat     <- stackMeta(mtlFile)
lsat
```

---

superClass *Supervised Classification*

Description

Supervised classification both for classification and regression mode based on vector training data (points or polygons).

Usage

```
superClass(
  img,
  trainData,
  valData = NULL,
  responseCol = NULL,
  nSamples = 1000,
  nSamplesV = 1000,
  polygonBasedCV = FALSE,
  trainPartition = NULL,
  model = "rf",
  tuneLength = 3,
  kfold = 5,
  sampling = NULL,
  minDist = 2,
```

```
  mode = "classification",
  predict = TRUE,
  predType = "raw",
  filename = NULL,
  verbose,
  overwrite = TRUE,
  ...
)
```

Arguments

img	SpatRaster. Typically remote sensing imagery, which is to be classified.
trainData	sf or sp spatial vector data containing the training locations (POINTs,or POLYGONs).
valData	Ssf or sp spatial vector data containing the validation locations (POINTs,or POLYGONs) (optional).
responseCol	Character or integer giving the column in trainData, which contains the response variable. Can be omitted, when trainData has only one column.
nSamples	Integer. Number of samples per land cover class. If NULL all pixels covered by training polygons are used (memory intensive!). Ignored if trainData consists of POINTs.
nSamplesV	Integer. Number of validation samples per land cover class. If NULL all pixels covered by validation polygons are used (memory intensive!). Ignored if valData consists of POINTs.
polygonBasedCV	Logical. If TRUE model tuning during cross-validation is conducted on a per-polygon basis. Use this to deal with overfitting issues. Does not affect training data supplied as SpatialPointsDataFrames.
trainPartition	Numeric. Partition (polygon based) of trainData that goes into the training data set between zero and one. Ignored if valData is provided.
model	Character. Which model to use. See train for options. Defaults to randomForest ('rf'). In addition to the standard caret models, a maximum likelihood classification is available via model = 'mlc'.
tuneLength	Integer. Number of levels for each tuning parameter (see train for details).
kfold	Integer. Number of cross-validation resamples during model tuning.
sampling	Character. Describes the type of additional sampling that is conducted after resampling (usually to resolve class imbalances), from caret. Currently supported are up, down, smote, and rose. Note, that smote requires the packages themis and rose the package ROSE. Latter is noly for binary classification problems.
minDist	Numeric. Minumum distance between training and validation data, e.g. minDist=1 clips validation polygons to ensure a minimal distance of one pixel (pixel size according to img) to the next training polygon. Requires all data to carry valid projection information.
mode	Character. Model type: 'regression' or 'classification'.
predict	Logical. Produce a map (TRUE, default) or only fit and validate the model (FALSE).

predType	Character. Type of the final output raster. Either "raw" for class predictions or "prob" for class probabilities. Class probabilities are not available for all classification models (predict.train).
filename	Path to output file (optional). If NULL, standard raster handling will apply, i.e. storage either in memory or in the raster temp directory.
verbose	Logical. prints progress and statistics during execution
overwrite	logical. Overwrite spatial prediction raster if it already exists.
...	further arguments to be passed to train

Details

Note that superClass automatically loads the lattice and randomForest package. SuperClass performs the following steps:

1. Ensure non-overlap between training and validation data. This is neccesary to avoid biased performance estimates. A minimum distance (minDist) in pixels can be provided to enforce a given distance between training and validation data.
2. Sample training coordinates. If trainData (and valData if present) are polygons superClass will calculate the area per polygon and sample nSamples locations per class within these polygons. The number of samples per individual polygon scales with the polygon area, i.e. the bigger the polygon, the more samples.
3. Split training/validation If valData was provided (reccomended) the samples from these polygons will be held-out and not used for model fitting but only for validation. If trainPartition is provided the trainingPolygons will be divided into training polygons and validation polygons.
4. Extract raster data The predictor values on the sample pixels are extracted from img
5. Fit the model. Using caret::train on the sampled training data the model will be fit, including parameter tuning (tuneLength) in kfold cross-validation. polygonBasedCV=TRUE will define cross-validation folds based on polygons (reccomended) otherwise it will be performed on a per-pixel basis.
6. Predict the classes of all pixels in img based on the final model.
7. Validate the model with the independent validation data.

Value

A superClass object (effectively a list) containing:

1. $model: the fitted model
2. $modelFit: model fit statistics
3. $training: indexes of samples used for training
4. $validation: list of
   (a) $performance: performance estimates based on independent validation (confusion matrix etc.)
   (b) $validationSamples: actual pixel coordinates plus reference and predicted values used for validation

(c) $validationGeometry: validation polygpns (clipped with mindist to training geometries)

5. $map: the predicted raster
6. $classMapping: a data.frame containing an integer <-> label mapping

See Also

train

Examples

```
library(RStoolbox)
library(caret)
library(randomForest)
library(e1071)
library(terra)
train <- readRDS(system.file("external/trainingPoints_rlogo.rds", package="RStoolbox"))

## Plot training data
olpar <- par(no.readonly = TRUE) # back-up par
par(mfrow=c(1,2))
colors <- c("yellow", "green", "deeppink")
plotRGB(rlogo)
plot(train, add = TRUE, col =  colors[train$class], pch = 19)

## Fit classifier (splitting training into 70\% training data, 30\% validation data)
SC       <- superClass(rlogo, trainData = train, responseCol = "class",
model = "rf", tuneLength = 1, trainPartition = 0.7)
SC

## Plots
plot(SC$map, col = colors, legend = FALSE, axes = FALSE, box = FALSE)
legend(1,1, legend = levels(train$class), fill = colors , title = "Classes",
horiz = TRUE,  bty = "n")
par(olpar) # reset par
```

---

tasseledCap *Tasseled Cap Transformation*

Description

Calculates brightness, greenness and wetness from multispectral imagery. Currently implemented Landsat 4 TM, Landsat 5 TM, Landsat 7ETM+, Landsat 8 OLI, MODIS, QuickBird, Spot5 and RapidEye.

Usage

```
tasseledCap(img, sat, ...)
```

Arguments

`img`	SpatRaster. Input image. Band order must correspond to sensor specifications (see Details and Examples)
`sat`	Character. Sensor; one of: c("Landsat4TM", "Landsat5TM", "Landsat7ETM", "Landsat8OLI", "MODIS", "QuickBird", "Spot5", "RapidEye"). Case is irrelevant.
`...`	Further arguments passed to writeRaster.

Details

Currently implemented: Landsat 4 TM, Landsat 5 TM, Landsat 7ETM+, Landsat 8 OLI, MODIS, QuickBird, Spot5, RapdiEye. Input data must be in top of atmosphere reflectance. Moreover, bands must be provided in ascending order as listed in the table below. Irrelevant bands, such as Landsat Thermal Bands or QuickBird/Spot5 Panchromatic Bands must be omitted. Required bands are:

sat	bands	coefficients	data unit
Landsat4TM	1,2,3,4,5,7	Crist 1985	reflectance
Landsat5TM	1,2,3,4,5,7	Crist 1985	reflectance
Landsat7ETM	1,2,3,4,5,7	Huang 2002	reflectance
Landsat8OLI	2,3,4,5,6,7	Baig 2014	reflectance
MODIS	1,2,3,4,5,6,7	Lobser 2007	reflectance
QuickBird	2,3,4,5	Yarbrough 2005	reflectance
Spot5	2,3,4,5	Ivtis 2008	reflectance
RapidEye	1,2,3,4,5	Schoenert 2014	reflectance

Value

Returns a SpatRaster with the thee bands: brigthness, greenness, and (soil) wetness.

References

Crist (1985) "A TM Tasseled Cap Equivalent Transformation for Reflectance Factor Data." Remote Sensing of Environment 17 (3): 301-306

Huang et al. (2002) "Derivation of a Tasselled Cap Transformation Based on Landsat 7 At-Satellite Reflectance." International Journal of Remote Sensing 23 (8): 1741-1748

Baig et al. (2014) "Derivation of a Tasselled Cap Transformation Based on Landsat 8 At-Satellite Reflectance." Remote Sensing Letters 5 (5): 423-431.

Lobser et al. (2007) "MODIS Tasselled Cap: Land Cover Characteristics Expressed through Transformed MODIS Data." International Journal of Remote Sensing 28 (22): 5079-5101.

Yarbrough et al. (2005) "QuickBird 2 tasseled cap transform coefficients: a comparison of derivation methods." Pecora 16 Global Priorities in Land Remote Sensing: 23-27.

Ivits et al. (2008) "Orthogonal transformation of segmented SPOT5 images." Photogrammetric Engineering & Remote Sensing 74 (11): 1351-1364.

Schoenert et al. (2014) "Derivation of tasseled cap coefficients for RapidEye data." Earth Resources and Environmental Remote Sensing/GIS Applications V (9245): 92450Qs.

Examples

```
library(terra)

## Run tasseled cap (exclude thermal band 6)
lsat_tc <- tasseledCap(lsat[[c(1:5,7)]], sat = "Landsat5TM")
lsat_tc
plot(lsat_tc)
```

---

topCor *Topographic Illumination Correction*

Description

account and correct for changes in illumination due to terrain elevation.

Usage

```
topCor(
  img,
  dem,
  metaData,
  solarAngles = c(),
  method = "C",
  stratImg = NULL,
  nStrat = 5,
  illu,
  ...
)
```

Arguments

img	SpatRaster. Imagery to correct
dem	SpatRaster. Either a digital elevation model as a RasterLayer or a RasterStack/Brick with pre-calculated slope and aspect (see terrain) in which case the layers must be named 'slope' and 'aspect'. Must have the same dimensions as img.
metaData	Character, ImageMetaData. Either a path to a Landsat meta-data file (MTL) or an ImageMetaData object (see readMeta)
solarAngles	Numeric vector containing sun azimuth and sun zenith (in radians and in that order). Not needed if metaData is provided
method	Character. One of c("cos", "avgcos", "minnaert", "C", "stat", "illu"). Choosing 'illu' will return only the local illumination map.
stratImg	RasterLayer or SpatRaster to define strata, e.g. NDVI. Or the string 'slope' in which case stratification will be on nStrat slope classes. Only relevant if method = 'minnaert'.
nStrat	Integer. Number of bins or quantiles to stratify by. If a bin has less than 50 samples it will be merged with the next bin. Only relevant if method = 'minnaert'.

illu	SpatRaster. Optional pre-calculated ilumination map. Run topCor with method="illu" to calculate an ilumination map
...	arguments passed to writeRaster

Details

For detailed discussion of the various approaches please see Riano et al. (2003).

The minnaert correction can be stratified for different landcover characteristics. If stratImg = 'slope' the analysis is stratified by the slope, i.e. the slope values are divided into nStrat classes and the correction coefficient k is calculated and applied separately for each slope class. An alternative could be to stratify by a vegetation index in which case an additional raster layer has to be provided via the stratImg argument.

Value

SpatRaster

References

Riano et al. (2003) Assessment of different topographic correction in Landsat-TM data for mapping vegetation types. IEEE Transactions on Geoscience and Remote Sensing.

Examples

```
## Load example data
metaData <- system.file("external/landsat/LT52240631988227CUB02_MTL.txt", package="RStoolbox")
metaData <- readMeta(metaData)

## Minnaert correction, solar angles from metaData
lsat_minnaert <- topCor(lsat, dem = srtm, metaData = metaData, method = "minnaert")

## C correction, solar angles provided manually
lsat_C <- topCor(lsat, dem = srtm, solarAngles = c(1.081533, 0.7023922), method = "C")
```

---

unsuperClass *Unsupervised Classification*

Description

Unsupervised clustering of SpatRaster data using kmeans clustering

Usage

```
unsuperClass(
  img,
  nSamples = 10000,
  nClasses = 5,
  nStarts = 25,
  nIter = 100,
  norm = FALSE,
  clusterMap = TRUE,
  algorithm = "Hartigan-Wong",
  output = "classes",
  ...
)
```

Arguments

img	SpatRaster.
nSamples	Integer. Number of random samples to draw to fit cluster map. Only relevant if clusterMap = TRUE.
nClasses	Integer. Number of classes.
nStarts	Integer. Number of random starts for kmeans algorithm.
nIter	Integer. Maximal number of iterations allowed.
norm	Logical. If TRUE will normalize img first using normImage. Normalizing is beneficial if your predictors have different scales.
clusterMap	Logical. Fit kmeans model to a random subset of the img (see Details).
algorithm	Character. kmeans algorithm. One of c("Hartigan-Wong", "Lloyd", "MacQueen")
output	Character. Either 'classes' (kmeans class; default) or 'distances' (euclidean distance to each cluster center).
...	further arguments to be passed to writeRaster, e.g. filename

Details

Clustering is done using kmeans. This can be done for all pixels of the image (clusterMap=FALSE), however this can be slow and is not memory safe. Therefore if you have large raster data (> memory), as is typically the case with remote sensing imagery it is advisable to choose clusterMap=TRUE (the default). This means that a kmeans cluster model is calculated based on a random subset of pixels (nSamples). Then the distance of *all* pixels to the cluster centers is calculated in a stepwise fashion using predict. Class assignment is based on minimum euclidean distance to the cluster centers.

The solution of the kmeans algorithm often depends on the initial configuration of class centers which is chosen randomly. Therefore, kmeans is usually run with multiple random starting configurations in order to find a convergent solution from different starting configurations. The nStarts argument allows to specify how many random starts are conducted.

Value

Returns an RStoolbox::unsuperClass object, which is a list containing the kmeans model ($model) and the raster map ($map). For output = "classes", $map contains a SpatRaster with discrete classes (kmeans clusters); for output = "distances" $map contains a SpatRaster, with ‘nClasses‘ layers, where each layer maps the euclidean distance to the corresponding class centroid.

Examples

```
## Not run:
library(terra)
input <- rlogo

## Plot
olpar <- par(no.readonly = TRUE) # back-up par
par(mfrow=c(1,2))
plotRGB(input)

## Run classification
set.seed(25)
unC <- unsuperClass(input, nSamples = 100, nClasses = 5, nStarts = 5)
unC

## Plots
colors <- rainbow(5)
plot(unC$map, col = colors, legend = FALSE, axes = FALSE, box = FALSE)
legend(1,1, legend = paste0("C",1:5), fill = colors, title = "Classes", horiz = TRUE,  bty = "n")

## Return the distance of each pixel to each class centroid
unC <- unsuperClass(input, nSamples = 100, nClasses = 3, output = "distances")
unC

ggR(unC$map, 1:3, geom_raster = TRUE)

par(olpar) # reset par

## End(Not run)
```

---

validateMap *Map accuracy assessment*

Description

validate a map from a classification or regression model. This can be useful to update the accuracy assessment after filtering, e.g. for a minimum mapping unit.

Usage

```
validateMap(
  map,
```

```
  valData,
  responseCol,
  nSamplesV = 500,
  mode = "classification",
  classMapping = NULL
)
```

Arguments

map	SpatRaster. The classified map.
valData	sf object with validation data (POLYGONs or POINTs).
responseCol	Character. Column containing the validation data in attribute table of valData.
nSamplesV	Integer. Number of pixels to sample for validation (only applies to polygons).
mode	Character. Either 'classification' or 'regression'.
classMapping	optional data.frame with columns 'class' and 'classID' defining the mapping from raster integers to class names.

Value

Returns a structured list includng the preformance and confusion-matrix of your then validated input data

Examples

```
library(caret)
library(terra)

## Training data
poly   <- readRDS(system.file("external/trainingPolygons_lsat.rds", package="RStoolbox"))

## Split training data in training and validation set (50%-50%)
splitIn   <- createDataPartition(poly$class, p = .5)[[1]]
train <- poly[splitIn,]
val   <- poly[-splitIn,]

## Classify (deliberately poorly)
sc <- superClass(lsat, trainData = train, responseCol = "class", nSamples = 50, model = "mlc")

## Polish map with majority filter

polishedMap <- focal(sc$map, matrix(1,3,3), fun = modal)

## Validation
## Before filtering
val0 <- validateMap(sc$map, valData = val, responseCol = "class",
                            classMapping = sc$classMapping)
## After filtering
val1 <- validateMap(polishedMap, valData = val, responseCol = "class",
                             classMapping = sc$classMapping)
```

writeSLI *Write ENVI spectral libraries*

Description

Writes binary ENVI spectral library files (sli) with accompanying header (.sli.hdr) files OR ASCII spectral library files in ENVI format.

Usage

```
writeSLI(
  x,
  path,
  wavl.units = "Micrometers",
  scaleF = 1,
  mode = "bin",
  endian = .Platform$endian
)
```

Arguments

x	data.frame with first column containing wavelengths and all other columns containing spectra.
path	path to spectral library file to be created.
wavl.units	wavelength units. Defaults to Micrometers. Nanometers is another typical option.
scaleF	optional reflectance scaling factor. Defaults to 1.
mode	character string specifying output file type. Must be one of "bin" for binary .sli files or "ASCII" for ASCII ENVI plot files.
endian	character. Optional. By default the endian is determined based on the platform, but can be forced manually by setting it to either "little" or "big".

Details

ENVI spectral libraries with ending .sli are binary arrays with spectra saved in rows.

Value

Does not return anything, write the SLI file directly to your drive for where your specified your path parameter

See Also

readSLI

Examples

```
## Example data
sliFile <- system.file("external/vegSpec.sli", package="RStoolbox")
sliTmpFile <- paste0(tempdir(),"/vegetationSpectra.sli")

## Read spectral library
sli <- readSLI(sliFile)
head(sli)
plot(sli[,1:2], col = "orange", type = "l")
lines(sli[,c(1,3)], col = "green")

## Write to binary spectral library
writeSLI(sli, path = sliTmpFile)
```

Index

∗ **datasets**
  lsat, 28
  rlogo, 47
  sen2, 52
  srtm, 54
  srtm_sen2, 55
∗ **earth-observation**
  RStoolbox, 48
∗ **remote-sensing**
  RStoolbox, 48
∗ **spatial-data-analysis**
  RStoolbox, 48

classifyQA, 3, *48*
cloudMask, 4, *7*, *48*
cloudShadowMask, 6, *6*, *48*
coregisterImages, *7*, 8, *48*

decodeQA, *4*, 10, *48*

encodeQA, *3, 4*, 11, *11*, *48*
estimateHaze, 12, *38*, *48*

fCover, 14, *49*
fortifySpatRaster, 16, *21*, *24*

getMeta, 17, *48*
getValidation, 18, *49*
ggR, 19, *24*, *49*
ggRGB, *21*, 22, *49*

histMatch, 24, *46*, *48*
hsv, *20*

ImageMetaData, 26

kmeans, *63*

lm, *35*
lsat, 28

mesma, 29, *49*

normImage, 31, *48*, *63*

oneHotEncode, 32, *48*

panSharpen, 33, *48*
pifMatch, 34, *48*
plotRGB, *23*
predict, *63*
predict.train, *58*
predict.unsuperClass, 36
princomp, *42*

radCor, *13*, 37, *48*
rasterCVA, 39, *49*
rasterEntropy, 41, *49*
rasterPCA, *34*, 41, *49*
readEE, 43, *48*
readMeta, 44, *48*, *61*
readRSTBX, *48*
readRSTBX *(*saveRSTBX*)*, 50
readSLI, 44, *48*, *66*
rescaleImage, 45, *48*
rlogo, 47
rsOpts, 47
RStoolbox, 48

sam, 49, *49*
saveRSTBX, *48*, 50
sen2, 52
spectralIndices, *49*, 52
srtm, 54
srtm_sen2, 55
stackMeta, *48*, 55
superClass, *14, 15*, *49*, 56

tasseledCap, *49*, 59
terra::writeRaster(), *46*
terrain, *61*
topCor, *48*, 61
train, *14, 15*, *57–59*
trainControl, *14, 15*

unsuperClass, *49*, 62

validateMap, *49*, 64

writeRaster, *3*, *9*, *15*, *25*, *30*, *32*, *36*, *42*, *49*, *53*, *62, 63*
writeSLI, *45*, *48*, 66