Package 'CAWaR'

August 3, 2019

Type Package

Title CAWa Project Tools

Version 0.0.1

Date 2019-07-30

URL https://github.com/RRemelgado/fieldRS/

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

Maintainer Ruben Remelgado <remelgado.ruben@gmail.com>

Description

Tools to process ground-truth data on crop types and perform a phenology based crop type classification. These tools were developed in the scope of the CAWa project and extend on the work of Conrad et al. (2011) <doi:10.1080/01431161.2010.550647>. Moreover, they introduce an innovative classification and validation scheme that utilizes spatially independent samples as proposed by Remelgado et al. (2017) <doi:10.1002/rse2.70>.

LazyData TRUE

Encoding UTF-8

Imports raster, sp, rgdal, ggplot2, grDevices, spatialEco, rgeos, lubridate, RStoolbox, fieldRS, rsMove

RoxygenNote 6.1.1

License GPL (>= 3)

Suggests knitr, rmarkdown, kableExtra, imager, lattice

VignetteBuilder knitr

NeedsCompilation no

Author Ruben Remelgado [aut, cre]

Repository CRAN

Date/Publication 2019-08-03 07:40:04 UTC

R topics documented:

`analyseTS` *analyseTS*

Description

Summarizes multi-band *raster* data within each element of a *SpatialPolygons* object.

Usage

```
analyseTS(x, y, out.plot = NULL)
```

Arguments

`x`	Object of class *data.frame*.
`y`	Vector of class *character* or *numeric* with a length equal to the number of rows in *x*.
`out.plot`	Specifies the data path where plots should be stored.

Details

For each unique value in *y*, the function will select the rows in *x* that correspond to it and estimate the median, Median Absolute Deviation (MAD), minimum, maximum, mean and standard deviation for each column. Then, the function will build a plot showing the median and draw a buffer that expresses the minimum and maximum. The final output is a list consisting of:

- *y.statistics* - Median, minimum and maximum values for each column in *x* over each unique class in *y*.
- *plots* - List of line plots for each unique element in *y*.

If *out.plot* is set, the function will save each plot as 10x10 cm PNG files within the specified path.

Value

A *SpatialPointsDataDrame* with the coordinate pairs for each of the sampled pixels.

See Also

`extractTS` `phenoCropVal` `phenoCropClass`

Examples

```
{

require(raster)
require(fieldRS)

# read raster data
r <- brick(system.file("extdata", "ndvi.tif", package="fieldRS"))

# read field data
data(fieldData)
data(fieldDataTS)

a.ts <- analyseTS(as.data.frame(fieldDataTS$weighted.mean), fieldData$crop)

}
```

`CAWaR` *CAWaR.*

Description

CAWaR.

checkSamples *checkSamples*

Description

checks if a shapefile with ground truth data contains all the necessary fields in the correct format.

Usage

```
checkSamples(x)
```

Arguments

x	An object or a list of class *sp* containing a *data.frame* (e.g. *SpatialPolygonsDataFrame*).

Details

Checks if a shapefile - or a list of - contains necessary columns and if these have the right format. It searches for:

- *sampler* - Character vector with name of responsible person.
- *date* - Date vector with the date on which each sample was collected (formatted as "yyyy-mm-dd").
- *label* - Character vector sample label (e.g. land cover class).

Value

A *data.frame* with the consistency checks for each element in *x*.

See Also

labelCheck

Examples

```
{

require(fieldRS)

# Example ground-truth data
data(fieldData)

# check shapefile content
cs <- checkSamples(fieldData)
head(cs)

}
```

compareLabel *compareLabel*

Description

Identifies samples that potentially require class label corrections.

Usage

```
compareLabel(x, y, x.label, y.label, na.count = 0, max.length = 0)
```

Arguments

x	Object of class *data.frame* with target profiles.
y	Object of class *data.frame* with reference profiles.
x.label	*Character vector* with classes of *x*.
y.label	*Character vector* with classes of *y*.
na.count	Maximum number of NA values accepted.
max.length	Maximum length of consecutive NA values accepted.

Details

The function cross-correlates *x* and *y*. Then,for each row, the function returns the element in *y.label* with the highest correlation. The final output of the function consists:

- *cross.cor* - Median, minimum and maximum values for each column in *x* over each unique class in *y*.
- *label.compare* - *data.frame* showing *y.label* and the best match in *y.label*.
- *na.stats* - *data.frame* showing the count and maximum number of consecutive NA values for each row in *x*.

Note that *na.count* and *max.length* determine which observations are judged. If These thresholds are exceeded, the function will return NA.

Value

A *list*.

See Also

extractTS analyseTS

Examples

```
{

require(raster)
require(fieldRS)

# read raster data
r <- brick(system.file("extdata", "ndvi.tif", package="fieldRS"))

# read field data
data(fieldData)
data(fieldDataTS)

a.ts <- analyseTS(as.data.frame(fieldDataTS$weighted.mean), fieldData$crop)

# extract reference profiles
rp <- as.data.frame(do.call(rbind, lapply(a.ts$y.statistics, function(i) {i$median})))

# compare labels
cl <- compareLabel(as.data.frame(fieldDataTS$weighted.mean), rp, fieldData$crop, a.ts$labels)

}
```

`countCropCycles` *countCropCycles*

Description

Matches two vectors with different lengths based on their maximum value.

Usage

```
countCropCycles(x, min.length = c(1, 1))
```

Arguments

`x`	Target numeric *vector*.
`min.length`	Two element numeric *vector*.

Details

The function counts the number of value segments in *x* that are above its mean effectively counting the number of crop cycles. Before reporting the final value, *min.length* is used to filter outliers. The first element filters segments that lie below the mean (i.e. recently cultivated/harvested). If the segment length is greater than the 1st element in *min.length* the segment is relabeled as "1 (i.e. "crop growth/maturity". This process is repeated for segments above the mean (i.e. crop growth/maturity). If the length of a segment is greater than the second element in *min.length* it is labeled as "recently cultivated/harvested".

Value

A *numeric* element with the number of crop cycles in *x*.

Examples

```
{

x <- c(293, 770, 1166, 1166, 1562, 2357, 3234,
5806, 5806, 5678, 5678, 5546, 5536, 5536, 5536,
5325, 5200, 4726, 3550, 2868, 2365, 2365, 2365)

n <- countCropCycles(x)

}
```

`extract2` *extract2*

Description

Extract of values from multi-extent raster objects with a spatial object.

Usage

```
extract2(x, y, x.date, out.date, time.buffer = c(365, 365))
```

Arguments

`x`	A *character* vector with the paths to *RasterLayer* objects or a *list* of *RasterLayers*.
`y`	An object of class *SpatialPoints* or *SpatialPolygons*.
`x.date`	Object of class *Date* with the acquisition dates of each element in *x*.
`out.date`	Object of class *Date* with the desired output dates.
`time.buffer`	Two-element, numeric vector.

Details

Creates a rectangular fishnet in a *SpatialPolygons* format based on the extent of *x* and the value of *y* which defines the spatial resolution.

Value

A *list* object.

`extractTS` *extractTS*

Description

Extracts time series data from a *RasterStack* for a *SpatialPolygons* or a *SpatialPolygonsDataFrame* object.

Usage

```
extractTS(x, y, z, id)
```

Arguments

`x`	Object of class *SpatialPolygons*, *SpatialPolygonsDataFrame*, *SpatialPoints* or *SpatialPointsDataFrame*.
`y`	A *raster* object, a list of *RasterLayer* objects or a numeric element.
`z`	*Numeric* vector with weights for each element in *x* (when points).
`id`	*Numeric* vector with unique identifiers for *x* (when points).

Details

For each polygon in *x* - if *x* is a *SpatialPolygons* and *SpatialPolygonsDataFrame* object - the function identifies the overlapping pixels in *y* and, for each pixel, estimates the percentage area covered by the polygon. Using this data as weights, the function calculates the weighted mean for each band in *y*. If *y* is a numeric element, the function will build a raster with resolution equal to *y* over which the pixel cover will be estimated. Moreover, if *x* is a *SpatialPoints* or a *SpatialPointsDataFrame* object, the function will skip the pixel extraction step. In this case, the user may provide a vector with sample weights through *z* and a vector of unique identifiers (reporting on e.g. the polygon membership) The function returns a list of three *data.frame* objects where each row represents a different polygon in *x*:

- *pixel.info* - *SpatialPointsDataFrame* with pixel-wise samples for each polygon (identified by the field *id*).
- *polygon.info* - Mean, min, max and standard deviation of the pixel cover; centroid coordinates.
- *weighted.mean* - Weighted mean raster values (if *y* is a raster object).

Value

A *list*.

See Also

`analyseTS`

Examples

```
{

require(raster)
require(fieldRS)

# read raster data
r <- brick(system.file("extdata", "ndvi.tif", package="fieldRS"))

# read field data
data(fieldData)

extractTS(fieldData[1:5,], r)

}
```

`fieldData2` *Point shapefile.*

Description

Ground truth data on crop types collected in Uzbekistan derived from the fieldData dataset with poly2sample.

Usage

```
data(fieldData2)
```

Format

A SpatialPointsDataFrame

Details

- xx coordinates.
- yy coordinates.
- coverPercent overlap between the the pixel each point was derived from and the corresponding polygon.
- idUnique identifier of the polygon the points belong to.

fieldDataCluster *Region labels for fieldData*

Description

Output of splitSamples for fieldData.

Usage

```
data(fieldDataCluster)
```

Format

A data.frame

Details

- region.idRegion identifier showing which samples are grouped.
- region.frequencyFrequency of samples per region.

fieldDataTS *Samples for fieldData*

Description

Output of extractTS for fieldData.

Usage

```
data(fieldDataTS)
```

Format

A data.frame

Details

- pixel.infoUnique pixels covered by fieldData with information of x and y coordinates and cover percent.
- polygon.infoPercent cover statistics and pixel count per polygon.
- weighted.meanWeighted-mean time-series for each polygon.

findBare *findBare*

Description

Shows is a vector is related to bare/fallow land..

Usage

```
findBare(x, y)
```

Arguments

x	A numeric *vector*.
y	A numeric element.

Details

The function computes the amplitude of *x* and returns TRUE if this value is below the threshold given by *y*.

Value

A logical element.

Examples

```
{

# vectors to test
x1 <- c(0.1,0.2,0.1,0.2,0.4,0.6,0.8,0.4,0.1) # simulated, crop profile
x2 <- c(0.1,0.3,0.2,0.2,0.1,0.3,0.1,0.2,0.2) # simulated, non-crop profile

# compare profiles
plot(x1, type="l")
lines(x2, col="red")

findBare(x1, 100) # returns TRUE
findBare(x2, 100) # returns FALSE

}
```

`matchIndices` *matchIndices*

Description

Matches two vectors with different lengths based on their maximum value.

Usage

```
matchIndices(x, y, z)
```

Arguments

`x`	Target numeric *vector*.
`y`	Reference numeric *vector*.
`z`	A *numeric* element.

Details

Uses Dynamic Time Wrapping (DTW) to match *x* and *y*. *z* determines the buffer size - expressed in number of data points - used to search for matching records.

Value

A *list* with selected indices for *x* and *y*.

Examples

```
{

x <- c(2200, 4500, 4600, 6400, 1600) # target
y <- c(1100, 1150, 1200, 6400, 1600) # reference

i <- matchIndices(x, y, 1) # find best match

# plot x (blue), and selected y (red)
plot(1:5, replicate(5,0), ylim=c(0,10000), type="l")
lines(i$x, x[i$x], type="l", col="blue")
lines(i$y, y[i$y], type="l", col="red")

}
```

meStack *meStack*

Description

Stacking of raster layers with different extents

Usage

```
meStack(x, y, z, agg.fun = mean, derive.stats = FALSE)
```

Arguments

x	A *list* of *RasterLayer* objects or a *character* vector with the paths to *raster* objects.
y	A spatial object from which an extent can be derived.
z	Object of class *Date* with the acquisition date for each element in *x*.
agg.fun	Function used to aggregate images collected in the same date. Default is the mean.
derive.stats	Logical argument. Default is FALSE.

Details

The function stacks the raster objects specified in *x*. For each element in *x*, the function crops it by the extent of *y* and, if their extents differ, fits the extent of *x* to the one of *y*. All new pixels are set to NA. If *z* is provided, the function will then aggregate all bands acquired in the same date using the function provide with *agg.fun*. If *derive.stats* is set to TRUE, the function will return basic statistics for each band (i.e. min, max, mean and sd) together with a plot of the mean values. The final output of the function is a list containing:

- *stack* - *RasterStack* object.
- *dates* - Acquisition dates for each layer in *stack*.
- *image.stats* - Statistics for each band in the output *RasterStack*.
- *stats.plot* - Plot showing the mean, minimum and maximum values per band.
- *control* - Logical vector showing which elements in *x* where used to build the *RasterStack*.

Value

A list containing a *RasterStack* and related statistics.

Examples

```
{

require(raster)

r1 <- raster(xmn=1, xmx=90, ymn=1, ymx=90, res=1, vals=1) # image 1
r2 <- raster(xmn=50, xmx=150, ymn=50, ymx=150, res=1, vals=1) # image 2
r0 <- raster(xmn=20, xmx=90, ymn=50, ymx=90, res=1, vals=1) # target extent

crs(r0) <- crs(r2) <- crs(r1)

mes <- meStack(list(r1, r2), r0)
plot(mes$stack)

}
```

`phenoCropClass` *phenoCropClass*

Description

Spatially explicit and phenology driven classification scheme for cropland mapping.

Usage

```
phenoCropClass(x, y, z, match = FALSE)
```

Arguments

`x`	A *matrix* or *data.frame*.
`y`	A *character* vector.
`z`	A *numeric* element. Default is 1.
`match`	logical argument.

Details

Correlates *x* with each row in *y*. The row in *y* with the highest correlation is reported as the selected class. If *match* is set to TRUE the function will use Dynamic Time Wrapping (DTW) *x* and *y* at each iteration. *z* sets the temporal buffer used to search to matching data points. The final output is a *data.frame* containing:

- *r2* - R^2 between *x* and each row *y*.
- *count* - Number of records used to estimate the R^2.
- *max.interval* - Maximum gap between data points when NA values exist.

Value

A *list* containing a set of reference profiles for each unique class in *y*.

See Also

analyseTS phenoCropVal

Examples

```
{

require(fieldRS)

# read reference profiles
data(referenceProfiles)

# target time series
x <- c(2200, 4500, 4600, 6400, 1600)
y <- referenceProfiles[,2:6]

# Perform classification
c <-phenoCropClass(x, y)
head(c)

}
```

---

phenoCropVal *phenoCropVal*

Description

Spatially explicit and phenology driven validation scheme for cropland mapping.

Usage

```
phenoCropVal(x, y, z)
```

Arguments

x	A *matrix* or *data.frame*.
y	A *character* vector.
z	A *character* vector.

Details

For each unique class in *y*, the function iterates through each unique element in *z* and keeps it for validation. Then, it calls analyseTS to derive reference profiles for each unique class in *y* and uses them to classify the validation samples using phenoCropClass. The final output consists of:

- *sample.validation* - A *logical* vector with the same length of *x* where TRUE means it was correctly classified.
- *predicted.class* - A *character* vector with the predicted classes for each sample.

- *sample.count* - A *numeric* vector with the number of non-NA used for validation per sample.
- *sample.r2* - A *numeric* vector with the r2 value between the target sample and the selected class profile.
- *class.accuracy* - A *data.frame* with sample count per class, precision, recall and F1-scores per unique class in *y*.

Value

A *list* containing a set of reference profiles for each unique class in *y*.

See Also

extractTS phenoCropClass

Examples

```
{

require(raster)
require(fieldRS)

# read raster data
r <- brick(system.file("extdata", "ndvi.tif", package="fieldRS"))

# read field data
data(fieldData)

# read reference profiles
data(referenceProfiles)

# read time series
data(fieldDataTS)
fieldDataTS <- as.data.frame(fieldDataTS$weighted.mean)

# read info. on sample spatial grouping
data(fieldDataCluster)

# derive validation results
cropVal <- phenoCropVal(fieldDataTS, fieldData$crop, fieldDataCluster$region.id)

# plot accuracy results
cropVal$accuracy.plot

# plot correctly classified polygons in red
plot(fieldData)
plot(fieldData[cropVal$sample.validation,], col="red", add=TRUE)

}
```

`segmentRaster` *segmentRaster*

Description

Segmentation of a raster object using k-means clustering and connected component labeling.

Usage

```
segmentRaster(x, change.threshold = 0, n.size = NULL)
```

Arguments

`x`	Object of class *RasterLayer*.
`change.threshold`	Numeric element (0 - 100).
`n.size`	Numeric element.

Details

The function clusters a raster object with k-means through unsuperClass and segments the output using ccLabel. The number of samples used to determine the number of clusters can be defined through *n.size* to reduce the required computational time. The optimal number of clusters is determined with the elbow method. he elbow method looks at the percentage of variance explained by the number of clusters. If adding a new cluster, referred here as k, leads to no improvement, the function will use k-1 to derive. This breakpoint is determined by *change.threshold* which controls the percent change between the amount of variance explained by two consequent k values. the output is a list consisting of:

- *class* - Cluster image.
- *regions* - Segmented region image.

Value

A *list*.

`splitSamples` *splitSamples*

Description

Aggregates a spatial object into regions.

Usage

```
splitSamples(x, y, z, agg.radius = agg.radius)
```

Arguments

x	A *SpatialPoints* or a *SpatialPolygons* object.
y	A *RasterLayer*.
z	A vector.
agg.radius	Numeric element.

Details

For each class in *z*, the function converts the elements in *x* into a raster layer using *y* as a basis. Then, it aggregates all pixels that are within a given distance of each other - defined by *agg.radius* using ccLabel. The output is a list consisting of:

- *region.id* - Class dependent region label for each element in *x*.
- *region.frequency* - Pixel count for each unique value in *region.id*.

Value

A list.

See Also

phenoCropVal phenoCropClass

Examples

```
{

require(raster)
require(fieldRS)

# read raster data
r <- brick(system.file("extdata", "ndvi.tif", package="fieldRS"))

# read field data
data(fieldData)
fieldData <- fieldData[3:4,]

# find polygon clusters
k <- splitSamples(fieldData, r, fieldData$crop, agg.radius=30)
fieldData$ID <- as.factor(k$region.id)

# plot regions with labels
spplot(fieldData["ID"])

# show pixel count per region
head(k$region.frequency)

}
```

Index