Package 'fieldRS'

September 16, 2018

Type Package

Title Remote Sensing Field Work Tools

Version 0.1.1

Date 2018-09-16

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

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

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

Description In remote sensing, designing a field campaign to collect ground-truth data can be a challenging task. We need to collect representative samples while accounting for issues such as budget constraints and limited accessibility created by e.g. poor infrastructure. As suggested by Olofsson et al. (2014) <doi:10.1016/j.rse.2014.02.015>, this demands the establishment of best-practices to collect ground-truth data that avoid the waste of time and funds. 'fieldRS' addresses this issue by helping scientists and practitioners design field campaigns through the identification of priority sampling sites, the extraction of potential sampling plots and the conversion of plots into consistent training and validation samples that can be used in e.g. land cover classification.

LazyData TRUE

Imports raster, sp, caret, ggplot2, grDevices, spatialEco, rgeos

RoxygenNote 6.1.0

License GPL (>= 3)

Suggests knitr, rmarkdown, kableExtra, imager, RStoolbox, randomForest, rgdal

VignetteBuilder knitr

NeedsCompilation no

Author Ruben Remelgado [aut, cre]

Repository CRAN

Date/Publication 2018-09-16 16:10:03 UTC

R topics documented:

`ccLabel` *ccLabel*

Description

Labels groups of pixels in a raster object that share similar attributes.

Usage

```
ccLabel(x, method = "simple", change.threshold = NULL)
```

Arguments

`x`	Object of class *RasterLayer*, *RasterStack* or *RasterBrick*.
`method`	Labeling method. Choose between 'simple' and 'change'. Default is 'simple'.
`change.threshold`	Numeric element.

Details

Uses a 8-neighbor connected component labeling algorithm (determined by *method*) to identify groups of pixels of the same value. Each group receives a distinct numeric label. The function provides two connected component labeling algorithms:

- *simple* - Connects neighboring pixels with the same value. Suitable for categorical data.
- *spatial* - Estimates the MAPE using a 3x3 moving window distinguishes neighboring pixels when the spatial change is lower than *change.threshold*.
- *temporal* - Estimates the MAPE among all bands in a raster object and distinguishes spatially neighboring pixels when the temporal change is higher than *change.threshold*.

The final output of the function is a list consisting of:

- *regions* - *RasterLayer* object with region labels.
- *frequency* - *data.frame* object with the pixel count for each unique value in *regions*.

Value

A list.

See Also

classModel rankPlots

Examples

```
{

require(raster)

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

# spatial change labeling
or <- ccLabel(r[[1]], method="spatial", change.threshold=10)
plot(or$regions)

# temporal change labeling
or <- ccLabel(r, method="temporal", change.threshold=80)
plot(or$regions)

}
```

checkOverlap *checkOverlap*

Description

Reports on how much two spatial objects overlap.

Usage

```
checkOverlap(x, y)
```

Arguments

x	A spatial object.
y	A spatial object.

Details

Uses intersect to report on the percentage of the area of *x* and *y* that coincides with their common spatial overlap.

Value

A two element numeric *vector*.

Examples

```
{

 require(raster)

 # build polygons
 df1 <- data.frame(x=c(1, 5, 10, 2, 1), y=c(10, 9, 8, 7, 10))
 df2 <- data.frame(x=c(2, 6, 5, 4, 2), y=c(10, 9, 7, 4, 10))
 p <- list(Polygons(list(Polygon(df1)), ID=1),
 Polygons(list(Polygon(df2)), ID=2))
 p <- SpatialPolygons(p)

 # check overlap %
 checkOverlap(p[1,], p[2,])

}
```

classModel *classModel*

Description

Derives a predictive model for

Usage

```
classModel(x, y, z, mode = "classification", ...)
```

Arguments

x	Object of class *data.frame*.
y	A vector of class *character* or *numeric*.
z	A vector of class *character* or *numeric*.
mode	One of "classification" or "regression".
...	Arguments passed to train.

Details

Uses train to derive a predictive model based on *x* - which contains the predictors - and *y* - which contains information on the target classes (if *mode* is "classification") or values (if *mode* is "regression"). This method iterates through all samples making sure that all contribute for the final accuracy. To specify how the samples should be split, the user should provide the sample-wise identifiers through *z*. For each unique value in *z*, the function keeps it for validation and the remaining samples for training. This process is repeated for all sample groups and a final accuracy is estimated from the overall set of results. If *mode* is "classification", the function will estimate the overall accuracy for each unique value in *y*. If "regression" is set, the output will be an the coefficient of determination. The classification algorithm and additional commands can be set through ... using inputs of the train function. Apart from the accuracy assessment, the function stores the performance for each sample. If *mode* is "classification", the function will return a logical vector reporting on the correctly assigned classes. If *mode* is "regression", the function will report on the percent deviation between the original value and it's predicted value. The final output of the function is a list consisting of:

- *sample.validation* - Accuracy assessment of each sample.
- *overall.validation* - Finally accuracy value for each class (if "classification").
- *r2* - Correlation between *y* and the predicted values (if "regression")

Value

A two element numeric *vector*.

See Also

raster2sample poly2sample ccLabel

Examples

```
{

require(raster)

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

# read field data
data(fieldData)
fieldData <- fieldData[c(1:3, 10),]

# extract values for polygon centroid
c <- spCentroid(fieldData)
ev <- as.data.frame(extract(r, c))

cm <- classModel(ev, fieldData$crop, as.character(1:length(fieldData)), method="rf")

}
```

`derivePlots` *derivePlots*

Description

Creates a fishnet from a spatial extent.

Usage

```
derivePlots(x, y)
```

Arguments

x	A spatial object.
y	A numeric element.

Details

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

Value

An object of class *SpatialPolygons*.

See Also

`rankPlots`

Examples

```
{

require(raster)

# read field data
data(fieldData)

# derive plots
g <- derivePlots(fieldData, 1000)

# compare original data and output
plot(fieldData)
plot(g, border="red", add=TRUE)

}
```

---

ecDistance *ecDistance*

Description

Calculates the Euclidean distance among all elements of a SpatialPoints object.

Usage

```
ecDistance(x)
```

Arguments

x	A *matrix*, *data.frame* or a *SpatialPoints* object.

Details

compares all elements of *x* and returns the minimum Euclidean distance between them.

Value

A *matrix*.

See Also

spCentroid

Examples

```
{

require(raster)

# read field data
data(fieldData)

# show distance matrix
head(ecDistance(fieldData))

}
```

`extractFields` *extractFields*

Description

Extracts and vectorizes clumps of pixels with equal value within a raster object.

Usage

```
extractFields(x)
```

Arguments

`x`	Object of class *RasterLayer*.

Details

Given a segmented image as *x*, the function extracts patches of pixels with equal value. For each pixel region, the function extracts the center pixel coordinates and derives their minimum convex polygon. Then, for each polygon, the derives a ratio between the area of the polygon and the area of the pixel region. Ratios below zero suggest that the region has a clearly defined shape (e.g. rectangular). Clumps is less than 3 points are ignored. The output of the function is a *SpatialPolygonsDataFrame* reporting on:

- *Region ID* - Unique polygon identifier.
- *Area* - Polygon Area (in square meters).
- *Perimeter* - Polygon perimeter (in square meters).
- *Pixel %* - Percentage of non-NA pixels.

Value

A *SpatialPolygonsDataFrame*.

Examples

```
{

require(raster)

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

# spatial change labeling
or <- ccLabel(r[[1]], method="spatial", change.threshold=10)$regions

# convert to polygons and plot
ef <- extractFields(or)
plot(ef)

}
```

`fieldData` *Polygon shapefile.*

Description

Ground truth data on crop types collected in Uzbekistan.

Usage

```
data(fieldData)
```

Format

A SpatialPointsDataFrame

Details

- cropCrop type.
- dateSampling date.
- areaArea in m^2.

`fieldRS` *fieldRS.*

Description

fieldRS.

geCheck *geCheck*

Description

Finds overlaps between polygons in the same shapefile.

Usage

```
geCheck(x)
```

Arguments

x	An object of class *SpatialPolygons* or *SpatialPolygonsDataFrame*.

Details

compares all elements of *x* and returns a a list containing:

- *overlap.df* - *data.frame* where each row shows the indices of which two polygons overlap.
- *overlap.shp* - *SpatialPointsDataFrame* with the actual overlap for each row in *overlap.df*.

Value

A *list*.

See Also

spCentroid ecDistance

Examples

```
{

require(raster)

# build polygons
df1 <- data.frame(x=c(1, 5, 10, 2, 1), y=c(10, 9, 8, 7, 10))
df2 <- data.frame(x=c(2, 6, 5, 4, 2), y=c(10, 9, 7, 4, 10))
p <- list(Polygons(list(Polygon(df1)), ID=1),
Polygons(list(Polygon(df2)), ID=2))
p <- SpatialPolygons(p)

# check overlap
co <- geCheck(p)

# show distance matrix
plot(p)
plot(co$overlap.shp, col="red", add=TRUE)
```

```
}
```

`labelCheck` *labelCheck*

Description

helps fix spelling mistakes in the labels of a set of samples.

Usage

```
labelCheck(x, y, z)
```

Arguments

x	Vector of class *character*.
y	Vector of class *character*.
z	Vector of class *character*.

Details

If *y* and *z* are missing, the function will return the unique values among all the elements of *y*. Otherwise, the function will provide a corrected copy of *y*. Additionally, the function will count the number of records for each of the unique labels from which a plot will be built. The final output consists of:

- *unique.labels* - Unique labels in the output.
- *corrected.labels* - Corrected labels in *x*.
- *label.count* - Count of occurrences in *unique.labels* per each element in *x*.
- *label.count.plot* - Plot of *label.count*.

Value

A *character* vector.

See Also

`extractFields`

mape *mape*

Description

Mean Absolute Percent Error (MAPE).

Usage

```
mape(x, na.rm = TRUE)
```

Arguments

x	A vector of class *numeric*.
na.rm	Logical. Should the NA values be excluded. Default is TRUE.

Details

Estimates the Mean Absolute Percent Error (MAPE) for a given vector. The MAPE compares the individual values against their mean and translates the mean of the differences into a percent deviation from the mean of the vector. The MAPE is estimated as:

$$100/length(x) * sum(abs((x - mean(x))/x))$$

Value

A *numeric* element.

See Also

relative.freq ccLabel

Examples

```
{

x <- c(0.1, 0.3, 0.4, 0.1, 0.2, 0.6)
m <- mape(x)

}
```

poly2sample *poly2sample*

Description

Converts a raster grid to points.

Usage

```
poly2sample(x, y)
```

Arguments

x	Object of class *SpatialPolygons* or *SpatialPolygonDataFrame*.
y	A raster object or a numeric element.

Details

poly2Sample extends on the `rasterize` function from the raster package making it more efficient over large areas and converting its output into point samples rather than a raster object. For each polygon in (*"x"*), *poly2sample* extracts the overlapping pixels of a reference grid. Then, for each pixel, the function estimates the percentage of it that is covered by the reference polygon. If *y* is a raster object, the function will use it as a reference grid. If *y* is a numeric element, the function will build a raster from the extent of *x* and a resolution equal to *y*.

Value

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

See Also

`raster2sample` `ccLabel`

Examples

```
{

 require(raster)

 # read raster data
 r <- raster(system.file("extdata", "ndvi.tif", package="fieldRS")[1])

 # read field data
 data(fieldData)
 fieldData <- fieldData[1,]

 # extract samples
 samples <- poly2sample(fieldData, r)

}
```

predictive.model1 *Predictive classification model accuracy (class-wise)*

Description

Sample-wise validation returned by classModel().

Usage

```
data(predictive.model1)
```

Format

A logical vector.

predictive.model2 *Predictive classification model accuracy (overall)*

Description

Class-wise F1-score returned by classModel().

Usage

```
data(predictive.model2)
```

Format

A data.frame

rankPlots *rankPlots*

Description

helps fix spelling mistakes in the labels of a set of samples.

Usage

```
rankPlots(x, y, z, min.size = 1, priority = c("class_count",
  "patch_count", "pixel_frequency", "road_distance"))
```

Arguments

x	Object of class *RasterLayer*, *RasterStack* or *RasterBrick*.
y	Object of class *SpatialPolygons* or *SpatialPolygonsDataFrame*.
z	Object of class *SpatialLines* or *SpatialLinesDataFrame*.
min.size	Numeric element.
priority	Character vector.

Details

For each polygon in *y*, the function will determine the distance between its centroid and the nearest road provided through *z*, count the number of classes in *x* and the number of patches of connected pixels and report on the proportion of non NA values. The patch count can be restricted to those with a size greater *min.size* which specifies the minimum number of pixels per patch. Then, the function will use this data to rank the elements of *y* according to the order of the keywords in *priority*. The user can choose one or more of the following keywords:

- *class_count* - Priority given to the highest class count.
- *pixel_frequency* - Priority given to the highest non-NA pixel count.
- *patch_count* - Priority given to the highest patch count.
- *road_distance* - Priority given to shortest distance.

The final output is a *data.frame* reporting on:

- *x* - Polygon centroid x coordinate.
- *y* - Polygon centroid y coordinate.
- *mape* - Mean Absolute Percent Error.
- *count* - Number of pixel regions.
- *frequency* - Number of non-NA pixels.
- *distance* - Linear distance to the closest road.
- *ranking* - Priority ranking

Value

A list.

See Also

derivePlots ccLabel

Examples

```
{

require(raster)
require(RStoolbox)
require(ggplot2)
```

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

# read road information
data(roads)

# unsupervised classification with kmeans
uc <- unsuperClass(r, nSamples=5000, nClasses=5)$map

# derive potential sampling plots
pp <- derivePlots(uc, 1000)

# plot ranking
pp@data <- rankPlots(uc, pp, roads)

# plot output
gp <- fortify(pp, region="ranking")
ggplot(gp, aes(x=long, y=lat, group=group, fill=as.numeric(gp$id))) +
geom_polygon() + scale_fill_continuous(name="Ranking")

}
```

`raster2sample` *raster2sample*

Description

Converts a raster grid to points.

Usage

```
raster2sample(x)
```

Arguments

x	Object of class *SpatialPolygons* or *SpatialPolygonDataFrame*.

Details

poly2Sample extends on the `rasterToPoints` function from the raster package. For each non-NA pixel in *x*, the function will use 3x3 moving window and report on the frequency of non-NA pixels. This can be useful to identify "pure" samples within a clump of pixels (i.e. high frequency) as well as mixed pixels along their borders (i.e. low frequency). The output is a *SpatialPointsDataFrame* reporting on:

- *x* - x coordinate.
- *y* - y coordinate.
- *cover* - Non-NA value frequency.
- *id* - Corresponding raster value in *x*.

Value

A *SpatialPointsDataFrame* with sampled pixels reporting on pixel compactness.

See Also

poly2sample ccLabel

Examples

```
{

 require(raster)

 # load example image
 r <- raster(system.file("extdata", "ndvi.tif", package="fieldRS")[1])
 r[r < 5000] <- NA

 # extract samples
 samples <- raster2sample(r)

}
```

---

referenceProfiles *Reference profiles.*

Description

Reference NDVI profiles of selected classes.

Usage

```
data(referenceProfiles)
```

Format

A data.frame

relative.freq *relative.freq*

Description

Estimate the relative frequency of non-NA pixels.

Usage

```
relative.freq(x, na.rm = TRUE)
```

Arguments

x	A vector of class *numeric* or an object of class *rasterLayer*.
na.rm	Logical. Should the NA values be excluded. Default is TRUE.

Details

Estimates the relative frequency of non-NA values.

Value

A *numeric* element or an object of class *RasterLayer*.

See Also

mape

Examples

```
{

x <- c(1, 1, 1, NA, NA, 1, NA)
f <- relative.freq(x, na.rm=TRUE)

}
```

roads *Road shapefile.*

Description

Road shapefile information extract from Open Street Map.

Usage

```
data(roads)
```

Format

A SpatialLinesDataFrame

samples1 *Pixel sample shapefile.*

Description

Samples derived from the fieldData dataset with poly2sample().

Usage

```
data(samples1)
```

Format

A SpatialLinesDataFrame

spCentroid *spCentroid*

Description

Aggregates a spatial object into regions.

Usage

```
spCentroid(x)
```

Arguments

x	An object of class *SpatialPoints* or *SpatialPolygons*.

Details

Returns the centroid of each element in *x*.

Value

A *spatialPointsDataFrame* object.

See Also

ecDistance

Examples

```
{

require(raster)

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

# read field data
data(fieldData)

# derive centroids
c <- spCentroid(fieldData)

# plot polygons and compare with centroids
plot(fieldData)
points(c, col="red")

}
```

Index