Package 'fieldRS'

July 29, 2019

Type Package

Title Remote Sensing Field Work Tools

Version 0.2.2

Date 2019-07-29

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

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

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

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.

Encoding UTF-8

LazyData TRUE

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

RoxygenNote 6.1.1

License GPL (>= 3)

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

VignetteBuilder knitr

NeedsCompilation no

Author Ruben Remelgado [aut, cre]

Repository CRAN

Date/Publication 2019-07-29 18:30:02 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 surpasses *change.threshold*.
- *temporal* - Estimates the MAPE among all bands in a raster object and distinguishes spatially neighboring pixels when the temporal change surpasses *change.threshold*.

When using the *spatial* and *temporal* methods, the value of *change.threshold* will influence the output. If the value is negative, the function will return the pixels that are below the threshold and vice versa when positive. Let's assume we are dealing with crop fields. When, using a negative threshold and the *spatial* method, the function will return homogeneous groups of pixels. This happens because the borders of between fields offer a contract and thus receive higher MAPE values. However, if we apply the same threshold when using the *temporal* method, the borders will be highlighted. This happens because the crop fields are more variant over time due to their fast growth while the borders remain virtually unchanged. 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 spatially-explicit predictive model as well as sample-wise validation.

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 *SpatialPolygons* 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, method = "simple", smooth.x = FALSE)
```

Arguments

`x`	Object of class *RasterLayer*.
`method`	One of "simple" or "complex".
`smooth.x`	A logical argument.

Details

Assuming *x* is a classified or segmented image, this function segments it using ccLabel and, draws polygons for each group of connected pixels. The way polygons are drawn depends on the *method* keyword. The function will accept one of the following options:

- *simple* - Extracts the center pixel coordinates and builds a polygon based on their minimum convex hull.
- *complex* - Extracts the center pixel coordinates and builds a polygon based on their minimum concave hull.

The "simple" approach is a faster but it can lead to poor results when dealing with very complex shapes. For example, crop fields can be rectangular in which case the "simple" method is sufficient. On the other hand, forest belts can have irregular shapes, in which case the "complex" method is more appropriate. By default, the function will use all pixels associated to a region. However, if *smooth.x* is set to TRUE, for each region, the function will remove samples with less than 5 neighbors efectively removing isolated pixels along the edge of a region. The final output is a *SpatialPolygonsDataFrame* reporting on:

- *region.id* - Unique polygon identifier corresponding to the original pixel region.
- *area* - Polygon Area (in square meters).
- *perimeter* - Polygon perimeter (in meters).
- *cover.ratio* - Ration between the polygon area and the area of the corresponding 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, method="temporal", change.threshold=50)$regions

# convert to polygons and plot (simple)
ef <- extractFields(or[1:50,1:50, drop=FALSE])
plot(ef)

# convert to polygons and plot (complex)
ef <- extractFields(or[1:50,1:50, drop=FALSE], method="complex")
plot(ef, border="red", add=TRUE)

# convert to polygons and plot (complex and smoothed)
ef <- extractFields(or[1:50,1:50, drop=FALSE], method="complex", smooth.x=TRUE)
plot(ef, border="blue", add=TRUE)

}
```

---

fieldData *Polygon shapefile.*

Description

Ground truth data on crop types collected in Uzbekistan.

Usage

```
data(fieldData)
```

Format

A SpatialPolygonsDataFrame

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, auto = FALSE)
```

Arguments

x	Vector of class *character*.
y	Vector of class *character*.
z	Vector of class *character*.
auto	Logical argument. Default is FALSE.

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*.

If *auto* is set to TRUE, the user can ignore *z* to correct the existing labels. Instead, the user can provide all the potential cases through *y*. Then, for each element in *x*, the function will return the most similar element in *y* using the `stringdist` function.

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)

}
```

`pixFilter` *pixFilter*

Description

Erosion and dilation filter of a raster image.

Usage

```
pixFilter(x, y, method)
```

Arguments

`x`	Object of class *RasterLayer*.
`y`	A *numeric* element.
`method`	A *character* element. One of "erode" or "dilate".

Details

Uses focal to filter *x* using either an erosion or a dilation filter, specified by *method*. If "erosion" is chosen, the function will identify and filter out border pixels around each cluster of pixels in *x*. Small or isolated groups of pixels will also be removed. If "dilation" is set, the function will increase the size of each cluster of pixels and simultaneously remove all gaps within them. The size of the buffer used in this function is defined by *y* and is expressed in number of pixels.

Value

A *RasterLayer*.

Examples

```
{

require(raster)

# read raster data
r <- raster(system.file("extdata", "ndvi.tif", package="fieldRS")) > 2000
r[r == 0] <- NA

# filter image (3x3 erosion)
```

```
or <- pixFilter(r, 1, "erode")
plot(r)
plot(or)

#' # filter image (3x3 dilation)
or <- pixFilter(r, 1, "dilate")
plot(r)
plot(or)

}
```

`poly2sample` *poly2sample*

Description

Converts a raster grid to point samples based on a overlapping polygon.

Usage

```
poly2sample(x, y, preserve.id = TRUE, min.cover = 0)
```

Arguments

`x`	Object of class *SpatialPolygons* or *SpatialPolygonDataFrame*.
`y`	A raster object or a numeric element.
`preserve.id`	Logical. Should the output be polygon-wide?
`min.cover`	A numeric element between 0 and 1.

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*. Sometimes, two or more polygons can cover the same pixel. When this happens, the function will assign this pixel to each of the overlapping polygons and thus replicate this sample. When we want to analyze each polygon individually this can be useful. However, if e.g. the polygons are all of same class and we want to avoid replicated samples, we can set *preserve.id* to FALSE. When doing so, the function will identify all the unique pixels covered polygons and for, the duplicated it will sum the percent cover values. Points with percent values below *min.cover* will be omitted from the output.

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("diversity", "richness",
  "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:

- *diversity* - Priority given to the highest Shannon, class diversity.
- *richness* - Priority given to the highest class richness (number of classes in plot / total number of classes).
- *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.
- *diversity* - Class diversity.

- *richness* - Class richeness.
- *pixel.frequency* - Number of non-NA pixels.
- *road.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 *SpatialPoints-DataFrame* 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")

}
```