Package ‘deforestable’

May 8, 2026

Type Package

Title Classify RGB Images into Forest or Non-Forest

Version 3.1.2

Author Jesper Muren [aut] (ORCID: <https://orcid.org/0000-0002-9208-5325>), Dmitry Otryakhin [aut, cre] (ORCID: <https://orcid.org/0000-0002-4700-7221>)

Maintainer Dmitry Otryakhin <d.otryakhin.acad@protonmail.ch>

Description Implements two out-of box classifiers presented in <doi:10.1002/env.2848> for distinguishing forest and non-forest terrain images. Under these algorithms, there are frequentist approaches: one parametric, using stable distributions, and another one-non-parametric, using the squared Mahalanobis distance. The package also contains functions for data handling and building of new classifiers as well as some test data set.

License GPL-3

Encoding UTF-8

Depends R (>= 4.1.0)

Imports terra, jpeg, plyr, StableEstim, Rcpp (>= 1.0.9)

LinkingTo Rcpp, RcppArmadillo

SystemRequirements GDAL (>= 2.2.3), GEOS (>= 3.4.0), PROJ (>= 4.9.3), sqlite3

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation yes

Repository CRAN

Date/Publication 2025-10-19 22:10:02 UTC

Contents

classify 2
Class_ForestTrain 3

createDataPartition 4
Koutparams 5
read_data 6
train 7

Index **10**

`classify` *Classify parts of images as forest / non-forest*

Description

Generic function classify dispatches methods according to the class of object Model. A chosen method takes raster object data and classifies parts of it as 1- forest or 0- non-forest.

Usage

```
classify(Model, ...)

## S3 method for class 'ForestTrainParam'
classify(Model, data, n_pts, parallel = FALSE, progress = "text", ...)

## S3 method for class 'ForestTrainNonParam'
classify(Model, data, n_pts, parallel = FALSE, progress = "text", ...)
```

Arguments

`Model`	trained model, e.g. by `train`
`...`	additional parameters passed to methods
`data`	raster object. `read_data_raster`
`n_pts`	size of sub-frames into which data is split
`parallel`	Boolean. Whether to use parallel setup
`progress`	progress bar. Works only when parallel=FALSE. Could be set to 'text' or 'none'

Details

Both classify.ForestTrainParam and classify.ForestTrainNonParam use parameter n_pts to split images into square sub-frames of the size n_pts. Those sub-frames are classified independently and all pixels from a sub-frame are tagged according to its classification result. When the image contained by data is of dimensions that are not divisible by n_pts, it is truncated from the right and the bottom to to make the largest divisible one. Thus, the result of classification can be of a different size than the original image.

Value

a black-and-white image of the terrain data where white represents forest and black is for non-forest.

Methods (by class)

- `classify(ForestTrainParam)`: Method for the class ForestTrainParam
- `classify(ForestTrainNonParam)`: Method for the class ForestTrainNonParam

Examples

```
library(deforestable)
n_pts <- 20

# Choosing folders with training data
Forestdir <- system.file('extdata/Forest/', package = "deforestable")
Nonforestdir <- system.file('extdata/Non-forest/', package = "deforestable")

#### Read the target image ####
tg_dir <- system.file('extdata/', package = "deforestable")
test_image <- read_data_raster('smpl_1.jpeg', dir = tg_dir)


# Simple training of the non-parametric model
Model_nonP_tr <- train(model='fr_Non-Param', Forestdir=Forestdir, Nonforestdir=Nonforestdir,
                       train_method='train', parallel=FALSE)

res <- classify(data=test_image, Model=Model_nonP_tr,
                n_pts=n_pts, parallel=FALSE, progress = 'text')

tmp_d <- tempdir(); tmp_d
jpeg::writeJPEG(image=res, target=paste(tmp_d,'Model_nonP_tr.jpeg', sep='/'))
```

`Class_ForestTrain` *S3 class ForestTrain*

Description

Class ForestTrain is the main class to contain models for binary classification forest/non-forest. It includes the following elements:

Details

In most cases objects of this class are generated by function `train`. Then, classification of terrain images is made by `classify`.

Slots

Element	Description
call	the function call with which it was created
tp	the number of true positives obtained during training
fp	the number of false positives obtained during training

tn	the number of true negatives obtained during training
fn	the number of false negatives obtained during training

`createDataPartition` *Data Partitioning*

Description

As input data, the functions need two folders- Nonforestdir with images of non-forest and forestdir with ones of forest. createDataPartition() splits data into training and testing partitions while keeping the relative sample size of the classes the same as in the original data. createFolds() splits the data into k folds for cross-validation.

Usage

```
createDataPartition(forestdir, Nonforestdir, times = 1, p = 0.5)

createFolds(forestdir, Nonforestdir, k = 5)
```

Arguments

`forestdir`	path to the directory with (only) forest images
`Nonforestdir`	path to the directory with (only) non-forest images
`times`	the number of data partitions to make
`p`	the percentage of data to set aside for training
`k`	the number of folds to split the data into

Value

createDataPartition returns a list of data partitions. Each partition consists of 4 sets- forest training, non-forest training, forest test and non-forest test set. createFolds returns lists $forest and $nonforest with k folds in each of them.

Functions

- `createFolds()`: Split data into folds

Examples

```
library(deforestable)
forestdir <- system.file('extdata/Forest/', package = "deforestable")
Nonforestdir <- system.file('extdata/Non-forest/', package = "deforestable")

trainPart <- createDataPartition(forestdir=forestdir, Nonforestdir=Nonforestdir, p = .7, times = 1)

folds <- createFolds(forestdir, Nonforestdir, k = 10)
```

`Koutparams` *Koutrouvelis parameter estimation of image data*

Description

In data, there are three columns and each column corresponds to the color intensity of one channel: red, green and blue correspondingly. The four parameters: alpha, beta, gamma and delta, of the stable distribution is estimated for each of these channels using the Koutrouvelis regressions-type technique.

Usage

```
Koutparams(data)
```

Arguments

`data`	matrix or data frame with color intensities of red, green and blue for an image.

Value

a data frame with columns alpha, beta, gamma, delta and rows red, green and blue.

Examples

```
library(deforestable)

Forestdir <- system.file('extdata/Forest/', package = "deforestable")
test_image <- read_data('_6_33_.jpeg', dir = Forestdir)

pars <- Koutparams(test_image)

pars
```

read_data *Import a jpeg image*

Description

All these functions are made to read jpeg images, the difference is in the class of objects they return

Usage

```
read_data(filename, dir)

read_data_matrix(filename, dir)

read_data_raster(filename, dir)
```

Arguments

filename	name of the jpeg file to import
dir	the directory where the image is located

Value

read_data returns a 3-column data.frame with pixels in rows and red, green, blue intensities in columns. read_data_matrix reads jpeg images and returns 3 matrices for each of red, green and blue colors. read_data_raster imports jpeg as a raster object `rast`.

Functions

- `read_data_matrix()`: returns three matrices
- `read_data_raster()`: returns a SpatRaster object

Examples

```
dir <- system.file('extdata/Forest/', package = "deforestable")

dd <- read_data(filename='_6_33_.jpeg', dir=dir)
hist(dd[,1])


dir <- system.file('extdata/Forest/', package = "deforestable")

dd <- read_data_matrix(filename='_6_33_.jpeg', dir=dir)


dir <- system.file('extdata/Forest/', package = "deforestable")

dd<-read_data_raster(filename='_8_46_.jpeg', dir=dir)
```

train *Train models for forest detection*

Description

As input data, the function needs two folders- Nonforestdir with images of non-forest and Forestdir with ones of forest. train() uses all images in both folders to train a model. Putting an image into an incorrect folder is equivalent to tagging the image incorrectly.

Usage

```
train(
  n_pts,
  model = c("fr_Non-Param", "fr_Param"),
  Forestdir,
  Nonforestdir,
  train_method = c("cv", "train"),
  k_folds,
  parallel = FALSE
)
```

Arguments

n_pts	matters only when train_method='cv'. Defines the size of the square sub-frames into which images would be split during cross-validation.
model	which model to train
Forestdir	path to the directory with (only) forest images
Nonforestdir	path to the directory with (only) non-forest images
train_method	how to train the model: simple training, cross-validation.
k_folds	matters only when train_method='cv'. The number of folds in the k-fold cross-validation setup.
parallel	matters only when train_method='cv'. Boolean. whether or not use a parallel setting during cross-validation

Details

Currently, both fr_Non-Param and fr_Param use parameter n_pts only in the testing part of cross-validation, not during training. Training is always done on whole original images in the training folders.

Value

object of class ForestTrain potentially with a sub-class. See Class_ForestTrain.

Examples

```
library(deforestable)
n_pts <- 20

# Choosing folders with training data
Forestdir <- system.file('extdata/Forest/', package = "deforestable")
Nonforestdir <- system.file('extdata/Non-forest/', package = "deforestable")

k_folds=3;

#### Read the target image ####
tg_dir <- system.file('extdata/', package = "deforestable")
test_image <- read_data_raster('smpl_1.jpeg', dir = tg_dir)


#### Models ####


# Simple training of the non-parametric model
Model_nonP_tr <- train(model='fr_Non-Param', Forestdir=Forestdir, Nonforestdir=Nonforestdir,
                       train_method='train', parallel=FALSE)

res <- classify(data=test_image, Model=Model_nonP_tr,
                n_pts=n_pts, parallel=FALSE, progress = 'text')

tmp_d <- tempdir(); tmp_d
jpeg::writeJPEG(image=res, target=paste(tmp_d,'Model_nonP_tr.jpeg', sep='/'))


# Cross-validation of the non-parametric model
Model_nonP_cv <- train(n_pts=n_pts, model='fr_Non-Param', Forestdir=Forestdir,
                       Nonforestdir=Nonforestdir, train_method='cv',
                       k_folds=k_folds, parallel=FALSE)

res <- classify(data=test_image, Model=Model_nonP_cv,
                n_pts=n_pts, parallel=FALSE, progress = 'text')

tmp_d <- tempdir(); tmp_d
jpeg::writeJPEG(image=res, target=paste(tmp_d,'Model_nonP_cv.jpeg', sep='/'))




# Cross-validation of the parametric model
Model_P_cv <- train(n_pts=n_pts, model='fr_Param', Forestdir=Forestdir,
                    Nonforestdir=Nonforestdir, train_method='cv',
                    k_folds=k_folds, parallel=FALSE)

res <- classify(data=test_image, Model=Model_P_cv,
                n_pts=n_pts, parallel=FALSE, progress = 'text')

tmp_d <- tempdir(); tmp_d
```

```
jpeg::writeJPEG(image=res, target=paste(tmp_d,'Model_P_cv.jpeg', sep='/'))


# Simple training of the parametric model
Model_P_tr <- train(model='fr_Param', Forestdir=Forestdir, Nonforestdir=Nonforestdir,
                    train_method='train', parallel=FALSE)
res <- classify(data=test_image, Model=Model_P_tr,
                n_pts=n_pts, parallel=FALSE, progress = 'text')

tmp_d <- tempdir(); tmp_d
jpeg::writeJPEG(image=res, target=paste(tmp_d,'Model_P_tr.jpeg', sep='/'))
```

Index

Class_ForestTrain, 3, *7*
classify, 2, *3*
createDataPartition, 4
createFolds *(*createDataPartition*)*, 4

Koutparams, 5

rast, *6*
read_data, 6
read_data_matrix *(*read_data*)*, 6
read_data_raster, *2*
read_data_raster *(*read_data*)*, 6

train, *2, 3*, 7