Package 'implant'

May 1, 2020

Type Package

Title A High-throughput Phenotyping Pipeline for Image Processing and Functional Growth Curve Analysis

Version 0.1.0

Author Ronghao Wang, Yumou Qiu, Yuzhen Zhou, Yuhang Xu, James Schnable

Maintainer Ronghao Wang <rhwang64@gmail.com>

Description The package ``implant'' is used for both image processing and functional data analysis, which is able to provide statistical inference for the plant traits directly from the input images. For image processing, the package provides methods including thresholding, hidden Markov random field model,etc. For the growth curve analysis, this package can produce nonparametric curve fitting with its confidence region for plant growth. A functional ANOVA model to test for the treatment and genotype effects of the growth curve dynamics is also provided.

Depends

Imports stats,
fda,
matrixcalc,
png,
MASS

Suggests imager, R.rsp

VignetteBuilder R.rsp

License GPL (>= 2)

Encoding UTF-8

LazyData true

R topics documented:

CI 2
CI_contrast 4
Color2Gray 5
ColorB 6
ColorG 7
Color_Exchange 8
dilation 9
downsize 9

downsize_avg 10
downsize_matrix 11
erosion 12
extract_pheno 13
fanova 14
HMRF 17
imageBinary 18
image_kmeans 19

Index **21**

`CI` *Estimating a linear combination of treatment effects and obtaining its confidence regions in FANOVA*

Description

This function is used for estimating a linear combination of treatment effects and its confidence regions in functional data analysis. Moreover, it can be used for testing the difference between two treatments.

Usage

```
CI(fit, L, alpha)
```

Arguments

`fit`	an object of output obtained by function "fanova".
`L`	a numeric contrast vector corresponding to the design matrix in "fanova", which specifies the linear combination of the parameters of interest. Users can use the design matrix output by the function fanova( ), to idenfity the value of L. See the example part for details.
`alpha`	a positive small number between 0 and 1. 1-alpha gives the confidence level. In default, alpha = 0.05.

Details

We can estimate a linear combination of treatment effects and its confidence regions with the function "$CI(fit, L \ldots)$", where fit is the output from "$fanova(\ldots)$", and L is a contrast vector under your fanova model specifying the linear combination of the parameters of interest. The parameter estimation $\hat{\beta}$ is implemented by the function "$fanova(\ldots)$" in our package. By specifying the linear vector L, the estimation of interest at time t is:

$$\hat{\tau}(t) = L \otimes B'(t)\hat{\beta},$$

where "$\otimes$" is the Kronecker product, and $B(t) = (B_{d,1}, B_{d,2}, \ldots, B_{d,K})'$. Here d is the order of the B-spline basis function and K is the rank of the B-spline basis function. More details can be found in the help documentation of function "fanova".

In the example part, we study the relationship between the plant sizes and treatments using functional data models. Let $y_i(t)$ be the size of the ith plant measured at time t, where $i = 1, \ldots, 9$. We treat genotype and block as fixed effects. There are 3 different genotypes $(1, 2, 3)$ and 3 blocks

$(1, 2, 3)$ in this study. We can use $Gi = (G_{ik})_{k=2}^{3}$ as the categorical indicator of the ith plant genotype. Specifically, G_{ik} is set to one if the plant has the kth genotype; otherwise, $G_{ik} = 0$. For example, if the ith plant has the 2nd genotype, then $Gi = (0, 1, 0)$. And Gis being zeros represents the $1st$ genotype, which is treated as the baseline. Similarly, $Pi = (P_{ik})_{k=2}^{3}$ is defined to indicate the block that the ith plant belongs to, and the first block is set as the baseline. The model can be written as:

$$y_i(t_{ij}) = \mu(t_{ij}) + \sum_{k=2}^{3} G_{ik} g_k(t_{ij}) + \sum_{k=2}^{3} P_{ik} p_k(t_{ij}) + \epsilon_i(t_{ij}),$$

where $\mu(t)$ is the growth function of the plant with the 1st genotype (Genotype 1) from the 1st block, $g_k(t)$s are the genotype effect functions, $p_k(t)$s are the fixed block effect functions, and the residuals $\epsilon_i(t)$s are modeled by independent random processes with mean zeros.

Using the function $CI(\ldots)$, we can estimate a linear combination of treatment effects, including estimating the average growth curve of a particular genotype over all the blocks. In the example part, we use this function to estimate the average growth curve of Genotype 1 over three blocks.

Value

`trt`	a t by 1 vector giving the estimated linear combination of coefficients at t observation time points.
`ub`	A t by 1 vector which indicates the upper bound of the (1-alpha) confidence band at different time points.
`lb`	A t by 1 vector which indicates the lower bound of the (1-alpha) confidence band at different time points.

See Also

`fanova`

Examples

```
#load the data
data_new = read.csv(system.file("extdata","data.csv",
                                package = "implant", mustWork = TRUE))
#The first three columns of data_new refer to the original position of the observations from the
#original dataset, genotype and block, respectively
Y = data_new[,-c(1:3)]
#This step is to factorize each factor
Genotype = as.factor(data_new$Genotype)
Block = as.factor(data_new$Block)
X = data.frame(Genotype,Block)
formula = "~ Genotype + Block"
tt = seq(from = 0, to = 44,by = 2)
#fit the model
fit = fanova(Y.na.mat = Y, X = X, tt = tt, formula, K.int = 6, order = 4, lower = -10, upper = 15)
fit$design_mat
> fit$design_mat
      (Intercept) Genotype2  Genotype  Block2  Block3
1           1          0          1      0      0
2           1          0          0      0      0
3           1          1          0      0      0
4           1          1          0      1      0
5           1          0          1      1      0
```

```
6           1          0          0      1      0
7           1          0          1      0      1
8           1          1          0      0      1
9           1          0          0      0      1
#We want to estimate the growth curve of genotype 1 averaging over three blocks,
#that is, we are interested in the 1st, the 4th and the 5th column of the design matrix.
#Therefore, we can define:
L1 = c(1, 0, 0, 1/3, 1/3)
ci1 = CI(fit = fit, L = L1, alpha = 0.05)
plot(tt,ci1$trt,type = "l")
lines(tt,ci1$lb, col = "blue")
lines(tt,ci1$ub, col = "blue")
```

`CI_contrast` *Constructing confidence regions for specific treatment effect in FANOVA*

Description

This is a special case of the function "CI" in this package. This function is easier to use when studying the difference between two specific treatments/genotype, without specifying the linear combination needed in "CI".

Usage

```
CI_contrast(fit, j1, j2, alpha)
```

Arguments

`fit`	An object of output obtained by function "fanova".
`j1`	A positive integer, specifying the columns of the design matrix corresponding to one treatment of interest. Users can use the design matrix output by the function fanova( ), to idenfity the value of j1. See the example part for details.
`j2`	A positive integer, specifying the columns of the design matrix corresponding to the other treatment of interest. Users can use the design matrix output by the function fanova( ), to idenfity the value of j2. See the example part for details.
`alpha`	A positive small number between 0 and 1. 1-alpha gives the confidence level. In default, alpha = 0.05.

Details

We can test the significance of the treatment effects of interest by constructing the corresponding confidence regions with the function "$CI\_contrast(fit, j1, j2, \ldots)$", where fit is the output from "$fanova(\ldots)$", and $j1$ and $j2$ specify the columns of the design matrix corresponding to the treatment of interest.

Value

`trt`	a t by 1 vector, which refers to the estimated coefficients, where t is the number of observation time points specified in the argument tt in the function fanova( ).

lb — a t by 1 vector, which refers to the lower bound of the (1-alpha) confidence band, where t is the number of observation time points specified in the argument tt in the function fanova( ).

ub — a t by 1 vector, which refers to the the upper bound of the (1-alpha) confidence band, where t is the number of observation time points specified in the argument tt in the function fanova( ).

See Also

fanova

Examples

```
#load the data
data_new = read.csv(system.file("extdata","data.csv",
                                package = "implant", mustWork = TRUE))
#The first three columns of data_new refer to the original position of the observations from the
#original dataset, genotype and block, respectively
Y = data_new[,-c(1:3)]
#This step is to factorize each factor
Genotype = as.factor(data_new$Genotype)
Block = as.factor(data_new$Block)
X = data.frame(Genotype,Block)
formula = "~ Genotype + Block"
tt = seq(from = 0, to = 44,by = 2)
#fit anova
fit = fanova(Y.na.mat = Y, X = X, tt = tt, formula, K.int = 6, order = 4, lower = -10, upper = 15)
fit$design_mat
> fit$design_mat
       (Intercept) Genotype2  Genotype  Block2  Block3
1           1          0           1      0      0
2           1          0           0      0      0
3           1          1           0      0      0
4           1          1           0      1      0
5           1          0           1      1      0
6           1          0           0      1      0
7           1          0           1      0      1
8           1          1           0      0      1
9           1          0           0      0      1
#We want to test the significance between genotype 2 and genotype 3,
#that is, we are interested in the 2nd, the 3rd column of the design matrix.
#Therefore, we can define:
ci_diff = CI_contrast(fit = fit,j1 = 2, j2 = 3, alpha = 0.05)
plot(tt,ci_diff$trt,type = "l")
lines(tt,ci_diff$lb, col = "blue")
lines(tt,ci_diff$ub, col = "blue")
```

Color2Gray *Converting an RGB image to a grayscale image*

Description

This function is used to convert an RGB image to Grayscale.

Usage

```
Color2Gray(image, weight = c(0.299,0.587,0.114))
```

Arguments

image	an array of an RGB image file for processing.
weight	a numeric vector, in which elements are weights for the Red channel of the image, Green channel of the image, and the Blue channel of the image, respectively.

Value

image	A matrix of pixels of the image converted from RGB to Grayscale.

Examples

```
#read an RGB image
library(png)
image = readPNG(system.file("extdata", "image1.png",
                            package = "implant",
                            mustWork = TRUE))[,,c(1:3)]
imageGray = Color2Gray(image)
```

ColorB *Identifying the region of interest of a plant*

Description

This function is specifically designed for processing the plant images taken in the University of Nebraska-Lincoln (UNL) greenhouse Innovation Center. It helps identify the left and right boundaries of the plant by removing the parts of the image that contains the black bars, but keep the black part of the pot. In a word, to help users identify the region of interest. This is based on locating the position of the black bars in the background. See the image1.png in the example data.

Usage

```
ColorB(image, colThreshold = 0.5)
```

Arguments

image	an RGB image of plant.
colThreshold	a positive real number, which is the used to identify the black bars appear in the columns of the image.

Value

lb	left bound of the part of the region of interest.
rb	right bound of the part of the region of the interest.
c	A modified image replacing all the area outside the region of interest by white color.

Examples

```
#read an RGB image
library(png)
image = readPNG(system.file("extdata", "image1.png",
                            package = "implant",
                            mustWork = TRUE))[,,c(1:3)]
writePNG(image,"~/imageOriginal.png")
resultColor = ColorB(image)
bound = c(resultColor$lb, resultColor$rb)
imageColor = resultColor$c
writePNG(imageColor,"~/imageColor.png")
```

`ColorG` *Identifying the region of interest (lower bound) for a plant image*

Description

This function is specifically designed for processing the plant images taken in the University of Nebraska-Lincoln (UNL) greenhouse Innovation Center. It helps identify the lower boundary of the plant by using images of empty pot with a plastic green bar. See the sample image: image_pot.png.

Usage

```
ColorG (imagefile, rowThreshold = 0.007, Bthreshold = 60 / 255,
EGThreshold = 0.1, weight = c(-1, 2, -1), changeto = c(1, 1, 1))
```

Arguments

`imagefile`	an input RGB image of the empty pot
`rowThreshold`	a positive real number, used to identify the position of the green strip in the empty pot.
`Bthreshold`	a value between 0 and 1. It is applied to the sum of the RGB intensities.
`EGThreshold`	a value between 0 and 1. It is applied to the contrast intensity by the specified weight in the function.
`weight`	a 3 by 1 numeric vector. The three elements indicate the weight of the pixel intensities of R, G, B, respectively. In default, it takes weight c(-1,2,-1), for contrast intensity.
`changeto`	a numeric vector, with values between 0 and 1, giving the color intensity to change the green trip in the empty pot.

Details

In the example part, this function identifies the green strip in an empty pot, which can be considered as the lower boundary of a plant, and change the color of the green strip from green to white.

Value

`lowb`	lower bound of the region of interest.
`c`	output image with changed color on the green strip.

Examples

```
#read an RGB image
library(png)
imageX = readPNG(system.file("extdata", "image_pot.png",
                             package = "implant",
                             mustWork = TRUE))[,,c(1:3)]
writePNG(imageX,"~/imageX.png")
tempG = ColorG (imagefile = imageX, rowThreshold = 0.007, Bthreshold = 60 / 255,
EGThreshold = 0.1, weight = c(-1, 2, -1), changeto = c(1, 1, 1))
tempG$c
tempG$lowb
```

`Color_Exchange` *Exchanging the color of the background and the foreground.*

Description

This function exchanges the color of the background and the foreground for a binary image.

Usage

```
color_exchange(image1)
```

Arguments

`image1`	A binary matrix of a segmented image

Details

Exchange 0 and 1 in the input binary matrix.

Value

A binary matrix with exchanged background and the foreground.

Examples

```
#read an RGB image
library(png)
image = readPNG(system.file("extdata", "image1.png",
                             package = "implant",
                             mustWork = TRUE))[,,c(1:3)]
imageB = imageBinary(image,  weight = c(-1, 2, -1),
threshold1 = 30 / 255, threshold2 = 0.05)
color_exchange(imageB)
```

dilation *Morphological Dilation*

Description

This function is used to perform morphological dilation of an image.

Usage

```
dilation(imagefile, mask = matrix(1, 3, 3))
```

Arguments

imagefile	a binary matrix of the segmented image.
mask	a matrix constructed by structuring elements.

Value

a binary matrix of the eroded image.

References

Image Analysis and Mathematical Morphology by Jean Serra, ISBN 0-12-637240-3 (1982)

Examples

```
#read an RGB image
library(png)
image = readPNG(system.file("extdata", "image1.png",
                            package = "implant",
                            mustWork = TRUE))[,,c(1:3)]

imageB = imageBinary(image, weight = c(-1, 2, -1),
                     threshold1 = 30 / 255,
                     threshold2 = 0.075)

imageBD = dilation(imageB, mask = matrix(1, 5, 5))
```

downsize *Reducing the size of an RGB image*

Description

This function reduces the size of an image by picking sample pixels from each row and each column of the original image, and use the selected pixels to construct a reduced image.

Usage

```
downsize(image, RowSample = 1, ColSample = 1)
```

Arguments

image	an input RGB image array
RowSample	a positive integer for the rows. Select one component from every RowSample components. For example, RowSample = 2 means you select the 1st, 3rd, ... element from each row of the original image to construct your new image.
ColSample	a positive integer for the columns. Select one component from every ColSample components. For example, ColSample = 2 means you select the 1st, 3rd, ... element from each column of the original image to construct your new image.

Details

This function can be used to reduce the size of an image by picking sample elements of the original image as the elements of the reduced image. For example, "RowSample = 2", and "ColSample = 2" reduce the original size of an image to a quarter.

Value

array of pixels of the reduced image.

Note

This function is different from another function in this package called "downsize_matrix( )". That function is used to reduce the size of matrices while this function, downsize( ), is used to reduce the size of 3-D arrays.

Examples

```
#read an RGB image
library(png)
image = readPNG(system.file("extdata", "image1.png",
                            package = "implant",
                            mustWork = TRUE))[,,c(1:3)]
downsize(image, RowSample = 2, ColSample = 2)
```

---

downsize_avg *Reducing size of an image using the method of averaging in blocks.*

Description

This function is used for reducing size of an image by averaging its pixels in blocks.

Usage

```
downsize_avg(image, block_nrow = 2, block_ncol = 2)
```

Arguments

image	a pixel matrix or an array of the image for processing.
block_nrow	an integer number, which is the number of rows to select to construct your new array. For example, if block_nrow = 2 means you select every two rows of the input image as a block. Note that this number needed to be divisible by number of rows of the array of the input image.
block_ncol	an integer number, which is the number of columns to select to construct your new array. For example, if block_ncol = 2 means you select every two columns of the input image as a block. Note that this number needed to be divisible by number of columns of the array of the input image.

Details

This function is used to reduce the size of an image by dividing the original array into several blocks and calculate the average values within each block.

Value

a pixel array of the reduced image.

Note

block_nrow and block_ncol must be divisible by number of rows and columns of the pixel-arrary of the image, respectively. Otherwise, Errors will be reported as: "block_nrow(block_ncol) must be divisible by number of rows(columns) of the pixel-arrary of the image."

Examples

```
#read an RGB image
library(png)
image = readPNG(system.file("extdata", "image1.png",
                            package = "implant",
                            mustWork = TRUE))[,,c(1:3)]
downsize_avg(image,2,2)
```

---

downsize_matrix *Reducing the Size of a Matrix*

Description

This function reduces the size of a matrix by picking one component from every k1 components in each row and one component from every k2 components in each column of the input matrix, and use the selected elements to construct a reduced matrix, where k1 and k2 are specified by "RowSample" and "ColSampe", respectively.

Usage

```
downsize_matrix(M, RowSample = 1, ColSample = 1)
```

Arguments

M	the original input matrix.
RowSample	a positive integer for the rows. Select one component from every RowSample components. For example, RowSample = 2 means you select the 1st, 3rd, ... element from each row of the original matrix to construct your new matrix.
ColSample	a positive integer for the columns. Select one component from every ColSample components. For example, ColSample = 2 means you select the 1st, 3rd, ... element from each column of the original matrix to construct your new matrix.

Details

This function can be used to reduce the size of a matrix by picking sample elements of the original matrix as the elements of the reduced matrix. For example, "RowSample = 2", and "ColSample = 2" reduce the original size of a matrix to a quarter.

Value

the reduced matrix.

Note

This function is different from another function in this package called "downsize( )". That function is used to reduce the size of 3-D arrays while this function, downsize_matrix( ), is used to reduce the size of matrices.

Examples

```
#read the red band of an RGB image
library(png)
image = readPNG(system.file("extdata", "image1.png",
                            package = "implant",
                            mustWork = TRUE))[,,1]
downsize_matrix(M = image, RowSample = 2, ColSample = 2)
```

---

erosion *Morphological Erosion*

Description

This function is used to perform morphological erosion of an image.

Usage

```
erosion(imagefile, mask = matrix(1, 3, 3))
```

Arguments

imagefile	a binary matrix of the segmented image.
mask	a matrix constructed by structuring elements.

Value

a binary matrix of the eroded image.

References

Image Analysis and Mathematical Morphology by Jean Serra, ISBN 0-12-637240-3 (1982)

Examples

```
#read an RGB image
library(png)
image = readPNG(system.file("extdata", "image1.png",
                            package = "implant",
                            mustWork = TRUE))[,,c(1:3)]

imageB = imageBinary(image, weight = c(-1, 2, -1),
                     threshold1 = 30 / 255,
                     threshold2 = 0.075)

imageBE = erosion(imageB, mask = matrix(1, 5, 5))
```

`extract_pheno` *Extract phenotypical features from segmented images*

Description

This function extracts phenotypical features from segmented images.

Usage

```
extract_pheno(processed_image, Xsize = 1, Ysize = 1, a = 1, b = 1)
```

Arguments

`processed_image`	a binary matrix contains only 0 and 1, giving the segmented image of a plant.
`Xsize, Ysize`	Xsize and Ysize are the actual horizontal and vertical lengths of one pixel, respectively. The default set is Xsize = 1, Ysize = 1 for the non-adjustment for the pixel size.
`a,b`	positive integers, the same as the values of RowSample and ColSample in the function: "downsize", respectively. This is only used if the function "downsize" was used to reduce the size of the image in image segmentation.

Value

`pixelCount`	The total number of pixels of the segmented plant of interest.
`plantheight`	The height of the segmented plant.
`plantwidth`	The width of the segmented plant.
`plantSize`	The size of the segmented plant based on the total number of pixels of the segmented plant of interest. i.e. Size = pixelCount * Xsize * Ysize.

See Also

downsize for reducing size of an image.

Examples

```
#read an RGB image
library(png)
imageOriginal = readPNG(system.file("extdata", "image1.png",
                            package = "implant",
                            mustWork = TRUE))[,,c(1:3)]
#reduce the size of an image
image_reduced = downsize(imageOriginal, RowSample = 2, ColSample = 2)
#segment the reduced image
imageB = imageBinary(image_reduced, weight = c(-1, 2, -1), threshold1 = 30 / 255, threshold2 = 0.05)
#obtain the size of the segmented image
extract_pheno(processed_image = imageB, Xsize = 1.5, Ysize = 1.5, a = 2, b = 2)$size
```

---

fanova *Fitting Functional ANOVA Models*

Description

This function is to fit fanova models by B-Spline basis expansion with a penalty term on the second derivative of of the estimated functions for the smoothness of the curves. The penalty parameter, λ, is chosen by GCV (generalized cross validation).

Usage

```
fanova(Y.na.mat, X, tt, formula, K.int = 6,
            order = 4, lower = -10, upper = 15)
```

Arguments

Y.na.mat	an n by t matrix of response variable (the extracted features), where each row contains the time series data of an observation, and each column contains all the observations at a given time. Here n is the number of biological units of the study (for example, in the study of plant growth curve, n is the number of plants and t is the number of observation time points. Any missing observation is filled by "NA" in the matrix. For example, if a measurement of plant is missing for plant i on date j, the (i, j) component in Y.na.mat should be filled by "NA".
X	an n by r dataframe or matrix of explanatory variables, where n is the number of observations and r is the number of explanatory variables.
tt	a t by 1 vector, with the same length as the rows in Y.na.mat. Each element implies the observation date.
formula	an object of class "formula", which specifies the model to use.
K.int	a positive integer, which refers to the number of interior knots of the B-spline.
order	a positive integer, which refers to the order of the B-spline. In default, order = 4, which implies that we use a cubic polynomial to connect each two adjacent knots.

d1 — a non-negative integer, which indicates the derivatives of the basis expansion function in the penalty matrix, Ω. See details in the "detail"" part. In default, d1 = 2. Note that d1 should be smaller than the order of the basis function.

lower — lower bound for the possible values of log (smoothing parameter), i.e., $log(\lambda)$, used in GCV.

upper — upper bound for the possible values of log (smoothing parameter), i.e., $log(\lambda)$, used in GCV.

Details

Suppose the trait is potentially affected by q factors. In convenience, let $q = 2$ in the help documentation. Denote by l_q the number of levels of the jth factor. We define $X_{ij} = (x_{ij2}, \ldots, x_{ijl_j})'$ as the categorical indicators of the qth factor of the ith plant. Specifically, x_{ijk} is set to one if the jth factor of the ith plant has level "k"; otherwise, $x_{ijk} = 0$.

Denote by $\otimes$ the Kronecker product of matrices. A functional ANOVA model with interactions can be written in the following form:

$$y_i(t) = \mu(t) + X'_{i1}a_1(t) + X'_{i2}a_2(t) + (X'_{i1} \otimes X'_{i2})a_{1,2}(t) + \epsilon_i(t),$$

where $a_j(t) = (a_{j2}(t), \ldots, a_{jl_j}(t))'$ are the treatment effect functions of the jth factor with dimension $l_j - 1$, $a_{j_1,j_2}(t)$ are the pairwise interaction effect functions between factor j_1 and j_2 with dimension $(l_{j_1} - 1)(l_{j_2} - 1)$, and $\epsilon_i(t)$s are temporal dependent random processes with zero means. We have implemented this multi-factor model ($q \geq 2$) in our package such that researchers can specify the main and interactions effects as needed.

With the model, we can estimate the temporal varying coefficient functions. First, we approximate all of the coefficient functions with rank K splines expansions. For example, $a_{1k}(t) = \sum_{v=1}^{K} \beta_{a_{jk},v} B_{d,v}(t)$, where $B_{d,v}(t)$ is an order d B-spline basis function, and $\beta_{a_1,l}$ is the corresponding coefficient. Then, the problem is reduced to estimate all those coefficients βs in front of basis functions. We apply the least squares estimation with penalizations on the $d1th$ derivatives of the rank K spline expansion functions to get rid of the overfitting problem. Denoting $y_i(t_{i,z})$ as the measurement for the ith ($i = 1, ..., n$) plant being observed at the zth ($z = 1, ..., m$) time point, we minimize the penalized error sum of squares:

$$\sum_{i=1}^{n}\sum_{z=1}^{m}\{y_i(t_{i,z}) - B(t_{i,z})'\beta_\mu - X'_{i,1}B(t_{i,z})'\beta_1 - X'_{i,2}B(t_{i,z})'\beta_2) - (X'_{i1} \otimes X'_{i2})B(t_{i,z})'\beta_{1,2})\}^2$$

$$+\lambda\beta'_\mu\Omega\beta_\mu + \lambda\beta'_1\Omega\beta_1 + \lambda\beta'_2\Omega\beta_2 + \lambda\beta'_{1,2}\Omega\beta_{1,2}$$

where λ is the smoothing parameter, and $\Omega = \int B^{(d1)}(t)\{B^{(d1)}(t)\}'dt$ is a penalty matrix. Here $B(t) = (B_{d,1}(t), \ldots, B_{d,K}(t))'$, and d1 is the corresponding derivitives. The smoothing parameter λ is estimated by generalized cross validation (GCV). By plugging it back to the rank K splines expansions, we get the estimated varying coefficient functions in the model. e.g., $\hat{\beta}_\mu(t) = B(t)'\hat{\beta}_\mu, \hat{a}_1(t) = B(t)'\hat{\beta}_1, \ldots$.

Value

est_fun — a t by u matrix of the estimated function of coefficients, where t is the number of observation time points, and u is the number of columns of the design matrix. The (z, j_l) element represents the estimated value of the j_lth coefficient at the time z (i.e., $\hat{a}_{j_l}(t_z)$) in the following example:

For example, if the model includes two categorical variables without interaction terms, each categorical variable contains two levels, the model can be expressed as

$$y(t) = \beta_0(t) + \beta_1(t)X_1 + \beta_2(t)X_2 + \epsilon(t).$$

In this case, $u = 3$. This means your output "est_fun" contains 3 columns, and the columns repesent $\hat{\beta}_0(t)$, $\hat{\beta}_1(t)$ and $\hat{\beta}_2(t)$, respectively.

`bhat` a $(u \times K) \times 1$ numeric vector, containing all the estimated parameters (i.e., $\hat{\beta}$, where u is the number of columns of the design matrix, and K is the rank of the spline expansion.

`design_mat` The design matrix under the model specified in the function. It can be used as reference to help users identify the contrast vector L in the function CI( ), and the values of j1 and j2 in the function CI_contrast( ).

`lambda` The penalty parameter, obatined by GCV.

Note

The rank of the B-spline basis functions, K, is equal to the order (degree+1) of the spline plus the number of interior knots. To avoid over-fitting, we choose the rank less than half of the number of observation time points, m.

References

Ramsay, James O., and Silverman, Bernard W. (2005), Functional Data Analysis, 2nd ed., Springer, New York.

Ramsay, James O., Hooker, Giles, and Graves, Spencer (2009) Functional Data Analysis in R and Matlab, Springer, New York.

Examples

```
#load the data
data_new = read.csv(system.file("extdata", "data.csv",
                                package = "implant", mustWork = TRUE))
#The first three columns of data_new refer to the original position of
#the observations from the original dataset, genotype and block, respectively
Y = data_new[, -c(1:3)]
#This step is to factorize each factor
Genotype = as.factor(data_new$Genotype)
Block = as.factor(data_new$Block)
X = data.frame(Genotype, Block)
formula = "~ Genotype + Block"
tt = seq(from = 0, to = 44, by = 2)
fit = fanova(Y.na.mat = Y, X = X, tt = tt, formula, K.int = 6, order = 4)
fit$lambda
fit$est_fun
```

HMRF *Image Segmentation using Hidden Markov Random Field and EM Algorithm Framework*

Description

This function can be used to obtain the segmented image using HMRF-EM.

Usage

```
HMRF(X, Y, Z, em_iter, map_iter, beta = 2,
        epsilon_em = 0.00001, epsilon_map = 0.00001)
```

Arguments

X	an m by n binary matrix of the inital labels for an image, which can be obtained using initital segmentation methods, such as K-means or thresholding methods. Note that X could be any binary matrix, say its element could be 0 & 1, or 1 & 2, or 2 & 3 ,..., etc.
Y	an m by n matrix of pixel intensity. For plant segmentation, we recommend to use relative green.
Z	an m by n binary matrix, giving an estimate for the object edges in Y. We can obtain Z using the Canny edge detector: Z = t(cannEdges (Y) [, , 1, 1]) from the package: imager. See the example for details.
em_iter	a positive integer, which is the number of iteration steps of the EM Algorithm.
map_iter	a positive integer, which is the number of iteration steps of calculating MAP (the maximum a posterior estimation).
beta	The clique potential parameter for neighbourhood dependence. In default, beta = 2. See details in the supplementary file on the HMRF Model. This beta is equivalent to the Psi in the supplentary file (see page 20, 21).
epsilon_em	a small positive number, which is the convergence criterion of the EM Algorithm.
epsilon_map	a small positive number, which is the convergence criterion of MAP (maximum a posterior estimation).

Details

1. More detailed explanation about this method can be found in the supplymentary file:

https://github.com/rwang14/implant/blob/master/vignettes/HMRF_EM.pdf

2. The arguement Z can be obatined by CannyEdge detector using function cannEdges( ) from the package: imager. However, since this package needs to involve Rcpp and other dependent packages which may increase installation complexity of our package, we recommend the users to install the package: imager by themselves if needed.

Value

image_matrix	A matrix giving labels for the segmented image.

Note

This function is modified based on the matlab code written by Quan Wang (see reference).

References

Wang, Quan (2012), "Hmrf-em-image: implementation of the hidden markov random field model and its expectation-maximization algorithm."arXivpreprintarXiv:1207.3510

See Also

image_kmeans

Examples

```
library(implant)
library(png)
orig = readPNG(system.file("extdata", "reduced.png", package = "implant", mustWork = TRUE))
#Define the response as relative green.
Y = orig[ , , 2]/(orig[ , , 1] + orig[ , , 2] + orig[ , , 3])
#Z is a matrix obtained by CannyEdge detector
Z = readPNG(system.file("extdata", "Z.png",
                        package = "implant", mustWork = TRUE))
##Note: Users can obtain Z using the package "imager" and the function
#CannyEdges( ) for different images
#Z = t(cannyEdges(orig)[ , , 1, 1])
#Take the initial label of EM algorithm using K-means
X = image_kmeans(Y, k = 2)$X
#Obtain the image produced by kmeans clustering
output = matrix(as.numeric(X), nrow = nrow(X), ncol = ncol(X)) - 1
writePNG(output,"~/kmeans.png")
#Run the HMRF Model. Note that it may take a lot of time ...
img = HMRF(X, Y, Z, em_iter = 20, map_iter = 20, beta = 2,
              epsilon_em = 0.00001, epsilon_map = 0.00001)
#Obtain the matrix of the segmented image
image = img$image_matrix
#Morphological Operations
imageD = dilation(image)
imageDE = erosion(imageD)
imageDEE = erosion(imageDE)
imageDEED = dilation(imageDEE)
writePNG("~/HMRF.png")
```

---

imageBinary *Segmentation and Binarization*

Description

This function uses the Double-Criterion Thresholding method (DCT) to segment the object of study from the background of an image, and tranform the image to a binary image, i.e., a black and white image.

Usage

```
imageBinary(image, weight = c(-1, 2, -1), threshold1 = 30 / 255,
threshold2 = 0.075)
```

Arguments

image	an array of pixels of the image for processing.
weight	a 3 by 1 vector. The three elements indicate the weight of the pixel intensities of R,G,B, respectively. In default, it takes the value of c(-1,2,-1), which helps to construct a relative green ratio.
threshold1,threshold2	Values between 0 and 1. threshold1 is applied to the sum of the RGB intensities. threshold2 is applied on the contrast intensity by the specified weight in the function.

Details

The plant pixels are classified as the intersection of sum of RGB intensities larger than threshold 1 and green intensities larger than threshold 2.

Value

A matrix of the processed image.

Examples

```
#read an RGB image
library(png)
image = readPNG(system.file("extdata", "image1.png",
                            package = "implant",
                            mustWork = TRUE))[,,c(1:3)]
imageB = imageBinary(image,  weight = c(-1, 2, -1),
threshold1 = 30 / 255, threshold2 = 0.05)
```

---

image_kmeans *Obtain the Matrix of the Segmented Image using K-means Clustering.*

Description

This function is to obtain a segmented plant image using K-means Clustering Method, together with the means and standard deviations of the pixel intensities for different classes.

Usage

```
image_kmeans(Y,k)
```

Arguments

Y	an input matrix. For plant segmentation, we recommend to use the relative green intensity of the image.
k	a positive integer, which refers to the cluster number. In default, k = 2 to seperate the plant from the background.

Value

X	a matrix of the class label of the segmented image.
mu	a k by 1 matrix. Each row represents the sample mean of each cluster.
sigma	a k by 1 matrix. Each row represents the sample standard deviation of each cluster.

Examples

```
library(png)
orig = readPNG(system.file("extdata", "reduced.png", package = "implant", mustWork = TRUE))
#Define the response as relative green.
Y = orig[ , , 2]/(orig[ , , 1]+orig[ , , 2]+orig[ , , 3])
#Take the initial label of EM algorithm using K-means
X = image_kmeans(Y, k = 2)$X
#Obtain the image produced by kmeans clustering
output = matrix(as.numeric(X), nrow = nrow(X), ncol = ncol(X)) - 1
writePNG(output,"~/kmeans.png")
```

Index

Average Reducing *(*downsize_avg*)*, 10

Background Exchange *(*Color_Exchange*)*, 8
Binarization *(*imageBinary*)*, 18

CI, 2
CI_contrast, 4
Color2Gray, 5
Color_Exchange, 8
ColorB, 6
ColorG, 7

Dilation *(*dilation*)*, 9
dilation, 9
downsize, 9, *14*
downsize_avg, 10
downsize_matrix, 11

Erosion *(*erosion*)*, 12
erosion, 12
extract_pheno, 13

fanova, *3*, *5*, 14

HMRF, 17

image_kmeans, *18*, 19
imageBinary, 18