Package 'FRESHD'

October 12, 2022

Type Package

Title Fast Robust Estimation of Signals in Heterogeneous Data

Version 1.0

Date 2022-05-09

Maintainer Adam Lund <adam.lund@math.ku.dk>

Description Procedure for solving the maximin problem for identical design across heterogeneous data groups. Particularly efficient when the design matrix is either orthogonal or has tensor structure. Orthogonal wavelets can be specified for 1d, 2d or 3d data simply by name. For tensor structured design the tensor components (two or three) may be supplied. The package also provides an efficient implementation of the generic magging estimator.

Imports Rcpp (>= 0.12.12), glamlasso

License GPL

LinkingTo Rcpp, RcppArmadillo, RcppEigen

RoxygenNote 7.1.2

NeedsCompilation yes

Author Adam Lund [aut, cre, ctb, cph],
Benjamin Stephens [ctb, cph],
Gael Guennebaud [ctb, cph],
Angelo Furfaro [ctb, cph],
Luca Di Gaspero [ctb, cph],
Brandon Whitcher [ctb, cph]

Repository CRAN

Date/Publication 2022-05-12 08:00:06 UTC

R topics documented:

`iwt` *Inverse discrete wavelet transform*

Description

This function performs a level J decomposition of the input array (1d, 2d, or 3d) using the pyramid algorithm (Mallat 1989).

Usage

```
iwt(x, wf = "la8", J = NULL)
```

Arguments

`x`	a 1, 2, or 3 dimensional data array. The size of each dimension must be dyadic.
`wf`	the type of wavelet family used. See R-package waveslim for options.
`J`	is the level (depth) of the decomposition. For default `NULL` the max depth is used making `iwt(x)` equal to multiplying `x` with the inverse of corresponding wavelet matrix.

Details

This is a C++/R wrapper function for a C implementation of the inverse discrete wavelet transform by Brandon Whitcher licensed under the BSD 3 license https://cran.r-project.org/web/licenses/BSD_3_clause, see the Waveslim package; Percival and Walden (2000); Gencay, Selcuk and Whitcher (2001).

Given a data array (1d, 2d or 3d) with dyadic dimensions sizes this transform is computed efficiently via the pyramid algorithm see Mallat (1989).

Value

...	An array with dimensions equal to those of `x`.

Author(s)

Adam Lund, Brandon Whitcher

References

Gencay, R., F. Selcuk and B. Whitcher (2001) An Introduction to Wavelets and Other Filtering Methods in Finance and Economics, Academic Press.

Mallat, S. G. (1989) A theory for multiresolution signal decomposition: the wavelet representation, IEEE Transactions on Pattern Analysis and Machine Intelligence, 11, No. 7, 674-693.

Percival, D. B. and A. T. Walden (2000) Wavelet Methods for Time Series Analysis, Cambridge University Press.

Examples

```
###1d
x <- as.matrix(rnorm(2^3))
range(x - iwt(wt(x)))

###2d
x <- matrix(rnorm(2^(3 + 4)), 2^3, 2^4)
range(x - iwt(wt(x)))

###3d
x <- array(rnorm(2^(3 + 4 + 5)), c(2^3, 2^4, 2^5))
range(x - iwt(wt(x)))
```

`magging` *Maximin Aggregation*

Description

R wrapper for a C++ implementation of the generic maximin aggregation procedure.

Usage

```
magging(B)
```

Arguments

`B`	array of size $p \times G$ containing the group parameter estimates where p is the number of model parameters and G is the number of groups.

Details

Following *Buhlmann 2016* this function computes the maximin aggregation estimator for given group estimates. This entails solving a convex quadratic optimization problem. The function wraps a C++ implementation of an algorithm by Goldfarb and Idnani for solving a (convex) quadratic programming problem by means of a dual method.

The underlying C++ program solving the convex quadratic optimization problem, eiquadprog.hpp, copyright (2011) Benjamin Stephens, GPL v2 see https://www.cs.cmu.edu/~bstephe1/eiquadprog.hpp, is based on previous libraries:

QuadProg++, Copyright (C) 2007-2016 Luca Di Gaspero, MIT License. See https://github.com/liuq/QuadProgpp

uQuadProg, Copyright (C) 2006 - 2017 Angelo Furfaro, LGPL v3, a port of QuadProg++ working with ublas data structures. See https://github.com/fx74/uQuadProg/blob/master/README.md

QuadProg Copyright (C) 2014-2015 Gael Guennebaud, LGPL v3, a modification of uQuadProg, working with Eigen data structures. See http://www.labri.fr/perso/guenneba/code/QuadProg/.

Value

An object with S3 Class "FRESHD".

... A p vector containing the maximin aggregated parameter estimates.

Author(s)

Adam Lund, Benjamin Stephens, Gael Guennebaud, Angelo Furfaro, Luca Di Gaspero

Maintainer: Adam Lund, <adam.lund@math.ku.dk>

References

Buhlmann, Peter and Meinshausen, Nicolai (2016). Magging: maximin aggregation for inhomogeneous large-scale data. Proceedings of the IEEE, 1, 104, 126-135

D. Goldfarb, A. Idnani. A numerically stable dual method for solving strictly convex quadratic programs (1983). Mathematical Programming, 27, 1-33.

Examples

```
##size of example
set.seed(42)
G <- 15; n <- c(50, 20, 13); p <- c(7, 5, 4)
nlambda <- 10

##marginal design matrices (Kronecker components)
x <- list()
for(i in 1:length(n)){
x[[i]] <- matrix(rnorm(n[i] * p[i], 0, 1), n[i], p[i])
}

##common features and effects
common_features <- rbinom(prod(p), 1, 0.1)
common_effects <- rnorm(prod(p), 0, 1) * common_features
system.time({
##group response and fit
lambda <- exp(seq(-1, -4, length.out = nlambda))
magbeta <- matrix(0, prod(p), nlambda)
B <- array(NA, c(prod(p), G, nlambda))
y <- array(NA, c(n, G))
for(g in 1:G){
bg <- rnorm(prod(p), 0, 0.1) * (1 - common_features) + common_effects
Bg <- array(bg, p)
mu <- RH(x[[3]], RH(x[[2]], RH(x[[1]], Bg)))
y[,,, g] <- array(rnorm(prod(n)), dim = n) + mu
B[, g, ] <- glamlasso::glamlasso(x, y[,,, g], lambda = lambda)$coef
}
})

##maximin aggregation for all lambdas (models)
for(l in 1:dim(B)[3]){
magbeta[, l] <- magging(B[, , l])
```

```
}

##estimated common effects for specific lambda
modelno <- 10
betafit <- magbeta[, modelno]
plot(common_effects, type = "h", ylim = range(betafit, common_effects), col = "red")
lines(betafit, type = "h")
```

`maximin` *Maximin signal estimation*

Description

Efficient procedure for solving the maximin estimation problem with identical design across groups, see (Lund, 2022).

Usage

```
maximin(y,
        x,
        penalty = "lasso",
        alg ="aradmm",
        kappa = 0.99,
        nlambda = 30,
        lambda_min_ratio = 1e-04,
        lambda = NULL,
        penalty_factor = NULL,
        standardize = TRUE,
        tol = 1e-05,
        maxiter = 1000,
        delta = 1,
        gamma = 1,
        eta = 0.1,
        aux_par = NULL)
```

Arguments

`y`	Array of size $n_1 \times \cdots \times n_d \times G$ containing the response values.
`x`	Either i) the design matrix, ii) a list containing the Kronecker components (2 or 3) if the design matrix has Kronecker structure or iii) a string indicating the name of the wavelet to use (see `wt` for options)
`penalty`	string specifying the penalty type. Possible values are `"lasso"`.
`alg`	string specifying the optimization algorithm. Possible values are `"admm"`, `"aradmm"`, `"tos"`, `"tosacc"`.
`kappa`	Strictly positive float controlling the maximum sparsity in the solution. Only used with ADMM type algorithms. Should be between 0 and 1.

`nlambda`	Positive integer giving the number of `lambda` values. Used when lambda is not specified.
`lambda_min_ratio`	strictly positive float giving the smallest value for `lambda`, as a fraction of λ_{max}; the (data dependent) smallest value for which all coefficients are zero. Used when lambda is not specified.
`lambda`	Sequence of strictly positive floats used as penalty parameters.
`penalty_factor`	A vector of length p containing positive floats that are multiplied with each element in `lambda` to allow differential penalization on the coefficients. For tensor models an array of size $p_1 \times \cdots \times p_d$.
`standardize`	Boolean indicating if response `y` should be scaled. Default is TRUE to avoid numerical problems.
`tol`	Strictly positive float controlling the convergence tolerance.
`maxiter`	Positive integer giving the maximum number of iterations allowed for each `lambda` value.
`delta`	Positive float controlling the step size in the algorithm.
`gamma`	Positive float controlling the relaxation parameter in the algorithm. Should be between 0 and 2.
`eta`	Scaling parameter for the step size in the accelerated TOS algorithm. Should be between 0 and 1.
`aux_par`	Auxiliary parameters for the algorithms.

Details

For n heterogeneous data points divided into G equal sized groups with $m < n$ data points in each, let $y_g = (y_{g,1}, \ldots, y_{g,m})$ denote the vector of observations in group g. For a $m \times p$ design matrix X consider the model

$$y_g = Xb_g + \epsilon_g$$

for b_g a random group specific coefficient vector and ϵ_g an error term, see Meinshausen and Buhlmann (2015). For the model above following Lund (2022) this package solves the maximin estimation problem

$$\min_{\beta} -\hat{V}_g(\beta)) + \lambda\|\beta\|_1, \lambda \geq 0$$

where

$$\hat{V}_g(\beta) := \frac{1}{n}(2\beta^\top X^\top y_g - \beta^\top X^\top X\beta),$$

is the empirical explained variance in group g. See *Lund, 2022* for more details and references.

The package solves the problem using different algorithms depending on X:

i) If X is orthogonal (e.g. the inverse wavelet transform) either an ADMM algorithm (standard or relaxed) or an adaptive relaxed ADMM (ARADMM) with auto tuned step size is used, see Xu et al (2017).

ii) For general X, a three operator splitting (TOS) algorithm is implemented, see Damek and Yin (2017). Note if the design is tensor structured, $X = \bigotimes_{i=1}^{d} X_i$ for $d \in \{1, 2, 3\}$, the procedure accepts a list containing the tensor components (matrices).

Value

An object with S3 Class "FRESHD".

spec	A string indicating the array dimension (1, 2 or 3) and the penalty.
coef	A $p_1 \cdots p_d \times$ nlambda matrix containing the estimates of the model coefficients (beta) for each lambda-value.
lambda	A vector containing the sequence of penalty values used in the estimation procedure.
df	The number of nonzero coefficients for each value of lambda.
dimcoef	A vector giving the dimension of the model coefficient array β.
dimobs	A vector giving the dimension of the observation (response) array Y.
dim	Integer indicating the dimension of of the array model. Equal to 1 for non array.
wf	A string indicating the wavelet name if used.
diagnostics	A list where item iter is a vector containing the number of iterations for each lambda value for which the algorithm converged. Item stop_maxiter is 1 if maximum iterations is reached otherwise zero. Item stop_sparse is 1 if maximum sparsity is reached otherwise zero.

Author(s)

Adam Lund

Maintainer: Adam Lund, <adam.lund@math.ku.dk>

References

Lund, Adam (2022). Fast Robust Signal Estimation for Heterogeneous data. *In preparation*.

Meinshausen, N and P. Buhlmann (2015). Maximin effects in inhomogeneous large-scale data. *The Annals of Statistics*. 43, 4, 1801-1830.

Davis, Damek and Yin, Wotao, (2017). A three-operator splitting scheme and its optimization applications. *Set-valued and variational analysis*. 25, 4, 829-858.

Xu, Zheng and Figueiredo, Mario AT and Yuan, Xiaoming and Studer, Christoph and Goldstein, Tom (2017). Adaptive relaxed admm: Convergence theory and practical implementation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* 7389-7398.

Examples

```
## general 3d tensor design matrix
set.seed(42)
G <- 20; n <- c(65, 26, 13)*3; p <- c(13, 5, 4)*3
sigma <- 1

##marginal design matrices (Kronecker components)
x <- list()
for(i in 1:length(n)){x[[i]] <- matrix(rnorm(n[i] * p[i], 0, sigma), n[i], p[i])}

##common features and effects
```

```
common_features <- rbinom(prod(p), 1, 0.1)
common_effects <- rnorm(prod(p), 0, 0.1) * common_features

##simulate group response
y <- array(NA, c(n, G))
for(g in 1:G){
bg <- rnorm(prod(p), 0, 0.1) * (1 - common_features) + common_effects
Bg <- array(bg, p)
mu <- RH(x[[3]], RH(x[[2]], RH(x[[1]], Bg)))
y[,,, g] <- array(rnorm(prod(n), 0, var(mu)), dim = n) + mu
}

##fit model for range of lambda
system.time(fit <- maximin(y, x, penalty = "lasso", alg = "tosacc"))

##estimated common effects for specific lambda
modelno <- 10
betafit <- fit$coef[, modelno]
plot(common_effects, type = "h", ylim = range(betafit, common_effects), col = "red")
lines(betafit, type = "h")

##size of example
set.seed(42)
G <- 50; p <- n <- c(2^6, 2^5, 2^6);

##common features and effects
common_features <- rbinom(prod(p), 1, 0.1) #sparsity of comm. feat.
common_effects <- rnorm(prod(p), 0, 1) * common_features

##group response
y <- array(NA, c(n, G))
for(g in 1:G){
bg <- rnorm(prod(p), 0, 0.1) * (1 - common_features) + common_effects
Bg <- array(bg, p)
mu <- iwt(Bg)
y[,,, g] <- array(rnorm(prod(n),0, 0.5), dim = n) + mu
}

##orthogonal wavelet design with 1d data
G = 50; N1 = 2^10; p = 101; J = 2; amp = 20; sigma2 = 10
y <- matrix(0, N1, G)
z <- seq(0, 2, length.out = N1)
sig <- cos(10 * pi * z) + 1.5 * sin(5 * pi * z)

for (i in 1:G){
freqs <- sample(1:100, size = J, replace = TRUE)
y[, i] <- sig * 2 + rnorm(N1, sd = sqrt(sigma2))
for (j in 1:J){
y[, i] <- y[, i] + amp * sin(freqs[j] * pi * z + runif(1, -pi, pi))
}
}

system.time(fit <- maximin(y, "la8", alg = "aradmm", kappa = 0.9))
```

```
mmy <- predict(fit, "la8")
plot(mmy[,1], type = "l")
lines(sig, col = "red")
```

---

predict.FRESHD *Make Prediction From a FRESHD Object*

Description

Given covariate data this function computes the linear predictors based on the estimated model coefficients in an object produced by the function `maximin` or `magging`. Note that the data can be supplied in two different formats: i) for wavelet based models as a string indicating the wavelet used to produce the model object. ii) for models with custom design as a list of one, two or three Kronecker component matrices each of size $n_i' \times p_i, i = 1, 2, 3$. Note `x` will typically be the original design (covariate data) that was used to produce `object` using `maximin` or `magging` so n_i' is the number of marginal data points in the ith dimension i.e. $n_i' = n_i$.

Usage

```
## S3 method for class 'FRESHD'
predict(object, x, ...)
```

Arguments

`object`	An object of class FRESHD, produced with `maximin` or `magging`.
`x`	An object that should be like the input to the call that produced `object`. For models with custom design a list like the one supplied to produce `object` and for a wavelet design the name of the wavelet used to produce `object`.
`...`	ignored.

Value

If `x` is a string indicating a wavelet an array of the same size as the input data used to produce `object`. Otherwise an array of size $n_1' \times \cdots \times n_d'$, with $d \in \{1, 2, 3\}$.

Author(s)

Adam Lund

Examples

```
##size of example
set.seed(42)
G = 50; N1 = 2^10; p = 101; J = 3; amp = 20; sigma2 = 10
y <- matrix(0, N1, G)
z <- seq(0, 2, length.out = N1)
sig <- cos(10 * pi * z) + 1.5 * sin(5 * pi * z)
```

```
for (i in 1:G){
freqs <- sample(1:100, size = J, replace = TRUE)
y[, i] <- sig * 2 + rnorm(N1, sd = sqrt(sigma2))
for (j in 1:J){
y[, i] <- y[, i] + amp * sin(freqs[j] * pi * z + runif(1, -pi, pi))
}
}
system.time(fitmm <- maximin(y, "la8", alg = "aradmm", kappa = 0.95))
mmy <- predict(fitmm, "la8")
plot(mmy[, 2], type = "l")
lines(sig, col = "red")
```

`print.FRESHD` *Print Function for objects of Class FRESHD*

Description

This function will print some information about the FRESHD object.

Usage

```
## S3 method for class 'FRESHD'
print(x, ...)
```

Arguments

x	a FRESHD object
...	ignored

Details

A three-column data.frame with columns 'sparsity', 'Df' and 'lambda'. The 'Df' column is the number of nonzero coefficients and 'sparsity' is the percentage of zeros in the solution.

Value

The data.frame above is silently returned

Author(s)

Adam Lund

Examples

```
##size of example
set.seed(42)
G <- 50; n <- c(65, 26, 13); p <- c(13, 5, 4)
sigma <-0.1
nlambda =30
##marginal design matrices (Kronecker components)
x <- list()
for(i in 1:length(n)){x[[i]] <- matrix(rnorm(n[i] * p[i],0,sigma), n[i], p[i])}

##common features and effects
common_features <- rbinom(prod(p), 1, 0.1)
common_effects <- rnorm(prod(p), 0, 0.1) * common_features

##group response and fit
lambda <- exp(seq(0, -5, length.out = nlambda))
B <- array(NA, c(prod(p), nlambda, G))
y <- array(NA, c(n, G))
for(g in 1:G){
bg <- rnorm(prod(p), 0, 0.1) * (1 - common_features) + common_effects
Bg <- array(bg, p)
mu <- RH(x[[3]], RH(x[[2]], RH(x[[1]], Bg)))
y[,,, g] <- array(rnorm(prod(n), 0, var(mu)), dim = n) + mu
}

##fit model for range of lambda
system.time(fit <- maximin(y, x, penalty = "lasso", alg = "tos"))
Betahat <- fit$coef

##estimated common effects for specific lambda
modelno <- 20;
m <- min(Betahat[, modelno], common_effects)
M <- max(Betahat[, modelno], common_effects)
plot(common_effects, type = "h", ylim = c(m, M), col = "red")
lines(Betahat[, modelno], type = "h")
```

RH *The Rotated H-transform of a 3d Array by a Matrix*

Description

This function is an implementation of the ρ-operator found in *Currie et al 2006.* It forms the basis of the GLAM arithmetic.

Usage

```
RH(M, A)
```

Arguments

M	a $n \times p_1$ matrix.
A	a 3d array of size $p_1 \times p_2 \times p_3$.

Details

For details see *Currie et al 2006.* Note that this particular implementation is not used in the routines underlying the optimization procedure.

Value

A 3d array of size $p_2 \times p_3 \times n$.

Author(s)

Adam Lund

References

Currie, I. D., M. Durban, and P. H. C. Eilers (2006). Generalized linear array models with applications to multidimensional smoothing. *Journal of the Royal Statistical Society. Series B*. 68, 259-280. url = http://dx.doi.org/10.1111/j.1467-9868.2006.00543.x.

Examples

```
n1 <- 65; n2 <- 26; n3 <- 13; p1 <- 13; p2 <- 5; p3 <- 4

##marginal design matrices (Kronecker components)
X1 <- matrix(rnorm(n1 * p1), n1, p1)
X2 <- matrix(rnorm(n2 * p2), n2, p2)
X3 <- matrix(rnorm(n3 * p3), n3, p3)

Beta <- array(rnorm(p1 * p2 * p3, 0, 1), c(p1 , p2, p3))
max(abs(c(RH(X3, RH(X2, RH(X1, Beta)))) - kronecker(X3, kronecker(X2, X1)) %*% c(Beta)))
```

---

wt *Discrete wavelet transform*

Description

This function performs a level J wavelet transform of the input array (1d, 2d, or 3d) using the pyramid algorithm (Mallat 1989).

Usage

```
wt(x, wf = "la8", J = NULL)
```

Arguments

x	a 1, 2, or 3 dimensional data array. The size of each dimension must be dyadic.
wf	the type of wavelet family used. See R-package waveslim for options.
J	is the level (depth) of the decomposition. For default NULL the max depth is used making wt(x) equal to multiplying x with the corresponding wavelet matrix.

Details

This is a C++/R wrapper function for a C implementation of the discrete wavelet transform by Brandon Whitcher licensed under the BSD 3 license https://cran.r-project.org/web/licenses/BSD_3_clause, see the Waveslim package; Percival and Walden (2000); Gencay, Selcuk and Whitcher (2001).

Given a data array (1d, 2d or 3d) with dyadic sizes this transform is computed efficiently via the pyramid algorithm see Mallat (1989).

Value

...	An array with dimensions equal to those of x.

Author(s)

Adam Lund, Brandon Whitcher

References

Gencay, R., F. Selcuk and B. Whitcher (2001) An Introduction to Wavelets and Other Filtering Methods in Finance and Economics, Academic Press.

Mallat, S. G. (1989) A theory for multiresolution signal decomposition: the wavelet representation, IEEE Transactions on Pattern Analysis and Machine Intelligence, 11, No. 7, 674-693.

Percival, D. B. and A. T. Walden (2000) Wavelet Methods for Time Series Analysis, Cambridge University Press.

Examples

```
###1d
x <- as.matrix(rnorm(2^3))
range(x - iwt(wt(x)))

###2d
x <- matrix(rnorm(2^(3 + 4)), 2^3, 2^4)
range(x - iwt(wt(x)))

###3d
x <- array(rnorm(2^(3 + 4 + 5)), c(2^3, 2^4, 2^5))
range(x - iwt(wt(x)))
```

Index