Package 'SMME'

January 20, 2025

Type Package

Title Soft Maximin Estimation for Large Scale Heterogeneous Data

Version 1.1.1

Date 2023-01-02

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

Description Efficient procedure for solving the soft maximin problem for large scale heterogeneous data, see Lund, Mogensen and Hansen (2022) <doi:10.1111/sjos.12580>. Currently Lasso and SCAD penalized estimation is implemented. Note this package subsumes and replaces the SMMA package.

License MIT + file LICENSE

Imports Rcpp (>= 0.12.12)

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.1.2

NeedsCompilation yes

Author Adam Lund [aut, cre]

Repository CRAN

Date/Publication 2023-01-08 10:30:02 UTC

Contents

iwt 2
predict.SMME 3
print.SMME 4
RH 5
SMME 6
wt 12

Index **14**

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, Rigorous Analytics Ltd, 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).

This functionality is used in the computations underlying `softmaximin` to perform multiplications involving the wavelet (design) matrix efficiently.

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)))
```

`predict.SMME` *Make Prediction From a SMME Object*

Description

Given new covariate data this function computes the linear predictors based on the estimated model coefficients in an object produced by the function `softmaximin`. Note that the data can be supplied in three different formats: i) for general models as a $n' \times p$ matrix (p is the number of model coefficients and n' is the number of new data points), ii) for array models with custom design as a list of one, two or three Kronecker component matrices each of size $n'_i \times p_i, i = 1, 2, 3$ (n'_i is the number of new marginal data points in the ith dimension), iii) for wavelet based models a string indicating the wavelet used to produce the model object.

Usage

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

Arguments

`object`	An object of class SMME, produced with `softmaximin` with m_ζ fitted models for each value of `zeta`.
`x`	An object that should be like the input to the `softmaximin` call that produced `object`. For general models a matrix with column dimension equal to that of the original input.For array models with custom design a list like the one supplied to `softmaximin` to produce `object` and for a wavelet design the name of the wavelet used to produce `object`.
`...`	ignored.

Value

A list of length `length(zeta)`. If `x` is a $n' \times p$ matrix each list item is a $n' \times m_\zeta$ matrix containing the linear predictors computed for each `lambda`. If `x` is a string or a list of tensor component matrices and `fit$dim = d`, each list item is a $d + 1$ array containing predictions computed for each `lambda`.

Author(s)

Adam Lund

Examples

```
##size of example
n1 <- 65; n2 <- 26; n3 <- 13; p1 <- 13; p2 <- 5; p3 <- 4

##marginal design matrices (Kronecker components)
X1 <- matrix(rnorm(n1 * p1, 0, 0.5), n1, p1)
X2 <- matrix(rnorm(n2 * p2, 0, 0.5), n2, p2)
X3 <- matrix(rnorm(n3 * p3, 0, 0.5), n3, p3)
X <- list(X1, X2, X3)

component <- rbinom(p1 * p2 * p3, 1, 0.1)
Beta1 <- array(rnorm(p1 * p2 * p3, 0, 0.1) + component, c(p1 , p2, p3))
Beta2 <- array(rnorm(p1 * p2 * p3, 0, 0.1) + component, c(p1 , p2, p3))
mu1 <- RH(X3, RH(X2, RH(X1, Beta1)))
mu2 <- RH(X3, RH(X2, RH(X1, Beta2)))
Y1 <- array(rnorm(n1 * n2 * n3, mu1), dim = c(n1, n2, n3))
Y2 <- array(rnorm(n1 * n2 * n3, mu2), dim = c(n1, n2, n3))

Y <- array(NA, c(dim(Y1), 2))
Y[,,, 1] <- Y1; Y[,,, 2] <- Y2;

fit <- softmaximin(X, Y, zeta = c(1, 10), penalty = "lasso", alg = "npg")

##new data in tensor component form
X1 <- matrix(rnorm(2 * p1), nrow = 2)
X2 <- matrix(rnorm(3 * p2), nrow = 3)
X3 <- matrix(rnorm(4 * p3), nrow = 4)
Yhat <- predict(fit, x = list(X1, X2, X3))
```

---

print.SMME *Print Function for objects of Class SMME*

Description

This function will print some information about the SMME object.

Usage

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

Arguments

x	a SMME object
...	ignored

Author(s)

Adam Lund

Examples

```
##size of example
n1 <- 65; n2 <- 26; n3 <- 13; p1 <- 13; p2 <- 5; p3 <- 4

##marginal design matrices (Kronecker components)
X1 <- matrix(rnorm(n1 * p1, 0, 0.5), n1, p1)
X2 <- matrix(rnorm(n2 * p2, 0, 0.5), n2, p2)
X3 <- matrix(rnorm(n3 * p3, 0, 0.5), n3, p3)
X <- list(X1, X2, X3)

component <- rbinom(p1 * p2 * p3, 1, 0.1)
Beta1 <- array(rnorm(p1 * p2 * p3, 0, .1) + component, c(p1 , p2, p3))
Beta2 <- array(rnorm(p1 * p2 * p3, 0, .1) + component, c(p1 , p2, p3))
mu1 <- RH(X3, RH(X2, RH(X1, Beta1)))
mu2 <- RH(X3, RH(X2, RH(X1, Beta2)))
Y1 <- array(rnorm(n1 * n2 * n3, mu1), dim = c(n1, n2, n3))
Y2 <- array(rnorm(n1 * n2 * n3, mu2), dim = c(n1, n2, n3))

Y <- array(NA, c(dim(Y1), 2))
Y[,,, 1] <- Y1; Y[,,, 2] <- Y2;

fit <- softmaximin(X, Y, zeta = 10, penalty = "lasso", alg = "npg")
fit
```

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)))
```

SMME *Soft Maximin Estimation for Large Scale Heterogenous Data*

Description

Efficient procedure for solving the Lasso or SCAD penalized soft maximin problem for large scale_y data. This software implements two proximal gradient based algorithms (NPG and FISTA) to solve different forms of the soft maximin problem from *Lund et al., 2022*. 1) For general group specific design the soft maximin problem is solved using the NPG algorithm. 2) For fixed identical d-array-tensor design across groups, where $d = 1, 2, 3$, the estimation procedure uses either the FISTA algorithm or the NPG algorithm and is implemented for the following two cases; i) For a tensor design matrix the algorithms use array arithmetic to speed up design matrix multiplications using only the tensor components ii) For a wavelet design matrix the algorithms use the pyramid algorithm to completely avoid the design matrix and speed up design matrix multiplications. Multi-threading is possible when openMP is available for R.

Note this package SMME replaces the SMMA package.

Usage

```
softmaximin(x,
            y,
            zeta,
            penalty = c("lasso", "scad"),
            alg = c("npg", "fista"),
            nlambda = 30,
            lambda.min.ratio = 1e-04,
            lambda = NULL,
            scale_y = 1,
            penalty.factor = NULL,
            reltol = 1e-05,
            maxiter = 1000,
            steps = 1,
            btmax = 100,
            c = 0.0001,
            tau = 2,
            M = 4,
            nu = 1,
            Lmin = 0,
            lse = TRUE,
            nthreads = 2)
```

Arguments

`x`	Either a list containing the G group specific design matrices of sizes $n_i \times p_i$ (general model), a list containing the d ($d \in \{1, 2, 3\}$) tensor components (tensor array model) or a string indicating which wavelet design to use (wavelet array model), see wt for options.
`y`	list containing the G group specific response vectors of sizes $n_i \times 1$. Alternatively for a model with identical tensor design across G groups, `y`is an array of size $n_1 \times \cdots \times n_d \times G$ ($d \in \{1, 2, 3\}$) containing the response values.
`zeta`	vector of strictly positive floats controlling the softmaximin approximation accuracy. When `length(zeta)` > 1 the procedure will distribute the computations using the `nthreads` parameter below when openMP is available.
`penalty`	string specifying the penalty type. Possible values are `"lasso"`, `"scad"`.
`alg`	string specifying the optimization algorithm. Possible values are `"npg"`, `"fista"`.
`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`	A sequence of strictly positive floats used as penalty parameters.
`scale_y`	strictly positive number that the response `y` is multiplied with.

`penalty.factor`	a length p vector of positive floats that are multiplied with each element in `lambda` to allow differential penalization on the coefficients. For tensor models an array of size $p_1 \times \cdots \times p_d$.
`reltol`	strictly positive float giving the convergence tolerance.
`maxiter`	positive integer giving the maximum number of iterations allowed for each `lambda` value.
`steps`	strictly positive integer giving the number of steps used in the multi-step adaptive lasso algorithm for non-convex penalties. Automatically set to 1 when `penalty = "lasso"`.
`btmax`	strictly positive integer giving the maximum number of backtracking steps allowed in each iteration. Default is `btmax = 100`.
`c`	strictly positive float used in the NPG algorithm. Default is `c = 0.0001`.
`tau`	strictly positive float used to control the stepsize for NPG. Default is `tau = 2`.
`M`	positive integer giving the look back for the NPG. Default is `M = 4`.
`nu`	strictly positive float used to control the stepsize. A value less that 1 will decrease the stepsize and a value larger than one will increase it. Default is `nu = 1`.
`Lmin`	non-negative float used by the NPG algorithm to control the stepsize. For the default `Lmin = 0` the maximum step size is the same as for the FISTA algorithm.
`lse`	logical variable indicating whether to use the log-sum-exp-loss. TRUE is default and yields the loss below and FALSE yields the exponential of this.
`nthreads`	integer giving the number of threads to use when openMP is available. Default is 2.

Details

Consider modeling heterogeneous data $y_1, \ldots, y_n$ by dividing it into G groups $\mathbf{y}_g = (y_1, \ldots, y_{n_g})$, $g \in \{1, \ldots, G\}$ and then using a linear model

$$\mathbf{y}_g = \mathbf{X}_g b_g + \epsilon_g, \quad g \in \{1, \ldots, G\},$$

to model the group response. Then b_g is a group specific $p \times 1$ coefficient, $\mathbf{X}_g$ an $n_g \times p$ group design matrix and ϵ_g an $n_g \times 1$ error term. The objective is to estimate a common coefficient β such that $\mathbf{X}_g\beta$ is a robust and good approximation to $\mathbf{X}_g b_g$ across groups.

Following *Lund et al., 2022*, this objective may be accomplished by solving the soft maximin estimation problem

$$\min_\beta \frac{1}{\zeta} \log\left(\sum_{g=1}^{G} \exp(-\zeta \hat{V}_g(\beta))\right) + \lambda\|\beta\|_1, \quad \zeta > 0, \lambda \geq 0.$$

Here ζ essentially controls the amount of pooling across groups ($\zeta \sim 0$ effectively ignores grouping and pools observations) and

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

is the empirical explained variance, see *Lund et al., 2022* for more details and references.

The function `softmaximin` solves the soft maximin estimation problem in large scale settings for a sequence of penalty parameters $\lambda_{max} > \ldots > \lambda_{min} > 0$ and a sequence of strictly positive softmaximin parameters $\zeta_1, \zeta_2, \ldots$.

The implementation also solves the problem above with the penalty given by the SCAD penalty, using the multiple step adaptive lasso procedure to loop over the inner proximal algorithm.

Two optimization algorithms are implemented in the SMME packages; a non-monotone proximal gradient (NPG) algorithm and a fast iterative soft thresholding algorithm (FISTA).

The implementation is particularly efficient for models where the design is identical across groups i.e. $\mathbf{X}_g = \mathbf{X}\ \forall g \in \{1, \ldots, G\}$ in the following two cases: i) first if $\mathbf{X}$ has tensor structure i.e.

$$\mathbf{X} = \bigotimes_{i=1}^{d} \mathbf{M}_i$$

for marginal $n_i \times p_i$ design matrices $\mathbf{M}_1, \ldots, \mathbf{M}_d$, $d \in \{1, 2, 3\}$, y is a $d+1$ dimensional response array and `x` is a list containing the d marginal matrices $\mathbf{M}_1, \ldots, \mathbf{M}_d$. In this case `softmaximin` solves the soft maximin problem using minimal memory by way of tensor optimized arithmetic, see also `RH`. ii) second, if the design matrix $\mathbf{X}$ is the inverse matrix of an orthogonal wavelet transform `softmaximin` solves the soft maximin problem given the $d+1$ dimensional response array `y` and `x` the name of the wavelet family `wt`, using the pyramid algorithm to compute multiplications involving $\mathbf{X}$.

Note that when multiple values for ζ is provided it is possible to distribute the computations across CPUs if openMP is available.

Value

An object with S3 Class "SMME".

`spec`	A string indicating the array dimension (1, 2 or 3) and the penalty.
`coef`	A `length(zeta)`-list of $p\times$ `nlambda` matrices containing the estimates of the model coefficients (β) for each `lambda`-value for which the procedure converged.
`lambda`	A `length(zeta)`-list vectors containing the sequence of penalty values used in the estimation procedure for which the procedure converged.
`df`	A `length(zeta)`-list of vectors indicating the nonzero model coefficients for each value of `lambda` for which the procedure converged.
`dimcoef`	An integer giving the number p of model parameters. For array data a vector giving the dimension of the model coefficient array β.
`dimobs`	An integer giving the number of observations. For array data 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 of length 3. Item `iter` is a `length(zeta)`-list of vectors containing the number of iterations for each `lambda` value for which the algorithm converged. Item `bt_iter` is a `length(zeta)` vector with total number of backtracking steps

performed across all (converged) `lambda` values for given `zeta` value. Key `bt_enter` is a `length(zeta)` vector with total number of times backtracking is initiated across all (converged) `lambda` values for given `zeta` value.

`endmod`	Vector of length `length(zeta)` with the number of models fitted for each `zeta`.
`Stops`	Convergence indicators.

Author(s)

Adam Lund

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

References

Lund, A., S. W. Mogensen and N. R. Hansen (2022). Soft Maximin Estimation for Heterogeneous Data. *Scandinavian Journal of Statistics*, vol. 49, no. 4, pp. 1761-1790. url = https://doi.org/10.1111/sjos.12580

Examples

```
#Non-array data

##size of example
set.seed(42)
G <- 10; n <- sample(100:500, G); p <- 60
x <- y <- list()

##group design matrices
for(g in 1:G){x[[g]] <- matrix(rnorm(n[g] * p), n[g], p)}

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

##group response
for(g in 1:G){
bg <- rnorm(p, 0, 0.5) * (1 - common_features) + common_effects
mu <- x[[g]] %*% bg
y[[g]] <- rnorm(n[g]) + mu
}

##fit model for range of lambda and zeta
system.time(fit <- softmaximin(x, y, zeta = c(0.1, 1), penalty = "lasso", alg = "npg"))
betahat <- fit$coef

##estimated common effects for specific lambda and zeta
zetano <- 2
modelno <- dim(betahat[[zetano]])[2]
m <- min(betahat[[zetano]][ , modelno], common_effects)
M <- max(betahat[[zetano]][ , modelno], common_effects)
plot(common_effects, type = "p", ylim = c(m, M), col = "red")
lines(betahat[[zetano]][ , modelno], type = "h")
```

```
#Array data
##size of example
set.seed(42)
G <- 50; n <- c(30, 20, 10); p <- c(7, 5, 4)

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

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

##group response
 y <- array(NA, c(n, G))
for(g in 1:G){
bg <- rnorm(prod(p), 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
}

##fit model for range of lambda and zeta
system.time(fit <- softmaximin(x, y, zeta = c(1, 10, 100), penalty = "lasso",
            alg = "npg"))
betahat <- fit$coef

##estimated common effects for specific lambda and zeta
zetano <- 1
modelno <- dim(betahat[[zetano]])[2]
m <- min(betahat[[zetano]][, modelno], common_effects)
M <- max(betahat[[zetano]][, modelno], common_effects)
plot(common_effects, type = "p", ylim = c(m, M), col = "red")
lines(betahat[[zetano]][ , modelno], type = "h")

#Array data and wavelets
##size of example
set.seed(42)
G <- 50; p <- n <- c(2^3, 2^4, 2^5);

##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
}
```

```
##fit model for range of lambda and zeta
system.time(fit <- softmaximin(x = "la8", y, zeta = c(0.1, 1, 10),
                               penalty = "lasso", alg = "fista"))
betahat <- fit$coef

##estimated common effects for specific lambda and zeta
zetano <- 3
modelno <- dim(betahat[[zetano]])[2]
m <- min(betahat[[zetano]][, modelno], common_effects)
M <- max(betahat[[zetano]][, modelno], common_effects)
plot(common_effects, type = "p", ylim = c(m, M), col = "red")
lines(betahat[[zetano]][ , modelno], type = "h")
```

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, Rigorous Analytics Ltd, licensed under the BSD 3 license https://cran.r-project.org/web/licenses/BSD_3_claus 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).

This functionality is used in the computations underlying `softmaximin` to perform multiplications involving the wavelet (design) matrix efficiently.

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

∗ **package**
    SMME, 6

H *(*RH*)*, 5

iwt, 2

pga *(*SMME*)*, 6
predict.SMME, 3
print.SMME, 4

RH, 5, *9*
Rotate *(*RH*)*, 5

SMME, 6
SMME.predict *(*predict.SMME*)*, 3
SMME_predict *(*predict.SMME*)*, 3
SMME_RH *(*RH*)*, 5
softmaximin, 2, *12*
softmaximin *(*SMME*)*, 6

wt, *7*, *9*, 12