Package 'SMME'

July 22, 2021

Type Package

Title Soft Maximin Estimation for Large Scale Heterogeneous Data

Version 1.0.1

Date 2021-07-21

Author Adam Lund

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 (2021) <arXiv:1805.02407>. Currently Lasso and SCAD penalized estimation is implemented. Note this package subsumes and replaces the SMMA package.

License GPL (>= 2)

Imports Rcpp (>= 0.12.12)

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.1.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2021-07-22 07:50:11 UTC

R topics documented:

`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 two different formats: i) as a $n' \times p$ matrix (p is the number of model coefficients and n' is the number of new data points) or ii) as a list of 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).

Usage

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

Arguments

`object`	An object of class SMME, produced with `softmaximin` with m_ζ fitted models for each value of `zeta`.
`x`	a matrix of size $n' \times p$ with n' is the number of new data points.
`X`	a list containing the data matrices each of size $n'_i \times p_i$, where n'_i is the number of new data points in the ith dimension.
`...`	ignored

Value

A list of length `length(zeta)`. If new covariate data is supplied as an $n' \times p$ matrix `x` each item in the list is an $n' \times m_\zeta$ matrix with the linear predictors computed for each model. If new covariate data is supplied as a list of matrices each of size $n'_i \times p_i$, each item is an array of size $n'_1 \times \cdots \times n'_d \times m_\zeta$, $d \in \{1, 2, 3\}$, with the linear predictors computed for each model.

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 matrix form
x <- matrix(rnorm(2 * p1 * p2 * p3), nrow = 2)
yhat <- predict(fit, x = x)

##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 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., 2021*. i) For general group specific design the soft maximin problem is solved using the NPG algorithm. ii) 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 for $d = 2, 3$ avoids using the tensor design matrix. 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,
          penalty.factor = NULL,
          reltol = 1e-05,
          maxiter = 1000,
          steps = 1,
          btmax = 100,
          c = 0.0001,
          tau = 2,
          M = 4,
          nu = 1,
          Lmin = 0,
          log = TRUE,
          nthreads = 4)
```

Arguments

x	list containing the G group specific design matrices of sizes $n_i \times p_i$. Alternatively for a model with identical tensor design across G groups, x is a list containing the d ($d \in \{1, 2, 3\}$) tensor components.
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, yis 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	sequence of strictly positive floats used as penalty parameters.
penalty.factor	array of size $p_1 \times \cdots \times p_d$ of positive floats. Is multiplied with each element in lambda to allow differential penalization on the coefficients.
reltol	strictly positive float giving the convergence tolerance for the inner loop.
maxiter	positive integer giving the maximum number of iterations allowed for each lambda value, when summing over all outer iterations for said lambda.
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.
`log`	logical variable indicating whether to use log-loss. TRUE is default and yields the loss below.
`nthreads`	integer giving the number of threads to use when openMP is available. Default is 4.

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., 2021*, 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 ($\zeta \sim 0$ 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 from *Meinshausen and Buhlmann, 2015.* See *Lund et al., 2021* 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 applications where the design is identical across groups i.e. $\mathbf{X}_g = \mathbf{X}$ $\forall g \in \{1, \ldots, G\}$ and where $\mathbf{X}$ has tensor structure i.e.

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

for marginal $n_i \times p_i$ design matrices $\mathbf{M}_1, \ldots, \mathbf{M}_d$.

For $d \in \{1, 2, 3\}$, provided only with the marginal matrices and the group response vectors, `softmaximin` solves the soft maximin problem with minimal memory footprint using tensor optimized arithmetic, see RH.

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 $p_1 \cdots p_d \times$ `nlambda` matrix containing the estimates of the model coefficients (`beta`) for each `lambda`-value for which the procedure converged. When `length(zeta) > 1` a `length(zeta)`-list of such matrices.
`lambda`	A vector containing the sequence of penalty values used in the estimation procedure for which the procedure converged. When `length(zeta) > 1` a `length(zeta)`-list of such vectors.
`Obj`	A matrix containing the objective values for each iteration and each model for which the procedure converged. When `length(zeta) > 1` a `length(zeta)`-list of such matrices.
`df`	A vector indicating the nonzero model coefficients for each value of `lambda` for which the procedure converged. When `length(zeta) > 1` a `length(zeta)`-list of such vectors.
`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`.
`iter`	A vector containing the number of iterations for each `lambda` value for which the procedure converged. When `length(zeta) > 1` a `length(zeta)`-list of such vectors.

Author(s)

Adam Lund

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

References

Lund, A., S. W. Mogensen and N. R. Hansen (2021). Soft Maximin Estimation for Heterogeneous Data. *Preprint*. url = https://arxiv.org/abs/1805.02407

Meinshausen, N and P. Buhlmann (2015). Maximin effects in inhomogeneous large-scale data. *The Annals of Statistics*. 43, 4, 1801-1830. url = https://doi.org/10.1214/15-AOS1325.

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)
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
modelno <- 6; 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 <- 5; n <- c(65, 26, 13); p <- c(13, 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)
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, 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(0.1, 1, 10, 100), penalty = "lasso", alg = "npg"))
Betahat <- fit$coef

##estimated common effects for specific lambda and zeta
modelno <- 10; zetano <- 3
m <- min(Betahat[[zetano]][, modelno], common_effects)
M <- max(Betahat[[zetano]][, modelno], common_effects)
plot(common_effects, type = "h", ylim = c(m, M), col = "red")
lines(Betahat[[zetano]][, modelno], type = "h")
```

Index