Package ‘SMMA’

January 18, 2018

Type Package

Title Soft Maximin Estimation for Large Scale Array-Tensor Models

Version 1.0.2

Date 2018-01-18

Author Adam Lund

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

Description
Efficient design matrix free procedure for solving a soft maximin problem for large scale array-tensor structured models. Currently Lasso and SCAD penalized estimation is implemented.

License GPL (>= 2)

Imports Rcpp (>= 0.12.12)

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 6.0.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-01-18 16:02:24 UTC

R topics documented:

`predict.SMMA` *Make Prediction From a SMMA 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 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 'SMMA'
predict(object, x = NULL, X = NULL, ...)
```

Arguments

`object`	An object of class SMMA, produced with `softmaximin`
`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 `nlambda` containing the linear predictors for each model. If new covariate data is supplied in one $n' \times p$ matrix `x` each item is a vector of length n'. If the data is supplied as a list of matrices each of size $n'_i \times p_i$, each item is an array of size $n'_1 \times \cdots \times n'_d$, with $d \in \{1, 2, 3\}$.

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, penalty = "lasso", alg = "npg")

##new data in matrix form
x <- matrix(rnorm(p1 * p2 * p3), nrow = 1)
predict(fit, x = x)[[15]]

##new data in tensor component form
X1 <- matrix(rnorm(p1), nrow = 1)
X2 <- matrix(rnorm(p2), nrow = 1)
X3 <- matrix(rnorm(p3), nrow = 1)
predict(fit, X = list(X1, X2, X3))[[15]]
```

---

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

Description

This function will print some information about the SMMA object.

Usage

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

Arguments

x	a SMMA 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, 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 optimization routines underlying the glamlasso 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.

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)))
```

SMMA — *Soft Maximin Estimation for Large Scale Array Data with Known Groups*

Description

Efficient design matrix free procedure for solving a soft maximin problem for large scale array-tensor structured models, see *Lund et al., 2017.* Currently Lasso and SCAD penalized estimation is implemented.

Usage

```
softmaximin(X,
            Y,
            penalty = c("lasso", "scad"),
            nlambda = 30,
            lambda.min.ratio = 1e-04,
            lambda = NULL,
            penalty.factor = NULL,
            reltol = 1e-05,
            maxiter = 15000,
```

```
          steps = 1,
          btmax = 100,
          zeta = 2,
          c = 0.001,
          Delta0 = 1,
          nu = 1,
          alg = c("npg", "mfista"),
          log = TRUE)
```

Arguments

X	A list containing the Kronecker components (1,2 or 3) of the Kronecker design matrix. These are matrices of sizes $n_i \times p_i$.
Y	The response values, an array of size $n_1 \times \cdots \times n_d \times G$.
penalty	A string specifying the penalty. Possible values are "lasso", "scad".
nlambda	The number of lambda values.
lambda.min.ratio	The smallest value for lambda, given as a fraction of λ_{max}; the (data dependent) smallest value for which all coefficients are zero.
lambda	The sequence of penalty parameters for the regularization path.
penalty.factor	An array of size $p_1 \times \cdots \times p_d$. Is multiplied with each element in lambda to allow differential shrinkage on the coefficients.
reltol	The convergence tolerance for the inner loop.
maxiter	The maximum number of iterations allowed for each lambda value, when summing over all outer iterations for said lambda.
steps	The number of steps used in the multi-step adaptive lasso algorithm for non-convex penalties. Automatically set to 1 when penalty = "lasso".
btmax	Maximum number of backtracking steps allowed in each iteration. Default is btmax = 100.
zeta	Constant controlling the softmax apprximation accuracy. Must be strictly positive. Default is zeta = 2.
c	constant used in the NPG algorithm. Must be strictly positive. Default is c = 0.001.
Delta0	constant used to bound the stepsize. Must be strictly positive. Default is Delta0 = 1.
nu	constant used to control the stepsize. Must be positive. A small value gives a big stepsize. Default is nu = 1.
alg	string indicating which algortihm to use. Possible values are "npg", "mfista".
log	logical variable indicating wheter to use log-loss to or not. TRUE is default and yields the problem described below.

Details

In *Lund et al., 2017* the following mixed model setup for d-dimensional array data, $d = 1, 2, 3$, with known fixed group structure and tensor structured design matrix is considered: With G groups,

$g \in \{1,\ldots,G\}$, n is the number of observations in each group, $Y_g := (y_i,\ldots,y_{i_n})^\top$ the group-specific $n_1 \times \cdots \times n_d$ response array and X a $n \times p$ design matrix, with tensor structure

$$X = \bigotimes_{i=1}^{d} X_i,$$

where for $d = 1, 2, 3$, $X_1, \ldots, X_d$ are the marginal $n_i \times p_i$ design matrices (Kronecker components). Using the GLAM framework the model equation is

$$Y_g = \rho(X_d, \rho(X_{d-1}, \ldots, \rho(X_1, B_g))) + E,$$

where ρ is the so called rotated H-transfrom (see *Currie et al., 2006*), B_g for each g is a random $p_1 \times \cdots \times p_d$ parameter array and E is $n_1 \times \cdots \times n_d$ error array uncorrelated with X. Note that for $d = 1$ the model is a GLM.

In *Lund et al., 2017* a penalized soft maximin problem, given as

$$\min_{\beta} \log\left(\sum_{g=1}^{G} \exp(-\zeta \hat{V}_g(\beta))\right) + \lambda J(\beta),$$

is proposed where J is a proper and convex penalty, $\zeta > 0$ and

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

$y_g := vec(Y_g)$, is the minimal empirical explained variance from *Meinshausen and Buhlmann, 2015*.

For $d = 1, 2, 3$, using only the marginal matrices $X_1, X_2, \ldots$ (for $d = 1$ there is only one marginal), the function `softmaximin` solves the soft maximin problem for a sequence of penalty parameters $\lambda_{max} > \ldots > \lambda_{min} > 0$. The underlying algorithm is based on a non-monotone proximal gradient method. We note that if J is not convex, as with the SCAD penalty, we use the multiple step adaptive lasso procedure to loop over the proximal algorithm, see *Lund et al., 2017* for more details.

Value

An object with S3 Class "SMMA".

`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.
`Obj`	A matrix containing the objective values for each iteration and each model.
`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`.
`Iter`	A list with 4 items: `bt_iter` is total number of backtracking steps performed, `bt_enter` is the number of times the backtracking is initiated, and `iter_mat` is a vector containing the number of iterations for each `lambda` value and `iter` is total number of iterations i.e. `sum(Iter)`.

Author(s)

Adam Lund

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

References

Lund, A., S. W. Mogensen and N. R. Hansen (2017). Estimating Soft Maximin Effects in Heterogeneous Large-scale Array Data. *Preprint*.

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

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.

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), n1, p1)
X2 <- matrix(rnorm(n2 * p2), n2, p2)
X3 <- matrix(rnorm(n3 * p3), 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))
mu1 <- RH(X3, RH(X2, RH(X1, Beta1)))
Y1 <- array(rnorm(n1 * n2 * n3), dim = c(n1, n2, n3)) + mu1
Beta2 <- array(rnorm(p1 * p2 * p3, 0, 0.1) + component, c(p1 , p2, p3))
mu2 <- RH(X3, RH(X2, RH(X1, Beta2)))
Y2 <- array(rnorm(n1 * n2 * n3), dim = c(n1, n2, n3)) + mu2
Beta3 <- array(rnorm(p1 * p2 * p3, 0, 0.1) + component, c(p1 , p2, p3))
mu3 <- RH(X3, RH(X2, RH(X1, Beta3)))
Y3 <- array(rnorm(n1 * n2 * n3), dim = c(n1, n2, n3)) + mu3
Beta4 <- array(rnorm(p1 * p2 * p3, 0, 0.1) + component, c(p1 , p2, p3))
mu4 <- RH(X3, RH(X2, RH(X1, Beta4)))
Y4 <- array(rnorm(n1 * n2 * n3), dim = c(n1, n2, n3)) + mu4
Beta5 <- array(rnorm(p1 * p2 * p3, 0, 0.1) + component, c(p1 , p2, p3))
mu5 <- RH(X3, RH(X2, RH(X1, Beta5)))
Y5 <- array(rnorm(n1 * n2 * n3), dim = c(n1, n2, n3)) + mu5

Y <- array(NA, c(dim(Y1), 5))
Y[,,, 1] <- Y1; Y[,,, 2] <- Y2; Y[,,, 3] <- Y3; Y[,,, 4] <- Y4; Y[,,, 5] <- Y5;

fit <- softmaximin(X, Y, penalty = "lasso", alg = "npg")
Betafit <- fit$coef

modelno <- 15
```

```
m <- min(Betafit[ , modelno], c(component))
M <- max(Betafit[ , modelno], c(component))
plot(c(component), type="l", ylim = c(m, M))
lines(Betafit[ , modelno], col = "red")
```

Index