Package 'glamlasso'

July 22, 2025

Type Package

Title Penalization in Large Scale Generalized Linear Array Models

Version 3.0.1

Date 2021-05-10

Author Adam Lund

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

Description Efficient design matrix free lasso penalized estimation in large scale 2 and 3-dimensional generalized linear array model framework. The procedure is based on the gdpg algorithm from Lund et al. (2017) <doi:10.1080/10618600.2017.1279548>. Currently Lasso or Smoothly Clipped Absolute Deviation (SCAD) penalized estimation is possible for the following models: The Gaussian model with identity link, the Binomial model with logit link, the Poisson model with log link and the Gamma model with log link. It is also possible to include a component in the model with non-tensor design e.g an intercept. Also provided are functions, glamlassoRR() and glamlassoS(), fitting special cases of GLAMs.

License GPL-3

Imports Rcpp (>= 0.11.2)

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.1.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2021-05-16 22:30:06 UTC

Contents

`glamlasso` *Penalization in Large Scale Generalized Linear Array Models*

Description

Efficient design matrix free procedure for fitting large scale penalized 2 or 3-dimensional generalized linear array models (GLAM). It is also possible to fit an additional non-tensor structured component - e.g an intercept - however this can reduce the computational efficiency of the procedure substantially. Currently the LASSO penalty and the SCAD penalty are both implemented. Furthermore, the Gaussian model with identity link, the Binomial model with logit link, the Poisson model with log link and the Gamma model with log link is currently implemented. The underlying algorithm combines gradient descent and proximal gradient (gdpg algorithm), see *Lund et al., 2017*.

Usage

```
glamlasso(X,
          Y,
          Z = NULL,
          family = "gaussian",
          penalty = "lasso",
          intercept = FALSE,
          weights = NULL,
          betainit = NULL,
          alphainit = NULL,
          nlambda = 100,
          lambdaminratio = 1e-04,
          lambda = NULL,
          penaltyfactor = NULL,
          penaltyfactoralpha = NULL,
          reltolinner = 1e-07,
          reltolouter = 1e-04,
          maxiter = 15000,
          steps = 1,
          maxiterinner = 3000,
          maxiterouter = 25,
          btinnermax = 100,
          btoutermax = 100,
          iwls = "exact",
          nu = 1)
```

Arguments

`X`	A list containing the tensor components (2 or 3) of the tensor design matrix. These are matrices of sizes $n_i \times p_i$.

Y	The response values, an array of size $n_1 \times \cdots \times n_d$. For option family = "binomial" this array must contain the proportion of successes and the number of trials is then specified as weights (see below).
Z	The non tensor structrured part of the design matrix. A matrix of size $n_1 \cdots n_d \times q$. Is set to NULL as default.
family	A string specifying the model family (essentially the response distribution). Possible values are "gaussian", "binomial", "poisson", "gamma".
penalty	A string specifying the penalty. Possible values are "lasso", "scad".
intercept	Logical variable indicating if the model includes an intercept. When intercept = TRUE the first coulmn in the non-tensor design component Z is all 1s. Default is FALSE.
weights	Observation weights, an array of size $n_1 \times \cdots \times n_d$. For option family = "binomial" this array must contain the number of trials and must be provided.
betainit	The initial parameter values. Default is NULL in which case all parameters are initialized at zero.
alphainit	A $q \times 1$ vector containing the initial parameter values for the non-tensor parameter. Default is NULL in which case all parameters are initialized at 0.
nlambda	The number of lambda values.
lambdaminratio	The smallest value for lambda, given as a fraction of λ_{max}; the (data derived) smallest value for which all coefficients are zero.
lambda	The sequence of penalty parameters for the regularization path.
penaltyfactor	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.
penaltyfactoralpha	A $q \times 1$ vector multiplied with each element in lambda to allow differential shrinkage on the non-tensor coefficients.
reltolinner	The convergence tolerance for the inner loop
reltolouter	The convergence tolerance for the outer loop.
maxiter	The maximum number of inner 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".
maxiterinner	The maximum number of inner iterations allowed for each outer iteration.
maxiterouter	The maximum number of outer iterations allowed for each lambda.
btinnermax	Maximum number of backtracking steps allowed in each inner iteration. Default is btinnermax = 100.
btoutermax	Maximum number of backtracking steps allowed in each outer iteration. Default is btoutermax = 100.
iwls	A string indicating whether to use the exact iwls weight matrix or use a kronecker structured approximation to it.

nu — A number between 0 and 1 that controls the step size δ in the proximal algorithm (inner loop) by scaling the upper bound $\hat{L}_h$ on the Lipschitz constant L_h (see *Lund et al., 2017*). For `nu = 1` backtracking never occurs and the proximal step size is always $\delta = 1/\hat{L}_h$. For `nu = 0` backtracking always occurs and the proximal step size is initially $\delta = 1$. For `0 < nu < 1` the proximal step size is initially $\delta = 1/(\nu\hat{L}_h)$ and backtracking is only employed if the objective function does not decrease. A `nu` close to 0 gives large step sizes and presumably more backtracking in the inner loop. The default is `nu = 1` and the option is only used if `iwls = "exact"`.

Details

Consider a (two component) generalized linear model (GLM)

$$g(\mu) = X\beta + Z\alpha =: \eta.$$

Here g is a link function, μ is a $n \times 1$ vector containing the mean of the response variable Y, Z is a $n \times q$ matrix and X a $n \times p$ matrix with tensor structure

$$X = X_d \otimes \cdots \otimes X_1,$$

where $X_1, \ldots, X_d$ are the marginal $n_i \times p_i$ design matrices (tensor factors) such that $p = p_1 \cdots p_d$ and $n = n_1 \cdots n_d$. Then β is the $p \times 1$ parameter associated with the tensor component X and α the $q \times 1$ parameter associated with the non-tensor component Z, e.g. the intercept.

Using the generalized linear array model (GLAM) framework the model equation is

$$g(\mu) = \text{vec}(\rho(X_d, \rho(X_{d-1}, \ldots, \rho(X_1, B)))) + Z\alpha,$$

where ρ is the so called rotated H-transform and B is the array version of β. See *Currie et al., 2006* for more details.

The log-likelihood is a function of $\theta := (\beta, \alpha)$ through the linear predictor η i.e. $\theta \mapsto l(\eta(\theta))$. In the usual exponential family framework this can be expressed as

$$l(\eta(\theta)) = \sum_{i=1}^{n} a_i \frac{y_i\vartheta(\eta_i(\theta)) - b(\vartheta(\eta_i(\theta)))}{\psi} + c(y_i, \psi)$$

where ϑ, the canonical parameter map, is linked to the linear predictor via the identity $\eta(\theta) = g(b'(\vartheta))$ with b the cumulant function. Here $a_i \geq 0, i = 1, \ldots, n$ are observation weights and ψ is the dispersion parameter.

For $d = 3$ or $d = 2$, using only the marginal matrices $X_1, X_2, \ldots$, the function `glamlasso` solves the penalized estimation problem

$$\min_{\theta} -l(\eta(\theta)) + \lambda J(\theta),$$

for J either the LASSO or SCAD penalty function, in the GLAM setup for a sequence of penalty parameters $\lambda > 0$. The underlying algorithm is based on an outer gradient descent loop and an inner proximal gradient based loop. 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 inner proximal algorithm, see *Lund et al., 2017* for more details.

Note that the package is optimized towards solving the estimation problem, for $\alpha = 0$. For $\alpha \neq 0$ the user incurs a potentially substantial computational cost. Especially it is not advisable to inlcude a very large non-tensor component in the model (large q) and even adding an intecept to the model ($q = 1$) will result in a reduction of computational efficiency.

Value

An object with S3 Class 'glamlasso'.

spec	A string indicating the GLAM dimension ($d = 2, 3$), the model family and the penalty.
beta	A $p_1 \cdots p_d \times$ nlambda matrix containing the estimates of the parameters for the tensor structured part of the model (beta) for each lambda-value.
alpha	A $q \times$ nlambda matrix containing the estimates of the parameters for the non tensor structured part of the model alpha for each lambda-value. If intercept = TRUE the first row contains the intercept estimate 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.
Iter	A list with 4 items: bt_iter_inner is total number of backtracking steps performed in the inner loop, bt_enter_inner is the number of times the backtracking is initiated in the inner loop, bt_iter_outer is total number of backtracking steps performed in the outer loop, and iter_mat is a nlambda $\times$ maxiterouter matrix containing the number of inner iterations for each lambda value and each outer iteration 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., M. Vincent, and N. R. Hansen (2017). Penalized estimation in large-scale generalized linear array models. *Journal of Computational and Graphical Statistics*, 26, 3, 709-724. url = https://doi.org/10.1080/10618600.2017.1279548.

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

```
##size of example
n1 <- 65; n2 <- 26; n3 <- 13; p1 <- 12; p2 <- 6; p3 <- 4

##marginal design matrices (tensor 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)
```

```
##############gaussian example
Beta <- array(rnorm(p1 * p2 * p3) * rbinom(p1 * p2 * p3, 1, 0.1), c(p1 , p2, p3))
Mu <- RH(X3, RH(X2, RH(X1, Beta)))
Y <- array(rnorm(n1 * n2 * n3, Mu), c(n1, n2, n3))

system.time(fit <- glamlasso(X, Y))

modelno <- length(fit$lambda)
plot(c(Beta), type = "h", ylim = range(Beta, fit$coef[, modelno]))
points(c(Beta))
lines(fit$coef[ , modelno], col = "red", type = "h")

###with non tensor design component Z
q <- 5
alpha <- matrix(rnorm(q)) * rbinom(q, 1, 0.5)
Z <- matrix(rnorm(n1 * n2 * n3 * q), n1 * n2 *n3, q)
Y <- array(rnorm(n1 * n2 * n3, Mu + array(Z %*% alpha, c(n1, n2, n3))), c(n1, n2, n3))
system.time(fit <- glamlasso(X, Y, Z))

modelno <- length(fit$lambda)
oldmfrow <- par()$mfrow
par(mfrow = c(1, 2))
plot(c(Beta), type = "l", ylim = range(Beta, fit$coef[, modelno]))
points(c(Beta))
lines(fit$coef[ , modelno], col = "red")
plot(c(alpha), type = "h", ylim = range(Beta, fit$alpha[, modelno]))
points(c(alpha))
lines(fit$alpha[ , modelno], col = "red", type = "h")
par(mfrow = oldmfrow)

################ poisson example
Beta <- array(rnorm(p1 * p2 * p3, 0, 0.1) * rbinom(p1 * p2 * p3, 1, 0.1), c(p1 , p2, p3))
Mu <- RH(X3, RH(X2, RH(X1, Beta)))
Y <- array(rpois(n1 * n2 * n3, exp(Mu)), dim = c(n1, n2, n3))
system.time(fit <- glamlasso(X, Y, family = "poisson", nu = 0.1))

modelno <- length(fit$lambda)
plot(c(Beta), type = "h", ylim = range(Beta, fit$coef[, modelno]))
points(c(Beta))
lines(fit$coef[ , modelno], col = "red", type = "h")

###with non tensor design component Z
q <- 5
alpha <- matrix(rnorm(q)) * rbinom(q, 1, 0.5)
Z <- matrix(rnorm(n1 * n2 * n3 * q), n1 * n2 *n3, q)
Y <- array(rpois(n1 * n2 * n3, exp(Mu + array(Z %*% alpha, c(n1, n2, n3)))), dim = c(n1, n2, n3))
system.time(fit <- glamlasso(X, Y, Z, family = "poisson", nu = 0.1))

modelno <- length(fit$lambda)
oldmfrow <- par()$mfrow
par(mfrow = c(1, 2))
plot(c(Beta), type = "l", ylim = range(Beta, fit$coef[, modelno]))
points(c(Beta))
```

```
lines(fit$coef[ , modelno], col = "red")
plot(c(alpha), type = "h", ylim = range(Beta, fit$alpha[, modelno]))
points(c(alpha))
lines(fit$alpha[ , modelno], col = "red", type = "h")
par(mfrow = oldmfrow)
```

`glamlassoRR` *Penalized reduced rank regression in a GLAM*

Description

Efficient design matrix free procedure for fitting large scale penalized reduced rank regressions in a 3-dimensional generalized linear array model. To obtain a factorization of the parameter array, the `glamlassoRR` function performes a block relaxation scheme within the gdpg algorithm, see *Lund and Hansen, 2018*.

Usage

```
glamlassoRR(X,
            Y,
            Z = NULL,
            family = "gaussian",
            penalty = "lasso",
            intercept = FALSE,
            weights = NULL,
            betainit = NULL,
            alphainit = NULL,
            nlambda = 100,
            lambdaminratio = 1e-04,
            lambda = NULL,
            penaltyfactor = NULL,
            penaltyfactoralpha = NULL,
            reltolinner = 1e-07,
            reltolouter = 1e-04,
            reltolalt = 1e-04,
            maxiter = 15000,
            steps = 1,
            maxiterinner = 3000,
            maxiterouter = 25,
            maxalt = 10,
            btinnermax = 100,
            btoutermax  = 100,
            iwls = "exact",
            nu = 1)
```

Arguments

X	A list containing the 3 tensor components of the tensor design matrix. These are matrices of sizes $n_i \times p_i$.
Y	The response values, an array of size $n_1 \times n_2 \times n_3$. For option family = "binomial" this array must contain the proportion of successes and the number of trials is then specified as weights (see below).
Z	The non tensor structrured part of the design matrix. A matrix of size $n_1 n_2 n_3 \times q$. Is set to NULL as default.
family	A string specifying the model family (essentially the response distribution). Possible values are "gaussian", "binomial", "poisson", "gamma".
penalty	A string specifying the penalty. Possible values are "lasso", "scad".
intercept	Logical variable indicating if the model includes an intercept. When intercept = TRUE the first coulmn in the non-tensor design component Z is all 1s. Default is FALSE.
weights	Observation weights, an array of size $n_1 \times \cdots \times n_d$. For option family = "binomial" this array must contain the number of trials and must be provided.
betainit	A list (length 2) containing the initial parameter values for each of the parameter factors. Default is NULL in which case all parameters are initialized at 0.01.
alphainit	A $q \times 1$ vector containing the initial parameter values for the non-tensor parameter. Default is NULL in which case all parameters are initialized at 0.
nlambda	The number of lambda values.
lambdaminratio	The smallest value for lambda, given as a fraction of λ_{max}; the (data derived) smallest value for which all coefficients are zero.
lambda	The sequence of penalty parameters for the regularization path.
penaltyfactor	A list of length two containing an array of size $p_1 \times p_2$ and a $p_3 \times 1$ vector. Multiplied with each element in lambda to allow differential shrinkage on the (tensor) coefficients blocks.
penaltyfactoralpha	A $q \times 1$ vector multiplied with each element in lambda to allow differential shrinkage on the non-tensor coefficients.
reltolinner	The convergence tolerance for the inner loop
reltolouter	The convergence tolerance for the outer loop.
reltolalt	The convergence tolerance for the alternation loop over the two parameter blocks.
maxiter	The maximum number of inner 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".
maxiterinner	The maximum number of inner iterations allowed for each outer iteration.
maxiterouter	The maximum number of outer iterations allowed for each lambda.
maxalt	The maximum number of alternations over parameter blocks.
btinnermax	Maximum number of backtracking steps allowed in each inner iteration. Default is btinnermax = 100.

`btoutermax` Maximum number of backtracking steps allowed in each outer iteration. Default is `btoutermax = 100`.

`iwls` A string indicating whether to use the exact iwls weight matrix or use a tensor structured approximation to it.

`nu` A number between 0 and 1 that controls the step size δ in the proximal algorithm (inner loop) by scaling the upper bound $\hat{L}_h$ on the Lipschitz constant L_h (see *Lund et al., 2017*). For `nu = 1` backtracking never occurs and the proximal step size is always $\delta = 1/\hat{L}_h$. For `nu = 0` backtracking always occurs and the proximal step size is initially $\delta = 1$. For `0 < nu < 1` the proximal step size is initially $\delta = 1/(\nu\hat{L}_h)$ and backtracking is only employed if the objective function does not decrease. A `nu` close to 0 gives large step sizes and presumably more backtracking in the inner loop. The default is `nu = 1` and the option is only used if `iwls = "exact"`.

Details

Given the setting from `glamlasso` we place a reduced rank restriction on the $p_1 \times p_2 \times p_3$ parameter array B given by

$$B = (B_{i,j,k})_{i,j,k} = (\gamma_k \kappa_{i,j})_{i,j,k}, \quad \gamma_k, \kappa_{i,j} \in \mathcal{R}.$$

The `glamlassoRR` function solves the PMLE problem by combining a block relaxation scheme with the gdpg algorithm. This scheme alternates between optimizing over the first parameter block $\kappa = (\kappa_{i,j})_{i,j}$ and the second block $\gamma = (\gamma_k)_k$ while fixing the second resp. first block.

Note that the individual parameter blocks are only identified up to a multiplicative constant. Also note that the algorithm is sensitive to inital values `betainit` which can prevent convergence.

Value

An object with S3 Class "glamlasso".

`spec` A string indicating the model family and the penalty.

`coef12` A $p_1 p_2 \times$ `nlambda` matrix containing the estimates of the first model coefficient factor (κ) for each `lambda`-value.

`coef3` A $p_3 \times$ `nlambda` matrix containing the estimates of the second model coefficient factor (γ) for each `lambda`-value.

`alpha` A $q \times$ `nlambda` matrix containing the estimates of the parameters for the non tensor structured part of the model (`alpha`) for each `lambda`-value. If `intercept = TRUE` the first row contains the intercept estimate 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`.

`Iter` A list with 4 items: `bt_iter_inner` is total number of backtracking steps performed in the inner loop, `bt_enter_inner` is the number of times the backtracking is initiated in the inner loop, `bt_iter_outer` is total number of backtracking

steps performed in the outer loop, and `iter_mat` is a `nlambda` $\times$ `maxiterouter` matrix containing the number of inner iterations for each `lambda` value and each outer iteration 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., M. Vincent, and N. R. Hansen (2017). Penalized estimation in large-scale generalized linear array models. *Journal of Computational and Graphical Statistics*, 26, 3, 709-724. url = https://doi.org/10.1080/10618600.2017.1279548.

Lund, A. and N. R. Hansen (2019). Sparse Network Estimation for Dynamical Spatio-temporal Array Models. *Journal of Multivariate Analysis*, 174. url = https://doi.org/10.1016/j.jmva.2019.104532.

Examples

```
##size of example
n1 <- 65; n2 <- 26; n3 <- 13; p1 <- 12; p2 <- 6; p3 <- 4

##marginal design matrices (tensor 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)
Beta12 <- matrix(rnorm(p1 * p2), p1, p2) * matrix(rbinom(p1 * p2, 1, 0.5), p1, p2)
Beta3 <- matrix(rnorm(p3) * rbinom(p3, 1, 0.5), p3, 1)
Beta <- outer(Beta12, c(Beta3))
Mu <- RH(X3, RH(X2, RH(X1, Beta)))
Y <- array(rnorm(n1 * n2 * n3, Mu), dim = c(n1, n2, n3))

system.time(fit <- glamlassoRR(X, Y))

modelno  <- length(fit$lambda)
oldmfrow <- par()$mfrow
par(mfrow = c(1, 3))
plot(c(Beta), type = "h")
points(c(Beta))
lines(c(outer(fit$coef12[, modelno], c(fit$coef3[, modelno]))), col = "red", type = "h")
plot(c(Beta12), ylim = range(Beta12, fit$coef12[, modelno]), type = "h")
points(c(Beta12))
lines(fit$coef12[, modelno], col = "red", type = "h")
plot(c(Beta3), ylim = range(Beta3, fit$coef3[, modelno]), type = "h")
points(c(Beta3))
lines(fit$coef3[, modelno], col = "red", type = "h")
par(mfrow = oldmfrow)

###with non tensor design component Z
q <- 5
```

```
alpha <- matrix(rnorm(q)) * rbinom(q, 1, 0.5)
Z <- matrix(rnorm(n1 * n2 * n3 * q), n1 * n2 * n3, q)
Y <- array(rnorm(n1 * n2 * n3, Mu + array(Z %*% alpha, c(n1, n2, n3))), c(n1, n2, n3))
system.time(fit <- glamlassoRR(X, Y, Z))

modelno <- length(fit$lambda)
oldmfrow <- par()$mfrow
par(mfrow = c(2, 2))
plot(c(Beta), type = "h")
points(c(Beta))
lines(c(outer(fit$coef12[, modelno], c(fit$coef3[, modelno]))), col = "red", type = "h")
plot(c(Beta12), ylim = range(Beta12,fit$coef12[, modelno]), type = "h")
points(c(Beta12))
lines(fit$coef12[, modelno], col = "red", type = "h")
plot(c(Beta3), ylim = range(Beta3, fit$coef3[, modelno]), type = "h")
points(c(Beta3))
lines(fit$coef3[, modelno], col = "red", type = "h")
plot(c(alpha), ylim = range(alpha, fit$alpha[, modelno]), type = "h")
points(c(alpha))
lines(fit$alpha[, modelno], col = "red", type = "h")
par(mfrow = oldmfrow)

################ poisson example
set.seed(7954) ## for this seed the algorithm fails to converge for default initial values!!
set.seed(42)
##size of example
n1 <- 65; n2 <- 26; n3 <- 13; p1 <- 12; p2 <- 6; p3 <- 4

##marginal design matrices (tensor 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)

Beta12 <- matrix(rnorm(p1 * p2, 0, 0.5) * rbinom(p1 * p2, 1, 0.1), p1, p2)
Beta3 <-  matrix(rnorm(p3, 0, 0.5) * rbinom(p3, 1, 0.5), p3, 1)
Beta <- outer(Beta12, c(Beta3))
Mu <- RH(X3, RH(X2, RH(X1, Beta)))
Y <- array(rpois(n1 * n2 * n3, exp(Mu)), dim = c(n1, n2, n3))
system.time(fit <- glamlassoRR(X, Y ,family = "poisson"))
modelno <- length(fit$lambda)
oldmfrow <- par()$mfrow
par(mfrow = c(1, 3))
plot(c(Beta), type = "h")
points(c(Beta))
lines(c(outer(fit$coef12[, modelno], c(fit$coef3[, modelno]))), col = "red", type = "h")
plot(c(Beta12), ylim = range(Beta12, fit$coef12[, modelno]), type = "h")
points(c(Beta12))
lines(fit$coef12[, modelno], col = "red", type = "h")
plot(c(Beta3), ylim = range(Beta3, fit$coef3[, modelno]), type = "h")
points(c(Beta3))
lines(fit$coef3[, modelno], col = "red", type = "h")
par(mfrow = oldmfrow)
```

glamlassoS *Penalization in Large Scale Generalized Linear Array Models*

Description

Efficient design matrix free procedure for fitting a special case of a generalized linear model with array structured response and partially tensor structured covariates. See *Lund and Hansen, 2019* for an application of this special purpose function.

Usage

```
glamlassoS(X,
           Y,
           V,
           Z = NULL,
           family = "gaussian",
           penalty = "lasso",
           intercept = FALSE,
           weights = NULL,
           betainit = NULL,
           alphainit = NULL,
           nlambda = 100,
           lambdaminratio = 1e-04,
           lambda = NULL,
           penaltyfactor = NULL,
           penaltyfactoralpha = NULL,
           reltolinner = 1e-07,
           reltolouter = 1e-04,
           maxiter = 15000,
           steps = 1,
           maxiterinner = 3000,
           maxiterouter = 25,
           btinnermax = 100,
           btoutermax = 100,
           iwls = "exact",
           nu = 1)
```

Arguments

X	A list containing the tensor components (2 or 3) of the tensor design matrix. These are matrices of sizes $n_i \times p_i$.
Y	The response values, an array of size $n_1 \times \cdots \times n_d$. For option `family = "binomial"` this array must contain the proportion of successes and the number of trials is then specified as `weights` (see below).

V	The weight values, an array of size $n_1 \times \cdots \times n_d$.
Z	The non tensor structrured part of the design matrix. A matrix of size $n_1 \cdots n_d \times q$. Is set to NULL as default.
family	A string specifying the model family (essentially the response distribution). Possible values are "gaussian", "binomial", "poisson", "gamma".
penalty	A string specifying the penalty. Possible values are "lasso", "scad".
intercept	Logical variable indicating if the model includes an intercept. When intercept = TRUE the first coulmn in the non-tensor design component Z is all 1s. Default is FALSE.
weights	Observation weights, an array of size $n_1 \times \cdots \times n_d$. For option family = "binomial" this array must contain the number of trials and must be provided.
betainit	The initial parameter values. Default is NULL in which case all parameters are initialized at zero.
alphainit	A $q \times 1$ vector containing the initial parameter values for the non-tensor parameter. Default is NULL in which case all parameters are initialized at 0.
nlambda	The number of lambda values.
lambdaminratio	The smallest value for lambda, given as a fraction of λ_{max}; the (data derived) smallest value for which all coefficients are zero.
lambda	The sequence of penalty parameters for the regularization path.
penaltyfactor	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.
penaltyfactoralpha	A $q \times 1$ vector multiplied with each element in lambda to allow differential shrinkage on the non-tensor coefficients.
reltolinner	The convergence tolerance for the inner loop
reltolouter	The convergence tolerance for the outer loop.
maxiter	The maximum number of inner 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".
maxiterinner	The maximum number of inner iterations allowed for each outer iteration.
maxiterouter	The maximum number of outer iterations allowed for each lambda.
btinnermax	Maximum number of backtracking steps allowed in each inner iteration. Default is btinnermax = 100.
btoutermax	Maximum number of backtracking steps allowed in each outer iteration. Default is btoutermax = 100.
iwls	A string indicating whether to use the exact iwls weight matrix or use a kronecker structured approximation to it.
nu	A number between 0 and 1 that controls the step size δ in the proximal algorithm (inner loop) by scaling the upper bound $\hat{L}_h$ on the Lipschitz constant L_h (see *Lund et al., 2017*). For nu = 1 backtracking never occurs and the proximal step size is always $\delta = 1/\hat{L}_h$. For nu = 0 backtracking always occurs and the

proximal step size is initially $\delta = 1$. For `0 < nu < 1` the proximal step size is initially $\delta = 1/(\nu \hat{L}_h)$ and backtracking is only employed if the objective function does not decrease. A `nu` close to 0 gives large step sizes and presumably more backtracking in the inner loop. The default is `nu = 1` and the option is only used if `iwls = "exact"`.

Details

Given the setting from `glamlasso` we consider a model where the tensor design component is only partially tensor structured as

$$X = [V_1 X_2^\top \otimes X_1^\top, \ldots, V_{n_3} X_2^\top \otimes X_1^\top]^\top.$$

Here X_i is a $n_i \times p_i$ matrix for $i = 1, 2$ and V_i is a $n_1 n_2 \times n_1 n_2$ diagonal matrix for $i = 1, \ldots, n_3$.

Letting Y denote the $n_1 \times n_2 \times n_3$ response array and V the $n_1 \times n_2 \times n_3$ weight array containing the diagonals of the V_is, the function `glamlassoS` solves the PMLE problem using Y, V, X_1, X_2 and the non-tensor component Z as input.

Value

An object with S3 Class "glamlasso".

`spec`	A string indicating the model family and the penalty.
`beta`	A $p_1 \cdots p_d \times$ `nlambda` matrix containing the estimates of the parameters for the tensor structured part of the model (`beta`) for each `lambda`-value.
`alpha`	A $q \times$ `nlambda` matrix containing the estimates of the parameters for the non tensor structured part of the model (`alpha`) for each `lambda`-value. If `intercept = TRUE` the first row contains the intercept estimate 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`.
`Iter`	A list with 4 items: `bt_iter_inner` is total number of backtracking steps performed in the inner loop, `bt_enter_inner` is the number of times the backtracking is initiated in the inner loop, `bt_iter_outer` is total number of backtracking steps performed in the outer loop, and `iter_mat` is a `nlambda` $\times$ `maxiterouter` matrix containing the number of inner iterations for each `lambda` value and each outer iteration 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., M. Vincent, and N. R. Hansen (2017). Penalized estimation in large-scale generalized linear array models. *Journal of Computational and Graphical Statistics*, 26, 3, 709-724. url = https://doi.org/10.1080/10618600.2017.1279548.

Lund, A. and N. R. Hansen (2019). Sparse Network Estimation for Dynamical Spatio-temporal Array Models. *Journal of Multivariate Analysis*, 174. url = https://doi.org/10.1016/j.jmva.2019.104532.

Examples

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

##marginal design matrices (tensor components)
X1 <- matrix(rnorm(n1 * p1), n1, p1)
X2 <- matrix(rnorm(n2 * p2), n2, p2)
X <- list(X1, X2)
V <- array(rnorm(n3 * n2 * n1), c(n1, n2, n3))

##gaussian example
Beta <- array(rnorm(p1 * p2) * rbinom(p1 * p2, 1, 0.1), c(p1 , p2))
Mu <- V * array(RH(X2, RH(X1, Beta)), c(n1, n2, n3))
Y <- array(rnorm(n1 * n2 * n3, Mu), c(n1, n2, n3))
system.time(fit <- glamlassoS(X, Y, V))

modelno <- length(fit$lambda)
plot(c(Beta), ylim = range(Beta, fit$coef[, modelno]), type = "h")
points(c(Beta))
lines(c(fit$coef[, modelno]), col = "red", type = "h")

###with non tensor design component Z
q <- 5
alpha <- matrix(rnorm(q)) * rbinom(q, 1, 0.5)
Z <- matrix(rnorm(n1 * n2 * n3 * q), n1 * n2 *n3, q)
Y <- array(rnorm(n1 * n2 * n3, Mu + array(Z %*% alpha, c(n1, n2, n3))), c(n1, n2, n3))
system.time(fit <- glamlassoS(X, Y, V , Z))

modelno <- length(fit$lambda)
oldmfrow <- par()$mfrow
par(mfrow = c(1, 2))
plot(c(Beta), type="h", ylim = range(Beta, fit$coef[, modelno]))
points(c(Beta))
lines(fit$coef[ , modelno], col = "red", type = "h")
plot(c(alpha), type = "h", ylim = range(alpha, fit$alpha[, modelno]))
points(c(alpha))
lines(fit$alpha[ , modelno], col = "red", type = "h")
par(mfrow = oldmfrow)

################ poisson example
Beta <- matrix(rnorm(p1 * p2, 0, 0.1) * rbinom(p1 * p2, 1, 0.1), p1 , p2)
Mu <- V * array(RH(X2, RH(X1, Beta)), c(n1, n2, n3))
Y <- array(rpois(n1 * n2 * n3, exp(Mu)), dim = c(n1, n2, n3))
```

```
system.time(fit <- glamlassoS(X, Y, V, family = "poisson", nu = 0.1))

modelno <- length(fit$lambda)
plot(c(Beta), type = "h", ylim = range(Beta, fit$coef[, modelno]))
points(c(Beta))
lines(fit$coef[ , modelno], col = "red", type = "h")
```

`objective` *Compute objective values*

Description

Computes the objective values of the penalized log-likelihood problem for the models implemented in the package glamlasso.

Usage

```
objective(Y,
          Weights,
          X,
          Beta,
          lambda,
          penalty.factor,
          family,
          penalty)
```

Arguments

`Y`	The response values, an array of size $n_1 \times \cdots \times n_d$.
`Weights`	Observation weights, an array of size $n_1 \times \cdots \times n_d$.
`X`	A list containing the tensor components of the tensor design matrix, each of size $n_i \times p_i$.
`Beta`	A coefficient matrix of size $p_1 \cdots p_d \times$ `nlambda`.
`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.
`family`	A string specifying the model family (essentially the response distribution).
`penalty`	A string specifying the penalty.

Value

A vector of length `length(lambda)` containing the objective values for each `lambda` value.

Examples

```
## Not run:
n1 <- 65; n2 <- 26; n3 <- 13; p1 <- 13; p2 <- 5; p3 <- 4
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) * rbinom(p1 * p2 * p3, 1, 0.1), c(p1 , p2, p3))
mu <- RH(X3, RH(X2, RH(X1, Beta)))
Y <- array(rnorm(n1 * n2 * n3, mu), dim = c(n1, n2, n3))
fit <- glamlasso(list(X1, X2, X3), Y, family = "gaussian", penalty = "lasso", iwls = "exact")
objfit <- objective(Y, NULL, list(X1, X2, X3), fit$coef, fit$lambda, NULL, fit$family)
plot(objfit, type = "l")

## End(Not run)
```

---

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

Arguments

`object`	An object of Class glamlasso, produced with `glamlasso`.
`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 $d \in \{2, 3\}$.

Author(s)

Adam Lund

Examples

```
n1 <- 65; n2 <- 26; n3 <- 13; p1 <- 13; p2 <- 5; p3 <- 4
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) * rbinom(p1 * p2 * p3, 1, 0.1), c(p1 , p2, p3))
mu <- RH(X3, RH(X2, RH(X1, Beta)))
Y <- array(rnorm(n1 * n2 * n3, mu), dim = c(n1, n2, n3))
fit <- glamlasso(list(X1, X2, X3), Y)

##new data in matrix form
x <- matrix(rnorm(p1 * p2 * p3), nrow = 1)
predict(fit, x = x)[[100]]

##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))[[100]]
```

---

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

Description

This function will print some information about the glamlasso object.

Usage

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

Arguments

x	A glamlasso object
...	ignored

Details

For the call that produced the object x a two-column data.frame with columns Df and lambda is created. The Df column is the number of nonzero coefficients and lambda is the sequence of penalty parameters.

Value

Prints the data.frame described under Details.

Author(s)

Adam Lund

Examples

```
n1 <- 65; n2 <- 26; n3 <- 13; p1 <- 13; p2 <- 5; p3 <- 4
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) * rbinom(p1 * p2 * p3, 1, 0.1), c(p1 , p2, p3))
mu <- RH(X3, RH(X2, RH(X1, Beta)))
Y <- array(rnorm(n1 * n2 * n3, mu), dim = c(n1, n2, n3))
fit <- glamlasso(list(X1, X2, X3), Y)
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.

Index