Package 'fuser'

May 8, 2026

Title Fused Lasso for High-Dimensional Regression over Groups

Version 1.0.1

Description Enables high-dimensional penalized regression across heterogeneous subgroups. Fusion penalties are used to share information about the linear parameters across subgroups. The underlying model is described in detail in Dondelinger and Mukherjee (2017) <doi:10.48550/arXiv.1611.00953>.

Depends R (>= 3.2.0)

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

Suggests testthat, ggplot2, knitr, rmarkdown

Imports Matrix, irlba, Rcpp, glmnet, RSpectra

LinkingTo Rcpp, RcppEigen

VignetteBuilder knitr

NeedsCompilation yes

Author Frank Dondelinger [aut, cre],
Olivier Wilkinson [aut]

Maintainer Frank Dondelinger <fdondelinger.work@gmail.com>

Repository CRAN

Date/Publication 2018-06-17 20:22:54 UTC

Contents

bigeigen 2
fusedL2DescentGLMNet 2
fusedLassoProximal 3
fusedLassoProximalIterationsTaken 5
generateBlockDiagonalMatrices 6

Index **8**

bigeigen *Big eigenvalue calculation*

Description

Calculate maximal eigenvalue of `t(X) %*% X` for big matrices using singular value decomposition.

Usage

```
bigeigen(X, method = "RSpectra")
```

Arguments

`X`	matrix to be evaluated (can be a Matrix object).
`method`	One of 'irlba' or 'RSpectra'

Value

The maximal eigenvalue.

fusedL2DescentGLMNet *Optimise the fused L2 model with glmnet (using transformed input data)*

Description

Optimise the fused L2 model with glmnet (using transformed input data)

Usage

```
fusedL2DescentGLMNet(transformed.x, transformed.x.f, transformed.y, groups,
  lambda, gamma = 1, ...)
```

Arguments

`transformed.x`	Transformed covariates (output of generateBlockDiagonalMatrices)
`transformed.x.f`	Transformed fusion constraints (output of generateBlockDiagonalMatrices)
`transformed.y`	Transformed response (output of generateBlockDiagonalMatrices)
`groups`	Grouping factors for samples (a vector of size n, with K factor levels)
`lambda`	Sparsity penalty hyperparameter
`gamma`	Fusion penalty hyperparameter
`...`	Further options passed to glmnet.

Value

Matrix of fitted beta values.

A matrix with the linear coefficients for each group (p by k).

Examples

```
#' set.seed(123)

# Generate simple heterogeneous dataset
k = 4 # number of groups
p = 100 # number of covariates
n.group = 15 # number of samples per group
sigma = 0.05 # observation noise sd
groups = rep(1:k, each=n.group) # group indicators
# sparse linear coefficients
beta = matrix(0, p, k)
nonzero.ind = rbinom(p*k, 1, 0.025/k) # Independent coefficients
nonzero.shared = rbinom(p, 1, 0.025) # shared coefficients
beta[which(nonzero.ind==1)] = rnorm(sum(nonzero.ind), 1, 0.25)
beta[which(nonzero.shared==1),] = rnorm(sum(nonzero.shared), -1, 0.25)

X = lapply(1:k,
           function(k.i) matrix(rnorm(n.group*p),
                                n.group, p)) # covariates
y = sapply(1:k,
           function(k.i) X[[k.i]] %*% beta[,k.i] +
                         rnorm(n.group, 0, sigma)) # response
X = do.call('rbind', X)

# Pairwise Fusion strength hyperparameters (tau(k,k'))
# Same for all pairs in this example
G = matrix(1, k, k)

# Generate block diagonal matrices
transformed.data = generateBlockDiagonalMatrices(X, y, groups, G)

# Use L2 fusion to estimate betas (with near-optimal information
# sharing among groups)
beta.estimate = fusedL2DescentGLMNet(transformed.data$X,
                                     transformed.data$X.fused,
                                     transformed.data$Y, groups,
                                     lambda=c(0,0.001,0.1,1),
                                     gamma=0.001)
```

`fusedLassoProximal` *Fused lasso optimisation with proximal-gradient method. (Chen et al. 2010)*

Description

Fused lasso optimisation with proximal-gradient method. (Chen et al. 2010)

Usage

```
fusedLassoProximal(X, Y, groups, lambda, gamma, G, mu = 1e-04, tol = 1e-06,
  num.it = 1000, lam.max = NULL, c.flag = FALSE, intercept = TRUE,
  penalty.factors = NULL, conserve.memory = p >= 10000, scaling = TRUE)
```

Arguments

`X`	matrix of covariates (n by p)
`Y`	vector of responses (length n)
`groups`	vector of group indicators (length n)
`lambda`	Sparsity hyperparameter (accepts scalar value only)
`gamma`	Fusion hyperparameter (accepts scalar value only)
`G`	Matrix of pairwise group information sharing weights (K by K)
`mu`	Smoothness parameter for proximal optimization
`tol`	Tolerance for optimization
`num.it`	Number of iterations
`lam.max`	Maximal eigenvalue of `t(X) %*% X` (will be calculate if not provided)
`c.flag`	Whether to use Rcpp for certain calculations (see Details).
`intercept`	Whether to include a (group-specific) intercept term.
`penalty.factors`	Weights for sparsity penalty.
`conserve.memory`	Whether to calculate XX and XY on the fly, conserving memory at the cost of speed. (True by default iff p >= 10000)
`scaling`	if TRUE, scale the sum-squared loss for each group by 1/n_k where n_k is the number of samples in group k.

Details

The proximal algorithm uses `t(X) %*% X` and `t(X) %*% Y`. The function will attempt to pre-calculate these values to speed up computation. This may not always be possible due to memory restrictions; at present this is only done for p < 10,000. When p > 10,000, crossproducts are calculated explicitly; calculation can be speeded up by using Rcpp code (setting c.flag=TRUE).

Value

A matrix with the linear coefficients for each group (p by k).

Examples

```
set.seed(123)
# Generate simple heterogeneous dataset
k = 4 # number of groups
p = 100 # number of covariates
n.group = 15 # number of samples per group
sigma = 0.05 # observation noise sd
groups = rep(1:k, each=n.group) # group indicators
# sparse linear coefficients
beta = matrix(0, p, k)
nonzero.ind = rbinom(p*k, 1, 0.025/k) # Independent coefficients
nonzero.shared = rbinom(p, 1, 0.025) # shared coefficients
beta[which(nonzero.ind==1)] = rnorm(sum(nonzero.ind), 1, 0.25)
beta[which(nonzero.shared==1),] = rnorm(sum(nonzero.shared), -1, 0.25)

X = lapply(1:k,
           function(k.i) matrix(rnorm(n.group*p),
                                n.group, p)) # covariates
y = sapply(1:k,
           function(k.i) X[[k.i]] %*% beta[,k.i] +
                         rnorm(n.group, 0, sigma)) # response
X = do.call('rbind', X)

# Pairwise Fusion strength hyperparameters (tau(k,k'))
# Same for all pairs in this example
G = matrix(1, k, k)

# Use L1 fusion to estimate betas (with near-optimal sparsity and
# information sharing among groups)
beta.estimate = fusedLassoProximal(X, y, groups, lambda=0.01, tol=3e-3,
                                   gamma=0.01, G, intercept=FALSE,
                                   num.it=500)
```

`fusedLassoProximalIterationsTaken` *Following a call to fusedLassoProximal, returns the actual number of iterations taken.*

Description

Following a call to fusedLassoProximal, returns the actual number of iterations taken.

Usage

```
fusedLassoProximalIterationsTaken()
```

Value

Number of iterations performed in the previous call to fusedLassoProximal.

`generateBlockDiagonalMatrices` *Generate block diagonal matrices to allow for fused L2 optimization with glmnet.*

Description

Generate block diagonal matrices to allow for fused L2 optimization with glmnet.

Usage

```
generateBlockDiagonalMatrices(X, Y, groups, G, intercept = FALSE,
  penalty.factors = rep(1, dim(X)[2]), scaling = TRUE)
```

Arguments

`X`	covariates matrix (n by p).
`Y`	response vector (length n).
`groups`	vector of group indicators (ideally factors, length n)
`G`	matrix representing the fusion strengths between pairs of groups (K by K). Zero entries are assumed to be independent pairs.
`intercept`	whether to include an (per-group) intercept in the model
`penalty.factors`	vector of weights for the penalization of each covariate (length p)
`scaling`	Whether to scale each subgroup by its size. See Details for an explanation.

Details

We use the `glmnet` package to perform fused subgroup regression. In order to achieve this, we need to reformulate the problem as Y' = X'beta', where Y' is a concatenation of the responses Y and a vector of zeros, X' is a a matrix consisting of the block-diagonal matrix n by pK matrix X, where each block contains the covariates for one subgroups, and the choose(K,2)*p by pK matrix encoding the fusion penalties between pairs of groups. The vector of parameters beta' of length pK can be rearranged as a p by K matrix giving the parameters for each subgroup. The lasso penalty on the parameters is handled by glmnet.

One weakness of the approach described above is that larger subgroups will have a larger influence on the global parameters lambda and gamma. In order to mitigate this, we introduce the `scaling` parameter. If `scaling=TRUE`, then we scale the responses and covariates for each subgroup by the number of samples in that group.

Value

A list with components X, Y, X.fused and penalty, where X is a n by pK block-diagonal bigmatrix, Y is a re-arranged bigvector of length n, and X.fused is a choose(K,2)*p by pK bigmatrix encoding the fusion penalties.

Examples

```
set.seed(123)

# Generate simple heterogeneous dataset
k = 4 # number of groups
p = 100 # number of covariates
n.group = 15 # number of samples per group
sigma = 0.05 # observation noise sd
groups = rep(1:k, each=n.group) # group indicators
# sparse linear coefficients
beta = matrix(0, p, k)
nonzero.ind = rbinom(p*k, 1, 0.025/k) # Independent coefficients
nonzero.shared = rbinom(p, 1, 0.025) # shared coefficients
beta[which(nonzero.ind==1)] = rnorm(sum(nonzero.ind), 1, 0.25)
beta[which(nonzero.shared==1),] = rnorm(sum(nonzero.shared), -1, 0.25)

X = lapply(1:k,
           function(k.i) matrix(rnorm(n.group*p),
                                n.group, p)) # covariates
y = sapply(1:k,
           function(k.i) X[[k.i]] %*% beta[,k.i] +
                         rnorm(n.group, 0, sigma)) # response
X = do.call('rbind', X)

# Pairwise Fusion strength hyperparameters (tau(k,k'))
# Same for all pairs in this example
G = matrix(1, k, k)

# Generate block diagonal matrices
transformed.data = generateBlockDiagonalMatrices(X, y, groups, G)
```

Index

bigeigen, 2

fusedL2DescentGLMNet, 2
fusedLassoProximal, 3
fusedLassoProximalIterationsTaken, 5

generateBlockDiagonalMatrices, 6