Package 'hsm'

March 22, 2018

Type Package

Title A Path-Based BCD for Proximal Function of Latent Group Lasso

Description Implementation of the block coordinate descent procedure for solving the proximal function of latent group Lasso, highlighted by decomposing a DAG into several non-overlapping path graphs, and getting closed-form solution for each path graph. The procedure was introduced as Algorithm 4 in Yan and Bien (2017) <doi:10.1214/17-STS622> ``Hierarchical Sparse Modeling: A Choice of Two Group Lasso Formulations", and the closed-form solution for each path graph is solved in Algorithm 3 of that paper.

Version 0.2.0

Author Xiaohan Yan [aut, cre], Jacob Bien [aut, cre]

Maintainer Xiaohan Yan <xy257@cornell.edu>

Depends R (>= 3.2.1)

Suggests knitr

License GPL-3

LazyData TRUE

VignetteBuilder knitr

RoxygenNote 6.0.1

URL https://github.com/yanxht/hsm

BugReports https://github.com/yanxht/hsm/issues

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-03-22 15:31:13 UTC

R topics documented:

`hsm-package` *Block coordinate descent based on path graphs for proximal operator of latent group Lasso*

Description

`hsm` is the R package implementing Algorithm 4 of Yan & Bien (2017) that uses path-graph-based BCD to solve the proximal operator of latent group Lasso in hierarchical sparse modeling (HSM). The algorithm solves the proximal operator using BCD that circles over path graphs decomposed a directed acyclic graph (DAG).

Details

The package is designed for situation in which latent group Lasso is used to achieve hierarchical sparsity pattern in a DAG. The hierarchical sparsity pattern is one when parameters embedded in a node being set to zero, all the parameters embedded in the descendant nodes in DAG are zeroed out as well.

Its main functions are `hsm`, `hsm.path`.

Author(s)

Xiaohan Yan <xy257@cornell.edu>, Jacob Bien

References

Yan, X. and Bien, J. (2017). Hierarchical Sparse Modeling: A Choice of Two Group Lasso Formulations. Statist. Sci. 32, no. 4, 531–560. doi:10.1214/17-STS622.

`ancestor.find` *Find ancestor nodes for a node in directed acyclic graph.*

Description

Recursively finds all ancestor nodes in DAG for node with the given index.

Usage

```
ancestor.find(index, map, n.nodes)
```

Arguments

index	Index of the node currently at.
map	Matrix of n.edges-by-2 dimension, where n.edges is the number of directed edges in DAG. The first column has indices of nodes that edges directing from, whereas the second column gives the indices of nodes the corresponding edges directing towards.
n.nodes	Number of nodes in DAG.

Value

Returns a length-n.nodes vector of binary values for which 1 indicates the corresponding node is an ancestor node and 0 indicates it is not.

---

hsm *Solves proximal operator of latent group lasso in Hierarchical Sparse Modeling.*

Description

Solves proximal operator of the latent group Lasso appearing in Yan & Bien (2017)

$$min_\beta ||y - \beta||_2^2 + lam * \Omega(\beta; w)$$

where $\Omega(\beta; w) = min_{sum_l v^{(l)} = \beta; supp(v^{(l)}) \subset g_l} w_l * ||v^{(l)}||_2$ is known as the latent group lasso penalty as defined in Jacob et al. (2009). In the problem, β is a length-p parameter vector and its elements are embedded in a directed acyclic graph (DAG). The desired sparsity pattern is a subgraph of the DAG such that if β_i embedded in node i are set to zero, all the parameters embedded in the descendant nodes of i are zeroed out as well. The problem is solved by breaking down the DAG into several path graphs for which closed-form solutions are available for the proximal operator corresponding with each path graph, and performing block coordinate descent across the path graphs. See Section 4.3 of the paper for more details and explanations.

Usage

```
hsm(y, lam, w = NULL, map, var, assign = NULL, w.assign = NULL,
  get.penalval = FALSE, tol = 1e-08, maxiter = 10000, beta.ma = NULL)
```

Arguments

y	Length-p vector used in proximal operator.
lam	Non-negative tuning parameter that controls the sparsity level.
w	Length-n.nodes vector of positive values for which w_l gives the weight for g_l, where n.nodes is the number of nodes in DAG. If this is NULL, w_l = sqrt(|g_l|) will be used. Necessary condition is w_l increases with |g_l|.

map	Matrix of n.edges-by-2 dimension, where n.edges is the number of directed edges in DAG. The first column has indices of nodes that edges directing from, whereas the second column gives the indices of nodes the corresponding edges directing towards. If a node indexed i does not have edges linked to it, record the corresponding row as map[i, NA].
var	Length-n.nodes list for which the lth element contains the indices of variables embedded in the lth node.
assign	Matrix of p columns that gives the assignments of variables over different path graphs. Each row of assign corresponds to a path graph decomposed from DAG. If this is NULL, hsm first break down DAG into different path graphs, and then give value to assign afterwards, based on map and var; otherwise, map and var are ignored. Refer to paths for more details.
w.assign	List of length nrow(assign), for which the lth element contains the weights corresponding to the lth row of assign (the lth path graph). For example, if the lth path graph is made up of three nodes indexed with {3, 4, 6, 8}, w.assign[[l]] = {w_3, w_4, w_6, w_8}. If this is NULL, hsm will give value to w.assign, along with assign; otherwise, map and var are ignored. Refer to paths for more details.
get.penalval	If TRUE, $lam * \Omega(\beta; w)$ are computed and returned, otherwise NA is returned.
tol	Tolerance level used in BCD. Convergence is assumed when no parameter of interest in each path graph changes by more than tol in BCD.
maxiter	Upperbound of the number of iterations that BCD to perform.
beta.ma	n.paths-by-p matrix of initialization of beta value in the n.paths path graphs decomposed from DAG. Do not use unless you know the decomposition of DAG.

Details

See Section 2.2 of the paper for problem setup and group structure specifications. See Figure 7 in Section 4.3 for an example of decomposing DAG into path graphs. See Algorithm 4 in paper for details of the path-based BCD.

Value

Returns an estimate of the solution to the proximal operator of the latent group Lasso. The returned value is an exact solution if the DAG is a directed path graph.

beta	A length-p vector giving solution to the proximal operator defined above.
ite	Number of cycles of BCD performed.
penalval	Value of the penalty $lam * \Omega(\beta; w)$ if get.penalval is TRUE, otherwise NA.
assign	Value of assign.
w.assign	Value of w.assign.
beta.ma	n.paths-by-p matrix of decomposed beta values for all the decomposed path graphs. The beta values are from the last iteration in hsm.

References

Yan, X. and Bien, J. (2017). Hierarchical Sparse Modeling: A Choice of Two Group Lasso Formulations. Statist. Sci. 32, no. 4, 531–560. doi:10.1214/17-STS622.

Jacob, L., Obozinski, G. and Vert, J. (2009). Group Lasso with Overlap and Graph Lasso. In Proceedings of the 26th Annual International Conference on Machine Learning. ICML'09 433-440. ACM, New York.

See Also

hsm.path

paths

Examples

```
# The following example appears in Figure 7 of Yan & Bien (2015).
# Generate map defining DAG.
map <- matrix(0, ncol=2, nrow=8)
map[1, ] <- c(1, 2)
map[2, ] <- c(2, 7)
map[3, ] <- c(3, 4)
map[4, ] <- c(4, 6)
map[5, ] <- c(6, 7)
map[6, ] <- c(6, 8)
map[7, ] <- c(3, 5)
map[8, ] <- c(5, 6)
# Assume one parameter per node.
# Let parameter and node share the same index.
var <- as.list(1:8)
set.seed(100)
y <- rnorm(8)
result <- hsm(y=y, lam=0.5, map=map, var=var, get.penalval=TRUE)

# Another example in which DAG contains two separate nodes
map <- matrix(0, ncol=2, nrow=2)
map[1, ] <- c(1, NA)
map[2, ] <- c(2, NA)
# Assume ten parameters per node.
var <- list(1:10, 11:20)
set.seed(100)
y <- rnorm(20)
lam <- 0.5
result <- hsm(y=y, lam=lam, map=map, var=var, get.penalval=TRUE)
# The solution is equivalent to performing group-wise soft-thresholdings
beta.st <- c(y[1:10] * max(0, 1 - lam * sqrt(10) / norm(y[1:10], "2")),
          y[11:20] * max(0, 1 - lam * sqrt(10) / norm(y[11:20], "2")))
all.equal(result$beta, beta.st)
```

hsm.path *Solves proximal operator of latent group Lasso over a grid of lam values.*

Description

See hsm for the problem that is solved. If lamlist is not provided, a grid of lam values will be constructed starting at lammax, the smallest value of lam for which the solution is completely sparse.

Usage

```
hsm.path(y, nlam = 20, flmin = 0.01, lamlist = NULL, w = NULL, map, var,
  assign = NULL, w.assign = NULL, get.penalval = FALSE, tol = 1e-08,
  maxiter = 10000)
```

Arguments

y	Length-p vector used in proximal operator.
nlam	Number of lam values to include in grid. Default value is 20.
flmin	Ratio between the smallest lam and largest lam in grid. Default value is 0.01. Increasing its value will give more sparse solutions.
lamlist	A grid of lam values to use. If this is NULL, then a grid of nlam lam values equally spaced in the logarithm scale between lammax and lammax * flmin are used; otherwise, nlam and flmin are ignored.
w	Length-n.nodes vector of positive values for which w_l gives the weight for g_l, where n.nodes is the number of nodes in DAG. If this is NULL, w_l = sqrt(|g_l|) will be used. Necessary condition is w_l increases with |g_l|.
map	Matrix of n.edges-by-2 dimension, where n.edges is the number of directed edges in DAG. The first column has indices of nodes that edges directing from, whereas the second column gives the indices of nodes the corresponding edges directing towards. If a node indexed i does not have edges linked to it, record the corresponding row as map[i, NA].
var	Length-n.nodes list for which the lth element contains the indices of variables embedded in the lth node.
assign	Matrix of p columns that gives the assignments of variables over different path graphs. Each row of assign corresponds to a path graph decomposed from DAG. If this is NULL, hsm first break down DAG into different path graphs, and then give value to assign afterwards, based on map and var; otherwise, map and var are ignored. Refer to paths for more details.
w.assign	List of length nrow(assign), for which the lth element contains the weights corresponding to the lth row of assign (the lth path graph). For example, if the lth path graph is made up of three nodes indexed with {3, 4, 6, 8}, w.assign[[l]] = {w_3, w_4, w_6, w_8}. If this is NULL, hsm will give value to w.assign, along with assign; otherwise, map and var are ignored. Refer to paths for more details.

get.penalval	If TRUE, $lam * \Omega(\beta; w)$ are computed and returned, otherwise NA is returned.
tol	Tolerance level used in BCD. Convergence is assumed when no parameter of interest in each path graph changes by more than tol in BCD.
maxiter	Upperbound of the number of iterations that BCD to perform.

Value

Returns a sequence of estimates of the solution to the proximal operator of the latent group Lasso. The returned solutions are exact ones if the DAG is a directed path graph.

lamlist	Grid of lam values used.
beta.m	A nlam-by-p matrix where beta.m[i,] gives the ith solution to the proximal operator, corresponding to the ith lam value in the grid.
penalval.m	Length-nlam vector of values of the penalty $lam*\Omega(\beta; w)$ where penalval.m[i,] correponds to the ith lam value in the grid, if get.penalval is TRUE. If get.penalval is FALSE, NA is returned.
assign	Value of assign.
w.assign	Value of w.assign.

See Also

hsm

paths

lam.max.hsm

Examples

```
# The following example appears in Figure 7 of Yan & Bien (2015).
# Generate map defining DAG.
map <- matrix(0, ncol=2, nrow=8)
map[1, ] <- c(1,2)
map[2, ] <- c(2,7)
map[3, ] <- c(3,4)
map[4, ] <- c(4,6)
map[5, ] <- c(6,7)
map[6, ] <- c(6,8)
map[7, ] <- c(3,5)
map[8, ] <- c(5,6)
# Assume one parameter per node.
# Let parameter and node share the same index.
var <- as.list(1:8)
set.seed(100)
y <- rnorm(8)
result <- hsm(y=y, lam=0.5, map=map, var=var, get.penalval=TRUE)
result.path <- hsm.path(y=y, map=map, var=var, get.penalval=TRUE)
```

lam.max.hsm *Computes the smallest lam value such that beta = 0.*

Description

Computes `lammax`, the smallest value of lam for which `hsm` gives a completely sparse solution.

Usage

```
lam.max.hsm(y, assign, w.assign)
```

Arguments

`y`	Length-`p` vector used in proximal operator.
`assign`	Matrix of `p` columns that gives the assignments of variables over different path graphs. Each row of `assign` corresponds to a path graph decomposed from DAG. Refer to `paths` for more details.
`w.assign`	List of length `nrow(assign)`, for which the `l`th element contains the weights corresponding to the `l`th row of `assign` (the `l`th path graph). For example, if the `l`th path graph is made up of three nodes indexed with `{3, 4, 6, 8}`, `w.assign[[l]] = {w_3, w_4, w_6, w_8}`. Refer to `paths` for more details.

See Also

`hsm.path`

path.find *Find all path graphs originated from a given root.*

Description

Recursively find all possible path graphs originated from a given root in DAG.

Usage

```
path.find(index, map)
```

Arguments

`index`	Index of a root node (a node whose index never appears in `map[, 2]`).
`map`	Matrix of `n.edges`-by-2 dimension, where `n.edges` is the number of directed edges in DAG. The first column has indices of nodes that edges directing from, whereas the second column gives the indices of nodes the corresponding edges directing towards.

Value

Returns a list of path graphs originated from root index, for which the ith element of the returned list is a vector of indices of nodes in the ith path graph.

---

paths *Generate* assign *and* w.assign.

Description

For every root node in DAG defined by map, paths circles over all possible path graphs, picks up the one that consists of the most unmarked node, and then marks the nodes in the path graph that have been selected. paths won't move to the next root node, until all the descendant nodes of the current root have been marked.

Usage

```
paths(map, var, w = NULL)
```

Arguments

map	Matrix of n.edges-by-2 dimension, where n.edges is the number of directed edges in DAG. The first column has indices of nodes that edges directing from, whereas the second column gives the indices of nodes the corresponding edges directing towards. If a node indexed i does not have edges linked to it, record the corresponding row as map[i, NA].
var	Length-n.nodes list where n.nodes is the number of nodes in DAG and for which the lth element contains the indices of variables embedded in the lth node.
w	Length-n.nodes vector of positive values for which w_l gives the weight for g_l, where n.nodes is the number of nodes in DAG. If this is NULL, w_l = sqrt(|g_l|) will be used. Necessary condition is w_l increases with |g_l|.

Value

Returns assign, a matrix of p columns that gives the assignments of variables over selected path graphs, and w.assign, a list of the same length as the number of rows in assign.

assign	Each row of assign corresponds to a path graph decomposed from DAG.
w.assign	The lth element of the list contains the weights corresponding to the lth row of assign (the lth path graph).

Examples

```
# The following example appears in Figure 7 of Yan & Bien (2015).
# Generate map defining DAG.
map <- matrix(0, ncol=2, nrow=8)
map[1, ] <- c(1, 2)
map[2, ] <- c(2, 7)
map[3, ] <- c(3, 4)
map[4, ] <- c(4, 6)
map[5, ] <- c(6, 7)
map[6, ] <- c(6, 8)
map[7, ] <- c(3, 5)
map[8, ] <- c(5, 6)
# Assume two parameters per node.
var <- as.list(data.frame(t(matrix(1:16, ncol=2, byrow=TRUE))))
paths.result <- paths(map, var)
paths.result$assign
paths.result$w.assign
```

Index