Package ‘flsa’

March 13, 2019

Type Package
Title Path Algorithm for the General Fused Lasso Signal Approximator
Version 1.5.1
Date 2019-03-11
Author Holger Hoefling
Maintainer Holger Hoefling <hhoeflin@gmail.com>
Description Implements a path algorithm for the Fused Lasso Signal Approximator.
For more details see the help files or the article by Hoefling (2009) <arXiv:0910.0526>.
License GPL-2
Depends R (>= 2.0.0)
SystemRequirements C++11
Suggests testthat
NeedsCompilation yes
Repository CRAN
Date/Publication 2019-03-13 22:50:03 UTC

R topics documented:

flsa ................................................................. 1
is.connListObj .................................................... 3

Index

    flsa  Fused Lasso Signal Approximator

Description

These functions are the main interface functions for calculating FLSA solutions
Usage

```r
flsa(y, lambda1=0, lambda2=NULL, connListObj = NULL, splitCheckSize=1e+09, verbose=FALSE, thr = 1e-09, maxGrpNum=4*length(y))
flsaTopDown(y, lambda1=0, groups=1:length(y), lambda2=NULL)
flsaGetSolution(solObj, lambda1=0, lambda2=NULL, dim=NULL)
```

Arguments

- **y**: response variable; numeric
- **lambda1**: penalty parameter vector (non-negative) for the absolute values of the coefficients; numeric
- **lambda2**: penalty parameter vector (non-negative) for the difference of certain coefficients; numeric
- **groups**: Return solutions for which the given number of groups is present - solutions found exactly at the breakpoint
- **connListObj**: an object specifying which differences are to be penalized by lambda2. If NULL, then the dimensionality of y is being used. If y is a vector, the differences of neighbouring coefficients are penalized. If y is a matrix, differences of neighbouring coefficients in 2 dimensions are being penalized. For more information see `connListObj`
- **splitCheckSize**: a parameter specifying from which size on, groups of variables are not being checked for breaking up; can be used to reduce computation time; may lead to inaccurate results
- **solObj**: Solution object as returned by FLSA if lambda2=NULL
- **dim**: dimensions how the result should be formatted for a specific lambda. Used to format the 2-dimensional FLSA as a matrix in the response. For this, just include the dimensions of y as `dim`
- **verbose**: print status messages during fitting
- **thr**: the error threshold used in the algorithm
- **maxGrpNum**: if every step of the algorithm, a group with a higher number is generated; this limits the number of steps the algorithm can take

Details

`flsa` is the main function for calculate a `flsa` fit. If lambda2=NULL, then it returns an object that encodes the whole solution path. Solutions for specific values of lambda1 and lambda2 can then be obtained by using `flsaGetSolution`.

`flsaTopDown` calculates the solution of the 1-dimensional FLSA, starting at large values of lambda2. If only solutions for large values of lambda2 are needed, this is more efficient.

Author(s)

Holger Hoefling
See Also

connListObj

Examples

library(flsa)
# generate some artificial data, 1 and 2 dimensional
y <- rnorm(100)
y2Dim = matrix(rnorm(100), ncol=10)

### apply function flsa and get solution directly
lambda2= 0:10/10
res <- flsa(y, lambda2=lambda2)
res2Dim <- flsa(y2Dim, lambda2=lambda2)

### apply the function and get the solution later
resSolObj <- flsa(y, lambda2=NULL)
resSolObjTopDown <- flsaTopDown(y)
resSolObj2Dim <- flsa(y2Dim, lambda2=NULL)

res2 <- flsaGetSolution(resSolObj, lambda2=lambda2)
### here note that the solution object does not store that the input was 2 dimensional
### therefore, res2Dim does not give out the solution as a 2
### dimensional matrix (unlike the direct version above)
res2Dim2 <- flsaGetSolution(resSolObj2Dim, lambda2=lambda2)
the coefficient corresponding to the current list element is linked (i.e. the difference of the two coefficients is being penalized). I.e., assume that value $j$ is a member of the list under list element $i$. Then this means that coefficient $i-1$ and coefficient $j$ are being penalized. To understand this, consider that R-lists when viewed in C-code are being numbered starting with 0, not 1 and note that all computation is being done in C-code.

Furthermore, the connection list object has class connListObj.

Also note that the vectors in the list are of type integer not numeric. An empty vector should be set to NULL.

Author(s)
Holger Hoefling

See Also
connListObj

Examples

```r
connList <- vector("list", 4)
y <- 1:4

class(connList) = "connListObj"
connList[[1]] = as.integer(c(1,2))
connList[[2]] = as.integer(c(0,3))
connList[[3]] = as.integer(c(3,0))
connList[[4]] = as.integer(c(2,1))
names(connList) <- as.character(0:3) # not necessary, just for illustration

res <- flsa(y, connListObj=connList)
res2 <- flsa(matrix(y, nrow=2))

res$beginLambda
res2$beginLambda
flsaGetSolution(res, lambda2=c(0, 0.5, 1))
flsaGetSolution(res2, lambda2=c(0, 0.5, 1))
```
Index

*Topic **multivariate**
  flsa, 1
  is.connListObj, 3

*Topic **regression**
  flsa, 1
  is.connListObj, 3

ConnListObj(is.connListObj), 3
connListObj, 2–4
connListObj(is.connListObj), 3

FLSA (flsa), 1
flsa, 1
flsaGetSolution (flsa), 1
flsaTopDown (flsa), 1

is.connListObj, 3