Package ‘isotone’

July 24, 2015

Type Package
Title Active Set and Generalized PAVA for Isotone Optimization
Version 1.1-0
Date 2015-07-24
Author Patrick Mair [aut, cre], Jan De Leeuw [aut], Kurt Hornik [aut]
Maintainer Patrick Mair <mair@fas.harvard.edu>
Description Contains two main functions: one for solving general isotone regression problems using the pool-adjacent-violators algorithm (PAVA); another one provides a framework for active set methods for isotone optimization problems with arbitrary order restrictions. Various types of loss functions are prespecified.
Imports graphics, stats, nnls
Depends R (>= 3.0.2)
License GPL-2
URL http://r-forge.r-project.org/projects/psychor/
LazyData yes
LazyLoad yes
ByteCompile yes
NeedsCompilation yes
Repository CRAN
Date/Publication 2015-07-24 12:16:22

R topics documented:

activeSet ................................................................. 2
aSolver ................................................................. 5
dSolver ................................................................. 6
eSolver ................................................................. 7
fSolver ................................................................. 8
### activeSet

**Active Set Methods for Isotone Optimization**

#### Description

Isotone optimization can be formulated as a convex programming problem with simple linear constraints. This function offers active set strategies for a collection of isotone optimization problems pre-specified in the package.

#### Usage

```
activeSet(isomat, mySolver = "LS", x0 = NULL, ups = 1e-12, check = TRUE, maxiter = 100, ...)  
```

#### Arguments

- **isomat**: Matrix with 2 columns that contains isotonicity conditions, i.e., for row i it holds that fitted value i column 1 <= fitted value i column 2 (see examples)
- **mySolver**: Various functions are pre-defined (see details). Either to function name or the corresponding string equivalent can be used. For user-specified functions `fsolver` with additional arguments can be used (see details as well).
- **x0**: Feasible starting solution. If NULL the null-vector is used internally.
- **ups**: Upper boundary
- **check**: If TRUE, KKT feasibility checks for isotonicity of the solution are performed
- **maxiter**: Iteration limit
- **...**: Additional arguments for the various solvers (see details)
Details

The following solvers are specified. Note that y as the vector of observed values and weights as the vector of weights need to provided through ... for each solver (except for fSolver() and sSolver()). Some solvers need additional arguments as described in the corresponding solver help files. More technical details can be found in the package vignette.

The pre-specified solvers are the following (we always give the corresponding string equivalent in brackets): lsSolver() ("LS") for least squares with diagonal weights, aSolver() ("asyLS") for asymmetric least squares, dSolver() ("L1") for the least absolute value, eSolver() ("L1eps") minimizes l1-approximation. hSolver() ("huber") for Huber loss function, iSolver() ("SILF") for SILF loss (support vector regression), lfSolver() ("GLS") for general least squares with non-diagonal weights, mSolver() ("chebyshev") for Chebyshev L-inf norm, oSolver() ("Lp") for L-p power norm, pSolver() ("quantile") for quantile loss function, and finally sSolver() ("poisson") for Poisson likelihood.

fSolver() for user-specified arbitrary differentiable functions. The arguments fobj (target function ) and gobj (first derivative) must be provided plus any additional arguments used in the definition of fobj.

Value

Generates an object of class activeset.

- x: Vector containing the fitted values
- y: Vector containing the observed values
- lambda: Vector with Lagrange multipliers
- fval: Value of the target function
- constr.vals: Vector with the values of isotonicity constraints
- Alambda: Constraint matrix multiplied by lambda (should be equal to gradient)
- gradient: Gradient
- isocheck: List containing the KKT checks for stationarity, primal feasibility, dual feasibility, and complementary slackness (>= 0 means feasible)
- niter: Number of iterations
- call: Matched call

Author(s)

Jan de Leeuw, Kurt Hornik, Patrick Mair

References


See Also

gpava, lsSolver, dSolver, mSolver, fSolver, pSolver, lfSolver, oSolver, aSolver, eSolver, sSolver, hSolver, iSolver
Examples

```r
## Data specification
set.seed(12345)
y <- rnorm(9)           ## normal distributed response values
w1 <- rep(1, 9)        ## unit weights
Atot <- cbind(1:8, 2:9) ## Matrix defining isotonicity (total order)
Atot

## Least squares solver (pre-specified and user-specified)
fit.ls1 <- activeSet(Atot, "LS", y = y, weights = w1)
fit.ls1
summary(fit.ls1)
fit.ls2 <- activeSet(Atot, fsSolver, fobj = function(x) sum(w1*(x-y)^2),
gobj = function(x) 2*drop(w1*(x-y)), y = y, weights = w1)

## LS vs. GLS solver (needs weight matrix)
set.seed(12345)
wvec <- 1:9
wmat <- crossprod(matrix(rnorm(81), 9, 9))/9
fit.wls <- activeSet(Atot, "LS", y = y, weights = wvec)
fit.gls <- activeSet(Atot, "GLS", y = y, weights = wmat)

## Quantile regression
fit.qua <- activeSet(Atot, "quantile", y = y, weights = wvec, aw = 0.3, bw = 0.7)

## Mean absolute value norm
fit.abs <- activeSet(Atot, "L1", y = y, weights = w1)

## Lp norm
fit.pow <- activeSet(Atot, "Lp", y = y, weights = w1, p = 1.2)

## Chebyshev norm
fit.che <- activeSet(Atot, "chebyshev", y = y, weights = w1)

## Efron's asymmetric LS
fit.asy <- activeSet(Atot, "asyLS", y = y, weights = w1, aw = 2, bw = 1)

## Huber and SILF loss
fit.hub <- activeSet(Atot, "huber", y = y, weights = w1, eps = 1)
fit.svm <- activeSet(Atot, "SILF", y = y, weights = w1, beta = 0.8, eps = 0.2)

## Negative Poisson log-likelihood
set.seed(12345)
yp <- rpois(9, 5)
x0 <- 1:9
fit.poi <- activeSet(Atot, "poisson", x0 = x0, y = yp)
```
aSolver

Asymmetric Least Squares

Description

Minimizes Efron’s asymmetric least squares regression.

Usage

aSolver(z, a, extra)

Arguments

z Vector containing observed response
a Matrix with active constraints
extra List with element y containing the observed response vector, weights with optional observation weights, weight aw for y > x, and weight bw for y <= x

Details

This function is called internally in activeSet by setting mySolver = aSolver.
**Value**

- **x**: Vector containing the fitted values
- **lbd**: Vector with Lagrange multipliers
- **f**: Value of the target function
- **gx**: Gradient at point x

**References**


**See Also**

`activeSet`

**Examples**

```r
# Fitting isotone regression using active set
set.seed(12345)
y <- rnorm(9)  # response values
w <- rep(1, 9)  # unit weights
btota <- cbind(1:8, 2:9)  # Matrix defining isotonicity (total order)
fit.asy <- activeSet(btota, aSolver, weights = w, y = y, aw = 0.3, bw = 0.5)
```

---

**dSolver**  

**Absolute Value Norm**

**Description**

Solver for the least absolute value norm with optional weights.

**Usage**

`dSolver(z, a, extra)`

**Arguments**

- **z**: Vector containing observed response
- **a**: Matrix with active constraints
- **extra**: List with element `y` containing the observed response vector and weights with optional observation weights

**Details**

This function is called internally in `activeSet` by setting `mySolver = dSolver`. 
Value

- \( x \)  Vector containing the fitted values
- \( lbd \)  Vector with Lagrange multipliers
- \( f \)  Value of the target function
- \( gx \)  Gradient at point \( x \)

See Also

activeSet

Examples

```r
# Fitting weighted absolute norm problem
set.seed(12345)
y <- rnorm(9)  # response values
w <- rep(1, 9)  # unit weights
btota <- cbind(1:8, 2:9)  # Matrix defining isotonicity (total order)
fit.abs <- activeSet(btota, dSolver, weights = w, y = y)
```

Description

Solves an L1 approximation.

Usage

eSolver(z, a, extra)

Arguments

- \( z \)  Vector containing observed response
- \( a \)  Matrix with active constraints
- \( \text{extra} \)  List with element \( y \) containing the observed response vector and weights with optional observation weights, \( \text{eps} \) for the error term

Details

This function is called internally in activeSet by setting \text{mySolver} = eSolver.
Value

- x: Vector containing the fitted values
- lb: Vector with Lagrange multipliers
- f: Value of the target function
- g: Gradient at point x

See Also

activeSet

Examples

```r
## Fitting isotone regression using active set
set.seed(12345)
y <- rnorm(9)               ## response values
w <- rep(1,9)              ## unit weights
eps = 0.01                 ## error term
btota <- cbind(1:8, 9:9)   ## Matrix defining isotonicity (total order)
fit.approx <- activeSet(btota, eSolver, weights = w, y = y, eps = eps)
```

fSolver

**User-Specified Loss Function**

Description

Specification of a differentiable convex loss function.

Usage

fSolver(z, a, extra)

Arguments

- z: Vector containing observed response
- a: Matrix with active constraints
- extra: List with element fobj containing the target function and gobj with the first derivative

Details

This function is called internally in activeSet by setting mySolver = fSolver. It uses optim() with "BFGS" for optimization.
**Value**

- **x**: Vector containing the fitted values
- **1bd**: Vector with Lagrange multipliers
- **f**: Value of the target function
- **gx**: Gradient at point x

**See Also**

- `activeSet`

**Examples**

```r
# Fitting isotone regression using active set (L2-norm user-specified)
set.seed(12345)
y <- rnorm(9)               # response values
w <- rep(1,9)              # unit weights
btota <- cbind(1:8, 2:9)   # Matrix defining isotonicity (total order)
fit.convex <- activeSet(btota, fSolver, fobj = function(x) sum(w*(x-y)^2),
gobj = function(x) 2*drop(w*(x-y)), y = y, weights = w)
```

**Description**

Generalized Pooled-Adjacent-Violators Algorithm (PAVA)

Pooled-adjacent-violators algorithm for general isotone regression problems. It allows for general convex target function, multiple measurements, and different approaches for handling ties.

**Usage**

```r
gpava(z, y, weights = NULL, solver = weighted.mean, ties = "primary", p = NA)
```

**Arguments**

- **z**: Vector of abscissae values
- **y**: Vector or list of vectors of responses
- **weights**: Vector of list of vectors of observation weights
- **solver**: Either `weighted.mean`, `weighted.median`, `weighted.fractile`, or a user-specified function (see below)
- **ties**: Treatment of ties, either "primary", "secondary", or "tertiary"
- **p**: Fractile value between 0 and 1 if `weighted.fractile` is used
Details

A Pool Adjacent Violators Algorithm framework for minimizing problems like

$$\sum_i \sum_j w_{ij} f(y_{ij}, m_i)$$

under the constraint $m_1 \leq \ldots \leq m_n$ with $f$ a convex function in $m$. Note that this formulation allows for repeated data in each block (i.e. each list element of $y$, and hence is more general than the usual pava/isoreg ones.

A solver for the unconstrained $\sum_k w_k f(y_k, m) - \geq min!$ can be specified. Typical cases are $f(y, m) = |y - m|^p$ for $p = 2$ (solved by weighted mean) and $p = 1$ (solved by weighted median), respectively.

Using the weighted.fractile solver corresponds to the classical minimization procedure in quantile regression.

The user can also specify his own function foo(y, w) with responses and weights as arguments. It should return a single numerical value.

Value

Generates an object of class gpava.

- x: Fitted values
- y: Observed response
- z: Observed predictors
- w: Weights
- solver: Convex function
- call: Matched call
- p: Fractile value

Author(s)

Kurt Hornik, Jan de Leeuw, Patrick Mair

References


Examples

data(pituitary)
# different tie approaches
gpava(pituitary[,1], pituitary[,2], ties = "primary")
gpava(pituitary[,1], pituitary[,2], ties = "secondary")
gpava(pituitary[,1], pituitary[,2], ties = "tertiary")
hSolver

---

### different target functions

```r
gpava(pituitary[, 1], pituitary[, 2], solver = weighted.mean)
gpava(pituitary[, 1], pituitary[, 2], solver = weighted.median)
gpava(pituitary[, 1], pituitary[, 2], solver = weighted.fractile, p = 0.25)
```

```r

### repeated measures

```r
data(posturo)
res <- gpava(posturo[, 1], posturo[, 2:4], ties = "secondary")
plot(res)
```

---

#### hSolver

**Huber Loss Function**

---

**Description**

Solver for Huber’s robust loss function.

**Usage**

```r
hSolver(z, a, extra)
```

**Arguments**

- `z` Vector containing observed response
- `a` Matrix with active constraints
- `extra` List with element `y` containing the observed response vector and weights with optional observation weights, and `eps`

**Details**

This function is called internally in `activeSet` by setting `mySolver = hSolver`.

**Value**

- `x` Vector containing the fitted values
- `lb` Vector with Lagrange multipliers
- `f` Value of the target function
- `gx` Gradient at point `x`

**References**


**See Also**

`activeSet`
Examples

```r
# Fitting isotone regression using active set
set.seed(12345)
y <- rnorm(9)  # response values
w <- rep(1,9)  # unit weights
eps <- 0.01
btota <- cbind(1:8, 2:9)  # Matrix defining isotonicity (total order)
fit.huber <- activeSet(btota, hSolver, weights = w, y = y, eps = eps)
```

---

### isSolver

**SILF Loss**

**Description**

Minimizes soft insensitive loss function (SILF) for support vector regression.

**Usage**

```r
isSolver(z, a, extra)
```

**Arguments**

- `z` Vector containing observed response
- `a` Matrix with active constraints
- `extra` List with element `y` containing the observed response vector, `weights` with optional observation weights, `beta` between 0 and 1, and `eps > 0`

**Details**

This function is called internally in `activeSet` by setting `mySolver = isSolver`.

**Value**

- `x` Vector containing the fitted values
- `lbd` Vector with Lagrange multipliers
- `f` Value of the target function
- `gx` Gradient at point `x`

**References**


**See Also**

`activeSet`
Examples

```r
set.seed(12345)
y <- rnorm(9)            ##response values
w <- rep(1,9)           ##unit weights
eps <- 2
beta <- 0.4
btota <- cbind(1:8, 2:9) ##Matrix defining isotonicity (total order)
fit.silf <- activeSet(btota, iSolver, weights = w, y = y, beta = beta, eps = eps)
```

Description

Solver for the general least squares monotone regression problem of the form \((y-x)'W(y-x)\).

Usage

```r
lfSolver(z, a, extra)
```

Arguments

- `z` Vector containing observed response
- `a` Matrix with active constraints
- `extra` List with element `y` containing the observed response vector and weights as weight matrix `W` which is not necessarily positive definite.

Details

This function is called internally in `activeSet` by setting `mySolver = lfSolver`.

Value

- `x` Vector containing the fitted values
- `lbd` Vector with Lagrange multipliers
- `f` Value of the target function
- `gx` Gradient at point `x`

See Also

`activeSet`
Examples

```r
# Fitting isotone regression
set.seed(12345)
y <- rnorm(9)  # response values
w <- diag(rep(1, 9))  # unit weight matrix
btota <- cbind(1:8, 2:9)  # Matrix defining isotonicity (total order)
# fit.ls <- activeSet(btota, lsSolver, weights = w, y = y)
```

**lsSolver**  
*Least Squares Loss Function*

**Description**
Solver for the least squares monotone regression problem with optional weights.

**Usage**

```r
lsSolver(z, a, extra)
```

**Arguments**
- `z`: Vector containing observed response
- `a`: Matrix with active constraints
- `extra`: List with element `y` containing the observed response vector and weights with optional observation weights

**Details**
This function is called internally in `activeSet` by setting `mySolver = lsSolver`.

**Value**
- `x`: Vector containing the fitted values
- `lbd`: Vector with Lagrange multipliers
- `f`: Value of the target function
- `gx`: Gradient at point `x`

**See Also**
- `activeSet`
Examples

```r
# Fitting isotone regression using active set
set.seed(12345)
y <- rnorm(9)  # response values
w <- rep(1, 9)  # unit weights
btota <- cbind(1:8, 2:9)  # Matrix defining isotonicity (total order)
fit.ls <- activeSet(btota, lsSolver, weights = w, y = y)
```

---

### mendota

**Number of freezing days at Lake Mendota**

---

**Description**

This dataset shows the number of freezing days at Lake Mendota measured from November, 23, in the year 1854.

**Usage**

```r
data(mendota)
```

**Format**

A data frame with 12 subjects.

**References**


### mregnn

**Regression with Linear Inequality Restrictions on Predicted Values**

### Description

The package contains three functions for fitting regressions with inequality restrictions: `mregnn` is the most general one, allowing basically for any partial orders, `mregnnM` poses a monotone restriction on the fitted values, `mregnnP` restricts the predicted values to be positive. More details can be found below.

### Usage

```r
mregnn(x, y, a)
mregnnM(x, y)
mregnnP(x, y)
```

### Arguments

- **x**: Can be a spline basis.
- **y**: Response.
- **a**: Matrix containing order restrictions.

### Details

These functions solve the problem

\[
f(b) = \frac{1}{2}(y - Xb)'(y - Xb)
\]

over all \( b \) for which \( A'Xb \geq 0 \). \( A \) can be used require the transformation to be non-negative, or increasing, or satisfying any partial order.

### Value

- **xb**: Predicted values.
- **lb**: Solution of the dual problem.
- **f**: Value of the target function

### References

Examples

```r
## Compute the best fitting quadratic polynomial (in black)
## and monotone quadratic polynomial (in blue)
set.seed(12345)
x <- outer(1:10,1:3,"")
x <- apply(x,2,mean)
x <- apply (x,2,function(x)
x / sqrt (sum(x ^ 2)))
y <- rowSums(x) + rnorm(10)
plot(x[,1], y, lwd = 3, col = "RED", xlab = "x", ylab = "P(x)"
o <- mregnnM(x,y)
lines(x[,1], o$xb, col = "BLUE", lwd = 2)
xb <- drop(x %*% qr.solve(x,y))
lines(x[,1],xb,col="BLACK", lwd = 2)
```

```r
## same monotone model through basic mregnn()
difmat <- function (n) {
m1 <- ifelse(outer(1:(n - 1),1:n,"-")) == -1, 1, 0)
m2 <- ifelse(outer(1:(n - 1),1:n,"-")) == 0,-1, 0)
return (m1 + m2)
}
a <- difmat(nrow(x))      ## order restriction
o2 <- mregnn(x, y, a)
```

---

mSolver: Chebyshev norm

Description

Solver for the Chebyshev norm.

Usage

```r
mSolver(z, a, extra)
```

Arguments

- **z**: Vector containing observed response
- **a**: Matrix with active constraints
- **extra**: List with element y containing the observed response vector and weights with optional observation weights

Details

This function is called internally in activeSet by setting mySolver = mSolver.
Value

- `x`: Vector containing the fitted values
- `lbd`: Vector with Lagrange multipliers
- `f`: Value of the target function
- `gx`: Gradient at point `x`

See Also

`activeSet`

Examples

```r
set.seed(12345)
y <- rnorm(9)  # response values
w <- rep(1, 9)  # unit weights
btota <- cbind(1:8, 2:9)  # Matrix defining isotonicity (total order)
fit.cheby <- activeSet(btota, mSolver, weights = w, y = y)
```

Description

Solver for Lp-norm.

Usage

`oSolver(z, a, extra)`

Arguments

- `z`: Vector containing observed response
- `a`: Matrix with active constraints
- `extra`: List with element `y` containing the observed response vector, `weights` as an optional weight vector, and `p` as the exponent for the Lp-norm.

Details

This function is called internally in `activeSet` by setting `mSolver = oSolver`.
**Value**

- **x**: Vector containing the fitted values
- **λd**: Vector with Lagrange multipliers
- **f**: Value of the target function
- **gx**: Gradient at point x

**See Also**

activeSet

**Examples**

```r
# Fitting isotone regression
set.seed(12345)
y <- rnorm(9)  # normal distributed response values
w1 <- rep(1, 9)  # unit weights
Atot <- cbind(1:8, 2:9)  # Matrix defining isotonicity (total order)
fit.pow <- activeSet(Atot, oSolver, y = y, weights = w1, p = 1.2)
```

### Description

The University of Carolina conducted a study in which the size (in mm) of the pituitary fissure was measured on girls between an age of 8 and 14.

### Usage

```r
data(pituitary)
```

### Format

A data frame with 11 subjects.

### References


### Examples

```r
data(pituitary)
```
posturo | Repeated posturographic measures

Description

This dataset represents a subset from the posturographic data collected in Leitner et al. (sensory organisation test SOT).

Usage

data(posturo)

Format

A data frame with 50 subjects, age as predictor and 3 repeated SOT measures as responses.

References


Examples

data(posturo)

---

pSolver | Quantile Regression

Description

Solver for the general p-quantile monotone regression problem with optional weights.

Usage

pSolver(z, a, extra)

Arguments

- `z`  
  Vector containing observed response

- `a`  
  Matrix with active constraints

- `extra`  
  List with element `y` containing the observed response vector, `weights` with optional observation weights, `aw` and `bw` as quantile weights.
Details

This function is called internally in activeSet by setting mySolver = pSolver. Note that if \( aw = bw \), we get the weighted median and therefore we solved the weighted absolute norm.

Value

- \( x \): Vector containing the fitted values
- \( \lambda \text{bd} \): Vector with Lagrange multipliers
- \( f \): Value of the target function
- \( g_x \): Gradient at point \( x \)

References


See Also

activeSet

Examples

```r
## Fitting quantile regression
set.seed(12345)
y <- rnorm(9)  # response values
w <- rep(1,9) # unit weights
btota <- cbind(1:8, 2:9) # Matrix defining isotonicity (total order)
fit.p <- activeSet(btota, pSolver, weights = w, y = y, aw = 0.3, bw = 0.7)
```

---

sSolver

**Negative Poisson Log-Likelihood**

Description

Solver for the negative Poisson log-likelihood

Usage

`sSolver(z, a, extra)`

Arguments

- \( z \): Vector containing observed response
- \( a \): Matrix with active constraints
- \( extra \): List with element \( y \) containing the observed response vector
weighted.fractile

Details

This function is called internally in activeSet by setting mySolver = sSolver.

Value

- **x**: Vector containing the fitted values
- **lbd**: Vector with Lagrange multipliers
- **f**: Value of the target function
- **gx**: Gradient at point x

See Also

- activeSet

Examples

```r
##Minimizing Poisson log-likelihood
set.seed(12345)
yp <- rpois(9,5)
Atot <- cbind(1:8, 2:9)  ##Matrix defining isotonicity (total order)
x0 <- 1:9               ##starting values
fit.poi <- activeSet(Atot, sSolver, x0 = x0, y = yp)
```

---

**weighted.fractile**  
*Weighted Median*

Description

Computes the weighted fractile of a numeric vector

Usage

`weighted.fractile(y, w, p)`

Arguments

- **y**: A numeric vector containing the values whose fractile is to be computed
- **w**: A vector of length `y` giving the weights to use for each element of `y`
- **p**: Fractile specification; value between 0 and 1

See Also

- weighted.mean, weighted.median
**weighted.median**

**Examples**

```r
y <- 1:9
w <- c(rep(1,5), rep(2,4))
res <- weighted.fractional(y, w, p = 0.33)
```

---

**Description**

Computes a weighted median of a numeric vector

**Usage**

```r
weighted.median(y, w)
```

**Arguments**

- `y`: A numeric vector containing the values whose median is to be computed
- `w`: A vector of length `y` giving the weights to use for each element of `y`

**See Also**

`weighted.mean`, `weighted.fractional`

**Examples**

```r
y <- 1:9
w <- c(rep(1,5), rep(2,4))
res <- weighted.median(y, w)
```
Index

*Topic datasets
mendota, 15
pituitary, 19
posturo, 20

*Topic models
activeSet, 2
aSolver, 5
dSolver, 6
eSolver, 7
fSolver, 8
gpava, 9
hSolver, 11
iSolver, 12
lfSolver, 13
lsSolver, 14
mregnn, 16
mSolver, 17
oSolver, 18
pSolver, 20
sSolver, 21
weighted.fractile, 22
weighted.median, 23

activeSet, 2, 6–9, 11–14, 18, 19, 21, 22
aSolver, 3, 5
dSolver, 3, 6
eSolver, 3, 7
fSolver, 3, 8
gpava, 3, 9
hSolver, 3, 11
iSolver, 3, 12
lfSolver, 3, 13
lsSolver, 3, 14
mendota, 15
mregnn, 16
mregnnM (mregnn), 16
mregnnP (mregnn), 16
mSolver, 3, 17
oSolver, 3, 18
pituitary, 19
plot.pava (gpava), 9
posturo, 20
print.activeset (activeSet), 2
print.pava (gpava), 9
pSolver, 3, 20
sSolver, 3, 21
summary.activeset (activeSet), 2
weighted.fractile, 22, 23
weighted.mean, 22, 23
weighted.median, 22, 23