Package ‘oem’

June 4, 2020

Type Package

Title Orthogonalizing EM: Penalized Regression for Big Tall Data

Version 2.0.10

Maintainer Jared Huling <jaredhuling@gmail.com>

Description Solves penalized least squares problems for big tall data using the orthogonalizing EM algorithm of Xiong et al. (2016) <doi:10.1080/00401706.2015.1054436>. The main fitting function is oem() and the functions cv.oem() and xval.oem() are for cross validation, the latter being an accelerated cross validation function for linear models. The big.oem() function allows for out of memory fitting.

URL https://arxiv.org/abs/1801.09661, https://github.com/jaredhuling/oem, https://jaredhuling.github.io/oem

BugReports https://github.com/jaredhuling/oem/issues

License GPL (>= 2)

Encoding UTF-8

LazyData TRUE

Depends R (>= 3.2.0), bigmemory

Imports Rcpp (>= 0.11.0), Matrix, foreach, methods

LinkingTo Rcpp, RcppEigen, BH, bigmemory, RcppArmadillo

RoxygenNote 7.1.0

Suggests knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation yes

Author Bin Dai [aut],
Jared Huling [aut, cre] (<https://orcid.org/0000-0003-0670-4845>),
Yixuan Qiu [ctb],
Gael Guennebaud [cph],
Jitse Niesen [cph]

Repository CRAN

Date/Publication 2020-06-04 12:00:05 UTC

R topics documented:

`big.oem` *Orthogonalizing EM for big.matrix objects*

Description

Orthogonalizing EM for big.matrix objects

Usage

```
big.oem(
  x,
  y,
  family = c("gaussian", "binomial"),
 penalty = c("elastic.net", "lasso", "ols", "mcp", "scad", "mcp.net", "scad.net",
  "grp.lasso", "grp.lasso.net", "grp.mcp", "grp.scad", "grp.mcp.net", "grp.scad.net",
   "sparse.grp.lasso"),
  weights = numeric(0),
  lambda = numeric(0),
  nlambda = 100L,
  lambda.min.ratio = NULL,
  alpha = 1,
  gamma = 3,
  tau = 0.5,
  groups = numeric(0),
  penalty.factor = NULL,
  group.weights = NULL,
  standardize = TRUE,
  intercept = TRUE,
  maxit = 500L,
  tol = 1e-07,
```

```
    irls.maxit = 100L,
    irls.tol = 0.001,
    compute.loss = FALSE,
    gigs = 4,
    hessian.type = c("full", "upper.bound")
)
```

Arguments

`x` input big.matrix object pointing to design matrix Each row is an observation, each column corresponds to a covariate

`y` numeric response vector of length nobs.

`family` `"gaussian"` for least squares problems, `"binomial"` for binary response. `"binomial"` currently not available.

`penalty` Specification of penalty type. Choices include:

- `"elastic.net"` - elastic net penalty, extra parameters: `"alpha"`
- `"lasso"` - lasso penalty
- `"ols"` - ordinary least squares
- `"mcp"` - minimax concave penalty, extra parameters: `"gamma"`
- `"scad"` - smoothly clipped absolute deviation, extra parameters: `"gamma"`
- `"mcp.net"` - minimax concave penalty + l2 penalty, extra parameters: `"gamma"`, `"alpha"`
- `"scad.net"` - smoothly clipped absolute deviation + l2 penalty, extra parameters: `"gamma"`, `"alpha"`
- `"grp.lasso"` - group lasso penalty
- `"grp.lasso.net"` - group lasso penalty + l2 penalty, extra parameters: `"alpha"`
- `"grp.mcp"` - group minimax concave penalty, extra parameters: `"gamma"`
- `"grp.scad"` - group smoothly clipped absolute deviation, extra parameters: `"gamma"`
- `"grp.mcp.net"` - group minimax concave penalty + l2 penalty, extra parameters: `"gamma"`, `"alpha"`
- `"grp.scad.net"` - group smoothly clipped absolute deviation + l2 penalty, extra parameters: `"gamma"`, `"alpha"`
- `"sparse.grp.lasso"` - sparse group lasso penalty (group lasso + lasso), extra parameters: `"tau"`

Careful consideration is required for the group lasso, group MCP, and group SCAD penalties. Groups as specified by the `groups` argument should be chosen in a sensible manner.

`weights` observation weights. Not implemented yet. Defaults to 1 for each observation (setting weight vector to length 0 will default all weights to 1)

`lambda` A user supplied lambda sequence. By default, the program computes its own lambda sequence based on `nlambda` and `lambda.min.ratio`. Supplying a value of lambda overrides this.

`nlambda`	The number of lambda values - default is 100.
`lambda.min.ratio`	Smallest value for lambda, as a fraction of `lambda.max`, the (data derived) entry value (i.e. the smallest value for which all coefficients are zero). The default depends on the sample size nobs relative to the number of variables nvars. If `nobs` > `nvars`, the default is 0.0001, close to zero. If `nobs` < `nvars`, the default is 0.01. A very small value of `lambda.min.ratio` will lead to a saturated fit when `nobs` < `nvars`.
`alpha`	mixing value for `elastic.net`, `mcp.net`, `scad.net`, `grp.mcp.net`, `grp.scad.net`. penalty applied is (1 - alpha) * (ridge penalty) + alpha * (lasso/mcp/mcp/grp.lasso penalty)
`gamma`	tuning parameter for SCAD and MCP penalties. must be >= 1
`tau`	mixing value for `sparse.grp.lasso`. penalty applied is (1 - tau) * (group lasso penalty) + tau * (lasso penalty)
`groups`	A vector of describing the grouping of the coefficients. See the example below. All unpenalized variables should be put in group 0
`penalty.factor`	Separate penalty factors can be applied to each coefficient. This is a number that multiplies lambda to allow differential shrinkage. Can be 0 for some variables, which implies no shrinkage, and that variable is always included in the model. Default is 1 for all variables.
`group.weights`	penalty factors applied to each group for the group lasso. Similar to `penalty.factor`, this is a number that multiplies lambda to allow differential shrinkage. Can be 0 for some groups, which implies no shrinkage, and that group is always included in the model. Default is sqrt(group size) for all groups.
`standardize`	Logical flag for x variable standardization, prior to fitting the models. The coefficients are always returned on the original scale. Default is `standardize = TRUE`. If variables are in the same units already, you might not wish to standardize. Keep in mind that standardization is done differently for sparse matrices, so results (when standardized) may be slightly different for a sparse matrix object and a dense matrix object
`intercept`	Should intercept(s) be fitted (`default = TRUE`) or set to zero (`FALSE`)
`maxit`	integer. Maximum number of OEM iterations
`tol`	convergence tolerance for OEM iterations
`irls.maxit`	integer. Maximum number of IRLS iterations
`irls.tol`	convergence tolerance for IRLS iterations. Only used if `family != "gaussian"`
`compute.loss`	should the loss be computed for each estimated tuning parameter? Defaults to `FALSE`. Setting to `TRUE` will dramatically increase computational time
`gigs`	maximum number of gigs of memory available. Used to figure out how to break up calculations involving the design matrix x
`hessian.type`	only for logistic regression. if `hessian.type = "full"`, then the full hessian is used. If `hessian.type = "upper.bound"`, then an upper bound of the hessian is used. The upper bound can be dramatically faster in certain situations, ie when n » p

Value

An object with S3 class "oem"

Examples

```
## Not run:
set.seed(123)
nrows <- 50000
ncols <- 100
bkFile <- "bigmat.bk"
descFile <- "bigmatk.desc"
bigmat <- filebacked.big.matrix(nrow=nrows, ncol=ncols, type="double",
                                backingfile=bkFile, backingpath=".",
                                descriptorfile=descFile,
                                dimnames=c(NULL,NULL))

# Each column value with be the column number multiplied by
# samples from a standard normal distribution.
set.seed(123)
for (i in 1:ncols) bigmat[,i] = rnorm(nrows)*i

y <- rnorm(nrows) + bigmat[,1] - bigmat[,2]

fit <- big.oem(x = bigmat, y = y,
               penalty = c("lasso", "elastic.net",
                           "ols",
                           "mcp",       "scad",
                           "mcp.net",   "scad.net",
                           "grp.lasso", "grp.lasso.net",
                           "grp.mcp",   "grp.scad",
                           "sparse.grp.lasso"),
               groups = rep(1:20, each = 5))

fit2 <- oem(x = bigmat[,], y = y,
            penalty = c("lasso", "grp.lasso"),
            groups = rep(1:20, each = 5))

max(abs(fit$beta[[1]] - fit2$beta[[1]]))

layout(matrix(1:2, ncol = 2))
plot(fit)
plot(fit, which.model = 2)

## End(Not run)
```

---

cv.oem *Cross validation for Orthogonalizing EM*

Description

Cross validation for Orthogonalizing EM

Usage

```
cv.oem(
  x,
  y,
 penalty = c("elastic.net", "lasso", "ols", "mcp", "scad", "mcp.net", "scad.net",
  "grp.lasso", "grp.lasso.net", "grp.mcp", "grp.scad", "grp.mcp.net", "grp.scad.net",
   "sparse.grp.lasso"),
  weights = numeric(0),
  lambda = NULL,
  type.measure = c("mse", "deviance", "class", "auc", "mae"),
  nfolds = 10,
  foldid = NULL,
  grouped = TRUE,
  keep = FALSE,
  parallel = FALSE,
  ncores = -1,
  ...
)
```

Arguments

`x`	input matrix of dimension n x p or `CsparseMatrix` objects of the **Matrix** (sparse not yet implemented. Each row is an observation, each column corresponds to a covariate. The cv.oem() function is optimized for n » p settings and may be very slow when p > n, so please use other packages such as `glmnet`, `ncvreg`, `grpreg`, or `gglasso` when p > n or p approx n.
`y`	numeric response vector of length nobs.
`penalty`	Specification of penalty type in lowercase letters. Choices include `"lasso"`, `"ols"` (Ordinary least squares, no penaly), `"elastic.net"`, `"scad"`, `"mcp"`, `"grp.lasso"`
`weights`	observation weights. defaults to 1 for each observation (setting weight vector to length 0 will default all weights to 1)
`lambda`	A user supplied lambda sequence. By default, the program computes its own lambda sequence based on nlambda and lambda.min.ratio. Supplying a value of lambda overrides this.
`type.measure`	measure to evaluate for cross-validation. The default is `type.measure = "deviance"`, which uses squared-error for gaussian models (a.k.a `type.measure = "mse"` there), deviance for logistic regression. `type.measure = "class"` applies to binomial only. `type.measure = "auc"` is for two-class logistic regression only. `type.measure = "mse"` or `type.measure = "mae"` (mean absolute error) can be used by all models; they measure the deviation from the fitted mean to the response.
`nfolds`	number of folds for cross-validation. default is 10. 3 is smallest value allowed.

foldid	an optional vector of values between 1 and nfold specifying which fold each observation belongs to.
grouped	Like in **glmnet**, this is an experimental argument, with default `TRUE`, and can be ignored by most users. For all models, this refers to computing nfolds separate statistics, and then using their mean and estimated standard error to describe the CV curve. If `grouped = FALSE`, an error matrix is built up at the observation level from the predictions from the `nfold` fits, and then summarized (does not apply to `type.measure = "auc"`).
keep	If `keep = TRUE`, a prevalidated list of arrasy is returned containing fitted values for each observation and each value of lambda for each model. This means these fits are computed with this observation and the rest of its fold omitted. The folid vector is also returned. Default is `keep = FALSE`
parallel	If TRUE, use parallel foreach to fit each fold. Must register parallel before hand, such as **doMC**.
ncores	Number of cores to use. If `parallel = TRUE`, then ncores will be automatically set to 1 to prevent conflicts
...	other parameters to be passed to `"oem"` function

Value

An object with S3 class `"cv.oem"`

Examples

```
set.seed(123)
n.obs <- 1e4
n.vars <- 100

true.beta <- c(runif(15, -0.25, 0.25), rep(0, n.vars - 15))

x <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
y <- rnorm(n.obs, sd = 3) + x %*% true.beta

fit <- cv.oem(x = x, y = y,
              penalty = c("lasso", "grp.lasso"),
              groups = rep(1:20, each = 5))

layout(matrix(1:2, ncol = 2))
plot(fit)
plot(fit, which.model = 2)
```

---

logLik.oem *log likelihood function for fitted oem objects*

Description

log likelihood function for fitted oem objects

log likelihood function for fitted cross validation oem objects

log likelihood function for fitted cross validation oem objects

Usage

```
## S3 method for class 'oem'
logLik(object, which.model = 1, ...)

## S3 method for class 'cv.oem'
logLik(object, which.model = 1, ...)

## S3 method for class 'xval.oem'
logLik(object, which.model = 1, ...)
```

Arguments

`object`	fitted "oem" model object.
`which.model`	If multiple penalties are fit and returned in the same `oem` object, the `which.model` argument is used to specify which model to plot. For example, if the oem object `"oemobj"` was fit with argument `penalty = c("lasso","grp.lasso")`, then `which.model = 2` provides a plot for the group lasso model.
`...`	not used

Examples

```
set.seed(123)
n.obs <- 2000
n.vars <- 50

true.beta <- c(runif(15, -0.25, 0.25), rep(0, n.vars - 15))
x <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
y <- rnorm(n.obs, sd = 3) + x %*% true.beta

fit <- oem(x = x, y = y, penalty = c("lasso", "mcp"), compute.loss = TRUE)

logLik(fit)

logLik(fit, which.model = "mcp")


fit <- cv.oem(x = x, y = y, penalty = c("lasso", "mcp"), compute.loss = TRUE,
              nlambda = 25)

logLik(fit)

logLik(fit, which.model = "mcp")
```

```
fit <- xval.oem(x = x, y = y, penalty = c("lasso", "mcp"), compute.loss = TRUE,
              nlambda = 25)

logLik(fit)

logLik(fit, which.model = "mcp")
```

---

oem *Orthogonalizing EM*

Description

Orthogonalizing EM

Usage

```
oem(
  x,
  y,
  family = c("gaussian", "binomial"),
 penalty = c("elastic.net", "lasso", "ols", "mcp", "scad", "mcp.net", "scad.net",
  "grp.lasso", "grp.lasso.net", "grp.mcp", "grp.scad", "grp.mcp.net", "grp.scad.net",
   "sparse.grp.lasso"),
  weights = numeric(0),
  lambda = numeric(0),
  nlambda = 100L,
  lambda.min.ratio = NULL,
  alpha = 1,
  gamma = 3,
  tau = 0.5,
  groups = numeric(0),
  penalty.factor = NULL,
  group.weights = NULL,
  standardize = TRUE,
  intercept = TRUE,
  maxit = 500L,
  tol = 1e-07,
  irls.maxit = 100L,
  irls.tol = 0.001,
  accelerate = FALSE,
  ncores = -1,
  compute.loss = FALSE,
  hessian.type = c("upper.bound", "full")
)
```

Arguments

`x` — input matrix of dimension n x p or `CsparseMatrix` object of the **Matrix** package. Each row is an observation, each column corresponds to a covariate. The oem() function is optimized for n » p settings and may be very slow when p > n, so please use other packages such as `glmnet`, `ncvreg`, `grpreg`, or `gglasso` when p > n or p approx n.

`y` — numeric response vector of length `nobs`.

`family` — `"gaussian"` for least squares problems, `"binomial"` for binary response.

`penalty` — Specification of penalty type. Choices include:

- `"elastic.net"` - elastic net penalty, extra parameters: `"alpha"`
- `"lasso"` - lasso penalty
- `"ols"` - ordinary least squares
- `"mcp"` - minimax concave penalty, extra parameters: `"gamma"`
- `"scad"` - smoothly clipped absolute deviation, extra parameters: `"gamma"`
- `"mcp.net"` - minimax concave penalty + l2 penalty, extra parameters: `"gamma"`, `"alpha"`
- `"scad.net"` - smoothly clipped absolute deviation + l2 penalty, extra parameters: `"gamma"`, `"alpha"`
- `"grp.lasso"` - group lasso penalty
- `"grp.lasso.net"` - group lasso penalty + l2 penalty, extra parameters: `"alpha"`
- `"grp.mcp"` - group minimax concave penalty, extra parameters: `"gamma"`
- `"grp.scad"` - group smoothly clipped absolute deviation, extra parameters: `"gamma"`
- `"grp.mcp.net"` - group minimax concave penalty + l2 penalty, extra parameters: `"gamma"`, `"alpha"`
- `"grp.scad.net"` - group smoothly clipped absolute deviation + l2 penalty, extra parameters: `"gamma"`, `"alpha"`
- `"sparse.grp.lasso"` - sparse group lasso penalty (group lasso + lasso), extra parameters: `"tau"`

Careful consideration is required for the group lasso, group MCP, and group SCAD penalties. Groups as specified by the `groups` argument should be chosen in a sensible manner.

`weights` — observation weights. Not implemented yet. Defaults to 1 for each observation (setting weight vector to length 0 will default all weights to 1)

`lambda` — A user supplied lambda sequence. By default, the program computes its own lambda sequence based on `nlambda` and `lambda.min.ratio`. Supplying a value of lambda overrides this.

`nlambda` — The number of lambda values. The default is 100.

`lambda.min.ratio` — Smallest value for lambda, as a fraction of `lambda.max`, the (data derived) entry value (i.e. the smallest value for which all coefficients are zero). The default depends on the sample size nobs relative to the number of variables nvars. If

	nobs > nvars, the default is 0.0001, close to zero. If nobs < nvars, the default is 0.01. A very small value of lambda.min.ratio will lead to a saturated fit when nobs < nvars.
alpha	mixing value for elastic.net, mcp.net, scad.net, grp.mcp.net, grp.scad.net. penalty applied is (1 - alpha) * (ridge penalty) + alpha * (lasso/mcp/mcp/grp.lasso penalty)
gamma	tuning parameter for SCAD and MCP penalties. must be >= 1
tau	mixing value for sparse.grp.lasso. penalty applied is (1 - tau) * (group lasso penalty) + tau * (lasso penalty)
groups	A vector of describing the grouping of the coefficients. See the example below. All unpenalized variables should be put in group 0
penalty.factor	Separate penalty factors can be applied to each coefficient. This is a number that multiplies lambda to allow differential shrinkage. Can be 0 for some variables, which implies no shrinkage, and that variable is always included in the model. Default is 1 for all variables.
group.weights	penalty factors applied to each group for the group lasso. Similar to penalty.factor, this is a number that multiplies lambda to allow differential shrinkage. Can be 0 for some groups, which implies no shrinkage, and that group is always included in the model. Default is sqrt(group size) for all groups.
standardize	Logical flag for x variable standardization, prior to fitting the models. The coefficients are always returned on the original scale. Default is standardize = TRUE. If variables are in the same units already, you might not wish to standardize. Keep in mind that standardization is done differently for sparse matrices, so results (when standardized) may be slightly different for a sparse matrix object and a dense matrix object
intercept	Should intercept(s) be fitted (default = TRUE) or set to zero (FALSE)
maxit	integer. Maximum number of OEM iterations
tol	convergence tolerance for OEM iterations
irls.maxit	integer. Maximum number of IRLS iterations
irls.tol	convergence tolerance for IRLS iterations. Only used if family != "gaussian"
accelerate	boolean argument. Whether or not to use Nesterov acceleration with adaptive restarting
ncores	Integer scalar that specifies the number of threads to be used
compute.loss	should the loss be computed for each estimated tuning parameter? Defaults to FALSE. Setting to TRUE will dramatically increase computational time
hessian.type	only for logistic regression. if hessian.type = "full", then the full hessian is used. If hessian.type = "upper.bound", then an upper bound of the hessian is used. The upper bound can be dramatically faster in certain situations, ie when n » p

Value

An object with S3 class "oem"

References

Shifeng Xiong, Bin Dai, Jared Huling, and Peter Z. G. Qian. Orthogonalizing EM: A design-based least squares algorithm. Technometrics, 58(3):285-293, 2016. http://amstat.tandfonline.com/doi/abs/10.1080/00401706.2015.1054436

Examples

```
set.seed(123)
n.obs <- 1e4
n.vars <- 50

true.beta <- c(runif(15, -0.25, 0.25), rep(0, n.vars - 15))

x <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
y <- rnorm(n.obs, sd = 3) + x %*% true.beta

fit <- oem(x = x, y = y,
           penalty = c("lasso", "grp.lasso", "sparse.grp.lasso"),
           groups = rep(1:10, each = 5))

layout(matrix(1:3, ncol = 3))
plot(fit)
plot(fit, which.model = 2)
plot(fit, which.model = "sparse.grp.lasso")

# the oem package has support for
# sparse design matrices

library(Matrix)

xs <- rsparsematrix(n.obs * 25, n.vars * 2, density = 0.01)
ys <- rnorm(n.obs * 25, sd = 3) + as.vector(xs %*% c(true.beta, rep(0, n.vars)) )
x.dense <- as.matrix(xs)

system.time(fit <- oem(x = x.dense, y = ys,
                       penalty = c("lasso", "grp.lasso"),
                       groups = rep(1:20, each = 5), intercept = FALSE,
                       standardize = FALSE))

system.time(fits <- oem(x = xs, y = ys,
                        penalty = c("lasso", "grp.lasso"),
                        groups = rep(1:20, each = 5), intercept = FALSE,
                        standardize = FALSE, lambda = fit$lambda))

max(abs(fit$beta[[1]] - fits$beta[[1]]))
max(abs(fit$beta[[2]] - fits$beta[[2]]))

# logistic
y <- rbinom(n.obs, 1, prob = 1 / (1 + exp(-x %*% true.beta)))

system.time(res <- oem(x, y, intercept = FALSE,
                       penalty = c("lasso", "sparse.grp.lasso", "mcp"),
```

```
                           family = "binomial",
                           groups = rep(1:10, each = 5),
                           nlambda = 10,
                           irls.tol = 1e-3, tol = 1e-8))

layout(matrix(1:3, ncol = 3))
plot(res)
plot(res, which.model = 2)
plot(res, which.model = "mcp")


# sparse design matrix
xs <- rsparsematrix(n.obs * 2, n.vars, density = 0.01)
x.dense <- as.matrix(xs)
ys <- rbinom(n.obs * 2, 1, prob = 1 / (1 + exp(-x %*% true.beta)))

system.time(res.gr <- oem(x.dense, ys, intercept = FALSE,
                          penalty = "grp.lasso",
                          family = "binomial",
                          nlambda = 10,
                          groups = rep(1:5, each = 10),
                          irls.tol = 1e-3, tol = 1e-8))

system.time(res.gr.s <- oem(xs, ys, intercept = FALSE,
                            penalty = "grp.lasso",
                            family = "binomial",
                            nlambda = 10,
                            groups = rep(1:5, each = 10),
                            irls.tol = 1e-3, tol = 1e-8))

max(abs(res.gr$beta[[1]] - res.gr.s$beta[[1]]))
```

`oem.xtx` *Orthogonalizing EM with precomputed XtX*

Description

Orthogonalizing EM with precomputed XtX

Usage

```
oem.xtx(
  xtx,
  xty,
  family = c("gaussian", "binomial"),
 penalty = c("elastic.net", "lasso", "ols", "mcp", "scad", "mcp.net", "scad.net",
  "grp.lasso", "grp.lasso.net", "grp.mcp", "grp.scad", "grp.mcp.net", "grp.scad.net",
    "sparse.grp.lasso"),
  lambda = numeric(0),
```

```
  nlambda = 100L,
  lambda.min.ratio = NULL,
  alpha = 1,
  gamma = 3,
  tau = 0.5,
  groups = numeric(0),
  scale.factor = numeric(0),
  penalty.factor = NULL,
  group.weights = NULL,
  maxit = 500L,
  tol = 1e-07,
  irls.maxit = 100L,
  irls.tol = 0.001
)
```

Arguments

`xtx` input matrix equal to `crossprod(x) / nrow(x)`. where `x` is the design matrix. It is highly recommended to scale by the number of rows in `x`. If `xtx` is scaled, `xty` must also be scaled or else results may be meaningless!

`xty` numeric vector of length `nvars`. Equal to `crosprod(x,y) / nobs`. It is highly recommended to scale by the number of rows in `x`.

`family` `"gaussian"` for least squares problems, `"binomial"` for binary response. (only `gaussian` implemented currently)

`penalty` Specification of penalty type. Choices include:

- `"elastic.net"` - elastic net penalty, extra parameters: `"alpha"`
- `"lasso"` - lasso penalty
- `"ols"` - ordinary least squares
- `"mcp"` - minimax concave penalty, extra parameters: `"gamma"`
- `"scad"` - smoothly clipped absolute deviation, extra parameters: `"gamma"`
- `"mcp.net"` - minimax concave penalty + l2 penalty, extra parameters: `"gamma"`, `"alpha"`
- `"scad.net"` - smoothly clipped absolute deviation + l2 penalty, extra parameters: `"gamma"`, `"alpha"`
- `"grp.lasso"` - group lasso penalty
- `"grp.lasso.net"` - group lasso penalty + l2 penalty, extra parameters: `"alpha"`
- `"grp.mcp"` - group minimax concave penalty, extra parameters: `"gamma"`
- `"grp.scad"` - group smoothly clipped absolute deviation, extra parameters: `"gamma"`
- `"grp.mcp.net"` - group minimax concave penalty + l2 penalty, extra parameters: `"gamma"`, `"alpha"`
- `"grp.scad.net"` - group smoothly clipped absolute deviation + l2 penalty, extra parameters: `"gamma"`, `"alpha"`
- `"sparse.grp.lasso"` - sparse group lasso penalty (group lasso + lasso), extra parameters: `"tau"`

Careful consideration is required for the group lasso, group MCP, and group SCAD penalties. Groups as specified by the `groups` argument should be chosen in a sensible manner.

`lambda` A user supplied lambda sequence. By default, the program computes its own lambda sequence based on `nlambda` and `lambda.min.ratio`. Supplying a value of lambda overrides this.

`nlambda` The number of lambda values - default is 100.

`lambda.min.ratio` Smallest value for lambda, as a fraction of `lambda.max`, the (data derived) entry value (i.e. the smallest value for which all coefficients are zero). The default depends on the sample size nobs relative to the number of variables nvars. The default is 0.0001

`alpha` mixing value for `elastic.net`, `mcp.net`, `scad.net`, `grp.mcp.net`, `grp.scad.net`. penalty applied is (1 - alpha) * (ridge penalty) + alpha * (lasso/mcp/mcp/grp.lasso penalty)

`gamma` tuning parameter for SCAD and MCP penalties. must be >= 1

`tau` mixing value for `sparse.grp.lasso`. penalty applied is (1 - tau) * (group lasso penalty) + tau * (lasso penalty)

`groups` A vector of describing the grouping of the coefficients. See the example below. All unpenalized variables should be put in group 0

`scale.factor` of length `nvars === ncol(xtx) == length(xty)` for scaling columns of `x`. The standard deviation for each column of `x` is a common choice for `scale.factor`. Coefficients will be returned on original scale. Default is no scaling.

`penalty.factor` Separate penalty factors can be applied to each coefficient. This is a number that multiplies lambda to allow differential shrinkage. Can be 0 for some variables, which implies no shrinkage, and that variable is always included in the model. Default is 1 for all variables.

`group.weights` penalty factors applied to each group for the group lasso. Similar to `penalty.factor`, this is a number that multiplies lambda to allow differential shrinkage. Can be 0 for some groups, which implies no shrinkage, and that group is always included in the model. Default is sqrt(group size) for all groups.

`maxit` integer. Maximum number of OEM iterations

`tol` convergence tolerance for OEM iterations

`irls.maxit` integer. Maximum number of IRLS iterations

`irls.tol` convergence tolerance for IRLS iterations. Only used if `family != "gaussian"`

Value

An object with S3 class `"oem"`

Examples

```
set.seed(123)
n.obs <- 1e4
n.vars <- 100
```

```
true.beta <- c(runif(15, -0.25, 0.25), rep(0, n.vars - 15))

x <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
y <- rnorm(n.obs, sd = 3) + x %*% true.beta

fit <- oem(x = x, y = y,
           penalty = c("lasso", "elastic.net",
                       "ols",
                       "mcp",       "scad",
                       "mcp.net",   "scad.net",
                       "grp.lasso", "grp.lasso.net",
                       "grp.mcp",   "grp.scad",
                       "sparse.grp.lasso"),
           standardize = FALSE, intercept = FALSE,
           groups = rep(1:20, each = 5))

xtx <- crossprod(x) / n.obs
xty <- crossprod(x, y) / n.obs

fit.xtx <- oem.xtx(xtx = xtx, xty = xty,
                   penalty = c("lasso", "elastic.net",
                               "ols",
                               "mcp",       "scad",
                               "mcp.net",   "scad.net",
                               "grp.lasso", "grp.lasso.net",
                               "grp.mcp",   "grp.scad",
                               "sparse.grp.lasso"),
                   groups = rep(1:20, each = 5))

max(abs(fit$beta[[1]][-1,] - fit.xtx$beta[[1]]))
max(abs(fit$beta[[2]][-1,] - fit.xtx$beta[[2]]))

layout(matrix(1:2, ncol = 2))
plot(fit.xtx)
plot(fit.xtx, which.model = 2)
```

`oemfit` *Deprecated functions*

Description

These functions have been renamed and deprecated in **oem**: `oemfit()` (use `oem()`), `cv.oemfit()` (use `cv.oem()`), `print.oemfit()`, `plot.oemfit()`, `predict.oemfit()`, and `coef.oemfit()`.

Usage

```
oemfit(
  formula,
```

```
    data = list(),
    lambda = NULL,
    nlambda = 100,
    lambda.min.ratio = NULL,
    tolerance = 0.001,
    maxIter = 1000,
    standardized = TRUE,
    numGroup = 1,
    penalty = c("lasso", "scad", "ols", "elastic.net", "ngarrote", "mcp"),
    alpha = 3,
    evaluate = 0,
    condition = -1
  )

  cv.oemfit(
    formula,
    data = list(),
    lambda = NULL,
    type.measure = c("mse", "mae"),
    ...,
    nfolds = 10,
    foldid,
    penalty = c("lasso", "scad", "elastic.net", "ngarrote", "mcp")
  )

  ## S3 method for class 'oemfit'
  plot(
    x,
    xvar = c("norm", "lambda", "loglambda", "dev"),
    xlab = iname,
    ylab = "Coefficients",
    ...
  )

  ## S3 method for class 'oemfit'
  predict(
    object,
    newx,
    s = NULL,
    type = c("response", "coefficients", "nonzero"),
    ...
  )

  ## S3 method for class 'oemfit'
  print(x, digits = max(3, getOption("digits") - 3), ...)
```

Arguments

`formula` an object of 'formula' (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'

`data` an optional data frame, list or environment (or object coercible by 'as.data.frame' to a data frame) containing the variables in the model. If not found in 'data', the variables are taken from 'environment(formula)', typically the environment from which 'oemfit' is called.

`lambda` A user supplied `lambda` sequence. Typical usage is to have the program compute its own `lambda` sequence based on `nlambda` and `lambda.min.ratio`. Supplying a value of `lambda` overrides this. WARNING: use with care. Do not supply a single value for `lambda` (for predictions after CV use `predict()` instead). Supply instead a decreasing sequence of `lambda` values. `oemfit` relies on its warms starts for speed, and its often faster to fit a whole path than compute a single fit.

`nlambda` The number of `lambda` values - default is 100.

`lambda.min.ratio` Smallest value for `lambda`, as a fraction of `lambda.max`, the (data derived) entry value (i.e. the smallest value for which all coefficients are zero). The default depends on the sample size `nobs` relative to the number of variables `nvars`. If `nobs > nvars`, the default is `0.0001`, close to zero. If `nobs < nvars`, the default is `0.01`. A very small value of `lambda.min.ratio` will lead to a saturated fit in the `nobs < nvars` case.

`tolerance` Convergence tolerance for OEM. Each inner OEM loop continues until the maximum change in the objective after any coefficient update is less than `tolerance`. Defaults value is `1E-3`.

`maxIter` Maximum number of passes over the data for all lambda values; default is 1000.

`standardized` Logical flag for x variable standardization, prior to fitting the model sequence. The coefficients are always returned on the original scale. Default is `standardize=TRUE`. If variables are in the same units already, you might not wish to standardize.

`numGroup` Integer value for the number of groups to use for OEM fitting. Default is 1.

`penalty` type in lower letters. Different types include 'lasso', 'scad', 'ols' (ordinary least square), 'elastic-net', 'ngarrote' (non-negative garrote) and 'mcp'.

`alpha` alpha value for scad and mcp.

`evaluate` debugging argument

`condition` Debugging for different ways of calculating OEM.

`type.measure` type.measure measure to evaluate for cross-validation. `type.measure = "mse"` (mean squared error) or `type.measure = "mae"` (mean absolute error)

`...` arguments to be passed to `oemfit()`

`nfolds` number of folds for cross-validation. default is 10.

`foldid` an optional vector of values between 1 and nfold specifying which fold each observation belongs to.

`x` fitted `oemfit` object

xvar	what is on the X-axis. "norm" plots against the L1-norm of the coefficients, "lambda" against the log-lambda sequence, and "dev" against the percent deviance explained.
xlab	x-axis label
ylab	y-axis label
object	fitted `oemfit` object
newx	matrix of new values for x at which predictions are to be made. Must be a matrix.
s	Value(s) of the penalty parameter lambda at which predictions are required. Default is the entire sequence used to create the model.
type	not used.
digits	significant digits in print out.

Details

The sequence of models implied by 'lambda' is fit by OEM algorithm.

Author(s)

Bin Dai

`plot.oem` *Plot method for Orthogonalizing EM fitted objects*

Description

Plot method for Orthogonalizing EM fitted objects

Plot method for Orthogonalizing EM fitted objects

Usage

```
## S3 method for class 'oem'
plot(
  x,
  which.model = 1,
  xvar = c("norm", "lambda", "loglambda", "dev"),
  labsize = 0.6,
  xlab = iname,
  ylab = NULL,
  main = x$penalty[which.model],
  ...
)

## S3 method for class 'cv.oem'
plot(x, which.model = 1, sign.lambda = 1, ...)
```

```
## S3 method for class 'xval.oem'
plot(
  x,
  which.model = 1,
  type = c("cv", "coefficients"),
  xvar = c("norm", "lambda", "loglambda", "dev"),
  labsize = 0.6,
  xlab = iname,
  ylab = NULL,
  main = x$penalty[which.model],
  sign.lambda = 1,
  ...
)
```

Arguments

`x`	fitted "oem" model object
`which.model`	If multiple penalties are fit and returned in the same oem object, the which.model argument is used to specify which model to plot. For example, if the oem object `"oemobj"` was fit with argument `penalty = c("lasso","grp.lasso")`, then `which.model = 2` provides a plot for the group lasso model.
`xvar`	What is on the X-axis. `"norm"` plots against the L1-norm of the coefficients, `"lambda"` against the log-lambda sequence, and `"dev"` against the percent deviance explained.
`labsize`	size of labels for variable names. If labsize = 0, then no variable names will be plotted
`xlab`	label for x-axis
`ylab`	label for y-axis
`main`	main title for plot
`...`	other graphical parameters for the plot
`sign.lambda`	Either plot against log(lambda) (default) or its negative if `sign.lambda = -1`.
`type`	one of `"cv"` or `"coefficients"`. `type = "cv"` will produce a plot of cross validation results like plot.cv.oem. `type = "coefficients"` will produce a coefficient path plot like `plot.oem()`

Examples

```
set.seed(123)
n.obs <- 1e4
n.vars <- 100
n.obs.test <- 1e3

true.beta <- c(runif(15, -0.5, 0.5), rep(0, n.vars - 15))

x <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
y <- rnorm(n.obs, sd = 3) + x %*% true.beta
```

```
fit <- oem(x = x, y = y, penalty = c("lasso", "grp.lasso"), groups = rep(1:10, each = 10))

layout(matrix(1:2, ncol = 2))
plot(fit, which.model = 1)
plot(fit, which.model = 2)

set.seed(123)
n.obs <- 1e4
n.vars <- 100
n.obs.test <- 1e3

true.beta <- c(runif(15, -0.5, 0.5), rep(0, n.vars - 15))

x <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
y <- rnorm(n.obs, sd = 3) + x %*% true.beta

fit <- cv.oem(x = x, y = y, penalty = c("lasso", "grp.lasso"), groups = rep(1:10, each = 10))

layout(matrix(1:2, ncol = 2))
plot(fit, which.model = 1)
plot(fit, which.model = "grp.lasso")

set.seed(123)
n.obs <- 1e4
n.vars <- 100
n.obs.test <- 1e3

true.beta <- c(runif(15, -0.5, 0.5), rep(0, n.vars - 15))

x <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
y <- rnorm(n.obs, sd = 3) + x %*% true.beta

fit <- xval.oem(x = x, y = y, penalty = c("lasso", "grp.lasso"), groups = rep(1:10, each = 10))

layout(matrix(1:4, ncol = 2))
plot(fit, which.model = 1)
plot(fit, which.model = 2)

plot(fit, which.model = 1, type = "coef")
plot(fit, which.model = 2, type = "coef")
```

`predict.cv.oem` *Prediction function for fitted cross validation oem objects*

Description

Prediction function for fitted cross validation oem objects

Usage

```
## S3 method for class 'cv.oem'
predict(
  object,
  newx,
  which.model = "best.model",
  s = c("lambda.min", "lambda.1se"),
  ...
)
```

Arguments

`object`	fitted "cv.oem" model object
`newx`	Matrix of new values for x at which predictions are to be made. Must be a matrix; can be sparse as in the `CsparseMatrix` objects of the **Matrix** package This argument is not used for `type = c("coefficients","nonzero")`
`which.model`	If multiple penalties are fit and returned in the same `oem` object, the `which.model` argument is used to specify which model to make predictions for. For example, if the oem object "oemobj" was fit with argument `penalty = c("lasso","grp.lasso")`, then `which.model = 2` provides predictions for the group lasso model. For `predict.cv.oem()`, can specify "best.model" to use the best model as estimated by cross-validation
`s`	Value(s) of the penalty parameter lambda at which predictions are required. Default is the entire sequence used to create the model. For `predict.cv.oem()`, can also specify "lambda.1se" or "lambda.min" for best lambdas estimated by cross validation
`...`	used to pass the other arguments for predict.oem

Value

An object depending on the type argument

Examples

```
set.seed(123)
n.obs <- 1e4
n.vars <- 100
n.obs.test <- 1e3

true.beta <- c(runif(15, -0.5, 0.5), rep(0, n.vars - 15))

x <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
y <- rnorm(n.obs, sd = 3) + x %*% true.beta
x.test <- matrix(rnorm(n.obs.test * n.vars), n.obs.test, n.vars)
y.test <- rnorm(n.obs.test, sd = 3) + x.test %*% true.beta

fit <- cv.oem(x = x, y = y,
              penalty = c("lasso", "grp.lasso"),
              groups = rep(1:10, each = 10),
```

```
              nlambda = 10)

preds.best <- predict(fit, newx = x.test, type = "response", which.model = "best.model")

apply(preds.best, 2, function(x) mean((y.test - x) ^ 2))

preds.gl <- predict(fit, newx = x.test, type = "response", which.model = "grp.lasso")

apply(preds.gl, 2, function(x) mean((y.test - x) ^ 2))

preds.l <- predict(fit, newx = x.test, type = "response", which.model = 1)

apply(preds.l, 2, function(x) mean((y.test - x) ^ 2))
```

---

predict.oem *Prediction method for Orthogonalizing EM fitted objects*

Description

Prediction method for Orthogonalizing EM fitted objects

Usage

```
## S3 method for class 'oem'
predict(
  object,
  newx,
  s = NULL,
  which.model = 1,
  type = c("link", "response", "coefficients", "nonzero", "class"),
  ...
)
```

Arguments

object	fitted "oem" model object
newx	Matrix of new values for x at which predictions are to be made. Must be a matrix; can be sparse as in the CsparseMatrix objects of the **Matrix** package. This argument is not used for type=c("coefficients","nonzero")
s	Value(s) of the penalty parameter lambda at which predictions are required. Default is the entire sequence used to create the model.
which.model	If multiple penalties are fit and returned in the same oem object, the which.model argument is used to specify which model to make predictions for. For example, if the oem object oemobj was fit with argument penalty = c("lasso","grp.lasso"), then which.model = 2 provides predictions for the group lasso model.

type
: Type of prediction required. `type = "link"` gives the linear predictors for the `"binomial"` model; for `"gaussian"` models it gives the fitted values. `type = "response"` gives the fitted probabilities for `"binomial"`. `type = "coefficients"` computes the coefficients at the requested values for `s`. `type = "class"` applies only to `"binomial"` and produces the class label corresponding to the maximum probability.

...
: not used

Value

An object depending on the type argument

Examples

```
set.seed(123)
n.obs <- 1e4
n.vars <- 100
n.obs.test <- 1e3

true.beta <- c(runif(15, -0.5, 0.5), rep(0, n.vars - 15))

x <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
y <- rnorm(n.obs, sd = 3) + x %*% true.beta
x.test <- matrix(rnorm(n.obs.test * n.vars), n.obs.test, n.vars)
y.test <- rnorm(n.obs.test, sd = 3) + x.test %*% true.beta

fit <- oem(x = x, y = y,
           penalty = c("lasso", "grp.lasso"),
           groups = rep(1:10, each = 10),
           nlambda = 10)

preds.lasso <- predict(fit, newx = x.test, type = "response", which.model = 1)
preds.grp.lasso <- predict(fit, newx = x.test, type = "response", which.model = 2)

apply(preds.lasso,     2, function(x) mean((y.test - x) ^ 2))
apply(preds.grp.lasso, 2, function(x) mean((y.test - x) ^ 2))
```

`predict.xval.oem` *Prediction function for fitted cross validation oem objects*

Description

Prediction function for fitted cross validation oem objects

Usage

```
## S3 method for class 'xval.oem'
predict(
  object,
  newx,
  which.model = "best.model",
  s = c("lambda.min", "lambda.1se"),
  ...
)
```

Arguments

`object`	fitted "cv.oem" model object
`newx`	Matrix of new values for x at which predictions are to be made. Must be a matrix; can be sparse as in the `CsparseMatrix` objects of the **Matrix** package This argument is not used for type=c("coefficients","nonzero")
`which.model`	If multiple penalties are fit and returned in the same `oem` object, the `which.model` argument is used to specify which model to make predictions for. For example, if the oem object "oemobj" was fit with argument `penalty = c("lasso", "grp.lasso")`, then `which.model = 2` provides predictions for the group lasso model. For `predict.cv.oem()`, can specify `"best.model"` to use the best model as estimated by cross-validation
`s`	Value(s) of the penalty parameter `lambda` at which predictions are required. Default is the entire sequence used to create the model. For predict.cv.oem, can also specify `"lambda.1se"` or `"lambda.min"` for best lambdas estimated by cross validation
`...`	used to pass the other arguments for `predict.oem()`

Value

An object depending on the type argument

Examples

```
set.seed(123)
n.obs <- 1e4
n.vars <- 100
n.obs.test <- 1e3

true.beta <- c(runif(15, -0.5, 0.5), rep(0, n.vars - 15))

x <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
y <- rnorm(n.obs, sd = 3) + x %*% true.beta
x.test <- matrix(rnorm(n.obs.test * n.vars), n.obs.test, n.vars)
y.test <- rnorm(n.obs.test, sd = 3) + x.test %*% true.beta

fit <- xval.oem(x = x, y = y,
                penalty = c("lasso", "grp.lasso"),
                groups = rep(1:10, each = 10),
```

```
                nlambda = 10)

preds.best <- predict(fit, newx = x.test, type = "response", which.model = "best.model")

apply(preds.best, 2, function(x) mean((y.test - x) ^ 2))

preds.gl <- predict(fit, newx = x.test, type = "response", which.model = "grp.lasso")

apply(preds.gl, 2, function(x) mean((y.test - x) ^ 2))

preds.l <- predict(fit, newx = x.test, type = "response", which.model = 1)

apply(preds.l, 2, function(x) mean((y.test - x) ^ 2))
```

---

print.summary.cv.oem *print method for* summary.cv.oem *objects*

Description

print method for summary.cv.oem objects

Usage

```
## S3 method for class 'summary.cv.oem'
print(x, digits, ...)
```

Arguments

x	a "summary.cv.oem" object
digits	digits to display
...	not used

---

summary.cv.oem *summary method for cross validation Orthogonalizing EM fitted objects*

Description

summary method for cross validation Orthogonalizing EM fitted objects

summary method for cross validation Orthogonalizing EM fitted objects

Usage

```
## S3 method for class 'cv.oem'
summary(object, ...)

## S3 method for class 'xval.oem'
summary(object, ...)
```

Arguments

`object`	fitted "`cv.oem`" object
`...`	not used

`xval.oem` *Fast cross validation for Orthogonalizing EM*

Description

Fast cross validation for Orthogonalizing EM

Usage

```
xval.oem(
  x,
  y,
  nfolds = 10L,
  foldid = NULL,
  type.measure = c("mse", "deviance", "class", "auc", "mae"),
  ncores = -1,
  family = c("gaussian", "binomial"),
 penalty = c("elastic.net", "lasso", "ols", "mcp", "scad", "mcp.net", "scad.net",
   "grp.lasso", "grp.lasso.net", "grp.mcp", "grp.scad", "grp.mcp.net", "grp.scad.net",
    "sparse.grp.lasso"),
  weights = numeric(0),
  lambda = numeric(0),
  nlambda = 100L,
  lambda.min.ratio = NULL,
  alpha = 1,
  gamma = 3,
  tau = 0.5,
  groups = numeric(0),
  penalty.factor = NULL,
  group.weights = NULL,
  standardize = TRUE,
  intercept = TRUE,
  maxit = 500L,
  tol = 1e-07,
  irls.maxit = 100L,
  irls.tol = 0.001,
  compute.loss = FALSE
)
```

Arguments

x — input matrix of dimension n x p (sparse matrices not yet implemented). Each row is an observation, each column corresponds to a covariate. The xval.oem() function is optimized for n » p settings and may be very slow when p > n, so please use other packages such as `glmnet`, `ncvreg`, `grpreg`, or `gglasso` when p > n or p approx n.

y — numeric response vector of length `nobs` = `nrow(x)`.

nfolds — integer number of cross validation folds. 3 is the minimum number allowed. defaults to 10

foldid — an optional vector of values between 1 and `nfold` specifying which fold each observation belongs to.

type.measure — measure to evaluate for cross-validation. The default is `type.measure = "deviance"`, which uses squared-error for gaussian models (a.k.a `type.measure = "mse"` there), deviance for logistic regression. `type.measure = "class"` applies to `binomial` only. `type.measure = "auc"` is for two-class logistic regression only. `type.measure="mse"` or `type.measure="mae"` (mean absolute error) can be used by all models; they measure the deviation from the fitted mean to the response.

ncores — Integer scalar that specifies the number of threads to be used

family — `"gaussian"` for least squares problems, `"binomial"` for binary response (not implemented yet).

penalty — Specification of penalty type. Choices include:

- `"elastic.net"` - elastic net penalty, extra parameters: `"alpha"`
- `"lasso"` - lasso penalty
- `"ols"` - ordinary least squares
- `"mcp"` - minimax concave penalty, extra parameters: `"gamma"`
- `"scad"` - smoothly clipped absolute deviation, extra parameters: `"gamma"`
- `"mcp.net"` - minimax concave penalty + l2 penalty, extra parameters: `"gamma"`, `"alpha"`
- `"scad.net"` - smoothly clipped absolute deviation + l2 penalty, extra parameters: `"gamma"`, `"alpha"`
- `"grp.lasso"` - group lasso penalty
- `"grp.lasso.net"` - group lasso penalty + l2 penalty, extra parameters: `"alpha"`
- `"grp.mcp"` - group minimax concave penalty, extra parameters: `"gamma"`
- `"grp.scad"` - group smoothly clipped absolute deviation, extra parameters: `"gamma"`
- `"grp.mcp.net"` - group minimax concave penalty + l2 penalty, extra parameters: `"gamma"`, `"alpha"`
- `"grp.scad.net"` - group smoothly clipped absolute deviation + l2 penalty, extra parameters: `"gamma"`, `"alpha"`
- `"sparse.grp.lasso"` - sparse group lasso penalty (group lasso + lasso), extra parameters: `"tau"`

Careful consideration is required for the group lasso, group MCP, and group SCAD penalties. Groups as specified by the `groups` argument should be chosen in a sensible manner.

`weights` observation weights. defaults to 1 for each observation (setting weight vector to length 0 will default all weights to 1)

`lambda` A user supplied lambda sequence. By default, the program computes its own lambda sequence based on `nlambda` and `lambda.min.ratio`. Supplying a value of lambda overrides this.

`nlambda` The number of lambda values - default is 100.

`lambda.min.ratio`
Smallest value for lambda, as a fraction of `lambda.max`, the (data derived) entry value (i.e. the smallest value for which all coefficients are zero). The default depends on the sample size nobs relative to the number of variables nvars. If `nobs` > `nvars`, the default is 0.0001, close to zero.

`alpha` mixing value for `elastic.net`, `mcp.net`, `scad.net`, `grp.mcp.net`, `grp.scad.net`. penalty applied is (1 - alpha) * (ridge penalty) + alpha * (lasso/mcp/mcp/grp.lasso penalty)

`gamma` tuning parameter for SCAD and MCP penalties. must be >= 1

`tau` mixing value for `sparse.grp.lasso`. penalty applied is (1 - tau) * (group lasso penalty) + tau * (lasso penalty)

`groups` A vector of describing the grouping of the coefficients. See the example below. All unpenalized variables should be put in group 0

`penalty.factor` Separate penalty factors can be applied to each coefficient. This is a number that multiplies lambda to allow differential shrinkage. Can be 0 for some variables, which implies no shrinkage, and that variable is always included in the model. Default is 1 for all variables.

`group.weights` penalty factors applied to each group for the group lasso. Similar to `penalty.factor`, this is a number that multiplies lambda to allow differential shrinkage. Can be 0 for some groups, which implies no shrinkage, and that group is always included in the model. Default is sqrt(group size) for all groups.

`standardize` Logical flag for `x` variable standardization, prior to fitting the models. The coefficients are always returned on the original scale. Default is `standardize = TRUE`. If variables are in the same units already, you might not wish to standardize.

`intercept` Should intercept(s) be fitted (`default = TRUE`) or set to zero (`FALSE`)

`maxit` integer. Maximum number of OEM iterations

`tol` convergence tolerance for OEM iterations

`irls.maxit` integer. Maximum number of IRLS iterations

`irls.tol` convergence tolerance for IRLS iterations. Only used if `family != "gaussian"`

`compute.loss` should the loss be computed for each estimated tuning parameter? Defaults to `FALSE`. Setting to `TRUE` will dramatically increase computational time

Value

An object with S3 class `"xval.oem"`

Examples

```
set.seed(123)
n.obs <- 1e4
n.vars <- 100

true.beta <- c(runif(15, -0.25, 0.25), rep(0, n.vars - 15))

x <- matrix(rnorm(n.obs * n.vars), n.obs, n.vars)
y <- rnorm(n.obs, sd = 3) + x %*% true.beta

system.time(fit <- oem(x = x, y = y,
                       penalty = c("lasso", "grp.lasso"),
                       groups = rep(1:20, each = 5)))

system.time(xfit <- xval.oem(x = x, y = y,
                             penalty = c("lasso", "grp.lasso"),
                             groups = rep(1:20, each = 5)))

system.time(xfit2 <- xval.oem(x = x, y = y,
                              penalty = c("lasso", "grp.lasso",
                                          "mcp",       "scad",
                                          "mcp.net",   "scad.net",
                                          "grp.lasso", "grp.lasso.net",
                                          "grp.mcp",   "grp.scad",
                                          "sparse.grp.lasso"),
                              groups = rep(1:20, each = 5)))
```

Index