Package 'lqa'

February 20, 2015

Type Package

Title Penalized Likelihood Inference for GLMs

Version 1.0-3

Date 2010-07-12

Author Jan Ulbricht

Maintainer Jan Ulbricht <jan.ulbricht@stat.uni-muenchen.de>

Description This package provides some basic infrastructure and tools to fit Generalized Linear Models (GLMs) via penalized likelihood inference. Estimating procedures already implemented are the LQA algorithm (that is where its name come from), P-IRLS, RidgeBoost, GBlockBoost and ForwardBoost.

License GPL-2

LazyLoad yes

Repository CRAN

Date/Publication 2012-10-29 08:59:07

NeedsCompilation no

R topics documented:

`lqa-package` *Fitting GLMs based on penalized likelihood inference.*

Description

The `lqa` package is designed to fit Generalized Linear Models (GLMs) based on penalized likelihood inference. That is we assume our objective to be

$$\min_{\mathbf{b}} -\ell(\mathbf{b}) + P_\lambda(\boldsymbol{\beta}),$$

where $\ell(\mathbf{b})$ is the log-likelihood of the underlying GLM with unknown coefficient vector $\mathbf{b} = (\beta_0, \boldsymbol{\beta}^\top)^\top$ and the penalty term has structure

$$P_\lambda(\boldsymbol{\beta}) = \sum_{j=1}^{J} p_{\lambda,j}(|\mathbf{a}_j^\top \boldsymbol{\beta}|)$$

with known vectors of constants $\mathbf{a}_j$. The subscript $\boldsymbol{\lambda}$ illustrates the dependency on a vector of tuning parameters. This structure allows for many different penalty terms, including polytopes (such as lasso, fused lasso, LICB) and quadratic penalties (ridge, penalreg etc.).

The main important fitting procedure is the LQA algorithm. Alternatively, you can apply GBlockBoost (including RidgeBoost as special case when you use the arguments `penalty = ridge` and `componentwise = TRUE`) or ForwardBoost. However, the P-IRLS algorithm is also implemented (indirectly) as this is a special case of the LQA algorithm. See Ulbricht (2010) or the accompanying 'User's Guide' for further details.

Details

Package:	lqa
Type:	Package
Version:	1.0-3
Date:	2010-07-12
License:	GPL-2
LazyLoad:	yes

Author(s)

Jan Ulbricht

Maintainer: Jan Ulbricht <jan.ulbricht@stat.uni-muenchen.de>

References

Ulbricht, Jan (2010) *Variable Selection in Generalized Linear Models*. Ph.D. Thesis. Ludwig Maximilians University Munich.

`adaptive.lasso` *Adaptive Lasso Penalty*

Description

Object of the `penalty` class to handle the adaptive lasso penalty (Zou, 2006).

Usage

```
adaptive.lasso (lambda = NULL, al.weights = NULL, ...)
```

Arguments

`lambda`	regularization parameter. This must be a nonnegative real number.
`al.weights`	weights used for the adaptive lasso penalty.
`...`	further arguments.

Details

The adaptive lasso penalty (Zou, 2006) is defined as

$$P_{\tilde{\lambda}}^{al}(\boldsymbol{\beta}) = \lambda \sum_{i=1}^{p} w_i |\beta_i|,$$

where adaptive weights w_i are used for penalizing different coefficients in the L_1-norm penalty. Based on a root-n-consistent estimator $\hat{\boldsymbol{\beta}}$ of the true parameter vector $\boldsymbol{\beta}$, Zou (2006) uses the weight vector $\hat{\mathbf{w}} = (\hat{w}_1, \ldots, \hat{w}_p)^\top$ with

$$\hat{w}_j = \frac{1}{|\hat{\beta}_j|^\gamma}, \quad j = 1, \ldots, p$$

as estimates for the adaptive weights, where $\gamma > 0$ can be chosen arbitrarily. In the $n > p$ case, the MLE can be used for the estimated weights. In the $n \ll p$ case, a ridge penalized MLE with an optimized regularization parameter might be a good alternative. You can commit any nonnegative weights by using `al.weights` in the initialization of the corresponding `penalty` object. If you left it unspecified then `al.weights = 1` will be used.

Value

An object of the class `penalty`. This is a list with elements

`penalty`	character: the penalty name.
`lambda`	double: the (nonnegative) regularization parameter.
`getpenmat`	function: computes the diagonal penalty matrix.
`first.derivative`	function: This returns the p-dimensional vector of the first derivative of the p penalty terms with respect to $\|\beta_i\|$. However, this function is not really required for adaptive lasso.

Author(s)

Jan Ulbricht

References

Zou, H. (2006) The adaptive lasso and its oracle properties. *Journal of the American Statistical Association* **101**, 1418–1429.

See Also

`penalty`, `lasso`

---

ao *Approximated Octagon Penalty*

Description

Object of the `penalty` class to handle the AO penalty (Ulbricht, 2010).

Usage

```
ao (lambda = NULL, ...)
```

Arguments

`lambda`	two dimensional tuning parameter parameter. The first component corresponds to the regularization parameter λ. This must be a nonnegative real number. The second component indicates the exponent γ of the bridge penalty term. See details below. It must hold that $\gamma > 1$.
`...`	further arguments.

Details

The basic idea of the AO penalty is to use a linear combination of L_1-norm and the bridge penalty with $\gamma > 1$ where the amount of the bridge penalty part is driven by empirical correlation. So, consider the penalty

$$P_{\tilde{\lambda}}^{ao}(\boldsymbol{\beta}) = \sum_{i=2}^{p} \sum_{j<i} p_{\tilde{\lambda},ij}(\boldsymbol{\beta}), \quad \tilde{\lambda} = (\lambda, \gamma)$$

where

$$p_{\tilde{\lambda},ij} = \lambda[(1 - |\varrho_{ij}|)(|\beta_i| + |\beta_j|) + |\varrho_{ij}|(|\beta_i|^\gamma + |\beta_j|^\gamma)],$$

and ϱ_{ij} denotes the value of the (empirical) correlation of the i-th and j-th regressor. Since we are going to approximate an octagonal polytope in two dimensions, we will refer to this penalty as *approximated octagon* (AO) penalty. Note that $P_{\tilde{\lambda}}^{ao}(\boldsymbol{\beta})$ leads to a dominating lasso term if the regressors are uncorrelated and to a dominating bridge term if they are nearly perfectly correlated.

The penalty can be rearranged as

$$P_{\tilde{\lambda}}^{ao}(\boldsymbol{\beta}) = \sum_{i=1}^{p} p_{\tilde{\lambda},i}^{ao}(\beta_i),$$

where

$$p_{\tilde{\lambda},i}^{ao}(\beta_i) = \lambda \left\{ |\beta_i| \sum_{j \neq i} (1 - |\varrho_{ij}|) + |\beta_i|^\gamma \sum_{j \neq i} |\varrho_{ij}| \right\}.$$

It uses two tuning parameters $\tilde{\lambda} = (\lambda, \gamma)$, where λ controls the penalty amount and γ manages the approximation of the pairwise L_∞-norm.

Value

An object of the class `penalty`. This is a list with elements

`penalty`	character: the penalty name.
`lambda`	double: the (nonnegative) regularization parameter.
`getpenmat`	function: computes the diagonal penalty matrix.

Author(s)

Jan Ulbricht

References

Ulbricht, Jan (2010) *Variable Selection in Generalized Linear Models*. Ph.D. Thesis. LMU Munich.

See Also

`penalty`, `genet`

`bridge` *Bridge Penalty*

Description

Object of the `penalty` to handle the bridge penalty (Frank \& Friedman, 1993, Fu, 1998)

Usage

```
bridge (lambda = NULL, ...)
```

Arguments

`lambda`	two dimensional tuning parameter parameter. The first component corresponds to the regularization parameter λ. This must be a nonnegative real number. The second component indicates the exponent γ of the penalty term. It must hold that $\gamma > 1$.
`...`	further arguments.

Details

The *bridge* penalty has been introduced in Frank \& Friedman (1993). See also Fu (1998). It is defined as

$$P_{\tilde{\lambda}}^{br}(\boldsymbol{\beta}) = \lambda \sum_{i=1}^{p} |\beta_i|^{\gamma}, \quad \gamma > 0,$$

where $\tilde{\lambda} = (\lambda, \gamma)$. It features an additional tuning parameter γ that controls the degree of preference for the estimated coefficient vector to align with the original, hence standardized, data axis directions in the regressor space. It comprises the lasso penalty ($\gamma = 1$) and the ridge penalty ($\gamma = 2$) as special cases.

Value

An object of the class `penalty`. This is a list with elements

`penalty`	character: the penalty name.
`lambda`	double: the (nonnegative) regularization parameter.
`getpenmat`	function: computes the diagonal penalty matrix.

Author(s)

Jan Ulbricht

References

Frank, I. E. \& J. H. Friedman (1993) A statistical view of some chemometrics regression tools (with discussion). *Technometrics* **35**, 109–148.

Fu, W. J. (1998) Penalized Regression: the bridge versus the lasso. *Journal of Computational and Graphical Statistics* **7**, 397–416.

See Also

penalty, lasso, ridge, ao, genet

`cv.lqa` *Finding Optimal Tuning Parameter via Cross-Validation or Validation Data*

Description

This function computes optimal tuning parameters for penalized GLMs that can be fitted by lqa. The optimal tuning parameter minimizes the loss function you have specified in the argument `loss.func`. However, to find the optimal one this function evaluates model performance for different tuning parameter candidates given in the argument `lambda.candidates`.

If you just give training data then a cross-validation will be applied. If you additionally provide validation data (`y.vali` and `x.vali`) then these will be used for measuring model performance while your training data (`y.train` and `x.train`) are entirely used for model fitting.

The cv.lqa function also returns `best.obj`. That is the `lqa` object as returned from lqa when it has been called with the chosen penalty family and the optimal tuning parameter.

Usage

```
  cv.lqa(y.train, x.train, intercept = TRUE, y.vali = NULL,
             x.vali = NULL, lambda.candidates, family, penalty.family,
             standardize = TRUE, n.fold, cv.folds,
             loss.func = aic.loss, control = lqa.control(), ...)
  ## S3 method for class 'cv.lqa'
print(x, ...)
```

Arguments

`y.train`	the vector of response training data.
`x.train`	the design matrix of training data. If `intercept = TRUE` then it does not matter whether a column of ones is already included in `x.train` or not. The function adjusts it if necessary.
`intercept`	logical. If 'TRUE' then an intercept is included in the model (this is recommended).
`y.vali`	an additional vector of response validation data. If given the validation data are used for evaluating the loss function.

x.vali	an additional design matrix of validation data. If given the validation data are used for evaluating the loss function. If intercept = TRUE then it does not matter whether a column of ones is already included in x.vali or not. The function adjusts it if necessary.
lambda.candidates	a list containing the tuning parameter candidates. The number of list elements must correspond to the dimension of the tuning parameter. See the accompanying 'User's Guide' for further details.
family	identifies the exponential family of the response and the link function of the model. See the description of the R function family() for further details.
penalty.family	a function or character argument identifying the penalty family. See examples below.
standardize	logical. If 'TRUE' the data are standardized (this is recommended).
n.fold	number of folds in cross-validation. This argument can be omitted if a validation set is used.
cv.folds	a list containing the indices of y.train to indicate the observations that might be used in the particular cross-validation folds. This can be omitted if a validation set is used. Moreover, it is optional as well if no validation set is given.
loss.func	a character indicating the loss function to be used in evaluating the model performance for the tuning parameter candidates. If it is missing then the aic.loss() function will be used. See details below.
control	a list of parameters for controlling the fitting process. See the documentation of lqa.control for details.
x	used in the 'print' method: a cv.lqa object as returned by cv.lqa.
...	Further arguments.

Details

This function can be used for evaluating model performance for different tuning parameter candidates. If you just give training data a cross-validation will be applied. If you additionally provide validation data then those data will be used for measuring the performance and the training data are completely used for model fitting.

You must specify a penalty family. This can be done by giving its name as a character (e.g. penalty.family = "lasso") or as a function call (e.g. penalty.family = lasso).

The tuning parameter candidates are given in the argument lambda.candidates. Usually one should a priori generate a sequence of equidistant points and then use them as exponents to Euler's number. See example below. Note that lambda.candidates must be a list in order to cope with different numbers of candidates among the elements of the tuning parameter vector.

For evaluation you must specify a loss function. The default value is aic.loss e.g. the AIC will be used to find an optimal tuning parameter. Other already implemented loss functions are bic.loss, gcv.loss, squared.loss (quadratic loss function), dev.loss (deviance as loss function).

Value

The function cv.lqa returns an object of class cv.lqa which is a list with the following components:

lambda.opt	the optimal tuning parameter(s).
beta.opt	the MLE corresponding to the optimal tuning parameter(s).
best.pos	the positions of the optimal tuning parameter(s) in the lambda.candidates argument.
loss.mat	the array containing the loss function values of all tuning parameter candidates (rows) and all folds (columns).
best.obj	a member of the class lqa where to optimal tuning parameters are used.
loss.func	the loss function used.
exist.vali	logical whether or not a validation data set has been given as argument.
cv.folds	the cv.folds (list containing the indices of the training data that has been used in the cross-validation folds) used.
n.fold	number of folds.
mean.array	the array containing the mean performances of all tuning parameter candidates.
lambda.candidates	the original lambda.candidates argument.

Author(s)

Jan Ulbricht

See Also

lqa, predict.lqa

Examples

```
## Gaussian response + lasso penalty + aic.loss:

set.seed (1111)

n <- 200
p <- 5
X <- matrix (rnorm (n * p), ncol = p)
X[,2] <- X[,1] + rnorm (n, sd = 0.1)
X[,3] <- X[,1] + rnorm (n, sd = 0.1)
true.beta <- c (1, 2, 0, 0, -1)
y <- drop (X %*% true.beta) + rnorm (n)

cv.obj1 <- cv.lqa (y, X, intercept = TRUE,
   lambda.candidates = list (c (0.001, 0.05, 1, 5, 10)), family = gaussian (),
   penalty.family = lasso, n.fold = 5,
   loss.func = "aic.loss")
```

```
cv.obj1


## Binary response + fused.lasso penalty + dev.loss:

 n <- 100
 p <- 5

 set.seed (1234)
 x <- matrix (rnorm (n * p), ncol = p)
 x[,2] <- x[,1] + rnorm (n, sd = 0.01)
 x[,3] <- x[,1] + rnorm (n, sd = 0.1)
 beta <- c (1, 2, 0, 0, -1)
 prob1 <- 1 / (1 + exp (drop (-x %*% beta)))
 y <- sapply (prob1, function (prob1) {rbinom (1, 1, prob1)})

cv.obj2 <- cv.lqa (y, x, family = binomial (), penalty.family =
 fused.lasso, lambda.candidates = list (c (0.001, 0.05, 0.5, 1, 5),
 c (0.001, 0.01, 0.5)), n.fold = 5, loss.func = "dev.loss")
cv.obj2
```

`cv.nng` *Finding Optimal Tuning Parameter via Cross-Validation or Validation Data for non-negative garrote penalization*

Description

This function computes optimal tuning parameters for GLMs with non-negative garrote penalization that can be fitted by the algorithm described in Ulbricht (2010), Subsection 3.4.1. The optimal tuning parameter minimizes the loss function you have specified in the argument `loss.func`. However, to find the optimal one this function evaluates model performance for different tuning parameter candidates given in the argument `lambda.candidates`.

If you just give training data then a cross-validation will be applied. If you additionally provide validation data (`y.vali` and `x.vali`) then these will be used for measuring model performance while your training data (`y.train` and `x.train`) are entirely used for model fitting.

The `cv.lqa` function also returns `best.obj`. That is the `lqa` object as returned from `lqa` when it has been called with the chosen penalty family and the optimal tuning parameter.

Usage

```
cv.nng(y.train, x.train, intercept = TRUE, y.vali = NULL,
           x.vali = NULL, lambda.nng, family, penalty,
           standardize = TRUE, n.fold, cv.folds,
           loss.func = aic.loss, control = lqa.control(), ...)
```

Arguments

y.train	the vector of response training data.
x.train	the design matrix of training data. If intercept = TRUE then it does not matter whether a column of ones is already included in x.train or not. The function adjusts it if necessary.
intercept	logical. If 'TRUE' then an intercept is included in the model (this is recommended).
y.vali	an additional vector of response validation data. If given the validation data are used for evaluating the loss function.
x.vali	an additional design matrix of validation data. If given the validation data are used for evaluating the loss function. If intercept = TRUE then it does not matter whether a column of ones is already included in x.vali or not. The function adjusts it if necessary.
lambda.nng	a list containing the tuning parameter candidates for the non-negative garrote penalization.
family	identifies the exponential family of the response and the link function of the model. See the description of the R function family() for further details.
penalty	a description of the penalty to be used in the fitting procedure. This must be a penalty object. See penalty for details on penalty functions.
standardize	logical. If 'TRUE' the data are standardized (this is recommended).
n.fold	number of folds in cross-validation. This argument can be omitted if a validation set is used.
cv.folds	a list containing the indices of y.train to indicate the observations that might be used in the particular cross-validation folds. This can be omitted if a validation set is used. Moreover, it is optional as well if no validation set is given.
loss.func	a character indicating the loss function to be used in evaluating the model performance for the tuning parameter candidates. If it is missing then the aic.loss() function will be used. See details below.
control	a list of parameters for controlling the fitting process. See the documentation of lqa.control for details.
x	used in the 'print' method: a cv.lqa object as returned by cv.lqa.
...	Further arguments.

Details

This function can be used for evaluating model performance of different tuning parameter candidates for the non-negative garrote penalty. If you just give training data a cross-validation will be applied. If you additionally provide validation data then those data will be used for measuring the performance and the training data are completely used for model fitting.

If the non-negative garrote estimate might be based on the maximum likelihood estimator (MLE) then you can leave the argume penalty unspecified. Otherwise, e.g. if you want to apply a ridge penalty, you should specify it as follows, e.g. penalty = ridge (lambda.opt), where lambda.opt is the optimized ridge regularization parameter as determined from a previous cross-validation procedure.

For evaluation you must specify a loss function. The default value is `aic.loss` e.g. the AIC will be used to find an optimal tuning parameter. Other already implemented loss functions are `bic.loss`, `gcv.loss`, `squared.loss` (quadratic loss function), `dev.loss` (deviance as loss function).

Value

The function `cv.nng` returns an object of class `cv.lqa` which is a list with the following components:

`lambda.opt`	the optimal tuning parameter(s).
`beta.opt`	the MLE corresponding to the optimal tuning parameter(s).
`best.pos`	the positions of the optimal tuning parameter(s) in the `lambda.candidates` argument.
`loss.mat`	the array containing the loss function values of all tuning parameter candidates (rows) and all folds (columns).
`best.obj`	a member of the class `lqa` where to optimal tuning parameters are used.
`loss.func`	the loss function used.
`exist.vali`	logical whether or not a validation data set has been given as argument.
`cv.folds`	the cv.folds (list containing the indices of the training data that has been used in the cross-validation folds) used.
`n.fold`	number of folds.
`mean.array`	the array containing the mean performances of all tuning parameter candidates.
`lambda.candidates`	the original `lambda.candidates` argument.

Author(s)

Jan Ulbricht

References

Ulbricht, Jan (2010) *Variable Selection in Generalized Linear Models*. Ph.D. Thesis. Ludwig Maximilians University Munich.

See Also

`lqa`, `predict.lqa`

Examples

```
set.seed (1234)

n <- 50
p <- 5
X <- matrix (rnorm (n * p), ncol = p)
X[,2] <- X[,1] + rnorm (n, sd = 0.1)
```

```
X[,3] <- X[,1] + rnorm (n, sd = 0.1)
true.beta <- c (1, 2, 0, 0, -1)
y <- drop (X %*% true.beta) + rnorm (n)

family1 <- binomial ()
eta.true <- drop (X %*% true.beta)
mu.true <- family1$linkinv (eta.true)
nvec <- 1 : n
y2 <- sapply (nvec, function (nvec) {rbinom (1, 1, mu.true[nvec])})


nng.obj <- cv.nng (y.train = y2, x.train = X, lambda.nng =
             list (c (0.01, 0.1, 1, 5, 10)), family = binomial (),
             n.fold = 5, penalty = NULL)

cv.obj <- cv.lqa (y.train = y2, x.train = X, lambda.candidates =
             list (c (0.01, 0.1, 1, 5, 10)), family = binomial (),
             n.fold = 5, penalty = ridge)

lambda.opt <- cv.obj$lambda.opt
nng.obj2 <- cv.nng (y.train = y2, x.train = X, lambda.nng =
             list (c (0.01, 0.1, 1, 5, 10)), family = binomial (),
             n.fold = 5, penalty = ridge (lambda.opt))

coef (nng.obj$best.obj)
coef (cv.obj$best.obj)
coef (nng.obj2$best.obj)
```

`enet` *Elastic Net Penalty*

Description

Object of the `penalty` to handle the elastic net (enet) penalty (Zou \& Hastie, 2005)

Usage

```
enet (lambda = NULL, ...)
```

Arguments

`lambda`	two-dimensional tuning parameter. The first component corresponds to the regularization parameter λ_1 of the lasso penalty, the second one to the regularization parameter λ_2 of the ridge penalty. Both must be nonnegative.
`...`	further arguments.

Details

The elastic net penalty has been introduced in the linear model context by Zou \& Hastie (2005). The elastic net enables simultaneous automatic variable selection and continuous shrinkage. Furthermore, contrary to the lasso it can select groups of correlated variables. This is related to the so called grouping effect, where strongly correlated regressors tend to be in or out of the model together.

The *elastic net* penalty

$$P_\lambda^{en}(\boldsymbol{\beta}) = \lambda_1 \sum_{i=1}^{p} |\beta_i| + \lambda_2 \sum_{i=1}^{p} \beta_i^2, \quad \lambda = (\lambda_1, \lambda_2)$$

is a linear combination of the lasso penalty and the ridge penalty. Therefore the penalty covers these both as extreme cases.

Value

An object of the class `penalty`. This is a list with elements

`penalty`	character: the penalty name.
`lambda`	double: the (nonnegative) tuning parameter.
`getpenmat`	function: computes the diagonal penalty matrix.

Author(s)

Jan Ulbricht

References

Zou, H. \& T. Hastie (2005) Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society B* **67**, 301–320.

See Also

`penalty`, `genet`, `lasso`, `ridge`

`ForwardBoost` *Computation of the ForwardBoost Algorithm*

Description

This function fits a GLM based on penalized likelihood inference by the ForwardBoost algorithm. However, it is primarily intended for internal use. You can access it via the argument setting `method = "ForwardBoost"` in `lqa`, `cv.lqa` or `plot.lqa`.

Usage

```
ForwardBoost (x, y, family = NULL, penalty = NULL, intercept =
     TRUE, weights = rep (1, nobs), control = lqa.control (),
     nu = 1, monotonic = TRUE, ...)
```

Arguments

x	matrix of standardized regressors. This matrix does not need to include a first column of ones when a GLM with intercept is to be fitted.
y	vector of observed response values.
family	a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. See `family()` for further details.
penalty	a description of the penalty to be used in the fitting procedure, e.g. `penalty = lasso (lambda = 1.7)`.
intercept	a logical object indicating whether the model should include an intercept (this is recommended) or not. The default value is `intercept = TRUE`.
weights	some additional weights for the observations.
control	a list of parameters for controlling the fitting process. See `lqa.control`.
nu	parameter ν from the ForwardBoost algorithm. This parameter manages the update step length. See Ulbricht (2010).
monotonic	a logical variable. If `TRUE` then the number of active regressors increases monotonically during the iterations. This is in line with the ForwardBoost algorithm and hence recommended.
...	further arguments.

Details

The ForwardBoost algorithm has been described in Ulbricht (2010). So see there for a more detailed technical description.

Value

`GBlockBoost` returns a list containing the following elements:

coefficients	the vector of standardized estimated coefficients.
beta.mat	matrix containing the estimated coefficients from all iterations (rowwise).
m.stop	the number of iterations until AIC reaches its minimum.
stop.at	the number of iterations until convergence.
aic.vec	vector of AIC criterion through all iterations.
bic.vec	vector of BIC criterion through all iterations.
converged	a logical variable. This will be `TRUE` if the algorithm has indeed converged.
min.aic	minimum value of AIC criterion.
min.bic	minimum value of BIC criterion.
tr.H	the trace of the hat matrix.
tr.Hatmat	vector of hat matrix traces through all iterations.
dev.m	vector of deviances through all iterations.

Author(s)

Jan Ulbricht

References

Ulbricht, Jan (2010) *Variable Selection in Generalized Linear Models*. Ph.D. Thesis. LMU Munich.

See Also

`lqa`, `GBlockBoost`

`fused.lasso` *Fused Lasso Penalty*

Description

Object of the `penalty` to handle the fused lasso penalty (Tibshirani et al., 2005)

Usage

```
fused.lasso (lambda = NULL, ...)
```

Arguments

`lambda`	two-dimensional tuning parameter. The first component corresponds to the regularization parameter λ_1 of the lasso penalty, the second one to the regularization parameter λ_2 of the fusion penalty. Both must be nonnegative.
`...`	further arguments

Details

The *fused lasso* penalty is defined as

$$P_{\tilde{\lambda}}^{fl}(\boldsymbol{\beta}) = \lambda_1 \sum_{i=1}^{p} |\beta_i| + \lambda_2 \sum_{i=2}^{p} |\beta_i - \beta_{i-1}|,$$

where $\tilde{\lambda} = (\lambda_1, \lambda_2)$ contains two regularization parameters. The main idea of the fused lasso penalty is to encourage sparsity in the coefficients by using the L_1-norm lasso penalty, and additionally to force sparsity in the differences of the coefficients by the L_1-norm of their differences as reflected in the second penalty term. As a result, the fused lasso penalty conveys the estimated coefficients to behave in a smooth manner, with only a small number of big jumps. See Tibshirani et al. (2005) for further details.

Value

An object of the class `penalty`. This is a list with elements

`penalty`	character: the penalty name.
`lambda`	double: the (nonnegative) regularization parameter.
`first.derivative`	function: This returns the J-dimensional vector of the first derivative of the J penalty terms with respect to $\lvert\mathbf{a}_j^\top \boldsymbol{\beta}\rvert$.
`a.coefs`	function: This returns the p-dimensional coefficient vector $\mathbf{a}_j$ of the J penalty terms.

Author(s)

Jan Ulbricht

References

Tibshirani, R., M. Saunders, S. Rosset, J. Zhu and K. Knight (2005) Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society B* **67**, 91–108.

See Also

`penalty`, `lasso`, `ridge`, `weighted.fusion`

`GBlockBoost` *Computation of the GBlockBoost Algorithm or Componentwise Boosting*

Description

This function fits a GLM based on penalized likelihood inference by the GBlockBoost algorithm. However, it is primarily intended for internal use. You can access it via the argument setting `method = "GBlockBoost"` in `lqa`, `cv.lqa` or `plot.lqa`. If you use `componentwise = TRUE` then componentwise boosting will be applied.

Usage

```
GBlockBoost (x, y, family = NULL, penalty = NULL, intercept =
    TRUE, weights = rep (1, nobs), control = lqa.control (),
    componentwise, ...)
```

Arguments

x	matrix of standardized regressors. This matrix does not need to include a first column of ones when a GLM with intercept is to be fitted.
y	vector of observed response values.
family	a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. See `family()` for further details.
penalty	a description of the penalty to be used in the fitting procedure, e.g. `penalty = lasso (lambda = 1.7)`.
intercept	a logical object indicating whether the model should include an intercept (this is recommended) or not. The default value is `intercept = TRUE`.
weights	some additional weights for the observations.
control	a list of parameters for controlling the fitting process. See `lqa.control`.
componentwise	if `TRUE` then componentwise boosting will be applied, e.g. there is just a single regressors updated during each iteration. Otherwise GBlockBoost will be applied. If this argument is missing and your penalty is `ridge` then it is set to `componentwise = TRUE`. Note that this will coincide with an application of the RidgeBoost algorithm. In all other cases of a missing argument `componentwise = FALSE` is used.
...	further arguments.

Details

The GBlockBoost algorithm has been introduced in Ulbricht \& Tutz (2008). For a more detailed technical description, also for componentwise boosting, see Ulbricht (2010).

Value

`GBlockBoost` returns a list containing the following elements:

coefficients	the vector of standardized estimated coefficients.
beta.mat	matrix containing the estimated coefficients from all iterations (rowwise).
m.stop	the number of iterations until AIC reaches its minimum.
stop.at	the number of iterations until convergence.
aic.vec	vector of AIC criterion through all iterations.
bic.vec	vector of BIC criterion through all iterations.
converged	a logical variable. This will be `TRUE` if the algorithm has indeed converged.
min.aic	minimum value of AIC criterion.
min.bic	minimum value of BIC criterion.
tr.H	the trace of the hat matrix.
tr.Hatmat	vector of hat matrix traces through all iterations.
dev.m	vector of deviances through all iterations.

Author(s)

Jan Ulbricht

References

Ulbricht, Jan (2010) *Variable Selection in Generalized Linear Models*. Ph.D. Thesis. LMU Munich.

Ulbricht, J. \& G. Tutz (2008) Boosting correlation based penalization in generalized linear models. In Shalabh \& C. Heumann (Eds.) *Recent Advances in Linear Models and Related Areas*. Heidelberg: Springer.

See Also

`lqa`, `ForwardBoost`

`genet` *Generalized Elastic Net Penalty*

Description

Object of the `penalty` to handle the Generalized Elastic Net (GENET) penalty (Ulbricht, 2010).

Usage

```
genet (lambda = NULL, ...)
```

Arguments

`lambda`	three-dimensional tuning parameter. The first component corresponds to the regularization parameter λ. This must be a nonnegative real number. The second component $0 \leq \alpha \leq 1$ drives the linear combination of L_1 penalty and the bridge penalty. The third component indicates the exponent γ of the bridge penalty term. See details below. It must hold that $\gamma > 1$.
`...`	further arguments.

Details

The GENET penalty can be defined as

$$P_{\bar{\lambda}}^{genet}(\boldsymbol{\beta}) = \lambda \left\{ \alpha \sum_{i=1}^{p} |\beta_i| + (1-\alpha) \sum_{i=1}^{p} |\beta_i|^{\gamma} \right\}, \quad 0 \leq \alpha \leq 1,\ \gamma > 1$$

with tuning parameter vector $\bar{\lambda} = (\lambda, \alpha, \gamma)$.

The regularization parameter λ determines the overall relevance of the GENET penalty. The balance between L_1-norm penalization, and hence variable selection, and bridge penalization for incorporating the grouping effect is managed by an overall tuning parameter α. For motivation and further details on the GENET penalty see Ulbricht (2010).

Value

An object of the class `penalty`. This is a list with elements

`penalty`	character: the penalty name.
`lambda`	double: the (nonnegative) tuning parameter.
`getpenmat`	function: computes the diagonal penalty matrix.

Author(s)

Jan Ulbricht

References

Ulbricht, Jan (2010) *Variable Selection in Generalized Linear Models*. Ph.D. Thesis. LMU Munich.

See Also

`penalty`, `ao`

`get.Amat` *Computation of the approximated penalty matrix.*

Description

The function `get.Amat` computes and returns

$$\mathbf{A}_\lambda = \sum_{j=1}^{J} \frac{p'_{\lambda,j}(|\mathbf{a}_j^\top \boldsymbol{\beta}|)}{\sqrt{(\mathbf{a}_j^\top \boldsymbol{\beta})^2 + c}} \mathbf{a}_j \mathbf{a}_j^\top,$$

where $c > 0$ is a small real number. However, this function is primarily intended for internal use. It acts as a link between `penalty` objects and methods which require the approximated penalty matrix $\mathbf{A}_\lambda$.

Usage

```
get.Amat (initial.beta = NULL, penalty = NULL, intercept = TRUE,
     c1 = lqa.control()$c1, x = NULL, ...)
```

Arguments

`initial.beta`	the current beta vector.
`penalty`	member of the `penalty` class, the penalty to be used.
`intercept`	logical. If 'TRUE' an intercept is included in the model.
`c1`	double: small positive real number used in the approximation of the linear combinations in the penalty.
`x`	optional argument containing the original regressor matrix. This will be used by some penalties, such as `penalreg` or `ao`.
`...`	further arguments.

Details

See penalty or the accompanying 'User's Guide' for further details on $\mathbf{A}_\lambda$.

Value

This function returns a $(p \times p)$-dimensional matrix or if an intercept is included a $((p+1) \times (p+1))$-dimensional matrix.

Author(s)

Jan Ulbricht

See Also

penalty, lqa

Examples

```
penalty <- lasso (lambda = 1.5)
beta <- c (1, -2, 3, -4)
get.Amat (initial.beta = beta, penalty = penalty, intercept = FALSE)
```

---

icb *Improved Correlation-based Penalty*

Description

Object of the penalty class to handle the Improved Correlation-Based (ICB) Penalty (Ulbricht, 2010).

Usage

```
icb(lambda = NULL, ...)
```

Arguments

lambda	two dimensional tuning parameter parameter. The first component corresponds to the regularization parameter λ_1 for the lasso penalty term, the second one λ_2 for the correlation-based penalty. Both parameters must be nonnegative.
...	further arguments

Details

The improved correlation-based (ICB) penalty is defined as

$$P_\lambda^{icb}(\boldsymbol{\beta}) = \lambda_1|\boldsymbol{\beta}|_1 + \frac{1}{2}\lambda_2\boldsymbol{\beta}^\top\mathbf{M}^{cb}\boldsymbol{\beta},$$

with tuning parameter $\lambda = (\lambda_1, \lambda_2)$, where $\mathbf{M}^{cb} = (m_{ij})$ is determined by $m_{ij} = 2\sum_{s\neq i}\frac{1}{1-\varrho_{is}^2}$ if $i = j$, and $m_{ij} = -2\frac{\varrho_{ij}}{1-\varrho_{ij}^2}$ otherwise. The ICB has been introduced to overcome the major drawback of the correlation based-penalized estimator, that is its lack of sparsity. See Ulbricht (2010) for details.

Value

An object of the class `penalty`. This is a list with elements

`penalty`	character: the penalty name.
`lambda`	double: the (nonnegative) regularization parameter.
`getpenmat`	function: computes the diagonal penalty matrix.

Author(s)

Jan Ulbricht

References

Ulbricht, Jan (2010) *Variable Selection in Generalized Linear Models*. Ph.D. Thesis. LMU Munich.

See Also

`penalty`, `penalreg`, `licb`, `weighted.fusion`

`lasso` *Lasso Penalty*

Description

Object of the `penalty` class to handle the lasso penalty (Tibshirani, 1996).

Usage

```
lasso (lambda = NULL, ...)
```

Arguments

`lambda`	regularization parameter. This must be a nonnegative real number.
`...`	further arguments

Details

The 'classic' penalty that incorporates variables selection. As introduced in Tibshirani (1996) the lasso penalty is defined as

$$P_\lambda^r(\boldsymbol{\beta}) = \lambda \sum_{i=1}^{p} |\beta_j|.$$

Value

An object of the class `penalty`. This is a list with elements

`penalty`	character: the penalty name.
`lambda`	double: the (nonnegative) regularization parameter.
`getpenmat`	function: computes the diagonal penalty matrix.

Author(s)

Jan Ulbricht

References

Tibshirani, R. (1996) Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society B* **58**, 267–288.

See Also

`penalty`, `ridge`, `penalreg`

---

licb *L1-Norm based Improved Correlation-based Penalty*

Description

Object of the `penalty` class to handle the L1-Norm based Improved Correlation-Based (LICB) Penalty (Ulbricht, 2010).

Usage

```
licb (lambda = NULL, ...)
```

Arguments

`lambda`	two-dimensional tuning parameter parameter. The first component corresponds to the regularization parameter λ_1 for the lasso penalty term, the second one λ_2 for the L_1-norm based correlation penalty term. Both parameters must be nonnegative.
`...`	further arguments

Details

The improved correlation-based (LICB) penalty is defined as

$$P_\lambda^{licb}(\boldsymbol{\beta}) = \lambda_1 \sum_{i=1}^{p} |\beta_i| + \lambda_2 \sum_{i=1}^{p-1} \sum_{j>i} \left\{ \frac{|\beta_i - \beta_j|}{1 - \varrho_{ij}} + \frac{|\beta_i + \beta_j|}{1 + \varrho_{ij}} \right\}.$$

The LICB has been introduced to overcome the major drawback of the correlation based-penalized estimator, that is its lack of sparsity. See Ulbricht (2010) for details.

Value

An object of the class `penalty`. This is a list with elements

`penalty` character: the penalty name.

`lambda` double: the (nonnegative) regularization parameter.

`first.derivative`
function: This returns the J-dimensional vector of the first derivative of the J penalty terms with respect to $|\mathbf{a}_j^\top \boldsymbol{\beta}|$.

`a.coefs` function: This returns the p-dimensional coefficient vector $\mathbf{a}_j$ of the J penalty terms.

Author(s)

Jan Ulbricht

References

Ulbricht, Jan (2010) *Variable Selection in Generalized Linear Models*. Ph.D. Thesis. LMU Munich.

See Also

`penalty`, `penalreg`, `icb`, `weighted.fusion`

`lqa` *Fitting penalized Generalized Linear Models with the LQA algorithm*

Description

'lqa' is used to fit penalized generalized linear models, specified by giving a symbolic description of the linear predictor and descriptions of the error distribution and the penalty.

Usage

```
lqa (x, ...)

lqa.update2 (x, y, family = NULL, penalty = NULL, intercept = TRUE,
             weights = rep (1, nobs), control = lqa.control (),
             initial.beta, mustart, eta.new, gamma1 = 1, ...)

## S3 method for class 'formula'
lqa(formula, data = list (), weights = rep (1, nobs), subset,
            na.action, start = NULL, etastart, mustart, offset, ...)

## Default S3 method:
lqa(x, y, family = gaussian (), penalty = NULL, method = "lqa.update2",
            weights = rep (1, nobs), start = NULL,
            etastart = NULL, mustart = NULL, offset = rep (0, nobs),
            control = lqa.control (), intercept = TRUE,
            standardize = TRUE, ...)
```

Arguments

formula	a symbolic description of the model to be fit. The details of model specification are given below.
data	an optional 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 'lqa' is called.
family	a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See `family()` for details of family functions.)
penalty	a description of the penalty to be used in the fitting procedure. This must be a penalty object. See `penalty` for details on penalty functions.
method	a character string naming the function used to estimate the model. The default value `method = lqa.update2` applies the LQA algorithm.
intercept	a logical object whether the model should include an intercept (this is recommended) or not. The default value is TRUE.
standardize	a logical object, whether the regressors should be standardized (this is recommended) or not. The default value is TRUE.
weights	an optional vector of weights to be used in the fitting process.
start	starting values for the parameters in the linear predictor.
etastart	starting values for the linear predictor.
mustart	starting values for the vector of means (response).

gamma1	additional step length parameter used in lqa.update2 to enforce convergence if necessary.
offset	this can be used to specify an a priori known component to be included in the linear predictor during fitting.
control	a list of parameters for controlling the fitting process. See the documentation of lqa.control for details.
na.action	a function which indicates what should happen when the data contain 'NA's.
subset	an optional vector specifying a subset of observations to be used in the fitting process.
x, y	Used in 'lqa.default': 'x' is a design matrix (with additional column of ones if an intercept should be included in the model) of dimension 'n * p', and 'y' is a vector of observations of length 'n'.
initial.beta	optional initial values of beta used in the fitting procedures.
eta.new	optional intial values of predictors used in the fitting procedures.
...	further arguments passed to or from other methods.

Details

A typical formula has the form 'response ~ terms' where 'response' is the (numeric) response vector and 'terms' is a series of terms which specifies a linear predictor for 'response'. The use is similar to that of the glm() function. As there, the right hand side of the model formula specifies the form of the linear predictor and hence gives the link function of the mean of the response, rather than the mean of the response directly. Per default an intercept is included in the model. If it should be removed then use formulae of the form 'response ~ 0 + terms' or 'response ~ terms - 1'.

Also lqa takes a family argument, which is used to specify the distribution from the exponential family to use, and the link function that is to go with it. The default value is the canonical link.

Value

lqa returns an object of class lqa which inherits from the classes glm and lm.

The generic accessor functions coefficients, fitted.values and residuals can be used to extract various useful features of the object returned by lqa.

Note it is highly recommended to include an intercept in the model (e.g. use Intercept = TRUE). If you use Intercept = FALSE in the classical linear model then make sure that your y argument is already centered! Otherwise the model would not be valid.

An object of class lqa is a list containing at least the following components:

coefficients	a named vector of unstandardized coefficients.
residuals	the residuals based on the estimated coefficients.
fitted.values	the fitted mean values, obtained by transforming the linear predictors by the inverse of the link function.
family	the family object used.
penalty	the penalty object used, indicating which penalty has been used.

`linear.predictors`	the linear fit on link scale.
`deviance`	up to a constant, minus twice the maximimized (unpenalized) log-likelihood.
`aic`	Akaike's Information Criterion, minus twice the maximized log-likelihood plus twice the trace of the hat matrix (so assuming that the dispersion is known).
`bic`	Bayesian Information Criterion, minus twice the maximized log-likelihood plus log (nobs) times the trace of the hat matrix (so assuming that the dispersion is known).
`null.deviance`	deviance of the null model (that only includes a constant)
`n.iter`	the number of iterations until convergence.
`best.iter`	the number of iterations until AIC reaches its minimum.
`weights`	diagonal elements of the weight matrix in GLMs.
`prior.weights`	the weights as optionally given as argument.
`df.residual`	the residual degrees of freedom.
`df.null`	the residual degrees of freedom for the null model.
`converged`	a logical variable. TRUE if the algorithm indeed converged.
`mean.x`	The vector of means of the regressors.
`norm.x`	The vector of Euclidean norms of the regressors.
`Amat`	The quadratically approximated penalty matrix corresponding to the penalty used.
`method`	The argument indicating the fitting method.
`rank`	The trace of the hat matrix.
`y`	the original response vector used to fit the model.
`x`	the original regressor matrix (including an intercept if given) used to fit the model.
`fit.obj`	the fitted object as returned from the fitting method (e.g. from `lqa.update2`).

Author(s)

Jan Ulbricht

See Also

`cv.lqa`, `penalty`

Examples

```
set.seed (1111)

n <- 200
p <- 5
X <- matrix (rnorm (n * p), ncol = p)
X[,2] <- X[,1] + rnorm (n, sd = 0.1)
X[,3] <- X[,1] + rnorm (n, sd = 0.1)
```

```
true.beta <- c (1, 2, 0, 0, -1)
y <- drop (X %*% true.beta) + rnorm (n)

obj1 <- lqa (y ~ X, family = gaussian (), penalty = lasso (1.5),
             control = lqa.control ())
obj1$coef


set.seed (4321)

n <- 25
p <- 5
X <- matrix (rnorm (n * p), ncol = p)
X[,2] <- X[,1] + rnorm (n, sd = 0.1)
X[,3] <- X[,1] + rnorm (n, sd = 0.1)
true.beta <- c (1, 2, 0, 0, -1)

family1 <- binomial ()
eta.true <- drop (X %*% true.beta)
mu.true <- family1$linkinv (eta.true)
prob1 <- sum (as.integer (y > 0)) / n
nvec <- 1 : n
y2 <- sapply (mu.true, function (n.vec) {rbinom (1, 1, mu.true)})

obj2 <- lqa (y2 ~ X, family = binomial (),
             penalty = fused.lasso (c (0.0001, 0.2)))
obj2$coef
```

`lqa.control` *Auxiliary for controlling lqa fitting*

Description

Auxiliary function as user interface for `lqa` fitting. Typically only used when calling `lqa` or `lqa.update2`.

Usage

```
lqa.control (x = NULL, var.eps = .Machine$double.eps, max.steps = 5000,
             conv.eps = 0.001, conv.stop = TRUE, c1 = 1e-08,
             digits = 5, ...)
```

Arguments

`x`	object of class 'lqa'. This optional argument is just included to be in line with the S3 class concept.
`var.eps`	tolerance in checking for zero variance of some regressors.
`max.steps`	maximum number of steps in the lqa algorithm.
`conv.eps`	tolerance for convergence break in parameter updating.

conv.stop	whether or not to stop the iterations when estimated coefficients are converged.
c1	controls the amount of approximation of linear combinations in the penalty term.
digits	number of digits of tuning parameter candidates to take into consideration when returning the loss array and mean array in cv.lqa.
...	further arguments.

Value

A list with the arguments as components.

Author(s)

Jan Ulbricht

See Also

lqa, cv.lqa

Examples

```
set.seed (1111)

n <- 200
p <- 5
X <- matrix (rnorm (n * p), ncol = p)
X[,2] <- X[,1] + rnorm (n, sd = 0.1)
X[,3] <- X[,1] + rnorm (n, sd = 0.1)
true.beta <- c (1, 2, 0, 0, -1)
y <- drop (X %*% true.beta) + rnorm (n)

control.obj <- lqa.control (max.steps = 200, conv.eps = 1e-3,
   conv.stop = FALSE)
obj <- lqa (y ~ X, family = gaussian (), penalty = lasso (1.5),
            control = control.obj)
obj$coef
```

---

oscar *OSCAR Penalty*

Description

Object of the penalty class to handle the OSCAR penalty (Bondell \& Reich, 2008)

Usage

```
oscar (lambda = NULL, ...)
```

Arguments

`lambda`	two-dimensional tuning parameter. The first component corresponds to the regularization parameter λ that drives the relevance of the OSCAR penalty for likelihood inference. The second component corresponds to c (see details below) Both must be nonnegative.
`...`	further arguments

Details

Bondell \& Reich (2008) propose a shrinkage method for linear models called OSCAR that simultaneously select variables while grouping them into predictive clusters. The OSCAR penalty is defined as

$$P_{\tilde{\lambda}}^{osc}(\boldsymbol{\beta}) = \lambda \left(\sum_{k=1}^{p} |\beta_k| + c \sum_{j<k} \max\{|\beta_j|, |\beta_k|\} \right), \quad \tilde{\lambda} = (\lambda, c)$$

where $c \geq 0$ and $\lambda > 0$ are tuning parameters with c controlling the relative weighting of the L_∞-norms and λ controlling the magnitude of penalization. The L_1-norm entails sparsity, while the pairwise maximum (L_∞-)norm encourages equality of coefficients.

Due to equation (3) in Bondell \& Reich (2008), we use the alternative formulation

$$P_{\tilde{\lambda}}^{osc}(\boldsymbol{\beta}) = \lambda \sum_{j=1}^{p} \{c(j-1)+1\} |\beta|_{(j)},$$

where $|\beta|_{(1)} \leq |\beta|_{(2)} \leq \ldots \leq |\beta|_{(p)}$ denote the ordered absolute values of the coefficients. However, there could be some difficulties in the LQA algorithm since we need an ordering of regressors which can differ between two adjacent iterations. In the worst case, this can lead to oscillations and hence to no convergence of the algorithm. Hence, for the OSCAR penalty it is recommend to use $\gamma < 1$, e.g. $\gamma = 0.01$ when to apply `lqa.update2` for fitting the GLM in order to facilitate convergence.

Value

An object of the class `penalty`. This is a list with elements

`penalty`	character: the penalty name.
`lambda`	double: the (nonnegative) regularization parameter.
`first.derivative`	function: This returns the J-dimensional vector of the first derivative of the J penalty terms with respect to $\lvert\mathbf{a}_j^\top \boldsymbol{\beta}\rvert$.
`a.coefs`	function: This returns the p-dimensional coefficient vector $\mathbf{a}_j$ of the J penalty terms.

Author(s)

Jan Ulbricht

References

Bondell, H. D. \& B. J. Reich (2008) Simultaneous regression shrinkage, variable selection and clustering of predictors with oscar. *Biometrics* **64**, 115–123.

See Also

`penalty`, `lasso`, `fused.lasso`, `weighted.fusion`

`penalreg` *Correlation-based Penalty*

Description

Object of the `penalty` to handle the correlation-based penalty (Tutz \& Ulbricht, 2009).

Usage

```
penalreg(lambda = NULL, ...)
```

Arguments

`lambda`	regularization parameter. This must be a nonnegative real number.
`...`	further arguments

Details

The method proposed in Tutz \& Ulbricht (2009) and Ulbricht \& Tutz (2008) utilizes the correlation between regressors explicitly in the penalty term. Coefficients which correspond to pairs of covariates are weighted according to their marginal correlation. The *correlation-based penalty* is given by

$$P_\lambda^{cb}(\boldsymbol{\beta}) = \frac{\lambda}{2} \sum_{i=1}^{p-1} \sum_{j>i} \left\{ \frac{(\beta_i - \beta_j)^2}{1 - \varrho_{ij}} + \frac{(\beta_i + \beta_j)^2}{1 + \varrho_{ij}} \right\}$$

where ϱ_{ij} denotes the (empirical) correlation between the i-th and the j-th regressor. It is designed in a way so that for strong positive correlation ($\varrho_{ij} \uparrow 1$) the first term becomes dominant having the effect that estimates for β_i and β_j are similar ($\hat{\beta}_i \approx \hat{\beta}_j$). For strong negative correlation ($\varrho_{ij} \downarrow -1$) the second term becomes dominant and $\hat{\beta}_i$ will be close to $-\hat{\beta}_j$. The effect is grouping, highly correlated effects show comparable values of estimates ($|\hat{\beta}_i| \approx |\hat{\beta}_j|$) with the sign being determined by positive or negative correlation. If the regressors are uncorrelated ($\varrho_{ij} = 0$) one obtains (up to a constant) the ridge penalty, i.e. $P_\lambda^{cb}(\boldsymbol{\beta}) \propto \lambda \sum_{i=1}^{p} \beta_i^2$. Consequently, for weakly correlated data the performance is quite close to the ridge estimator. Therefore, as for the elastic net ridge regression is a limiting case.

The correlation-based penalty is a quadratic penalty. Consequently, in general it will not be able to select variables. For this reason there have been introduced some advanced boosting techniques, such as GBlockBoost or ForwardBoost. See `GBlockBoost` and `ForwardBoost` for further details.

Value

An object of the class `penalty`. This is a list with elements

`penalty`	character: the penalty name.
`lambda`	double: the (nonnegative) regularization parameter.
`getpenmat`	function: computes the diagonal penalty matrix.

Author(s)

Jan Ulbricht

References

Tutz, G. \& J. Ulbricht (2009) Penalized Regression with correlation based penalty. *Statistics and Computing* **19**, 239–253.

Ulbricht, J. \& G. Tutz (2008) Boosting correlation based penalization in generalized linear models. In Shalabh \& C. Heumann (Eds.) *Recent Advances in Linear Models and Related Areas*. Heidelberg: Springer.

See Also

`penalty`, `ridge`, `lasso`, `GBlockBoost`, `ForwardBoost`

`penalty` *Penalty Objects*

Description

Penalty objects provide a convenient way to specify the details of the penalty terms used by functions for penalized regression problems as in `lqa`. See the documentation for `lqa` for the details on how such model fitting takes place.

Usage

```
penalty (x, ...)
```

Arguments

`x`	the function `penalty` accesses the `penalty` objects which are stored within objects created by modelling functions (e.g. `lqa`).
`...`	further arguments passed to methods.

Details

penalty is a generic function with methods for objects of the lqa class. The most crucial issue of penalty objects is to compute

$$\mathbf{A}_\lambda = \sum_{j=1}^{J} \frac{p'_{\lambda,j}(|\mathbf{a}_j^\top \boldsymbol{\beta}|)}{\sqrt{(\mathbf{a}_j^\top \boldsymbol{\beta})^2 + c}} \mathbf{a}_j \mathbf{a}_j^\top,$$

where $c > 0$ is a small real number. This approximated penalty matrix will be used in the fitting procedures lqa.update2, GBlockBoost or ForwardBoost.

There are five basic methods for penalty objects: penalty, lambda, getpenmat, first.derivative, a.coefs. The methods penalty and lambda are mandatory. They are necessary to identify the penalty family and, respectively, the tuning parameter vector in the other functions of the lqa-package. But they just appear as list elements in the structure() environment. The function getpenmat() and the functions first.derivative() and a.coefs() are mutually exclusive. Whether we need the first one or the last two depends on the nature of the penalty. Hence we have to distinguish two cases

1. (i)The use of a function getpenmat() is more efficient (in a numerical sense) if
   - •the penalty matrix $\mathbf{A}_\lambda$ as given above is a diagonal matrix, e.g. if $J = p$ and $\mathbf{a}_j$, $j = 1, \ldots, J$ just contains one non-zero element, or
   - •the penalty is quadratic.

   Then the (approximate) penalty matrix $\mathbf{A}_\lambda$ can be computed directly. Most implemented penalties are of those types, e.g. ridge, lasso, scad and penalreg.
2. (ii)The combination of the functions first.derivative and a.coefs is necessary in all other cases. The fused.lasso penalty is an example for it.

Value

An object of the class penalty (which has a concise print method). This is a list with elements

penalty	character: the penalty name.
lambda	double: the (non-negative) tuning parameter.
getpenmat	function: the penalty matrix. Note this element is optional. Either getpenmat or first.derivative (and if necessary a.coefs) must be given.
first.derivative	function: This returns a J dimensional vector of the first derivative of the J penalty terms with respect to ξ_j not(!!!) to β.
a.coefs	a $p \times J$ matrix containing the coefficients of the linear combinations.

Author(s)

Jan Ulbricht

Examples

```
penalty <- lasso (lambda = 1.5)
penalty
beta <- c (1, -2, 3, -4)
penalty$first.deriv (beta)
```

`plot.lqa` *Coefficient build-ups for penalized GLMs*

Description

This function plots coefficient build-ups for GLMs that can be estimated with `lqa`.

Usage

```
plot.lqa (x, y, family, penalty.family, intercept = TRUE,
  standardize = TRUE, lambdaseq = NULL, offset.values = NULL,
  show.standardized = FALSE, add.MLE = TRUE, control = lqa.control(),
  plot.type = "l", ret.true = FALSE, really.plot = TRUE, ...)
```

Arguments

`x`	the augmented design matrix. It must contain a column of ones if an intercept is included in the model.
`y`	the vector of observed responses.
`family`	a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See `family` for details of family functions.)
`penalty.family`	a function argument identifying the penalty family. See examples below.
`intercept`	a logical object whether the model should include an intercept (this is recommended) or not. The default value is TRUE.
`standardize`	a logical object, whether the regressors should be standardized (this is recommended) or not. The default value is TRUE.
`lambdaseq`	a sequence of tuning parameter candidates for the dimension you want to plot.
`offset.values`	a vector of the same dimension as your tuning parameter. At the position of the dimension you want to plot there must be entry 'NA'. The other positions should be filled with given (and fixed) tuning parameter values, as e.g. returned from `cv.lqa`. See examples below.
`show.standardized`	logical. If 'TRUE' the standardized coefficients are plotted, otherwise the unstandardized coefficients are plotted.
`add.MLE`	logical. If 'TRUE' the unrestricted MLE is also plotted. Note this only works for 'n > p' settings. Otherwise this argument is set to 'FALSE' automatically.
`control`	list of control parameters as returned by `lqa.control`. See there for details.

plot.type	determines the line type.
ret.true	logical. If 'TRUE' then plot.lqa returns a list ret.obj with the abscissa (s1) and ordinate (beta.mat) values of the coefficient build-up.
really.plot	logical. If 'FALSE' then plot.lqa just computes and returns the ret.obj object.
...	further arguments.

Details

This function plots coefficient build-ups for a given dimension of your tuning parameter(s). The argument lambdaseq can be omitted. In this case a default sequence lambdaseq <- exp (seq (-10, 6, length = 60) is used. If your penalty consists of more than one tuning parameter you must identify the relevant dimension to plot using offset.values where you state the fixed values for the other tuning parameters. See examples below for further details.

Value

This returns a plot object, if really.plot = TRUE, or ret.obj, if really.plot = FALSE.

Author(s)

Jan Ulbricht

See Also

lqa, cv.lqa

Examples

```
set.seed (434534)
family <- binomial ()

nobs <- 50
nvars <- 5
beta.true <- c (1, 2, 0, 0, -1)
intercept <- TRUE
standardize <- TRUE

x <- matrix (rnorm (nvars * nobs), ncol = nvars)
x[,2] <- x[,1] + rnorm (nobs, sd = 0.1)
x[,3] <- x[,1] + rnorm (nobs, sd = 0.2)

eta.true <- drop (x %*% beta.true)
mu.true <- family$linkinv (eta.true)
vec1 <- 1 : nobs
y <- sapply (mu.true, function (vec1) {rbinom (1, 1, vec1)})

pdf ("fusedlasso_lambda1.pdf", width = 6, height = 6)
    # here lambda1 'lambda1' is getting varied:
plot.lqa (y = y, x = x, family = binomial (), penalty.family = fused.lasso,
```

```
          offset.values = c (NA, 0.2), add.MLE = FALSE, really.plot = TRUE)
dev.off ()
```

`predict.lqa` *Prediction Method for lqa Fits*

Description

This function computes predictions based on an `lqa` object.

Usage

```
  ## S3 method for class 'lqa'
predict(object, new.x = NULL, new.y = NULL,
          weights = rep(1, n.newobs), ...)

  ## S3 method for class 'pred.lqa'
print(x, ...)
```

Arguments

`object`	a fitted object of class `lqa`. Usually this will be an object returned from the function `lqa`.
`new.x`	Optionally, a new data frame from which to make the predictions. If omitted, the fitted linear predictors are used. Note, if given `new.x` must have the same number of entries as the estimated coefficient vectors has. That is it must include $\hat{\beta}_0$ if your model includes an intercept.
`new.y`	Optionally, a vector of new responses. If given, the deviance can be computed.
`weights`	an optional vector including weights of the new observations.
`x`	an object of class `pred.lqa` as returned from `predict.lqa`.
`...`	additional arguments.

Value

`predict.lqa` returns an object of class `pred.lqa`, i.e. this is a list with the following elements

`deviance`	the deviance based on the new observations. This element is NULL if new.y = NULL, i.e. no new responses are used in `predict.lqa`.
`tr.H`	the trace of the hat matrix of the design matrix used to fit the model. This is just an extraction from the `lqa.obj` object that is used as argument of the internal loss functions.
`n.newobs`	the number of new observations.
`eta.new`	the estimated new predictors.
`mu.new`	the estimated new responses.
`lqa.obj`	the `lqa.obj` argument.
`new.y`	the `new.y` argument.

Author(s)

Jan Ulbricht

See Also

lqa, cv.lqa

Examples

```
set.seed (1111)

n <- 200
p <- 5
X <- matrix (rnorm (n * p), ncol = p)
X[,2] <- X[,1] + rnorm (n, sd = 0.1)
X[,3] <- X[,1] + rnorm (n, sd = 0.1)
true.beta <- c (1, 2, 0, 0, -1)
y <- drop (X %*% true.beta) + rnorm (n)

cv.obj1 <- cv.lqa (y, X, intercept = TRUE, lambda.candidates =
   list (c (0.001, 0.05, 1, 5, 10), c (0.1, 0.5, 1)), family = gaussian (),
   penalty.family = fused.lasso, loss.func = "gcv.loss")
cv.obj1

beta0.hat <- coef (cv.obj1$best.obj)[1]   # extracts the estimated intercept
pred.obj <- predict.lqa (cv.obj1$best.obj, new.x = c (beta0.hat, 1, 2, 3, 4, 5))
pred.obj

cv.obj2 <- cv.lqa (y, X, intercept = TRUE, lambda.candidates =
   list (c (0.001, 0.05, 1, 5, 10), c (0.1, 0.5, 1)), family = gaussian (),
   penalty.family = fused.lasso, n.fold = 5, loss.func = "squared.loss")
cv.obj2

beta0.hat <- coef (cv.obj2$best.obj)[1]  # extracts the estimated intercept
predict.lqa (cv.obj2$best.obj, new.x = cbind (beta0.hat, matrix (1 : 10, nrow = 2)))
```

---

ridge *Ridge Penalty*

Description

Object of the penalty class to handle the ridge penalty (Hoerl \& Kennard, 1970).

Usage

```
ridge(lambda = NULL, ...)
```

Arguments

`lambda`	regularization parameter. This must be a nonnegative real number.
`...`	further arguments.

Details

The 'classic' penalty as introduced in Hoerl \& Kennard (1970). The ridge penalty is defined as

$$P_\lambda^r(\boldsymbol{\beta}) = \lambda \sum_{i=1}^{p} \beta_j^2.$$

Value

An object of the class `penalty`. This is a list with elements

`penalty`	character: the penalty name.
`lambda`	double: the (nonnegative) regularization parameter.
`getpenmat`	function: computes the diagonal penalty matrix.

Author(s)

Jan Ulbricht

References

Hoerl, A. E. \& R. W. Kennard (1970) Ridge Regression: Bias estimation for nonorthogonal problems. *Technometrics* **12**, 55–67.

See Also

`penalty`, `lasso`, `penalreg`, `ForwardBoost`

`scad` *The SCAD Penalty*

Description

Object of the `penalty` class to handle the SCAD penalty (Fan \& Li, 2001)

Usage

```
scad(lambda = NULL, ...)
```

Arguments

`lambda`	two-dimensional tuning parameter. The first component corresponds to the regularization parameter λ that drives the relevance of the SCAD penalty for likelihood inference. It must be nonnegative. The second component corresponds to a (see details below). It must be greater than two.
...	further arguments.

Details

The *SCAD* penalty is formally defined as

$$P_{\tilde{\lambda}}^{sc}(\boldsymbol{\beta}) = \sum_{j=1}^{p} p_{\tilde{\lambda},j}^{sc}(|\beta_j|), \quad \tilde{\lambda} = (\lambda, s),$$

where $p_{\tilde{\lambda},j}^{sc}(|\beta_j|)$ is complicated to be specified directly. Fan \& Li (2001) just give the penalty by the first derivatives of its components as

$$\frac{dp_{\tilde{\lambda},j}^{sc}(|\beta_j|)}{d|\beta_j|} = \lambda \left\{ 1_{|\beta_j| \leq \lambda}(|\beta_j|) + \frac{(a\lambda - |\beta_j|)_+}{(a-1)\lambda} 1_{|\beta_j| > \lambda}(|\beta_j|) \right\},$$

where we use the notation $b_+ := \max\{0, b\}$ and $1_A(x)$ denotes the indicator function. The penalty depends on two tuning parameters, $\lambda > 0$ and $a > 2$. It is continuously differentiable in β_j, but not in their tuning parameters. If $|\beta_j| \leq \lambda$ then the lasso penalty is applied to β_j. Afterwards this penalization smoothly clipped apart until the threshold a is reached. For $|\beta_j| > a$ there is no penalization at all at this coefficient. Fan \& Li (2001) suggest to use $a = 3.7$. The SCAD penalty leaves large values of β_j not excessively penalized and makes the solution continuous.

Value

An object of the class `penalty`. This is a list with elements

`penalty`	character: the penalty name.
`lambda`	double: the (nonnegative) tuning parameter.
`getpenmat`	function: computes the diagonal penalty matrix.

Author(s)

Jan Ulbricht

References

Fan, J. \& R. Li (2001) Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association* **96**, 1348–1360.

See Also

`penalty`, `lasso`, `ridge`

`weighted.fusion` *Weighted Fusion Penalty*

Description

Object of the `penalty` class to handle the weighted fusion penalty (Daye \& Jeng, 2009)

Usage

```
weighted.fusion(lambda = NULL, ...)
```

Arguments

`lambda`	three-dimensional tuning parameter. The first component corresponds to the regularization parameter λ_1 of the lasso penalty. The second component corresponds to the regularization parameter λ_2 of the fusion penalty. Both components must be nonnegative. The third component corresponds to $\gamma > 0$ that determines the fusion penalty.
`...`	further arguments.

Details

Another extension of correlation-based penalization has been proposed by Daye \& Jeng (2009). They introduce the *weighted fusion penalty* to utilize the correlation information from the data by penalizing the pairwise differences of coefficients via correlation-driven weights. As a consequence, highly correlated regressors are allowed to be treated similarly in regression. The weighted fusion penalty is defined as

$$P_\lambda^{wf}(\boldsymbol{\beta}) = \lambda_1 \sum_{j=1}^{p} |\beta_j| + P_{\lambda_2}^{cd}(\boldsymbol{\beta}),$$

where

$$P_{\lambda_2}^{cd}(\boldsymbol{\beta}) = \frac{\lambda_2}{p} \sum_{i<j} \omega_{ij} \{\beta_i - \text{sign}(\varrho_{ij})\beta_j\}^2$$

is referred to as *correlation-driven penalty* function. Daye \& Jeng (2009) propose to use

$$\omega_{ij} = \frac{|\varrho_{ij}|^\gamma}{1 - |\varrho_{ij}|},$$

where $\gamma > 0$ is an additional tuning parameter. Consequently, the weighted fusion penalty consists of three tuning parameters $\lambda = (\lambda_1, \lambda_2, \gamma)$. The effect is that $\omega_{ij} \to \infty$ as $|\varrho_{ij}| \to 1$ so that the correlation-driven penalty function tends to equate the magnitude of the coefficients of the corresponding regressors x_i and x_j. Note that the lasso penalty term in the weighted fusion penalty is responsible for variable selection.

Value

An object of the class `penalty`. This is a list with elements

`penalty`	character: the penalty name.
`lambda`	double: the (nonnegative) regularization parameter.
`getpenmat`	function: computes the diagonal penalty matrix.

Author(s)

Jan Ulbricht

References

Daye, Z. J. \& X. J. Jeng (2009) Shrinkage and model selection with correlated variabeles via weighted fusion. *Computational Statistics and Data Analysis* **53**, 1284–1298.

See Also

`penalty`, `penalreg`, `icb`, `licb`, `ForwardBoost`

Index