Package ‘quantgen’

June 27, 2020

Type Package

Title Tools for generalized quantile modeling

Version 1.0.0

Date 2020-06-06

Author Ryan Tibshirani

Maintainer Ryan Tibshirani <ryantibs@stat.cmu.edu>

Depends Matrix, gurobi

Description
Tools for quantile modeling: penalized quantile regression, cross-validation, and ensembles.

License GPL-2

RoxygenNote 6.1.1

R topics documented:

`coef.quantile_ensemble` *Coef function for quantile_ensemble object*

Description

Retrieve ensemble coefficients for estimating the conditional quantiles at given tau values.

Usage

```
## S3 method for class 'quantile_ensemble'
coef(obj)
```

Arguments

`obj`	The `quantile_ensemble` object.

`coef.quantile_genlasso` *Coef function for quantile_genlasso object*

Description

Retrieve generalized lasso coefficients for estimating the conditional quantiles at specified tau or lambda values.

Usage

```
## S3 method for class 'quantile_genlasso'
coef(obj, s = NULL)
```

Arguments

`obj`	The `quantile_genlasso` object.
`s`	Vector of integers specifying the tau and lambda values to consider for coefficients; for each `i` in this vector, coefficients are returned at quantile level `tau[i]` and tuning parameter value `lambda[i]`, according to the `tau` and `lambda` vectors stored in the given `quantile_genlasso` object `obj`. (Said differently, `s` specifies the columns of `obj$beta` to retrieve for the coefficients.) Default is NULL, which means that all tau and lambda values will be considered.

`combine_into_array` *Combine matrices into an array*

Description

Combine (say) p matrices, each of dimension n x r, into an n x p x r array.

Usage

```
combine_into_array(mat, ...)
```

Arguments

`mat`	First matrix to combine into an array. Alternatively, a list of matrices to combine into an array.
`...`	Additional matrices to combine into an array. These additional arguments will be ignored if `mat` is a list.

`cv_quantile_genlasso` *Cross-validation for quantile generalized lasso*

Description

Run cross-validation for the quantile generalized lasso on a tau by lambda grid. For each tau, the lambda value minimizing the cross-validation error is reported.

Usage

```
cv_quantile_genlasso(x, y, d, tau, lambda = NULL, nlambda = 30,
  lambda_min_ratio = 0.001, weights = NULL, nfolds = 5,
  train_test_inds = NULL, intercept = TRUE, standardize = TRUE,
  lp_solver = c("gurobi", "glpk"), time_limit = NULL,
  warm_starts = TRUE, params = list(), transform = NULL,
  inv_trans = NULL, jitter = NULL, verbose = FALSE, sort = FALSE,
  iso = FALSE, nonneg = FALSE, round = FALSE)
```

Arguments

`nfolds`	Number of cross-validation folds. Default is 5.
`train_test_inds`	List of length two, with components named `train` and `test`. Each of `train` and `test` are themselves lists, of the same length; for each `i`, we will consider `train[[i]]` the indices (which index the rows of `x` and elements of `y`) to use for training, and `test[[i]]` as the indices to use for testing (validation). The validation error will then be summed up over all `i`. This allows for fine control of the "cross-validation" process (in quotes, because there need not be any crossing going on here). Default is NULL; if specified, takes priority over `nfolds`.

Details

All arguments through verbose (except for nfolds and train_test_inds) are as in quantile_genlasso_grid and quantile_genlasso. Past verbose, the arguments are as in predict.quantile_genlasso, and control what happens with the predictions made on the validation sets.

Value

A list with the following components:

- qgl_obj: a quantile_genlasso object obtained by fitting on the full training set, at all quantile levels and their corresponding optimal lambda values
- cv_mat: a matrix of cross-validation errors (as measured by quantile loss), of dimension (number of tuning parameter values) x (number of quantile levels)
- lambda_min: a vector of optimum lambda values, one per quantile level
- tau, lambda: the sequences of tau and lambda values considered

---

cv_quantile_lasso *Cross-validation for quantile lasso*

Description

Run cross-validation for the quantile lasso on a tau by lambda grid. For each tau, the lambda value minimizing the cross-validation error is reported.

Usage

```
cv_quantile_lasso(x, y, tau, lambda = NULL, nlambda = 30,
  lambda_min_ratio = 0.001, weights = NULL, no_pen_vars = c(),
  nfolds = 5, train_test_inds = NULL, intercept = TRUE,
  standardize = TRUE, lp_solver = c("gurobi", "glpk"),
  time_limit = NULL, warm_starts = TRUE, params = list(),
  transform = NULL, inv_trans = NULL, jitter = NULL,
  verbose = FALSE, sort = FALSE, iso = FALSE, nonneg = FALSE,
  round = FALSE)
```

Arguments

nfolds	Number of cross-validation folds. Default is 5.
train_test_inds	List of length two, with components named train and test. Each of train and test are themselves lists, of the same length; for each i, we will consider train[[i]] the indices (which index the rows of x and elements of y) to use for training, and test[[i]] as the indices to use for testing (validation). The validation error will then be summed up over all i. This allows for fine control of the "cross-validation" process (in quotes, because there need not be any crossing going on here). Default is NULL; if specified, takes priority over nfolds.

Details

All arguments through verbose (except for nfolds and train_test_inds) are as in quantile_lasso_grid and quantile_lasso. Past verbose, the arguments are as in predict.quantile_lasso, and control what happens with the predictions made on the validation sets.

Value

A list with the following components:

- qgl_obj: a `quantile_genlasso` object obtained by fitting on the full training set, at all quantile levels and their corresponding optimal lambda values
- cv_mat: a matrix of cross-validation errors (as measured by quantile loss), of dimension (number of tuning parameter values) x (number of quantile levels)
- lambda_min: a vector of optimum lambda values, one per quantile level

---

`get_diff_mat` *Difference matrix*

Description

Construct a difference operator, of a given order, for use in trend filtering penalties.

Usage

```
get_diff_mat(p, k)
```

Arguments

`p`	Dimension (number of columns) of the difference matrix.
`k`	Order of the difference matrix.

Value

A sparse matrix of dimension (p - k) x p.

`get_lambda_max` *Lambda max for quantile generalized lasso*

Description

Compute lambda max for a quantile generalized lasso problem.

Usage

```
get_lambda_max(x, y, d, weights = NULL, lp_solver = c("gurobi",
  "glpk"))
```

Details

This is not exact, but should be close to the exact value of $lambda$ such that $D\hat{\beta} = 0$ at the solution $\hat{\beta}$ of the quantile generalized lasso problem. It is derived from the KKT conditions when $\tau = 1/2$.

`get_lambda_seq` *Lambda sequence for quantile generalized lasso*

Description

Compute a lambda sequence for a quantile generalized lasso problem.

Usage

```
get_lambda_seq(x, y, d, nlambda, lambda_min_ratio, weights = NULL,
  intercept = TRUE, standardize = TRUE, lp_solver = c("gurobi",
  "glpk"), transform = NULL)
```

Details

This function returns `nlambda` values log-spaced in between `lambda_max`, as computed by `get_lambda_max`, and `lamdba_max * lambda_min_ratio`. If `d` is not specified, we will set it equal to the identity (hence interpret the problem as a quantile lasso problem).

`log_pad` *Convenience functions for log padding*

Description

Functions to map $y \mapsto \log(a + y)$ and $x \mapsto \exp(x) - a$.

Usage

```
log_pad(a = 1)

inv_log_pad(a = 1)
```

`plot.cv_quantile_genlasso` *Plot function for quantile_genlasso object*

Description

Plot the cross-validation error curves, for each quantile level, as functions of the tuning parameter value.

Usage

```
## S3 method for class 'cv_quantile_genlasso'
plot(obj, legend_pos = "topleft")
```

Arguments

`obj`	The `cv_quantile_genlasso` object.
`legend_pos`	Position for the legend; default is "topleft"; use NULL to suppress the legend.

`predict.cv_quantile_genlasso`
Predict function for cv_quantile_genlasso object

Description

Predict the conditional quantiles at a new set of predictor variables, using the generalized lasso coefficients tuned by cross-validation.

Usage

```
## S3 method for class 'cv_quantile_genlasso'
predict(obj, newx, s = NULL,
  sort = FALSE, iso = FALSE, nonneg = FALSE, round = FALSE)
```

Details

This just calls the `predict` function on the `quantile_genlasso` that is stored within the given `cv_quantile_genlasso` object.

`predict.quantile_ensemble`
Predict function for quantile_ensemble object

Description

Predict the conditional quantiles at a new set of ensemble realizations, using the ensemble coefficients at given tau values.

Usage

```
## S3 method for class 'quantile_ensemble'
predict(obj, newq, s = NULL, sort = TRUE,
  iso = FALSE, nonneg = FALSE, round = FALSE)
```

Arguments

`obj`	The `quantile_ensemble` object.
`newq`	Array of new predicted quantiles, of dimension (number of new prediction points) x (number or ensemble components) x (number of quantile levels).
`sort`	Should the returned quantile estimates be sorted? Default is TRUE.
`iso`	Should the returned quantile estimates be passed through isotonic regression? Default is FALSE; if TRUE, takes priority over `sort`.
`nonneg`	Should the returned quantile estimates be truncated at 0? Natural for count data. Default is FALSE.
`round`	Should the returned quantile estimates be rounded? Natural for count data. Default is FALSE.

`predict.quantile_genlasso`
Predict function for quantile_genlasso object

Description

Predict the conditional quantiles at a new set of predictor variables, using the generalized lasso coefficients at specified tau or lambda values.

Usage

```
## S3 method for class 'quantile_genlasso'
predict(obj, newx, s = NULL, sort = FALSE,
  iso = FALSE, nonneg = FALSE, round = FALSE)
```

Arguments

`obj`	The `quantile_genlasso` object.
`newx`	Matrix of new predictor variables at which predictions should be made.
`s`	Vector of integers specifying the tau and lambda values to consider for predictions; for each `i` in this vector, predictions are made at quantile level `tau[i]` and tuning parameter value `lambda[i]`, according to the `tau` and `lambda` vectors stored in the given `quantile_genlasso` object `obj`. (Said differently, `s` specifies the columns of `object$beta` to use for the predictions.) Default is NULL, which means that all tau and lambda values will be considered.
`sort`	Should the returned quantile estimates be sorted? Default is FALSE. Note: this option only makes sense if the values in the stored `tau` vector are distinct, and sorted in increasing order.
`iso`	Should the returned quantile estimates be passed through isotonic regression? Default is FALSE; if TRUE, takes priority over `sort`. Note: this option only makes sense if the values in the stored `tau` vector are distinct, and sorted in increasing order.
`nonneg`	Should the returned quantile estimates be truncated at 0? Natural for count data. Default is FALSE.
`round`	Should the returned quantile estimates be rounded? Natural for count data. Default is FALSE.

`predict.quantile_genlasso_grid`
Predict function for quantile_genlasso_grid object

Description

Predict the conditional quantiles at a new set of predictor variables, using the generalized lasso coefficients at given tau or lambda values.

Usage

```
## S3 method for class 'quantile_genlasso_grid'
predict(obj, newx, sort = FALSE,
  iso = FALSE, nonneg = FALSE, round = FALSE)
```

Details

This function operates as in the `predict.quantile_genlasso` function for a `quantile_genlasso` object, but with a few key differences. First, the output is reformatted so that it is an array of dimension (number of prediction points) x (number of tuning parameter values) x (number of quantile levels). This output is generated from the full set of tau and lambda pairs stored in the given `quantile_genlasso_grid` object `obj` (selecting a subset is disallowed). Second, the arguments `sort` and `iso` operate on the appropriate slices of this array: for a fixed lambda value, we sort or run isotonic regression across all tau values.

`quantile_ensemble` *Quantile ensemble*

Description

Fit ensemble weights, given a set of quantile predictions.

Usage

```
quantile_ensemble(qarr, y, tau, weights = NULL, tau_groups = rep(1,
  length(tau)), intercept = FALSE, nonneg = TRUE, unit_sum = TRUE,
  noncross = TRUE, q0 = NULL, lp_solver = "gurobi",
  time_limit = NULL, params = list(), verbose = FALSE)
```

Arguments

`qarr`	Array of predicted quantiles, of dimension (number of prediction points) x (number or ensemble components) x (number of quantile levels).
`y`	Vector of responses (whose quantiles are being predicted by `qarr`).
`tau`	Vector of quantile levels at which predictions are made. Assumed to be distinct, and sorted in increasing order.
`weights`	Vector of observation weights (to be used in the loss function). Default is NULL, which is interpreted as a weight of 1 for each observation.
`tau_groups`	Vector of group labels, having the same length as `tau`. Common labels indicate that the ensemble weights for the corresponding quantile levels should be tied together. Default is `rep(1,length(tau))`, which means that a common set of ensemble weights should be used across all levels. See details.
`intercept`	Should an intercept be included in the ensemble model? Default is FALSE.
`nonneg`	Should the ensemble weights be constrained to be nonnegative? Default is TRUE.
`unit_sum`	Should the ensemble weights be constrained to sum to 1? Default is TRUE.
`noncross`	Should noncrossing constraints be enforced? Default is TRUE. Note: this option only matters when there is more than group of ensemble weights, as determined by `tau_groups`. See details.

q0	Array of points used to define the noncrossing constraints. Must have dimension (number of points) x (number of ensemble components) x (number of quantile levels). Default is NULL, which means that we consider noncrossing constraints at the training points qarr.
lp_solver	One of "gurobi" or "glpk", indicating which LP solver to use. Default is "gurobi".
time_limit	This sets the maximum amount of time (in seconds) to allow Gurobi or GLPK to solve any single quantile generalized lasso problem (for a single tau and lambda value). Default is NULL, which means unlimited time.
params	A list of control parameters to pass to Gurobi or GLPK. Default is `list()` which means no additional parameters are passed. For example: with Gurobi, we can use `list(Threads=4)` to specify that Gurobi should use 4 threads when available. (Note that if a time limit is specified through this `params` list, then its value will be overriden by the last argument `time_limit`, assuming the latter is not NULL.)
verbose	Should progress be printed out to the console? Default is FALSE.

Details

This function solves the following quantile ensemble optimization problem, over quantile levels $\tau_k, k = 1, \dots, r$:

$$\underset{\alpha_j, j=1,\dots,p}{\text{minimize}} \sum_{k=1}^{r} \sum_{i=1}^{n} w_i \psi_{\tau_k} \left(y_i - \sum_{j=1}^{p} \alpha_j q_{ijk} \right)$$

$$\text{subject to } \sum_{j=1}^{p} \alpha_j = 1,\ \alpha_j \geq 0,\ j = 1, \dots, p$$

for a response vector y and quantile array q, where q_{ijk} is an estimate of the quantile of y_i at the level τ_k, from ensemble component member j. Here $\psi_\tau(v) = \max\{\tau v, (\tau - 1)v\}$ is the "pinball" or "tilted ℓ_1" loss. A more advanced version allows us to estimate a separate ensemble weight α_{jk} per component method j, per quantile level k:

$$\text{minimize}\, \alpha_{jk}, j = 1, \dots, p, k = 1, \dots, r \ \sum_{k=1}^{r} \sum_{i=1}^{n} w_i \psi_{\tau_k} \left(y_i - \sum_{j=1}^{p} \alpha_{jk} q_{ijk} \right)$$

$$\text{subject to } \sum_{j=1}^{p} \alpha_{jk} = 1,\ k = 1, \dots, r,\ \alpha_{jk} \geq 0,\ j = 1, \dots, p,\ k = 1, \dots, r$$

As a form of regularization, we can additionally incorporate noncrossing constraints into the above optimization, which take the form:

$$\alpha_{\bullet,k}^T q \leq \alpha_{\bullet,k+1}^T q,\ k = 1, \dots, r - 1,\ q \in \mathcal{Q}$$

where the quantile levels $\tau_k, k = 1, \dots, r$ are assumed to be in increasing order, and $\mathcal{Q}$ is a collection of points over which to enforce the noncrossing constraints. Finally, somewhere in between these two extremes is to allow one ensemble weight per component member j, per quantile group g. This can be interpreted as a set of further constraints which enforce equality between α_{jk} and $\alpha_{j\ell}$, for all k, ℓ that are in the same group g.

Value

A list with the following components:

- alpha: vector or matrix of ensemble weights. If `tau_groups` has only one unique label, then this is a vector of length = (number of ensemble components); otherwise, it is a matrix, of dimension (number of ensemble components) x (number of quantile levels)
- tau: vector of quantile levels used
- weights, tau_groups, ..., params: values of these other arguments used in the function call

---

`quantile_extrapolate` *Quantile extrapolater*

Description

Extrapolate a set of quantiles at new quantile levels: parametric in the tails, nonparametric in the middle.

Usage

```
quantile_extrapolate(tau, qvals, tau_out = c(0.01, 0.025, seq(0.05, 0.95,
  by = 0.05), 0.975, 0.99), sort = TRUE, iso = FALSE, nonneg = FALSE,
  round = FALSE, qfun_left = qnorm, qfun_right = qnorm,
  n_tau_left = 1, n_tau_right = 1, middle = c("cubic", "linear"),
  param0 = NULL, param1 = NULL, grid_size = 1000, tol = 0.01,
  max_iter = 10)
```

Arguments

`tau`	Vector of quantile levels. Assumed to be distinct, and sorted in increasing order.
`qvals`	Vector or matrix quantiles; if a matrix, each row is a separate set of quantiles, at the same (common) quantile levels, given by `tau`.
`tau_out`	Vector of quantile levels at which to perform extrapolation. Default is a sequence of 23 quantile levels from 0.01 to 0.99.
`sort`	Should the returned quantile estimates be sorted? Default is TRUE.
`iso`	Should the returned quantile estimates be passed through isotonic regression? Default is FALSE; if TRUE, takes priority over `sort`.
`nonneg`	Should the returned quantile estimates be truncated at 0? Natural for count data. Default is FALSE.
`round`	Should the returned quantile estimates be rounded? Natural for count data. Default is FALSE.
`qfun_left, qfun_right`	Quantile functions on which to base extrapolation in the left and right tails, respectively; each must be a function whose first two arguments are a quantile level and a distribution parameter (such as a mean parameter); these are assumed to be vectorized in the first argument when the second argument is fixed, and also vectorized in the second argument when the first argument is fixed. Default is `qnorm`. See details for further explanation.

n_tau_left, n_tau_right
: Integers between 1 and the length of tau, indicating how many elements quantile levels from the left and right ends, respectively, to use in defining the tails. For example, if n_tau_left=1, the default, then only the leftmost quantile is used for the left tail extrapolation; if n_tau_left=2, then the two leftmost quantiles are used, etc; and similarly for n_tau_right. See details for further explanation.

middle
: One of "cubic" or "linear", indicating the interpolation method to use in the middle (outside of the tails, as determined by n_tau_left and n_tau_right). If "cubic", the default, then a monotone cubic spline interpolant is fit to the given quantiles, and used to estimate quantiles in the middle. If "linear", then linear interpolation is used to estimate quantiles in the middle.

param0, param1
: TODO

grid_size, tol, maxiter
: TODO

Details

TODO

Value

A matrix of dimension (number of rows in qvals) x (length of tau_out), where each row is the extrapolation of the set of quantiles in the corresponding row of qvals, at the quantile levels specified in tau_out.

---

quantile_genlasso *Quantile generalized lasso*

Description

Compute quantile generalized lasso solutions.

Usage

```
quantile_genlasso(x, y, d, tau, lambda, weights = NULL,
  intercept = TRUE, standardize = TRUE, noncross = FALSE,
  x0 = NULL, lp_solver = c("gurobi", "glpk"), time_limit = NULL,
  warm_starts = TRUE, params = list(), transform = NULL,
  inv_trans = NULL, jitter = NULL, verbose = FALSE)
```

Arguments

x
: Matrix of predictors. If sparse, then passing it an appropriate sparse Matrix class can greatly help optimization.

y
: Vector of responses.

d
: Matrix defining the generalized lasso penalty; see details. If sparse, then passing it an appropriate sparse Matrix class can greatly help optimization. A convenience function get_diff_mat for constructing trend filtering penalties is provided.

tau, lambda	Vectors of quantile levels and tuning parameter values. If these are not of the same length, the shorter of the two is recycled so that they become the same length. Then, for each i, we solve a separate quantile generalized lasso problem at quantile level tau[i] and tuning parameter value lambda[i]. The most common use cases are: specifying one tau value and a sequence of lambda values; or specifying a sequence of tau values and one lambda value.
weights	Vector of observation weights (to be used in the loss function). Default is NULL, which is interpreted as a weight of 1 for each observation.
intercept	Should an intercept be included in the regression model? Default is TRUE.
standardize	Should the predictors be standardized (to have zero mean and unit variance) before fitting? Default is TRUE.
noncross	Should noncrossing constraints be applied? These force the estimated quantiles to be properly ordered across all quantile levels being considered. The default is FALSE. If TRUE, then noncrossing constraints are applied to the estimated quantiles at all points specified by the next argument x0. Note: this option only makes sense if the values in the tau vector are distinct, and sorted in increasing order.
x0	Matrix of points used to define the noncrossing constraints. Default is NULL, which means that we consider noncrossing constraints at the training points x.
lp_solver	One of "gurobi" or "glpk", indicating which LP solver to use. Default is "gurobi".
time_limit	This sets the maximum amount of time (in seconds) to allow Gurobi or GLPK to solve any single quantile generalized lasso problem (for a single tau and lambda value). Default is NULL, which means unlimited time.
warm_starts	Should warm starts be used in the LP solver (from one LP solve to the next)? Only supported for Gurobi.
params	A list of control parameters to pass to Gurobi or GLPK. Default is list() which means no additional parameters are passed. For example: with Gurobi, we can use list(Threads=4) to specify that Gurobi should use 4 threads when available. (Note that if a time limit is specified through this params list, then its value will be overriden by the last argument time_limit, assuming the latter is not NULL.)
transform, inv_trans	The first is a function to transform y before solving the quantile generalized lasso; the second is the corresponding inverse transform. For example: for count data, we might want to model log(1+y) (which would be the transform, and the inverse transform would be exp(x)-1). Both transform and inv_trans should be vectorized. Convenience functions log_pad and inv_log_pad are provided.
jitter	Function for applying random jitter to y, which might help optimization. For example: for count data, there can be lots of ties (with or without transformation of y), which can make optimization more difficult. The function jitter should take an integer n and return n random draws. A convenience function unif_jitter is provided.
verbose	Should progress be printed out to the console? Default is FALSE.

Details

This function solves the quantile generalized lasso problem, for each pair of quantile level τ and tuning parameter λ:

$$\underset{\beta_0,\beta}{\mathrm{minimize}} \; \sum_{i=1}^n w_i \psi_\tau(y_i - \beta_0 - x_i^T\beta) + \lambda \|D\beta\|_1$$

for a response vector y with components y_i, predictor matrix X with rows x_i, and penalty matrix D. Here $\psi_\tau(v) = \max\{\tau v, (\tau - 1)v\}$ is the "pinball" or "tilted ℓ_1" loss. When noncrossing constraints are applied, we instead solve one big joint optimization, over all quantile levels and tuning parameter values:

$$\underset{\beta_{0k}, \beta_k, k=1,\dots,r}{\text{minimize}} \; \sum_{k=1}^{r} \left(\sum_{i=1}^{n} w_i \psi_{\tau_k}(y_i - \beta_{0k} - x_i^T \beta_k) + \lambda_k \|D\beta_k\|_1 \right)$$

$$\text{subject to} \;\; \beta_{0k} + x^T \beta_k \le \beta_{0,k+1} + x^T \beta_{k+1} \;\; k = 1, \dots, r-1, \; x \in \mathcal{X}$$

where the quantile levels $\tau_k, k = 1, \dots, r$ are assumed to be in increasing order, and $\mathcal{X}$ is a collection of points over which to enforce the noncrossing constraints.

Either problem is readily converted into a linear program (LP), and solved using either Gurobi (which is free for academic use, and generally fast) or GLPK (which free for everyone, but slower).

Value

A list with the following components:

- beta: a matrix of generalized lasso coefficients, of dimension = (number of features + 1) x (number of quantile levels) assuming `intercept=TRUE`, else (number of features) x (number of quantile levels). Note: these coefficients will always be on the appropriate scale; they are always on the scale of original features, even if `standardize=TRUE`
- status: vector of status flags returned by Gurobi's or GLPK's LP solver, of length = (number of quantile levels)
- tau, lambda: vectors of tau and lambda values used
- weights, intercept, ..., jitter: values of these other arguments used in the function call

Author(s)

Ryan Tibshirani

`quantile_genlasso_grid`

Quantile generalized lasso on a tau by lambda grid

Description

Convenience function for computing quantile generalized lasso solutions on a tau by lambda grid.

Usage

```
quantile_genlasso_grid(x, y, d, tau, lambda = NULL, nlambda = 30,
  lambda_min_ratio = 0.001, weights = NULL, intercept = TRUE,
  standardize = TRUE, lp_solver = c("gurobi", "glpk"),
  time_limit = NULL, warm_starts = TRUE, params = list(),
  transform = NULL, inv_trans = NULL, jitter = NULL,
  verbose = FALSE)
```

Arguments

nlambda	Number of lambda values to consider, for each quantile level. Default is 30.
lambda_min_ratio	Ratio of the minimum to maximum lambda value, for each quantile levels. Default is 1e-3.

Details

This function forms a `lambda` vector either determined by the `nlambda` and `lambda_min_ratio` arguments, or the `lambda` argument; if the latter is specified, then it takes priority. Then, for each `i` and `j`, we solve a separate quantile generalized lasso problem at quantile level `tau[i]` and tuning parameter value `lambda[j]`, using the `quantile_genlasso` function. All arguments (aside from `nlambda` and `lambda_min_ratio`) are as in the latter function; noncrossing constraints are disallowed.

---

quantile_genlasso_objective

Quantile generalized lasso objective

Description

Compute generalized lasso objective for a single tau and lambda value.

Usage

```
quantile_genlasso_objective(x, y, d, beta, tau, lambda)
```

---

quantile_lasso

Quantile lasso.

Description

Compute quantile lasso solutions.

Usage

```
quantile_lasso(x, y, tau, lambda, weights = NULL, no_pen_vars = c(),
  intercept = TRUE, standardize = TRUE, noncross = FALSE,
  x0 = NULL, lp_solver = c("gurobi", "glpk"), time_limit = NULL,
  warm_starts = TRUE, params = list(), transform = NULL,
  inv_trans = NULL, jitter = NULL, verbose = FALSE)
```

Arguments

`x`	Matrix of predictors. If sparse, then passing it an appropriate sparse `Matrix` class can greatly help optimization.
`y`	Vector of responses.
`tau, lambda`	Vectors of quantile levels and tuning parameter values. If these are not of the same length, the shorter of the two is recycled so that they become the same length. Then, for each `i`, we solve a separate quantile lasso problem at quantile level `tau[i]` and tuning parameter value `lambda[i]`. The most common use cases are: specifying one tau value and a sequence of lambda values; or specifying a sequence of tau values and one lambda value.
`weights`	Vector of observation weights (to be used in the loss function). Default is NULL, which is interpreted as a weight of 1 for each observation.
`no_pen_vars`	Indices of the variables that should be excluded from the lasso penalty. Default is `c()`, which means that no variables are to be excluded.

Details

This function solves the quantile lasso problem, for each pair of quantile level τ and tuning parameter λ:

$$\underset{\beta_0,\beta}{\text{minimize}} \; \sum_{i=1}^n w_i \psi_\tau(y_i - \beta_0 - x_i^T \beta) + \lambda \|\beta\|_1$$

for a response vector y with components y_i, and predictor matrix X with rows x_i. Here $\psi_\tau(v) = \max\{\tau v, (\tau-1)v\}$ is the "pinball" or "tilted ℓ_1" loss. When noncrossing constraints are applied, we instead solve one big joint optimization, over all quantile levels and tuning parameter values:

$$\underset{\beta_{0k},\beta_k,k=1,\ldots,r}{\text{minimize}} \; \sum_{k=1}^r \left(\sum_{i=1}^n w_i \psi_{\tau_k}(y_i - \beta_{0k} - x_i^T \beta_k) + \lambda_k \|\beta_k\|_1 \right)$$

$$\text{subject to} \;\; \beta_{0k} + x^T \beta_k \le \beta_{0,k+1} + x^T \beta_{k+1} \;\; k = 1, \ldots, r-1, \; x \in \mathcal{X}$$

where the quantile levels $\tau_j, j = 1, \ldots, k$ are assumed to be in increasing order, and $\mathcal{X}$ is a collection of points over which to enforce the noncrossing constraints.

Either problem is readily converted into a linear program (LP), and solved using either Gurobi (which is free for academic use, and generally fast) or GLPK (which free for everyone, but slower).

All arguments not described above are as in the `quantile_genlasso` function. The associated `coef` and `predict` functions are just those for the `quantile_genlasso` class.

Value

A list with the following components:

- beta: a matrix of lasso coefficients, of dimension = (number of features + 1) x (number of quantile levels) assuming `intercept=TRUE`, else (number of features) x (number of quantile levels). Note: these coefficients will always be on the appropriate scale; they are always on the scale of original features, even if `standardize=TRUE`
- status: vector of status flags returned by Gurobi's or GLPK's LP solver, of length = (number of quantile levels)
- tau, lambda: vectors of tau and lambda values used
- weights, no_pen_vars, ..., jitter: values of these other arguments used in the function call

Author(s)

Ryan Tibshirani

`quantile_lasso_grid` *Quantile lasso on a tau by lambda grid*

Description

Convenience function for computing quantile lasso solutions on a tau by lambda grid.

Usage

```
quantile_lasso_grid(x, y, tau, lambda = NULL, nlambda = 30,
  lambda_min_ratio = 0.001, weights = NULL, no_pen_vars = c(),
  intercept = TRUE, standardize = TRUE, lp_solver = c("gurobi",
  "glpk"), time_limit = NULL, warm_starts = TRUE, params = list(),
  transform = NULL, inv_trans = NULL, jitter = NULL,
  verbose = FALSE)
```

Arguments

`nlambda` Number of lambda values to consider, for each quantile level. Default is 30.

`lambda_min_ratio` Ratio of the minimum to maximum lambda value, for each quantile levels. Default is 1e-3.

Details

This function forms a `lambda` vector either determined by the `nlambda` and `lambda_min_ratio` arguments, or the `lambda` argument; if the latter is specified, then it takes priority. Then, for each `i` and `j`, we solve a separate quantile lasso problem at quantile level `tau[i]` and tuning parameter value `lambda[j]`, using the `quantile_lasso` function. All arguments (aside from `nlambda` and `lambda_min_ratio`) are as in the latter function; noncrossing constraints are disallowed. The associated `predict` function is just that for the `quantile_genlasso_grid` class.

`quantile_lasso_objective` *Quantile lasso objective*

Description

Compute lasso objective for a single tau and lambda value.

Usage

```
quantile_lasso_objective(x, y, beta, tau, lambda)
```

`quantile_loss` *Quantile loss*

Description

Compute the quantile (tilted absolute) loss for a single tau value.

Usage

```
quantile_loss(yhat, y, tau)
```

`refit_quantile_genlasso` *Refit function for cv_quantile_genlasso object*

Description

Refit generalized lasso solutions at a new set of quantile levels, given an existing `cv_quantile_genlasso` object.

Usage

```
refit_quantile_genlasso(obj, x, y, d, tau_new = c(0.01, 0.025, seq(0.05,
  0.95, by = 0.05), 0.975, 0.99), weights = NULL, intercept = TRUE,
  standardize = TRUE, noncross = FALSE, x0 = NULL,
  lp_solver = NULL, time_limit = NULL, warm_starts = NULL,
  params = NULL, transform = NULL, inv_trans = NULL, jitter = NULL,
  verbose = FALSE)
```

Arguments

`obj`	The `cv_quantile_genlasso` object to start from.
`x`	Matrix of predictors.
`y`	Vector of responses.
`d`	Matrix defining the generalized lasso penalty.
`tau_new`	Vector of new quantile levels at which to fit new solutions. Default is a sequence of 23 quantile levels from 0.01 to 0.99.
`noncross`	Should noncrossing constraints be applied? These force the estimated quantiles to be properly ordered across all quantile levels being considered. The default is FALSE. If TRUE, then noncrossing constraints are applied to the estimated quantiles at all points specified by the next argument `x0`.
`x0`	Matrix of points used to define the noncrossing constraints. Default is NULL, which means that we consider noncrossing constraints at the training points `x`.
`verbose`	Should progress be printed out to the console? Default is FALSE.

Details

This function simply infers, for each quantile level in tau_new, a (very) roughly-CV-optimal tuning parameter value, then calls quantile_genlasso at the new quantile levels and corresponding tuning parameter values. If not specified, the arguments weights, intercept, standardize, lp_solver, time_limit, warm_starts, params, transform, inv_transorm, jitter are all inherited from the given cv_quantile_genlasso object.

Value

A quantile_genlasso object, with solutions at quantile levels tau_new.

---

refit_quantile_lasso *Refit function for cv_quantile_lasso object*

Description

Refit lasso solutions at a new set of quantile levels, given an existing cv_quantile_lasso object.

Usage

```
refit_quantile_lasso(obj, x, y, tau_new = c(0.01, 0.025, seq(0.05, 0.95,
  by = 0.05), 0.975, 0.99), weights = NULL, no_pen_vars = NULL,
  intercept = TRUE, standardize = TRUE, noncross = FALSE,
  x0 = NULL, lp_solver = NULL, time_limit = NULL,
  warm_starts = NULL, params = NULL, transform = NULL,
  inv_trans = NULL, jitter = NULL, verbose = FALSE)
```

Arguments

obj	The cv_quantile_lasso object to start from.
x	Matrix of predictors.
y	Vector of responses.
tau_new	Vector of new quantile levels at which to fit new solutions. Default is a sequence of 23 quantile levels from 0.01 to 0.99.
noncross	Should noncrossing constraints be applied? These force the estimated quantiles to be properly ordered across all quantile levels being considered. The default is FALSE. If TRUE, then noncrossing constraints are applied to the estimated quantiles at all points specified by the next argument x0.
x0	Matrix of points used to define the noncrossing constraints. Default is NULL, which means that we consider noncrossing constraints at the training points x.
verbose	Should progress be printed out to the console? Default is FALSE.

Details

This function simply infers, for each quantile level in tau_new, a (very) roughly-CV-optimal tuning parameter value, then calls quantile_lasso at the new quantile levels and corresponding tuning parameter values. If not specified, the arguments weights, no_pen_vars, intercept, standardize, lp_solver, time_limit, warm_start, params, transform, inv_transorm, jitter are all inherited from the given cv_quantile_lasso object.

Value

A `quantile_lasso` object, with solutions at quantile levels `tau_new`.

`unif_jitter` *Convenience function for uniform jitter*

Description

Function to generate random draws from $\mathrm{Unif}[a, b]$.

Usage

```
unif_jitter(a = 0, b = 0.01)
```

Index