

Package ‘quantgen’

December 2, 2020

Type Package

Title Tools for generalized quantile modeling

Version 1.0.0

Date 2020-06-06

URL <https://ryantibs.github.io/quantgen/>, <https://github.com/ryantibs/quantgen/>

BugReports <https://github.com/ryantibs/quantgen/>

Description Tools for generalized quantile modeling: regularized quantile regression (with generalized lasso penalties and noncrossing constraints), cross-validation, quantile extrapolation, and quantile ensembles.

Encoding UTF-8

License GPL-2

RoxygenNote 7.1.1

Imports Matrix,
Rglpk

Suggests gurobi

R topics documented:

coef.quantile_ensemble	2
coef.quantile_genlasso	2
combine_into_array	3
cv_quantile_genlasso	3
cv_quantile_lasso	5
get_diff_mat	6
get_lambda_max	6
get_lambda_seq	7
logit_pad	7
log_pad	8
plot.cv_quantile_genlasso	8
predict.cv_quantile_genlasso	8
predict.quantile_ensemble	9
predict.quantile_genlasso	10
predict.quantile_genlasso_grid	11
quantgen	11
quantile_ensemble	12

quantile_extrapolate	14
quantile_genlasso	16
quantile_genlasso_grid	19
quantile_genlasso_objective	20
quantile_lasso	20
quantile_lasso_grid	22
quantile_lasso_objective	23
quantile_loss	23
refit_quantile_genlasso	23
refit_quantile_lasso	24
unif_jitter	26

Index	27
--------------	-----------

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(object, ...)
```

Arguments

object	The quantile_ensemble object.
...	Additional arguments (not used).

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(object, s = NULL, ...)
```

Arguments

object	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.
...	Additional arguments (not used).

combine_into_array	<i>Combine matrices into an array</i>
--------------------	---------------------------------------

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	<i>Cross-validation for quantile generalized lasso</i>
----------------------	--

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("glpk", "gurobi"),
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

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

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:

<code>qgl_obj</code>	A <code>quantile_genlasso</code> object obtained by fitting on the full training set, at all quantile levels and their corresponding optimal lambda values
<code>cv_mat</code>	Matrix of cross-validation errors (as measured by quantile loss), of dimension (number of tuning parameter values) x (number of quantile levels)
<code>lambda_min</code>	Vector of optimum lambda values, one per quantile level
<code>tau, lambda</code>	Vectors of tau and lambda values used

cv_quantile_lasso	<i>Cross-validation for quantile lasso</i>
-------------------	--

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,  
  nlambdas = 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("glpk", "gurobi"),  
  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:

<code>qgl_obj</code>	A <code>quantile_genlasso</code> object obtained by fitting on the full training set, at all quantile levels and their corresponding optimal lambda values
<code>cv_mat</code>	Matrix of cross-validation errors (as measured by quantile loss), of dimension (number of tuning parameter values) x (number of quantile levels)
<code>lambda_min</code>	Vector of optimum lambda values, one per quantile level

<code>get_diff_mat</code>	<i>Difference matrix</i>
---------------------------	--------------------------

Description

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

Usage

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

Arguments

<code>p</code>	Dimension (number of columns) of the difference matrix.
<code>k</code>	Order of the difference matrix.

Value

A sparse matrix of dimension $(p - k) \times p$.

<code>get_lambda_max</code>	<i>Lambda max for quantile generalized lasso</i>
-----------------------------	--

Description

Compute lambda max for a quantile generalized lasso problem.

Usage

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

Details

This is not exact, but should be close to the exact value of λ 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("glpk", "gurobi"),
  transform = NULL
)
```

Details

This function returns nlambda values log-spaced in between lambda_max, as computed by get_lambda_max, and lambda_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).

logit_pad

*Convenience functions for logit/sigmoid mappings***Description**

Returns functions that map $x \mapsto \log\left(\frac{ax+b}{1-ax+b}\right)$ and $x \mapsto \frac{\exp(x)(1+b)-b}{a(1+\exp(x))}$. (These are inverses.)

Usage

```
logit_pad(a = 1, b = 1)
```

```
sigmd_pad(a = 1, b = 1)
```

log_pad	<i>Convenience functions for log/exp mappings</i>
---------	---

Description

Returns functions that map $x \mapsto \log(ax + b)$ and $x \mapsto (\exp(x) - b)/a$. (These are inverses.)

Usage

```
log_pad(a = 1, b = 1)
```

```
exp_pad(a = 1, b = 1)
```

plot.cv_quantile_genlasso	<i>Plot function for quantile_genlasso object</i>
---------------------------	---

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(x, legend_pos = "topleft", ...)
```

Arguments

x	The cv_quantile_genlasso object.
legend_pos	Position for the legend; default is "topleft"; use NULL to suppress the legend.
...	Additional arguments (not used).

predict.cv_quantile_genlasso	<i>Predict function for cv_quantile_genlasso object</i>
------------------------------	---

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(
  object,
  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(
  object,
  newq,
  s = NULL,
  sort = TRUE,
  iso = FALSE,
  nonneg = FALSE,
  round = FALSE,
  ...
)
```

Arguments

object	The quantile_ensemble object.
newq	Array of new predicted quantiles, of dimension (number of new prediction points) x (number of 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.
...	Additional arguments (not used).

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(
  object,
  newx,
  s = NULL,
  sort = FALSE,
  iso = FALSE,
  nonneg = FALSE,
  round = FALSE,
  ...
)
```

Arguments

object	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.
...	Additional arguments (not used).

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(
  object,
  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.

quantgen

quantgen: Tools for generalized quantile modeling

Description

This package provides tools for generalized quantile modeling: regularized quantile regression (with generalized lasso penalties and noncrossing constraints), cross-validation, quantile extrapolation, and quantile ensembles.

Details

We recommend the "getting started" and other vignettes, provided online: <https://ryantibs.github.io/quantgen/>.

quantile_ensemble	<i>Quantile ensemble</i>
-------------------	--------------------------

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 = c("glpk", "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.

<code>q0</code>	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 <code>qarr</code> .
<code>lp_solver</code>	One of "glpk" or "gurobi", indicating which LP solver to use. If possible, "gurobi" should be used because it is much faster and more stable; default is "glpk"; however, because it is open-source.
<code>time_limit</code>	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.
<code>params</code>	List of control parameters to pass to Gurobi or GLPK. Default is <code>list()</code> which means no additional parameters are passed. For example: with Gurobi, we can use <code>list(Threads=4)</code> 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 overridden by the last argument <code>time_limit</code> , assuming the latter is not NULL.)
<code>verbose</code>	Should progress be printed out to the console? Default is FALSE.

Details

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

$$\begin{aligned} & \underset{\alpha_{j,j=1,\dots,p}}{\text{minimize}} \sum_{k=1}^r \sum_{i=1}^n w_i \psi_{\tau_k} \left(y_i - \sum_{j=1}^p \alpha_j q_{ijk} \right) \\ & \text{subject to} \quad \sum_{j=1}^p \alpha_j = 1, \quad \alpha_j \geq 0, \quad j = 1, \dots, p \end{aligned}$$

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 :

$$\begin{aligned} & \underset{\alpha_{jk}, j=1,\dots,p, k=1,\dots,r}{\text{minimize}} \sum_{k=1}^r \sum_{i=1}^n w_i \psi_{\tau_k} \left(y_i - \sum_{j=1}^p \alpha_{jk} q_{ijk} \right) \\ & \text{subject to} \quad \sum_{j=1}^p \alpha_{jk} = 1, \quad k = 1, \dots, r, \quad \alpha_{jk} \geq 0, \quad j = 1, \dots, p, \quad k = 1, \dots, r \end{aligned}$$

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

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

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	<i>Quantile extrapolater</i>
----------------------	------------------------------

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, grid_size, tol, max_iter	Arguments for the algorithm used for parameter-fitting for tail extrapolation. See details.

Details

This function interpolates/extrapolates an initial sparser set of quantiles, say q_1, \dots, q_m at the levels $\tau_1 < \dots < \tau_m$ into a denser set of quantiles, say q_1^*, \dots, q_n^* at the levels $\tau_1^* < \dots < \tau_n^*$. At a high-level, the strategy is to nonparametrically interpolate the quantiles whose levels fall in the interval $[\tau_1, \tau_m]$, and parametrically extrapolate the quantiles whose levels fall in $[0, \tau_1)$ or $(\tau_m, 1]$. Let us call these the "middle" and "tail" strategies, respectively.

To give more details on the middle strategy: a monotone spline interpolant—either a cubic spline (if middle="cubic") or linear spline interpolant (if middle="linear")—is fit to the points

$$(\tau_i, q_i), \quad i = 1, \dots, m.$$

Denoting f by this interpolant, we then set

$$q_i^* = f(\tau_i^*), \quad \tau_i^* \in [\tau_1, \tau_m].$$

To give more details on the tail strategy: in each tail, left and right, the user specifies a tail function $q(\tau; \theta)$ which depends on a parameter θ . This is done via the functions qfun_left and qfun_right; the default is qnorm for both, in which case θ represents the mean of the normal distribution (and the standard deviation is fixed at 1, as per the default in qnorm). Given this tail function, we then find the parameter value $\hat{\theta}$ that best matches the given quantile, and use this for extrapolation. That is, for the left tail, we first fit $\hat{\theta}$ such that

$$q(\tau_1; \hat{\theta}) \approx q_1$$

and we then set

$$q_i^* = q(\tau_i^*; \hat{\theta}), \quad \tau_i^* < \tau_1.$$

The right tail is similar.

The fitting algorithm used for determining $\hat{\theta}$ in each tail is a kind of iterative grid search that proceeds in "rounds". The arguments `param0`, `param1` give the left and right endpoints of the initial interval used in the first round of the search—this interval typically contracts as the rounds proceed, but can also expand as needed; the argument `grid_size` is the number of grid points to consider in each round; the argument `tol` is the error tolerance for stopping; and the argument `max_iter` is the maximum number of rounds to consider. This fitting algorithm is robust to the case when the optimal parameter value that matches the given quantile, as per the above display, is not unique; in this case we take the mean of the range of optimal parameter values.

Finally, when the arguments `n_tau_left` and `n_tau_right` are changed from their defaults, then this changes the definition of the "middle" and the "tail" ranges, but otherwise the analogous strategies are employed. In fact, the middle strategy is unchanged, just applied to a different range. The tail strategy is similar, but now in each tail, left and right, we fit a separate parameter value $\hat{\theta}$ for each given quantile level in the tail range (for example, for each of the two leftmost quantile levels if `n_tau_left=2`), and then take the mean of these parameters as a single parameter value on which to base tail extrapolation.

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	<i>Quantile generalized lasso</i>
-------------------	-----------------------------------

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("glpk", "gurobi"),
  time_limit = NULL,
  warm_starts = TRUE,
  params = list(),
  transform = NULL,
```



```

    inv_trans = NULL,
    jitter = NULL,
    verbose = FALSE
)

```

Arguments

<code>x</code>	Matrix of predictors. If sparse, then passing it an appropriate sparse Matrix class can greatly help optimization.
<code>y</code>	Vector of responses.
<code>d</code>	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 <code>get_diff_mat</code> for constructing trend filtering penalties is provided.
<code>tau, lambda</code>	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 <code>i</code> , we solve a separate quantile generalized lasso problem at quantile level <code>tau[i]</code> and tuning parameter value <code>lambda[i]</code> . The most common use cases are: specifying one <code>tau</code> value and a sequence of <code>lambda</code> values; or specifying a sequence of <code>tau</code> values and one <code>lambda</code> value.
<code>weights</code>	Vector of observation weights (to be used in the loss function). Default is <code>NULL</code> , which is interpreted as a weight of 1 for each observation.
<code>intercept</code>	Should an intercept be included in the regression model? Default is <code>TRUE</code> .
<code>standardize</code>	Should the predictors be standardized (to have zero mean and unit variance) before fitting? Default is <code>TRUE</code> .
<code>noncross</code>	Should noncrossing constraints be applied? These force the estimated quantiles to be properly ordered across all quantile levels being considered. The default is <code>FALSE</code> . If <code>TRUE</code> , then noncrossing constraints are applied to the estimated quantiles at all points specified by the next argument <code>x0</code> . Note: this option only makes sense if the values in the <code>tau</code> vector are distinct, and sorted in increasing order.
<code>x0</code>	Matrix of points used to define the noncrossing constraints. Default is <code>NULL</code> , which means that we consider noncrossing constraints at the training points <code>x</code> .
<code>lp_solver</code>	One of "glpk" or "gurobi", indicating which LP solver to use. If possible, "gurobi" should be used because it is much faster and more stable; default is "glpk"; however, because it is open-source.
<code>time_limit</code>	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 <code>tau</code> and <code>lambda</code> value). Default is <code>NULL</code> , which means unlimited time.
<code>warm_starts</code>	Should warm starts be used in the LP solver (from one LP solve to the next)? Only supported for Gurobi.
<code>params</code>	List of control parameters to pass to Gurobi or GLPK. Default is <code>list()</code> which means no additional parameters are passed. For example: with Gurobi, we can use <code>list(Threads=4)</code> to specify that Gurobi should use 4 threads when available. (Note that if a time limit is specified through this <code>params</code> list, then its value will be overridden by the last argument <code>time_limit</code> , assuming the latter is not <code>NULL</code> .)

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 exp_pad are provided (these are inverses), as well as logit_pad and sigmd_pad (these are inverses).
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 :

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

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:

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

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

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 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,  
  nlambdas = 30,  
  lambda_min_ratio = 0.001,  
  weights = NULL,  
  intercept = TRUE,  
  standardize = TRUE,  
  lp_solver = c("glpk", "gurobi"),  
  time_limit = NULL,  
  warm_starts = TRUE,  
  params = list(),  
  transform = NULL,  
  inv_trans = NULL,  
  jitter = NULL,  
  verbose = FALSE  
)
```

Arguments

nlambdas	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 nlambdas 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 nlambdas and lambda_min_ratio) are as in the latter function; noncrossing constraints are disallowed.

quantile_genlasso_objective	<i>Quantile generalized lasso objective</i>
-----------------------------	---

Description

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

Usage

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

quantile_lasso	<i>Quantile lasso</i>
----------------	-----------------------

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("glpk", "gurobi"),
  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:

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

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

where the quantile levels $\tau_j, j = 1, \dots, k$ 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	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	<i>Quantile lasso on a tau by lambda grid</i>
---------------------	---

Description

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

Usage

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

Arguments

nlambdas	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 nlambdas 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 nlambdas 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	<i>Quantile lasso objective</i>
--------------------------	---------------------------------

Description

Compute lasso objective for a single tau and lambda value.

Usage

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

quantile_loss	<i>Quantile loss</i>
---------------	----------------------

Description

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

Usage

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

refit_quantile_genlasso	<i>Refit function for cv_quantile_genlasso object</i>
-------------------------	---

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

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

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

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

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 $\text{Unif}[a, b]$.

Usage

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

Index

`coef.quantile_ensemble`, [2](#)
`coef.quantile_genlasso`, [2](#)
`combine_into_array`, [3](#)
`cv_quantile_genlasso`, [3](#)
`cv_quantile_lasso`, [5](#)

`exp_pad(log_pad)`, [8](#)

`get_diff_mat`, [6](#)
`get_lambda_max`, [6](#)
`get_lambda_seq`, [7](#)

`log_pad`, [8](#)
`logit_pad`, [7](#)

`plot.cv_quantile_genlasso`, [8](#)
`predict.cv_quantile_genlasso`, [8](#)
`predict.quantile_ensemble`, [9](#)
`predict.quantile_genlasso`, [10](#)
`predict.quantile_genlasso_grid`, [11](#)

`quantgen`, [11](#)
`quantile_ensemble`, [12](#)
`quantile_extrapolate`, [14](#)
`quantile_genlasso`, [16](#)
`quantile_genlasso_grid`, [19](#)
`quantile_genlasso_objective`, [20](#)
`quantile_lasso`, [20](#)
`quantile_lasso_grid`, [22](#)
`quantile_lasso_objective`, [23](#)
`quantile_loss`, [23](#)

`refit_quantile_genlasso`, [23](#)
`refit_quantile_lasso`, [24](#)

`sigmd_pad(logit_pad)`, [7](#)

`unif_jitter`, [26](#)