Package: SeBR (via r-universe)

September 3, 2024

Type Package

Title Semiparametric Bayesian Regression Analysis

Version 1.0.0

Description Monte Carlo sampling algorithms for semiparametric Bayesian regression analysis. These models feature a nonparametric (unknown) transformation of the data paired with widely-used regression models including linear regression, spline regression, quantile regression, and Gaussian processes. The transformation enables broader applicability of these key models, including for real-valued, positive, and compactly-supported data with challenging distributional features. The samplers prioritize computational scalability and, for most cases, Monte Carlo (not MCMC) sampling for greater efficiency. Details of the methods and algorithms are provided in Kowal and Wu (2024) <doi:10.1080/01621459.2024.2395586>.

License MIT + file LICENSE

URL https://github.com/drkowal/SeBR, https://drkowal.github.io/SeBR/

BugReports https://github.com/drkowal/SeBR/issues

Imports graphics, stats

Suggests fields, GpGp, knitr, MASS, quantreg, rmarkdown, spikeSlabGAM, statmod

VignetteBuilder knitr

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

Repository https://drkowal.r-universe.dev

RemoteUrl https://github.com/drkowal/sebr

RemoteRef HEAD

RemoteSha bf1405f7fe453b8616474d911638ee64a1dbb52b

Contents

---

bgp_bc *Bayesian Gaussian processes with a Box-Cox transformation*

Description

MCMC sampling for Bayesian Gaussian process regression with a (known or unknown) Box-Cox transformation.

Usage

```
bgp_bc(
  y,
  locs,
  X = NULL,
  covfun_name = "matern_isotropic",
  locs_test = locs,
  X_test = NULL,
  nn = 30,
  emp_bayes = TRUE,
  lambda = NULL,
  sample_lambda = TRUE,
  nsave = 1000,
  nburn = 1000,
```

```
  nskip = 0
)
```

Arguments

y	n x 1 response vector
locs	n x d matrix of locations
X	n x p design matrix; if unspecified, use intercept only
covfun_name	string name of a covariance function; see ?GpGp
locs_test	n_test x d matrix of locations at which predictions are needed; default is locs
X_test	n_test x p design matrix for test data; default is X
nn	number of nearest neighbors to use; default is 30 (larger values improve the approximation but increase computing cost)
emp_bayes	logical; if TRUE, use a (faster!) empirical Bayes approach for estimating the mean function
lambda	Box-Cox transformation; if NULL, estimate this parameter
sample_lambda	logical; if TRUE, sample lambda, otherwise use the fixed value of lambda above or the MLE (if lambda unspecified)
nsave	number of MCMC iterations to save
nburn	number of MCMC iterations to discard
nskip	number of MCMC iterations to skip between saving iterations, i.e., save every (nskip + 1)th draw

Details

This function provides Bayesian inference for transformed Gaussian processes. The transformation is parametric from the Box-Cox family, which has one parameter lambda. That parameter may be fixed in advanced or learned from the data. For computational efficiency, the Gaussian process parameters are fixed at point estimates, and the latent Gaussian process is only sampled when emp_bayes = FALSE.

Value

a list with the following elements:

- coefficients the posterior mean of the regression coefficients
- fitted.values the posterior predictive mean at the test points locs_test
- fit_gp the fitted GpGp_fit object, which includes covariance parameter estimates and other model information
- post_ypred: nsave x n_test samples from the posterior predictive distribution at locs_test
- post_g: nsave posterior samples of the transformation evaluated at the unique y values
- post_lambda nsave posterior samples of lambda
- model: the model fit (here, bgp_bc)

as well as the arguments passed in.

Note

Box-Cox transformations may be useful in some cases, but in general we recommend the nonparametric transformation (with Monte Carlo, not MCMC sampling) in sbgp.

Examples

```
# Simulate some data:
n = 200 # sample size
x = seq(0, 1, length = n) # observation points

# Transform a noisy, periodic function:
y = g_inv_bc(
  sin(2*pi*x) + sin(4*pi*x) + rnorm(n, sd = .5),
              lambda = .5) # Signed square-root transformation

# Package we use for fast computing w/ Gaussian processes:
library(GpGp)

# Fit a Bayesian Gaussian process with Box-Cox transformation:
fit = bgp_bc(y = y, locs = x)
names(fit) # what is returned
coef(fit) # estimated regression coefficients (here, just an intercept)
class(fit$fit_gp) # the GpGp object is also returned
round(quantile(fit$post_lambda), 3) # summary of unknown Box-Cox parameter

# Plot the model predictions (point and interval estimates):
pi_y = t(apply(fit$post_ypred, 2, quantile, c(0.05, .95))) # 90% PI
plot(x, y, type='n', ylim = range(pi_y,y),
     xlab = 'x', ylab = 'y', main = paste('Fitted values and prediction intervals'))
polygon(c(x, rev(x)),c(pi_y[,2], rev(pi_y[,1])),col='gray', border=NA)
lines(x, y, type='p')
lines(x, fitted(fit), lwd = 3)
```

`blm_bc` *Bayesian linear model with a Box-Cox transformation*

Description

MCMC sampling for Bayesian linear regression with a (known or unknown) Box-Cox transformation. A g-prior is assumed for the regression coefficients.

Usage

```
blm_bc(
  y,
  X,
  X_test = X,
```

```
  psi = length(y),
  lambda = NULL,
  sample_lambda = TRUE,
  nsave = 1000,
  nburn = 1000,
  nskip = 0,
  verbose = TRUE
)
```

Arguments

y	n x 1 vector of observed counts
X	n x p matrix of predictors
X_test	n_test x p matrix of predictors for test data; default is the observed covariates X
psi	prior variance (g-prior)
lambda	Box-Cox transformation; if NULL, estimate this parameter
sample_lambda	logical; if TRUE, sample lambda, otherwise use the fixed value of lambda above or the MLE (if lambda unspecified)
nsave	number of MCMC iterations to save
nburn	number of MCMC iterations to discard
nskip	number of MCMC iterations to skip between saving iterations, i.e., save every (nskip + 1)th draw
verbose	logical; if TRUE, print time remaining

Details

This function provides fully Bayesian inference for a transformed linear model via MCMC sampling. The transformation is parametric from the Box-Cox family, which has one parameter lambda. That parameter may be fixed in advanced or learned from the data.

Value

a list with the following elements:

- coefficients the posterior mean of the regression coefficients
- fitted.values the posterior predictive mean at the test points X_test
- post_theta: nsave x p samples from the posterior distribution of the regression coefficients
- post_ypred: nsave x n_test samples from the posterior predictive distribution at test points X_test
- post_g: nsave posterior samples of the transformation evaluated at the unique y values
- post_lambda nsave posterior samples of lambda
- post_sigma nsave posterior samples of sigma
- model: the model fit (here, blm_bc)

as well as the arguments passed in.

Note

Box-Cox transformations may be useful in some cases, but in general we recommend the nonparametric transformation (with Monte Carlo, not MCMC sampling) in sblm.

Examples

```
# Simulate some data:
dat = simulate_tlm(n = 100, p = 5, g_type = 'step')
y = dat$y; X = dat$X # training data
y_test = dat$y_test; X_test = dat$X_test # testing data

hist(y, breaks = 25) # marginal distribution

# Fit the Bayesian linear model with a Box-Cox transformation:
fit = blm_bc(y = y, X = X, X_test = X_test)
names(fit) # what is returned
round(quantile(fit$post_lambda), 3) # summary of unknown Box-Cox parameter
```

---

bqr *Bayesian quantile regression*

Description

MCMC sampling for Bayesian quantile regression. An asymmetric Laplace distribution is assumed for the errors, so the regression models targets the specified quantile. A g-prior is assumed for the regression coefficients.

Usage

```
bqr(
  y,
  X,
  tau = 0.5,
  X_test = X,
  psi = length(y),
  nsave = 1000,
  nburn = 1000,
  nskip = 0,
  verbose = TRUE
)
```

Arguments

y	n x 1 vector of observed counts
X	n x p matrix of predictors
tau	the target quantile (between zero and one)

X_test	n_test x p matrix of predictors for test data; default is the observed covariates X
psi	prior variance (g-prior)
nsave	number of MCMC iterations to save
nburn	number of MCMC iterations to discard
nskip	number of MCMC iterations to skip between saving iterations, i.e., save every (nskip + 1)th draw
verbose	logical; if TRUE, print time remaining

Value

a list with the following elements:

- coefficients the posterior mean of the regression coefficients
- fitted.values the estimated tauth quantile at test points X_test
- post_theta: nsave x p samples from the posterior distribution of the regression coefficients
- post_ypred: nsave x n_test samples from the posterior predictive distribution at test points X_test
- model: the model fit (here, bqr)

as well as the arguments passed

Note

The asymmetric Laplace distribution is advantageous because it links the regression model (X%*%theta) to a pre-specified quantile (tau). However, it is often a poor model for observed data, and the semi-parametric version sbqr is recommended in general.

Examples

```
# Simulate some heteroskedastic data (no transformation):
dat = simulate_tlm(n = 100, p = 5, g_type = 'box-cox', heterosked = TRUE, lambda = 1)
y = dat$y; X = dat$X # training data
y_test = dat$y_test; X_test = dat$X_test # testing data

# Target this quantile:
tau = 0.05

# Fit the Bayesian quantile regression model:
fit = bqr(y = y, X = X, tau = tau, X_test = X_test)
names(fit) # what is returned

# Posterior predictive checks on testing data: empirical CDF
y0 = sort(unique(y_test))
plot(y0, y0, type='n', ylim = c(0,1),
     xlab='y', ylab='F_y', main = 'Posterior predictive ECDF')
temp = sapply(1:nrow(fit$post_ypred), function(s)
  lines(y0, ecdf(fit$post_ypred[s,])(y0), # ECDF of posterior predictive draws
        col='gray', type ='s'))
```

```
lines(y0, ecdf(y_test)(y0),  # ECDF of testing data
     col='black', type = 's', lwd = 3)

# The posterior predictive checks usually do not pass!
# try ?sbqr instead...
```

---

bsm_bc *Bayesian spline model with a Box-Cox transformation*

Description

MCMC sampling for Bayesian spline regression with a (known or unknown) Box-Cox transformation.

Usage

```
bsm_bc(
  y,
  x = NULL,
  x_test = NULL,
  psi = NULL,
  lambda = NULL,
  sample_lambda = TRUE,
  nsave = 1000,
  nburn = 1000,
  nskip = 0,
  verbose = TRUE
)
```

Arguments

y	n x 1 vector of observed counts
x	n x 1 vector of observation points; if NULL, assume equally-spaced on [0,1]
x_test	n_test x 1 vector of testing points; if NULL, assume equal to x
psi	prior variance (inverse smoothing parameter); if NULL, sample this parameter
lambda	Box-Cox transformation; if NULL, estimate this parameter
sample_lambda	logical; if TRUE, sample lambda, otherwise use the fixed value of lambda above or the MLE (if lambda unspecified)
nsave	number of MCMC iterations to save
nburn	number of MCMC iterations to discard
nskip	number of MCMC iterations to skip between saving iterations, i.e., save every (nskip + 1)th draw
verbose	logical; if TRUE, print time remaining

Details

This function provides fully Bayesian inference for a transformed spline model via MCMC sampling. The transformation is parametric from the Box-Cox family, which has one parameter `lambda`. That parameter may be fixed in advanced or learned from the data.

Value

a list with the following elements:

- `coefficients` the posterior mean of the regression coefficients
- `fitted.values` the posterior predictive mean at the test points `x_test`
- `post_theta`: `nsave` x `p` samples from the posterior distribution of the regression coefficients
- `post_ypred`: `nsave` x `n_test` samples from the posterior predictive distribution at `x_test`
- `post_g`: `nsave` posterior samples of the transformation evaluated at the unique `y` values
- `post_lambda` `nsave` posterior samples of lambda
- `model`: the model fit (here, `sbsm_bc`)

as well as the arguments passed in.

Note

Box-Cox transformations may be useful in some cases, but in general we recommend the nonparametric transformation (with Monte Carlo, not MCMC sampling) in `sbsm`.

Examples

```
# Simulate some data:
n = 100 # sample size
x = sort(runif(n)) # observation points

# Transform a noisy, periodic function:
y = g_inv_bc(
  sin(2*pi*x) + sin(4*pi*x) + rnorm(n, sd = .5),
              lambda = .5) # Signed square-root transformation

# Fit the Bayesian spline model with a Box-Cox transformation:
fit = bsm_bc(y = y, x = x)
names(fit) # what is returned
round(quantile(fit$post_lambda), 3) # summary of unknown Box-Cox parameter

# Plot the model predictions (point and interval estimates):
pi_y = t(apply(fit$post_ypred, 2, quantile, c(0.05, .95))) # 90% PI
plot(x, y, type='n', ylim = range(pi_y,y),
     xlab = 'x', ylab = 'y', main = paste('Fitted values and prediction intervals'))
polygon(c(x, rev(x)),c(pi_y[,2], rev(pi_y[,1])),col='gray', border=NA)
lines(x, y, type='p')
lines(x, fitted(fit), lwd = 3)
```

computeTimeRemaining *Estimate the remaining time in the MCMC based on previous samples*

Description

Estimate the remaining time in the MCMC based on previous samples

Usage

```
computeTimeRemaining(nsi, timer0, nsims, nrep = 1000)
```

Arguments

nsi	Current iteration
timer0	Initial timer value, returned from proc.time()[3]
nsims	Total number of simulations
nrep	Print the estimated time remaining every nrep iterations

Value

Table of summary statistics using the function summary

contract_grid *Grid contraction*

Description

Contract the grid if the evaluation points exceed some threshold. This removes the corresponding z values. We can add points back to achieve the same (approximate) length.

Usage

```
contract_grid(z, Fz, lower, upper, add_back = TRUE, monotone = TRUE)
```

Arguments

z	grid points (ordered)
Fz	function evaluated at those grid points
lower	lower threshold at which to check Fz
upper	upper threshold at which to check Fz
add_back	logical; if true, expand the grid to (about) the original size
monotone	logical; if true, enforce monotonicity on the expanded grid

Value

a list containing the grid points z and the (interpolated) function Fz at those points

Fz_fun *Compute the latent data CDF*

Description

Assuming a Gaussian latent data distribution (given x), compute the CDF on a grid of points

Usage

```
Fz_fun(z, weights = NULL, mean_vec = NULL, sd_vec)
```

Arguments

z	vector of points at which the CDF of z is evaluated
weights	n-dimensional vector of weights; if NULL, assume 1/n
mean_vec	n-dimensional vector of means; if NULL, assume mean zero
sd_vec	n-dimensional vector of standard deviations

Value

CDF of z evaluated at z

g_bc *Box-Cox transformation*

Description

Evaluate the Box-Cox transformation, which is a scaled power transformation to preserve continuity in the index lambda at zero. Negative values are permitted.

Usage

```
g_bc(t, lambda)
```

Arguments

t	argument(s) at which to evaluate the function
lambda	Box-Cox parameter

Value

The evaluation(s) of the Box-Cox function at the given input(s) t.

Note

Special cases include the identity transformation (`lambda = 1`), the square-root transformation (`lambda = 1/2`), and the log transformation (`lambda = 0`).

Examples

```
# Log-transformation:
g_bc(1:5, lambda = 0); log(1:5)

# Square-root transformation: note the shift and scaling
g_bc(1:5, lambda = 1/2); sqrt(1:5)
```

---

g_fun *Compute the transformation*

Description

Given the CDFs of z and y, compute a smoothed function to evaluate the transformation

Usage

```
g_fun(y, Fy_eval, z, Fz_eval)
```

Arguments

`y`	vector of points at which the CDF of y is evaluated
`Fy_eval`	CDF of y evaluated at `y`
`z`	vector of points at which the CDF of z is evaluated
`Fz_eval`	CDF of z evaluated at `z`

Value

A smooth monotone function which can be used for evaluations of the transformation.

g_inv_approx *Approximate inverse transformation*

Description

Compute the inverse function of a transformation g based on a grid search.

Usage

```
g_inv_approx(g, t_grid)
```

Arguments

g	the transformation function
t_grid	grid of arguments at which to evaluate the transformation function

Value

A function which can be used for evaluations of the (approximate) inverse transformation function.

g_inv_bc *Inverse Box-Cox transformation*

Description

Evaluate the inverse Box-Cox transformation. Negative values are permitted.

Usage

```
g_inv_bc(s, lambda)
```

Arguments

s	argument(s) at which to evaluate the function
lambda	Box-Cox parameter

Value

The evaluation(s) of the inverse Box-Cox function at the given input(s) s.

Note

Special cases include the identity transformation (lambda = 1), the square-root transformation (lambda = 1/2), and the log transformation (lambda = 0).

#' @examples # (Inverse) log-transformation: g_inv_bc(1:5, lambda = 0); exp(1:5)

(Inverse) square-root transformation: note the shift and scaling g_inv_bc(1:5, lambda = 1/2); (1:5)^2

plot_pptest *Plot point and interval predictions on testing data*

Description

Given posterior predictive samples at `X_test`, plot the point and interval estimates and compare to the actual testing data `y_test`.

Usage

```
plot_pptest(post_ypred, y_test, alpha_level = 0.1)
```

Arguments

`post_ypred`	`nsave` x `n_test` samples from the posterior predictive distribution at test points `X_test`
`y_test`	`n_test` testing points
`alpha_level`	alpha-level for prediction intervals

Value

plot of the testing data, point and interval predictions, and a summary of the empirical coverage

Examples

```
# Simulate some data:
dat = simulate_tlm(n = 100, p = 5, g_type = 'step')

# Fit a semiparametric Bayesian linear model:
fit = sblm(y = dat$y, X = dat$X, X_test = dat$X_test)

# Evaluate posterior predictive means and intervals on the testing data:
plot_pptest(fit$post_ypred, dat$y_test,
            alpha_level = 0.10) # coverage should be about 90%
```

rank_approx *Rank-based estimation of the linear regression coefficients*

Description

For a transformed Gaussian linear model, compute point estimates of the regression coefficients. This approach uses the ranks of the data and does not require the transformation, but must expand the sample size to `n^2` and thus can be slow.

Usage

```
rank_approx(y, X)
```

Arguments

y	n x 1 response vector
X	n x p matrix of predictors (should not include an intercept!)

Value

the estimated linear coefficients

Examples

```
# Simulate some data:
dat = simulate_tlm(n = 200, p = 10, g_type = 'step')

# Point estimates for the linear coefficients:
theta_hat = suppressWarnings(
  rank_approx(y = dat$y,
              X = dat$X[,-1]) # remove intercept
) # warnings occur from glm.fit (fitted probabilities 0 or 1)

# Check: correlation with true coefficients
cor(dat$beta_true[-1], # excluding the intercept
    theta_hat)
```

---

sbgp *Semiparametric Bayesian Gaussian processes*

Description

Monte Carlo sampling for Bayesian Gaussian process regression with an unknown (nonparametric) transformation.

Usage

```
sbgp(
  y,
  locs,
  X = NULL,
  covfun_name = "matern_isotropic",
  locs_test = locs,
  X_test = NULL,
  nn = 30,
  emp_bayes = TRUE,
  approx_g = FALSE,
```

```
  nsave = 1000,
  ngrid = 100
)
```

Arguments

y	n x 1 response vector
locs	n x d matrix of locations
X	n x p design matrix; if unspecified, use intercept only
covfun_name	string name of a covariance function; see ?GpGp
locs_test	n_test x d matrix of locations at which predictions are needed; default is locs
X_test	n_test x p design matrix for test data; default is X
nn	number of nearest neighbors to use; default is 30 (larger values improve the approximation but increase computing cost)
emp_bayes	logical; if TRUE, use a (faster!) empirical Bayes approach for estimating the mean function
approx_g	logical; if TRUE, apply large-sample approximation for the transformation
nsave	number of Monte Carlo simulations
ngrid	number of grid points for inverse approximations

Details

This function provides Bayesian inference for a transformed Gaussian process model using Monte Carlo (not MCMC) sampling. The transformation is modeled as unknown and learned jointly with the regression function (unless approx_g = TRUE, which then uses a point approximation). This model applies for real-valued data, positive data, and compactly-supported data (the support is automatically deduced from the observed y values). The results are typically unchanged whether laplace_approx is TRUE/FALSE; setting it to TRUE may reduce sensitivity to the prior, while setting it to FALSE may speed up computations for very large datasets. For computational efficiency, the Gaussian process parameters are fixed at point estimates, and the latent Gaussian process is only sampled when emp_bayes = FALSE. However, the uncertainty from this term is often negligible compared to the observation errors, and the transformation serves as an additional layer of robustness.

Value

a list with the following elements:

- coefficients the estimated regression coefficients
- fitted.values the posterior predictive mean at the test points locs_test
- fit_gp the fitted GpGp_fit object, which includes covariance parameter estimates and other model information
- post_ypred: nsave x ntest samples from the posterior predictive distribution at locs_test
- post_g: nsave posterior samples of the transformation evaluated at the unique y values
- model: the model fit (here, sbgp)

as well as the arguments passed in.

Examples

```
# Simulate some data:
n = 200 # sample size
x = seq(0, 1, length = n) # observation points

# Transform a noisy, periodic function:
y = g_inv_bc(
  sin(2*pi*x) + sin(4*pi*x) + rnorm(n, sd = .5),
              lambda = .5) # Signed square-root transformation

# Package we use for fast computing w/ Gaussian processes:
library(GpGp)

# Fit the semiparametric Bayesian Gaussian process:
fit = sbgp(y = y, locs = x)
names(fit) # what is returned
coef(fit) # estimated regression coefficients (here, just an intercept)
class(fit$fit_gp) # the GpGp object is also returned

# Plot the model predictions (point and interval estimates):
pi_y = t(apply(fit$post_ypred, 2, quantile, c(0.05, .95))) # 90% PI
plot(x, y, type='n', ylim = range(pi_y,y),
     xlab = 'x', ylab = 'y', main = paste('Fitted values and prediction intervals'))
polygon(c(x, rev(x)),c(pi_y[,2], rev(pi_y[,1])),col='gray', border=NA)
lines(x, y, type='p')
lines(x, fitted(fit), lwd = 3)
```

---

sblm *Semiparametric Bayesian linear model*

Description

Monte Carlo sampling for Bayesian linear regression with an unknown (nonparametric) transformation. A g-prior is assumed for the regression coefficients.

Usage

```
sblm(
  y,
  X,
  X_test = X,
  psi = length(y),
  laplace_approx = TRUE,
  approx_g = FALSE,
  nsave = 1000,
  ngrid = 100,
  verbose = TRUE
)
```

Arguments

y	n x 1 response vector
X	n x p matrix of predictors
X_test	n_test x p matrix of predictors for test data; default is the observed covariates X
psi	prior variance (g-prior)
laplace_approx	logical; if TRUE, use a normal approximation to the posterior in the definition of the transformation; otherwise the prior is used
approx_g	logical; if TRUE, apply large-sample approximation for the transformation
nsave	number of Monte Carlo simulations
ngrid	number of grid points for inverse approximations
verbose	logical; if TRUE, print time remaining

Details

This function provides fully Bayesian inference for a transformed linear model using Monte Carlo (not MCMC) sampling. The transformation is modeled as unknown and learned jointly with the regression coefficients (unless approx_g = TRUE, which then uses a point approximation). This model applies for real-valued data, positive data, and compactly-supported data (the support is automatically deduced from the observed y values). The results are typically unchanged whether laplace_approx is TRUE/FALSE; setting it to TRUE may reduce sensitivity to the prior, while setting it to FALSE may speed up computations for very large datasets.

Value

a list with the following elements:

- coefficients the posterior mean of the regression coefficients
- fitted.values the posterior predictive mean at the test points X_test
- post_theta: nsave x p samples from the posterior distribution of the regression coefficients
- post_ypred: nsave x n_test samples from the posterior predictive distribution at test points X_test
- post_g: nsave posterior samples of the transformation evaluated at the unique y values
- model: the model fit (here, sblm)

as well as the arguments passed in.

Examples

```
# Simulate some data:
dat = simulate_tlm(n = 100, p = 5, g_type = 'step')
y = dat$y; X = dat$X # training data
y_test = dat$y_test; X_test = dat$X_test # testing data

hist(y, breaks = 25) # marginal distribution
```

```
# Fit the semiparametric Bayesian linear model:
fit = sblm(y = y, X = X, X_test = X_test)
names(fit) # what is returned

# Note: this is Monte Carlo sampling, so no need for MCMC diagnostics!

# Evaluate posterior predictive means and intervals on the testing data:
plot_pptest(fit$post_ypred, y_test,
            alpha_level = 0.10) # coverage should be about 90%

# Check: correlation with true coefficients
cor(dat$beta_true[-1],
    coef(fit)[-1]) # excluding the intercept

# Summarize the transformation:
y0 = sort(unique(y)) # posterior draws of g are evaluated at the unique y observations
plot(y0, fit$post_g[1,], type='n', ylim = range(fit$post_g),
     xlab = 'y', ylab = 'g(y)', main = "Posterior draws of the transformation")
temp = sapply(1:nrow(fit$post_g), function(s)
  lines(y0, fit$post_g[s,], col='gray')) # posterior draws
lines(y0, colMeans(fit$post_g), lwd = 3) # posterior mean

# Add the true transformation, rescaled for easier comparisons:
lines(y,
    scale(dat$g_true)*sd(colMeans(fit$post_g)) + mean(colMeans(fit$post_g)), type='p', pch=2)
legend('bottomright', c('Truth'), pch = 2) # annotate the true transformation

# Posterior predictive checks on testing data: empirical CDF
y0 = sort(unique(y_test))
plot(y0, y0, type='n', ylim = c(0,1),
     xlab='y', ylab='F_y', main = 'Posterior predictive ECDF')
temp = sapply(1:nrow(fit$post_ypred), function(s)
  lines(y0, ecdf(fit$post_ypred[s,])(y0), # ECDF of posterior predictive draws
        col='gray', type ='s'))
lines(y0, ecdf(y_test)(y0),  # ECDF of testing data
     col='black', type = 's', lwd = 3)
```

---

sbqr *Semiparametric Bayesian quantile regression*

Description

MCMC sampling for Bayesian quantile regression with an unknown (nonparametric) transformation. Like in traditional Bayesian quantile regression, an asymmetric Laplace distribution is assumed for the errors, so the regression models targets the specified quantile. However, these models are often woefully inadequate for describing observed data. We introduce a nonparametric transformation to improve model adequacy while still providing inference for the regression coefficients and the specified quantile. A g-prior is assumed for the regression coefficients.

Usage

```
sbqr(
  y,
  X,
  tau = 0.5,
  X_test = X,
  psi = length(y),
  laplace_approx = TRUE,
  approx_g = FALSE,
  nsave = 1000,
  nburn = 100,
  ngrid = 100,
  verbose = TRUE
)
```

Arguments

y	n x 1 response vector
X	n x p matrix of predictors
tau	the target quantile (between zero and one)
X_test	n_test x p matrix of predictors for test data; default is the observed covariates X
psi	prior variance (g-prior)
laplace_approx	logical; if TRUE, use a normal approximation to the posterior in the definition of the transformation; otherwise the prior is used
approx_g	logical; if TRUE, apply large-sample approximation for the transformation
nsave	number of MCMC iterations to save
nburn	number of MCMC iterations to discard
ngrid	number of grid points for inverse approximations
verbose	logical; if TRUE, print time remaining

Details

This function provides fully Bayesian inference for a transformed quantile linear model. The transformation is modeled as unknown and learned jointly with the regression coefficients (unless approx_g = TRUE, which then uses a point approximation). This model applies for real-valued data, positive data, and compactly-supported data (the support is automatically deduced from the observed y values). The results are typically unchanged whether laplace_approx is TRUE/FALSE; setting it to TRUE may reduce sensitivity to the prior, while setting it to FALSE may speed up computations for very large datasets.

Value

a list with the following elements:

- coefficients the posterior mean of the regression coefficients

- fitted.values the estimated tauth quantile at test points X_test
- post_theta: nsave x p samples from the posterior distribution of the regression coefficients
- post_ypred: nsave x n_test samples from the posterior predictive distribution at test points X_test
- post_qtau: nsave x n_test samples of the tauth conditional quantile at test points X_test
- post_g: nsave posterior samples of the transformation evaluated at the unique y values
- model: the model fit (here, sbqr)

as well as the arguments passed in.

Examples

```
# Simulate some heteroskedastic data (no transformation):
dat = simulate_tlm(n = 200, p = 10, g_type = 'box-cox', heterosked = TRUE, lambda = 1)
y = dat$y; X = dat$X # training data
y_test = dat$y_test; X_test = dat$X_test # testing data

# Target this quantile:
tau = 0.05

# Fit the semiparametric Bayesian quantile regression model:
fit = sbqr(y = y, X = X, tau = tau, X_test = X_test)
names(fit) # what is returned

# Posterior predictive checks on testing data: empirical CDF
y0 = sort(unique(y_test))
plot(y0, y0, type='n', ylim = c(0,1),
     xlab='y', ylab='F_y', main = 'Posterior predictive ECDF')
temp = sapply(1:nrow(fit$post_ypred), function(s)
  lines(y0, ecdf(fit$post_ypred[s,])(y0), # ECDF of posterior predictive draws
        col='gray', type ='s'))
lines(y0, ecdf(y_test)(y0),  # ECDF of testing data
     col='black', type = 's', lwd = 3)
```

---

sbsm *Semiparametric Bayesian spline model*

Description

Monte Carlo sampling for Bayesian spline regression with an unknown (nonparametric) transformation.

Usage

```
sbsm(
  y,
  x = NULL,
  x_test = NULL,
  psi = NULL,
  laplace_approx = TRUE,
  approx_g = FALSE,
  nsave = 1000,
  ngrid = 100,
  verbose = TRUE
)
```

Arguments

y	n x 1 response vector
x	n x 1 vector of observation points; if NULL, assume equally-spaced on [0,1]
x_test	n_test x 1 vector of testing points; if NULL, assume equal to x
psi	prior variance (inverse smoothing parameter); if NULL, sample this parameter
laplace_approx	logical; if TRUE, use a normal approximation to the posterior in the definition of the transformation; otherwise the prior is used
approx_g	logical; if TRUE, apply large-sample approximation for the transformation
nsave	number of Monte Carlo simulations
ngrid	number of grid points for inverse approximations
verbose	logical; if TRUE, print time remaining

Details

This function provides fully Bayesian inference for a transformed spline regression model using Monte Carlo (not MCMC) sampling. The transformation is modeled as unknown and learned jointly with the regression function (unless approx_g = TRUE, which then uses a point approximation). This model applies for real-valued data, positive data, and compactly-supported data (the support is automatically deduced from the observed y values). The results are typically unchanged whether laplace_approx is TRUE/FALSE; setting it to TRUE may reduce sensitivity to the prior, while setting it to FALSE may speed up computations for very large datasets.

Value

a list with the following elements:

- coefficients the posterior mean of the regression coefficients
- fitted.values the posterior predictive mean at the test points x_test
- post_theta: nsave x p samples from the posterior distribution of the regression coefficients
- post_ypred: nsave x n_test samples from the posterior predictive distribution at x_test
- post_g: nsave posterior samples of the transformation evaluated at the unique y values

- model: the model fit (here, sbsm)

as well as the arguments passed in.

Examples

```
# Simulate some data:
n = 100 # sample size
x = sort(runif(n)) # observation points

# Transform a noisy, periodic function:
y = g_inv_bc(
  sin(2*pi*x) + sin(4*pi*x) + rnorm(n, sd = .5),
              lambda = .5) # Signed square-root transformation

# Fit the semiparametric Bayesian spline model:
fit = sbsm(y = y, x = x)
names(fit) # what is returned

# Note: this is Monte Carlo sampling, so no need for MCMC diagnostics!

# Plot the model predictions (point and interval estimates):
pi_y = t(apply(fit$post_ypred, 2, quantile, c(0.05, .95))) # 90% PI
plot(x, y, type='n', ylim = range(pi_y,y),
     xlab = 'x', ylab = 'y', main = paste('Fitted values and prediction intervals'))
polygon(c(x, rev(x)),c(pi_y[,2], rev(pi_y[,1])),col='gray', border=NA)
lines(x, y, type='p')
lines(x, fitted(fit), lwd = 3)
```

---

simulate_tlm *Simulate a transformed linear model*

Description

Generate training data (X, y) and testing data (X_test, y_test) for a transformed linear model. The covariates are correlated Gaussian variables. Half of the true regression coefficients are zero and the other half are one. There are multiple options for the transformation, which define the support of the data (see below).

Usage

```
simulate_tlm(
  n,
  p,
  g_type = "beta",
  n_test = 1000,
  heterosked = FALSE,
  lambda = 1
)
```

Arguments

n	number of observations in the training data
p	number of covariates
g_type	type of transformation; must be one of beta, step, or box-cox
n_test	number of observations in the testing data
heterosked	logical; if TRUE, simulate the latent data with heteroskedasticity
lambda	Box-Cox parameter (only applies for g_type = 'box-cox')

Details

The transformations vary in complexity and support for the observed data, and include the following options: beta yields marginally Beta(0.1, 0.5) data supported on [0,1]; step generates a locally-linear inverse transformation and produces positive data; and box-cox refers to the signed Box-Cox family indexed by lambda, which generates real-valued data with examples including identity, square-root, and log transformations.

Value

a list with the following elements:

- y: the response variable in the training data
- X: the covariates in the training data
- y_test: the response variable in the testing data
- X_test: the covariates in the testing data
- beta_true: the true regression coefficients
- g_true: the true transformation, evaluated at y

Examples

```
# Simulate data:
dat = simulate_tlm(n = 100, p = 5, g_type = 'beta')
names(dat) # what is returned
hist(dat$y, breaks = 25) # marginal distribution
```

---

sir_adjust *Post-processing with importance sampling*

Description

Given Monte Carlo draws from the surrogate posterior, apply sampling importance reweighting (SIR) to correct for the true model likelihood.

Usage

```
sir_adjust(fit, sir_frac = 0.3, nsims_prior = 100, verbose = TRUE)
```

Arguments

fit	a fitted model object that includes • coefficients the posterior mean of the regression coefficients • post_theta: nsave x p samples from the posterior distribution of the regression coefficients • post_ypred: nsave x n_test samples from the posterior predictive distribution at test points X_test • post_g: nsave posterior samples of the transformation evaluated at the unique y values • model: the model fit (sblm or sbsm)
sir_frac	fraction of draws to sample for SIR
nsims_prior	number of draws from the prior
verbose	logical; if TRUE, print time remaining

Details

The Monte Carlo sampling for sblm and sbsm uses a surrogate likelihood for posterior inference, which enables much faster and easier computing. SIR provides a correction for the actual (specified) likelihood. However, this correction step is quite slow and typically does not produce any noticeable discrepancies, even for small sample sizes.

Value

the fitted model object with the posterior draws subsampled based on the SIR adjustment

Note

SIR sampling is done WITHOUT replacement, so sir_frac is typically between 0.1 and 0.5. The nsims_priors draws are used to approximate a prior expectation, but larger values can significantly slow down this function.

Examples

```
# Simulate some data:
dat = simulate_tlm(n = 50, p = 5, g_type = 'step')
y = dat$y; X = dat$X # training data
y_test = dat$y_test; X_test = dat$X_test # testing data

hist(y, breaks = 10) # marginal distribution

# Fit the semiparametric Bayesian linear model:
fit = sblm(y = y, X = X, X_test = X_test)
names(fit) # what is returned
```

```
# Update with SIR:
fit_sir = sir_adjust(fit)

# Prediction: unadjusted vs. adjusted?

# Point estimates:
y_hat = fitted(fit)
y_hat_sir = fitted(fit_sir)
cor(y_hat, y_hat_sir) # similar

# Interval estimates:
pi_y = t(apply(fit$post_ypred, 2, quantile, c(0.05, .95))) # 90% PI
pi_y_sir = t(apply(fit_sir$post_ypred, 2, quantile, c(0.05, .95))) # 90% PI

# PI overlap (%):
overlaps = 100*sapply(1:length(y_test), function(i){
  # innermost part
  (min(pi_y[i,2], pi_y_sir[i,2]) - max(pi_y[i,1], pi_y_sir[i,1]))/
    # outermost part
    (max(pi_y[i,2], pi_y_sir[i,2]) - min(pi_y[i,1], pi_y_sir[i,1]))
})
summary(overlaps) # mostly close to 100%

# Coverage of PIs on testing data (should be ~ 90%)
mean((pi_y[,1] <= y_test)*(pi_y[,2] >= y_test)) # unadjusted
mean((pi_y_sir[,1] <= y_test)*(pi_y_sir[,2] >= y_test)) # adjusted

# Plot together with testing data:
plot(y_test, y_test, type='n', ylim = range(pi_y, pi_y_sir, y_test),
     xlab = 'y_test', ylab = 'y_hat', main = paste('Prediction intervals: testing data'))
abline(0,1) # reference line
suppressWarnings(
  arrows(y_test, pi_y[,1], y_test, pi_y[,2],
         length=0.15, angle=90, code=3, col='gray', lwd=2)
) # plot the PIs (unadjusted)
suppressWarnings(
  arrows(y_test, pi_y_sir[,1], y_test, pi_y_sir[,2],
         length=0.15, angle=90, code=3, col='darkgray', lwd=2)
) # plot the PIs (adjusted)
lines(y_test, y_hat, type='p', pch=2) # plot the means (unadjusted)
lines(y_test, y_hat_sir, type='p', pch=3) # plot the means (adjusted)
```

`uni.slice` *Univariate Slice Sampler from Neal (2008)*

Description

Compute a draw from a univariate distribution using the code provided by Radford M. Neal. The documentation below is also reproduced from Neal (2008).

Usage

```
uni.slice(x0, g, w = 1, m = Inf, lower = -Inf, upper = +Inf, gx0 = NULL)
```

Arguments

x0	Initial point
g	Function returning the log of the probability density (plus constant)
w	Size of the steps for creating interval (default 1)
m	Limit on steps (default infinite)
lower	Lower bound on support of the distribution (default -Inf)
upper	Upper bound on support of the distribution (default +Inf)
gx0	Value of g(x0), if known (default is not known)

Value

The point sampled, with its log density attached as an attribute.

Note

The log density function may return -Inf for points outside the support of the distribution. If a lower and/or upper bound is specified for the support, the log density function will not be called outside such limits.

Index