Package 'countSTAR'

July 22, 2025

Type Package

Title Flexible Modeling of Count Data

Version 1.0.2

Description For Bayesian and classical inference and prediction with count-valued data, Simultaneous Transformation and Rounding (STAR) Models provide a flexible, interpretable, and easy-to-use approach. STAR models the observed count data using a rounded continuous data model and incorporates a transformation for greater flexibility. Implicitly, STAR formalizes the commonly-applied yet incoherent procedure of (i) transforming count-valued data and subsequently (ii) modeling the transformed data using Gaussian models. STAR is well-defined for count-valued data, which is reflected in predictive accuracy, and is designed to account for zero-inflation, bounded or censored data, and over- or underdispersion. Importantly, STAR is easy to combine with existing MCMC or point estimation methods for continuous data, which allows seamless adaptation of continuous data models (such as linear regressions, additive models, BART, random forests, and gradient boosting machines) for count-valued data. The package also includes several methods for modeling count time series data, namely via warped Dynamic Linear Models. For more details and background on these methodologies, see the works of Kowal and Canale (2020) <doi:10.1214/20-EJS1707>, Kowal and Wu (2022) <doi:10.1111/biom.13617>, King and Kowal (2022) <doi:10.48550/arXiv.2110.14790>, and Kowal and Wu (2023) <doi:10.48550/arXiv.2110.12316>.

License GPL (>= 2)

Encoding UTF-8

LinkingTo Rcpp, RcppArmadillo

Imports stats, utils, coda, dbarts, FastGP, gbm, graphics, Matrix, spikeSlabGAM, splines2, randomForest, Rcpp, TruncatedNormal, truncdist, KFAS

RoxygenNote 7.2.3

Suggests knitr, rmarkdown, bayesplot, mgcv

VignetteBuilder knitr

URL https://bking124.github.io/countSTAR/
https://github.com/bking124/countSTAR

BugReports https://github.com/bking124/countSTAR/issues

Depends R (>= 2.10)

LazyData true

NeedsCompilation yes

Author Brian King [aut, cre],
Dan Kowal [aut]

Maintainer Brian King <brianking387@gmail.com>

Repository CRAN

Date/Publication 2023-06-30 18:30:08 UTC

Contents

`a_j` *Inverse rounding function*

Description

Define the intervals associated with `y = j` based on the flooring function. The function returns `-Inf` for `j = 0` (or smaller) and `Inf` for any `j >= y_max + 1`, where `y_max` is a known upper bound on the data `y` (if specified).

Usage

```
a_j(j, y_max = Inf)
```

Arguments

`j`	the integer-valued input(s)
`y_max`	a fixed and known upper bound for all observations; default is `Inf`

Value

The (lower) interval endpoint(s) associated with `j`.

Examples

```
# Standard cases:
a_j(1)
a_j(20)

# Boundary cases:
a_j(0)
a_j(20, y_max = 15)
```

bam_star *Fit Bayesian Additive STAR Model with MCMC*

Description

Run the MCMC algorithm for a STAR Bayesian additive model The transformation can be known (e.g., log or sqrt) or unknown (Box-Cox or estimated nonparametrically) for greater flexibility.

Usage

```
bam_star(
  y,
  X_lin,
  X_nonlin,
  splinetype = "orthogonal",
  transformation = "np",
  y_max = Inf,
  nsave = 5000,
  nburn = 5000,
  nskip = 2,
  save_y_hat = FALSE,
  verbose = TRUE
)
```

Arguments

y	n x 1 vector of observed counts
X_lin	n x pL matrix of predictors to be modelled as linear
X_nonlin	n x pNL matrix of predictors to be modelled as nonlinear
splinetype	Type of spline to use for modelling the nonlinear predictors; must be either "orthogonal" (orthogonalized splines–the default) or "thinplate" (low-rank thin plate splines)
transformation	transformation to use for the latent data; must be one of • "identity" (identity transformation) • "log" (log transformation) • "sqrt" (square root transformation) • "np" (nonparametric transformation estimated from empirical CDF) • "pois" (transformation for moment-matched marginal Poisson CDF) • "neg-bin" (transformation for moment-matched marginal Negative Binomial CDF) • "box-cox" (box-cox transformation with learned parameter) • "ispline" (transformation is modeled as unknown, monotone function using I-splines)
y_max	a fixed and known upper bound for all observations; default is Inf

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
save_y_hat	logical; if TRUE, compute and save the posterior draws of the expected counts, E(y), which may be slow to compute
verbose	logical; if TRUE, print time remaining

Details

STAR defines a count-valued probability model by (1) specifying a Gaussian model for continuous *latent* data and (2) connecting the latent data to the observed data via a *transformation and rounding* operation.

Posterior and predictive inference is obtained via a Gibbs sampler that combines (i) a latent data augmentation step (like in probit regression) and (ii) an existing sampler for a continuous data model.

There are several options for the transformation. First, the transformation can belong to the *Box-Cox* family, which includes the known transformations 'identity', 'log', and 'sqrt', as well as a version in which the Box-Cox parameter is inferred within the MCMC sampler ('box-cox'). Second, the transformation can be estimated (before model fitting) using the empirical distribution of the data y. Options in this case include the empirical cumulative distribution function (CDF), which is fully nonparametric ('np'), or the parametric alternatives based on Poisson ('pois') or Negative-Binomial ('neg-bin') distributions. For the parametric distributions, the parameters of the distribution are estimated using moments (means and variances) of y. Third, the transformation can be modeled as an unknown, monotone function using I-splines ('ispline'). The Robust Adaptive Metropolis (RAM) sampler is used for drawing the parameter of the transformation function.

Value

a list with at least the following elements:

- coefficients: the posterior mean of the coefficients
- fitted.values: the posterior mean of the conditional expectation of the counts y
- post.coefficients: posterior draws of the coefficients
- post.fitted.values: posterior draws of the conditional mean of the counts y
- post.pred: draws from the posterior predictive distribution of y
- post.lambda: draws from the posterior distribution of lambda
- post.sigma: draws from the posterior distribution of sigma
- post.log.like.point: draws of the log-likelihood for each of the n observations
- WAIC: Widely-Applicable/Watanabe-Akaike Information Criterion
- p_waic: Effective number of parameters based on WAIC

In the case of transformation="ispline", the list also contains

- post.g: draws from the posterior distribution of the transformation g
- post.sigma.gamma: draws from the posterior distribution of sigma.gamma, the prior standard deviation of the transformation g() coefficients

Examples

```
# Simulate data with count-valued response y:
sim_dat = simulate_nb_friedman(n = 100, p = 5, seed=32)
y = sim_dat$y; X = sim_dat$X

# Linear and nonlinear components:
X_lin = as.matrix(X[,-(1:3)])
X_nonlin = as.matrix(X[,(1:3)])

# STAR: nonparametric transformation
fit <- bam_star(y,X_lin, X_nonlin, nburn=1000, nskip=0)

# Posterior mean of each coefficient:
coef(fit)

# WAIC:
fit$WAIC

# MCMC diagnostics:
plot(as.ts(fit$post.coefficients[,1:3]))

# Posterior predictive check:
hist(apply(fit$post.pred, 1,
           function(x) mean(x==0)), main = 'Proportion of Zeros', xlab='');
abline(v = mean(y==0), lwd=4, col ='blue')
```

---

bart_star *MCMC Algorithm for BART-STAR*

Description

Run the MCMC algorithm for a BART model for count-valued responses using STAR. The transformation can be known (e.g., log or sqrt) or unknown (Box-Cox or estimated nonparametrically) for greater flexibility.

Usage

```
bart_star(
  y,
  X,
  X_test = NULL,
  y_test = NULL,
  transformation = "np",
  y_max = Inf,
  n.trees = 200,
  sigest = NULL,
  sigdf = 3,
```

```
  sigquant = 0.9,
  k = 2,
  power = 2,
  base = 0.95,
  nsave = 5000,
  nburn = 5000,
  nskip = 2,
  save_y_hat = FALSE,
  verbose = TRUE
)
```

Arguments

y	n x 1 vector of observed counts
X	n x p matrix of predictors
X_test	n0 x p matrix of predictors for test data
y_test	n0 x 1 vector of the test data responses (used for computing log-predictive scores)
transformation	transformation to use for the latent process; must be one of • "identity" (identity transformation) • "log" (log transformation) • "sqrt" (square root transformation) • "np" (nonparametric transformation estimated from empirical CDF) • "pois" (transformation for moment-matched marginal Poisson CDF) • "neg-bin" (transformation for moment-matched marginal Negative Binomial CDF) • "box-cox" (box-cox transformation with learned parameter) • "ispline" (transformation is modeled as unknown, monotone function using I-splines)
y_max	a fixed and known upper bound for all observations; default is Inf
n.trees	number of trees to use in BART; default is 200
sigest	positive numeric estimate of the residual standard deviation (see ?bart)
sigdf	degrees of freedom for error variance prior (see ?bart)
sigquant	quantile of the error variance prior that the rough estimate (sigest) is placed at. The closer the quantile is to 1, the more aggressive the fit will be (see ?bart)
k	the number of prior standard deviations E(Y|x) = f(x) is away from +/- 0.5. The response is internally scaled to range from -0.5 to 0.5. The bigger k is, the more conservative the fitting will be (see ?bart)
power	power parameter for tree prior (see ?bart)
base	base parameter for tree prior (see ?bart)
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

save_y_hat	logical; if TRUE, compute and save the posterior draws of the expected counts, E(y), which may be slow to compute
verbose	logical; if TRUE, print time remaining

Details

STAR defines a count-valued probability model by (1) specifying a Gaussian model for continuous *latent* data and (2) connecting the latent data to the observed data via a *transformation and rounding* operation. Here, the model in (1) is a Bayesian additive regression tree (BART) model.

Posterior and predictive inference is obtained via a Gibbs sampler that combines (i) a latent data augmentation step (like in probit regression) and (ii) an existing sampler for a continuous data model.

There are several options for the transformation. First, the transformation can belong to the *Box-Cox* family, which includes the known transformations 'identity', 'log', and 'sqrt', as well as a version in which the Box-Cox parameter is inferred within the MCMC sampler ('box-cox'). Second, the transformation can be estimated (before model fitting) using the empirical distribution of the data y. Options in this case include the empirical cumulative distribution function (CDF), which is fully nonparametric ('np'), or the parametric alternatives based on Poisson ('pois') or Negative-Binomial ('neg-bin') distributions. For the parametric distributions, the parameters of the distribution are estimated using moments (means and variances) of y. Third, the transformation can be modeled as an unknown, monotone function using I-splines ('ispline'). The Robust Adaptive Metropolis (RAM) sampler is used for drawing the parameter of the transformation function.

Value

a list with the following elements:

- post.pred: draws from the posterior predictive distribution of y
- post.sigma: draws from the posterior distribution of sigma
- post.log.like.point: draws of the log-likelihood for each of the n observations
- WAIC: Widely-Applicable/Watanabe-Akaike Information Criterion
- p_waic: Effective number of parameters based on WAIC
- post.pred.test: draws from the posterior predictive distribution at the test points X_test (NULL if X_test is not given)
- post.fitted.values.test: posterior draws of the conditional mean at the test points X_test (NULL if X_test is not given)
- post.mu.test: draws of the conditional mean of z_star at the test points X_test (NULL if X_test is not given)
- post.log.pred.test: draws of the log-predictive distribution for each of the n0 test cases (NULL if X_test is not given)
- fitted.values: the posterior mean of the conditional expectation of the counts y (NULL if save_y_hat=FALSE)
- post.fitted.values: posterior draws of the conditional mean of the counts y (NULL if save_y_hat=FALSE)

In the case of transformation="ispline", the list also contains

- post.g: draws from the posterior distribution of the transformation g
- post.sigma.gamma: draws from the posterior distribution of sigma.gamma, the prior standard deviation of the transformation g() coefficients

If transformation="box-cox", then the list also contains

- post.lambda: draws from the posterior distribution of lambda

Examples

```
# Simulate data with count-valued response y:
sim_dat = simulate_nb_friedman(n = 100, p = 10)
y = sim_dat$y; X = sim_dat$X

# BART-STAR with log-transformation:
fit_log = bart_star(y = y, X = X, transformation = 'log',
                    save_y_hat = TRUE, nburn=1000, nskip=0)

# Fitted values
plot_fitted(y = sim_dat$Ey,
            post_y = fit_log$post.fitted.values,
            main = 'Fitted Values: BART-STAR-log')

# WAIC for BART-STAR-log:
fit_log$WAIC

# MCMC diagnostics:
plot(as.ts(fit_log$post.fitted.values[,1:10]))

# Posterior predictive check:
hist(apply(fit_log$post.pred, 1,
           function(x) mean(x==0)), main = 'Proportion of Zeros', xlab='');
abline(v = mean(y==0), lwd=4, col ='blue')

# BART-STAR with nonparametric transformation:
fit = bart_star(y = y, X = X,
                    transformation = 'np', save_y_hat = TRUE)

# Fitted values
plot_fitted(y = sim_dat$Ey,
            post_y = fit$post.fitted.values,
            main = 'Fitted Values: BART-STAR-np')

# WAIC for BART-STAR-np:
fit$WAIC

# MCMC diagnostics:
plot(as.ts(fit$post.fitted.values[,1:10]))

# Posterior predictive check:
hist(apply(fit$post.pred, 1,
           function(x) mean(x==0)), main = 'Proportion of Zeros', xlab='');
abline(v = mean(y==0), lwd=4, col ='blue')
```

blm_star *STAR Bayesian Linear Regression*

Description

Posterior inference for STAR linear model

Usage

```
blm_star(
  y,
  X,
  X_test = NULL,
  transformation = "np",
  y_max = Inf,
  prior = "gprior",
  use_MCMC = TRUE,
  nsave = 5000,
  nburn = 5000,
  nskip = 0,
  method_sigma = "mle",
  approx_Fz = FALSE,
  approx_Fy = FALSE,
  psi = NULL,
  compute_marg = FALSE
)
```

Arguments

y	n x 1 vector of observed counts
X	n x p matrix of predictors
X_test	n0 x p matrix of predictors for test data
transformation	transformation to use for the latent process; must be one of • "identity" (identity transformation) • "log" (log transformation) • "sqrt" (square root transformation) • "np" (nonparametric transformation estimated from empirical CDF) • "pois" (transformation for moment-matched marginal Poisson CDF) • "neg-bin" (transformation for moment-matched marginal Negative Binomial CDF) • "box-cox" (box-cox transformation with learned parameter)

- "ispline" (transformation is modeled as unknown, monotone function using I-splines)
- "bnp" (Bayesian nonparametric transformation using the Bayesian bootstrap)

y_max	a fixed and known upper bound for all observations; default is Inf
prior	prior to use for the latent linear regression; currently implemented options are "gprior", "horseshoe", and "ridge". Not all modeling options and transformations are available with the latter two priors.
use_MCMC	= TRUE,
nsave	number of MCMC iterations to save (or MC samples to draw if use_MCMC=FALSE)
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
method_sigma	method to estimate the latent data standard deviation in exact sampler; must be one of

- "mle" use the MLE from the STAR EM algorithm
- "mmle" use the marginal MLE (Note: slower!)

approx_Fz	logical; in BNP transformation, apply a (fast and stable) normal approximation for the marginal CDF of the latent data
approx_Fy	logical; in BNP transformation, approximate the marginal CDF of y using the empirical CDF
psi	prior variance (g-prior)
compute_marg	logical; if TRUE, compute and return the marginal likelihood (only available when using exact sampler, i.e. use_MCMC=FALSE)

Details

STAR defines a count-valued probability model by (1) specifying a Gaussian model for continuous *latent* data and (2) connecting the latent data to the observed data via a *transformation and rounding* operation. Here, the continuous latent data model is a linear regression.

There are several options for the transformation. First, the transformation can belong to the *Box-Cox* family, which includes the known transformations 'identity', 'log', and 'sqrt', as well as a version in which the Box-Cox parameter is inferred within the MCMC sampler ('box-cox'). Second, the transformation can be estimated (before model fitting) using the empirical distribution of the data y. Options in this case include the empirical cumulative distribution function (CDF), which is fully nonparametric ('np'), or the parametric alternatives based on Poisson ('pois') or Negative-Binomial ('neg-bin') distributions. For the parametric distributions, the parameters of the distribution are estimated using moments (means and variances) of y. The distribution-based transformations approximately preserve the mean and variance of the count data y on the latent data scale, which lends interpretability to the model parameters. Lastly, the transformation can be modeled using the Bayesian bootstrap ('bnp'), which is a Bayesian nonparametric model and incorporates the uncertainty about the transformation into posterior and predictive inference.

The Monte Carlo sampler (use_MCMC=FALSE) produces direct, discrete, and joint draws from the posterior distribution and the posterior predictive distribution of the linear regression model with a g-prior.

Value

a list with at least the following elements:

- `coefficients`: the posterior mean of the regression coefficients
- `post.beta`: posterior draws of the regression coefficients
- `post.pred`: draws from the posterior predictive distribution of `y`
- `post.log.like.point`: draws of the log-likelihood for each of the `n` observations
- `WAIC`: Widely-Applicable/Watanabe-Akaike Information Criterion
- `p_waic`: Effective number of parameters based on WAIC

If test points are passed in, then the list will also have `post.predtest`, which contains draws from the posterior predictive distribution at test points.

Other elements may be present depending on the choice of prior, transformation, and sampling approach.

Note

The 'bnp' transformation (without the `Fy` approximation) is slower than the other transformations because of the way the `TruncatedNormal` sampler must be updated as the lower and upper limits change (due to the sampling of `g`). Thus, computational improvements are likely available.

`confint.lmstar` *Compute asymptotic confidence intervals for STAR linear regression*

Description

For a linear regression model within the STAR framework, compute (asymptotic) confidence intervals for a regression coefficient of interest. Confidence intervals are computed by inverting the likelihood ratio test and profiling the log-likelihood.

Usage

```
## S3 method for class 'lmstar'
confint(object, parm, level = 0.95, ...)
```

Arguments

`object`	Object of class "lmstar" as output by `lm_star`
`parm`	a specification of which parameters are to be given confidence intervals, either a vector of numbers or a vector of names. If missing, all parameters are considered.
`level`	confidence level; default is 0.95
`...`	Ignored

Value

A matrix (or vector) with columns giving lower and upper confidence limits for each parameter. These will be labelled as (1-level)/2 and 1 - (1-level)/2 in

Examples

```
#Simulate data with count-valued response y:
sim_dat = simulate_nb_lm(n = 100, p = 2)
y = sim_dat$y; X = sim_dat$X

#Select a transformation:
transformation = 'np'

#Estimate model
fit = lm_star(y~X, transformation=transformation)

#Confidence interval for all parameters
confint(fit)
```

---

credBands *Compute Simultaneous Credible Bands*

Description

Compute (1-alpha)% credible BANDS for a function based on MCMC samples using Crainiceanu et al. (2007)

Usage

```
credBands(sampFuns, alpha = 0.05)
```

Arguments

sampFuns	Nsims x m matrix of Nsims MCMC samples and m points along the curve
alpha	confidence level

Value

m x 2 matrix of credible bands; the first column is the lower band, the second is the upper band

Note

The input needs not be curves: the simultaneous credible "bands" may be computed for vectors. The resulting credible intervals will provide joint coverage at the (1-alpha) level across all components of the vector.

ergMean *Compute the ergodic (running) mean.*

Description

Compute the ergodic (running) mean.

Usage

```
ergMean(x)
```

Arguments

x	vector for which to compute the running mean

Value

A vector y with each element defined by `y[i] = mean(x[1:i])`

Examples

```
# Compare:
ergMean(1:10)
mean(1:10)

# Running mean for iid N(5, 1) samples:
x = rnorm(n = 10^4, mean = 5, sd = 1)
plot(ergMean(x))
abline(h=5)
```

gbm_star *Fitting STAR Gradient Boosting Machines via EM algorithm*

Description

Compute the MLEs and log-likelihood for the Gradient Boosting Machines (GBM) STAR model. The STAR model requires a *transformation* and an *estimation function* for the conditional mean given observed data. The transformation can be known (e.g., log or sqrt) or unknown (Box-Cox or estimated nonparametrically) for greater flexibility. The estimator in this case is a GBM. Standard function calls including `fitted` and `residuals` apply.

Usage

```
gbm_star(
  y,
  X,
  X.test = NULL,
  transformation = "np",
  y_max = Inf,
  sd_init = 10,
  tol = 10^-10,
  max_iters = 1000,
  n.trees = 100,
  interaction.depth = 1,
  shrinkage = 0.1,
  bag.fraction = 1
)
```

Arguments

y	n x 1 vector of observed counts
X	n x p matrix of predictors
X.test	m x p matrix of out-of-sample predictors
transformation	transformation to use for the latent data; must be one of • "identity" (identity transformation) • "log" (log transformation) • "sqrt" (square root transformation) • "np" (nonparametric transformation estimated from empirical CDF) • "pois" (transformation for moment-matched marginal Poisson CDF) • "neg-bin" (transformation for moment-matched marginal Negative Binomial CDF) • "box-cox" (box-cox transformation with learned parameter)
y_max	a fixed and known upper bound for all observations; default is Inf
sd_init	add random noise for EM algorithm initialization scaled by sd_init times the Gaussian MLE standard deviation; default is 10
tol	tolerance for stopping the EM algorithm; default is 10^-10;
max_iters	maximum number of EM iterations before stopping; default is 1000
n.trees	Integer specifying the total number of trees to fit. This is equivalent to the number of iterations and the number of basis functions in the additive expansion. Default is 100.
interaction.depth	Integer specifying the maximum depth of each tree (i.e., the highest level of variable interactions allowed). A value of 1 implies an additive model, a value of 2 implies a model with up to 2-way interactions, etc. Default is 1.
shrinkage	a shrinkage parameter applied to each tree in the expansion. Also known as the learning rate or step-size reduction; 0.001 to 0.1 usually work, but a smaller learning rate typically requires more trees. Default is 0.1.

`bag.fraction` the fraction of the training set observations randomly selected to propose the next tree in the expansion. This introduces randomnesses into the model fit. If bag.fraction < 1 then running the same model twice will result in similar but different fits. Default is 1 (for a deterministic prediction).

Details

STAR defines a count-valued probability model by (1) specifying a Gaussian model for continuous *latent* data and (2) connecting the latent data to the observed data via a *transformation and rounding* operation. The Gaussian model in this case is a GBM.

Value

a list with the following elements:

- `fitted.values`: the fitted values at the MLEs (training)
- `fitted.values.test`: the fitted values at the MLEs (testing)
- `g.hat` a function containing the (known or estimated) transformation
- `sigma.hat` the MLE of the standard deviation
- `mu.hat` the MLE of the conditional mean (on the transformed scale)
- `z.hat` the estimated latent data (on the transformed scale) at the MLEs
- `residuals` the Dunn-Smyth residuals (randomized)
- `residuals_rep` the Dunn-Smyth residuals (randomized) for 10 replicates
- `logLik` the log-likelihood at the MLEs
- `logLik0` the log-likelihood at the MLEs for the *unrounded* initialization
- `lambda` the Box-Cox nonlinear parameter
- `gbmObj`: the object returned by gbm() at the MLEs
- and other parameters that (1) track the parameters across EM iterations and (2) record the model specifications

Note

Infinite latent data values may occur when the transformed Gaussian model is highly inadequate. In that case, the function returns the *indices* of the data points with infinite latent values, which are significant outliers under the model. Deletion of these indices and re-running the model is one option, but care must be taken to ensure that (i) it is appropriate to treat these observations as outliers and (ii) the model is adequate for the remaining data points.

References

Kowal, D. R., & Wu, B. (2021). Semiparametric count data regression for self-reported mental health. *Biometrics*. doi:10.1111/biom.13617

Examples

```
# Simulate data with count-valued response y:
sim_dat = simulate_nb_friedman(n = 100, p = 10)
y = sim_dat$y; X = sim_dat$X

# EM algorithm for STAR (using the log-link)
fit_em = gbm_star(y = y, X = X,
                  transformation = 'log')

# Evaluate convergence:
plot(fit_em$logLik_all, type='l', main = 'GBM-STAR-log', xlab = 'Iteration', ylab = 'log-lik')

# Fitted values:
y_hat = fitted(fit_em)
plot(y_hat, y);

# Residuals:
plot(residuals(fit_em))
qqnorm(residuals(fit_em)); qqline(residuals(fit_em))

# Log-likelihood at MLEs:
fit_em$logLik
```

---

genEM_star *Generalized EM estimation for STAR*

Description

Compute MLEs and log-likelihood for a generalized STAR model. The STAR model requires a *transformation* and an *estimation function* for the conditional mean given observed data. The transformation can be known (e.g., log or sqrt) or unknown (Box-Cox or estimated nonparametrically) for greater flexibility. The estimator can be any least squares estimator, including nonlinear models. Standard function calls including `coefficients()`, `fitted()`, and `residuals()` apply.

Usage

```
genEM_star(
  y,
  estimator,
  transformation = "np",
  y_max = Inf,
  sd_init = 10,
  tol = 10^-10,
  max_iters = 1000
)
```

Arguments

`y`	`n x 1` vector of observed counts
`estimator`	a function that inputs data `y` and outputs a list with two elements: 1. The fitted values `fitted.values` 2. The parameter estimates `coefficients`
`transformation`	transformation to use for the latent data; must be one of • "identity" (identity transformation) • "log" (log transformation) • "sqrt" (square root transformation) • "np" (nonparametric transformation estimated from empirical CDF) • "pois" (transformation for moment-matched marginal Poisson CDF) • "neg-bin" (transformation for moment-matched marginal Negative Binomial CDF) • "box-cox" (box-cox transformation with learned parameter)
`y_max`	a fixed and known upper bound for all observations; default is `Inf`
`sd_init`	add random noise for EM algorithm initialization scaled by `sd_init` times the Gaussian MLE standard deviation; default is 10
`tol`	tolerance for stopping the EM algorithm; default is 10^-10;
`max_iters`	maximum number of EM iterations before stopping; default is 1000

Details

STAR defines a count-valued probability model by (1) specifying a Gaussian model for continuous *latent* data and (2) connecting the latent data to the observed data via a *transformation and rounding* operation.

The expectation-maximization (EM) algorithm is used to produce maximum likelihood estimators (MLEs) for the parameters defined in the `estimator` function, such as linear regression coefficients, which define the Gaussian model for the continuous latent data. Fitted values (point predictions), residuals, and log-likelihood values are also available. Inference for the estimators proceeds via classical maximum likelihood. Initialization of the EM algorithm can be randomized to monitor convergence. However, the log-likelihood is concave for all transformations (except 'box-cox'), so global convergence is guaranteed.

There are several options for the transformation. First, the transformation can belong to the *Box-Cox* family, which includes the known transformations 'identity', 'log', and 'sqrt', as well as a version in which the Box-Cox parameter is estimated within the EM algorithm ('box-cox'). Second, the transformation can be estimated (before model fitting) using the empirical distribution of the data `y`. Options in this case include the empirical cumulative distribution function (CDF), which is fully nonparametric ('np'), or the parametric alternatives based on Poisson ('pois') or Negative-Binomial ('neg-bin') distributions. For the parametric distributions, the parameters of the distribution are estimated using moments (means and variances) of `y`.

Value

a list with the following elements:

- `coefficients` the MLEs of the coefficients
- `fitted.values` the fitted values at the MLEs
- `g.hat` a function containing the (known or estimated) transformation
- `sigma.hat` the MLE of the standard deviation
- `mu.hat` the MLE of the conditional mean (on the transformed scale)
- `z.hat` the estimated latent data (on the transformed scale) at the MLEs
- `residuals` the Dunn-Smyth residuals (randomized)
- `residuals_rep` the Dunn-Smyth residuals (randomized) for 10 replicates
- `logLik` the log-likelihood at the MLEs
- `logLik0` the log-likelihood at the MLEs for the *unrounded* initialization
- `lambda` the Box-Cox nonlinear parameter
- and other parameters that (1) track the parameters across EM iterations and (2) record the model specifications

Note

Infinite latent data values may occur when the transformed Gaussian model is highly inadequate. In that case, the function returns the *indices* of the data points with infinite latent values, which are significant outliers under the model. Deletion of these indices and re-running the model is one option, but care must be taken to ensure that (i) it is appropriate to treat these observations as outliers and (ii) the model is adequate for the remaining data points.

References

Kowal, D. R., & Wu, B. (2021). Semiparametric count data regression for self-reported mental health. *Biometrics*. doi:10.1111/biom.13617

Examples

```
# Simulate data with count-valued response y:
sim_dat = simulate_nb_friedman(n = 100, p = 5)
y = sim_dat$y; X = sim_dat$X

# Select a transformation:
transformation = 'np'

# Example using GAM as underlying estimator (for illustration purposes only)
if(require("mgcv")){
  fit_em = genEM_star(y = y,
                      estimator = function(y) gam(y ~ s(X1)+s(X2),
                      data=data.frame(y,X)),
                      transformation = transformation)
}
```

```
# Fitted coefficients:
coef(fit_em)

# Fitted values:
y_hat = fitted(fit_em)
plot(y_hat, y);

# Log-likelihood at MLEs:
fit_em$logLik
```

---

genMCMC_star *Generalized MCMC Algorithm for STAR*

Description

Run the MCMC algorithm for STAR given

1. a function to initialize model parameters; and
2. a function to sample (i.e., update) model parameters.

The transformation can be known (e.g., log or sqrt) or unknown (Box-Cox or estimated nonparametrically) for greater flexibility.

Usage

```
genMCMC_star(
  y,
  sample_params,
  init_params,
  transformation = "np",
  y_max = Inf,
  nsave = 5000,
  nburn = 5000,
  nskip = 0,
  save_y_hat = FALSE,
  verbose = TRUE
)
```

Arguments

y	n x 1 vector of observed counts
sample_params	a function that inputs data y and a named list params containing 1. mu: the n x 1 vector of conditional means (fitted values) 2. sigma: the conditional standard deviation 3. coefficients: a named list of parameters that determine mu

	and outputs an updated list `params` of samples from the full conditional posterior distribution of `coefficients` and `sigma` (and updates `mu`)
`init_params`	an initializing function that inputs data `y` and initializes the named list `params` of `mu`, `sigma`, and `coefficients`
`transformation`	transformation to use for the latent data; must be one of • "identity" (identity transformation) • "log" (log transformation) • "sqrt" (square root transformation) • "np" (nonparametric transformation estimated from empirical CDF) • "pois" (transformation for moment-matched marginal Poisson CDF) • "neg-bin" (transformation for moment-matched marginal Negative Binomial CDF) • "box-cox" (box-cox transformation with learned parameter)
`y_max`	a fixed and known upper bound for all observations; default is `Inf`
`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
`save_y_hat`	logical; if TRUE, compute and save the posterior draws of the expected counts, E(y), which may be slow to compute
`verbose`	logical; if TRUE, print time remaining

Details

STAR defines a count-valued probability model by (1) specifying a Gaussian model for continuous *latent* data and (2) connecting the latent data to the observed data via a *transformation and rounding* operation.

Posterior and predictive inference is obtained via a Gibbs sampler that combines (i) a latent data augmentation step (like in probit regression) and (ii) an existing sampler for a continuous data model.

There are several options for the transformation. First, the transformation can belong to the *Box-Cox* family, which includes the known transformations 'identity', 'log', and 'sqrt', as well as a version in which the Box-Cox parameter is inferred within the MCMC sampler ('box-cox'). Second, the transformation can be estimated (before model fitting) using the empirical distribution of the data `y`. Options in this case include the empirical cumulative distribution function (CDF), which is fully nonparametric ('np'), or the parametric alternatives based on Poisson ('pois') or Negative-Binomial ('neg-bin') distributions. For the parametric distributions, the parameters of the distribution are estimated using moments (means and variances) of `y`.

Value

a list with at least the following elements:

- `post.pred`: draws from the posterior predictive distribution of `y`
- `post.sigma`: draws from the posterior distribution of `sigma`

- post.log.like.point: draws of the log-likelihood for each of the n observations
- WAIC: Widely-Applicable/Watanabe-Akaike Information Criterion
- p_waic: Effective number of parameters based on WAIC
- post.lambda: draws from the posterior distribution of lambda (NULL unless transformation='box-cox')
- fitted.values: the posterior mean of the conditional expectation of the counts y (NULL if save_y_hat=FALSE)
- post.fitted.values: posterior draws of the conditional mean of the counts y (NULL if save_y_hat=FALSE)

If the coefficients list from init_params and sample_params contains a named element beta, e.g. for linear regression, then the function output contains

- coefficients: the posterior mean of the beta coefficients
- post.beta: draws from the posterior distribution of beta
- post.othercoefs: draws from the posterior distribution of any other sampled coefficients, e.g. variance terms

If no beta exists in the parameter coefficients, then the output list just contains

- coefficients: the posterior mean of all coefficients
- post.beta: draws from the posterior distribution of all coefficients

Additionally, if init_params and sample_params have output mu_test, then the sampler will output post.predtest, which contains draws from the posterior predictive distribution at test points.

Examples

```
# Simulate data with count-valued response y:
sim_dat = simulate_nb_lm(n = 100, p = 5)
y = sim_dat$y; X = sim_dat$X

# STAR: log-transformation:
fit_log = genMCMC_star(y = y,
                       sample_params = function(y, params) sample_lm_gprior(y, X, params),
                          init_params = function(y) init_lm_gprior(y, X),
                          transformation = 'log')
# Posterior mean of each coefficient:
coef(fit_log)

# WAIC for STAR-log:
fit_log$WAIC

# MCMC diagnostics:
plot(as.ts(fit_log$post.beta[,1:3]))

# Posterior predictive check:
hist(apply(fit_log$post.pred, 1,
           function(x) mean(x==0)), main = 'Proportion of Zeros', xlab='');
abline(v = mean(y==0), lwd=4, col ='blue')
```

getEffSize *Summarize of effective sample size*

Description

Compute the summary statistics for the effective sample size (ESS) across posterior samples for possibly many variables

Usage

```
getEffSize(postX)
```

Arguments

postX	An array of arbitrary dimension (nsims x ... x ...), where nsims is the number of posterior samples

Value

Table of summary statistics using the function summary().

Examples

```
# ESS for iid simulations:
rand_iid = rnorm(n = 10^4)
getEffSize(rand_iid)

# ESS for several AR(1) simulations with coefficients 0.1, 0.2,...,0.9:
rand_ar1 = sapply(seq(0.1, 0.9, by = 0.1), function(x) arima.sim(n = 10^4, list(ar = x)))
getEffSize(rand_ar1)
```

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_bnp *Bayesian bootstrap-based transformation*

Description

Compute one posterior draw from the smoothed transformation implied by (separate) Bayesian bootstrap models for the CDFs of y and X.

Usage

```
g_bnp(
  y,
  xtSigmax = rep(0, length(y)),
  zgrid = NULL,
  sigma_epsilon = 1,
  approx_Fz = FALSE
)
```

Arguments

y	n x 1 vector of observed counts
xtSigmax	n x 1 vector of t(X_i) Sigma_theta X_i, where Sigma_theta is the prior variance
zgrid	optional vector of grid points for evaluating the CDF of z (Fz)

sigma_epsilon	latent standard deviation
approx_Fz	logical; if TRUE, use a normal approximation for Fz, the marginal CDF of the latent z, which is faster and more stable

Value

A smooth monotone function which can be used for evaluations of the transformation at each posterior draw.

Examples

```
# Sample some data:
y = rpois(n = 200, lambda = 5)
# Compute 200 draws of g on a grid:
t = seq(0, max(y), length.out = 100) # grid
g_post = t(sapply(1:500, function(s) g_bnp(y, approx_Fz = TRUE)(t)))
# Plot together:
plot(t, t, ylim = range(g_post), type='n', ylab = 'g(t)', main = 'Bayesian bootstrap posterior: g')
apply(g_post, 1, function(g) lines(t, g, col='gray'))
# And the posterior mean of g:
lines(t, colMeans(g_post), lwd=3)
```

---

g_cdf *Cumulative distribution function (CDF)-based transformation*

Description

Compute a CDF-based transformation using the observed count data. The CDF can be estimated nonparametrically or parametrically based on the Poisson or Negative Binomial distributions. In the parametric case, the parameters are determined based on the moments of y. Note that this is a fixed quantity and does not come with uncertainty quantification.

Usage

```
g_cdf(y, distribution = "np")
```

Arguments

y	n x 1 vector of observed counts
distribution	the distribution used for the CDF; must be one of • "np" (empirical CDF) • "pois" (moment-matched marginal Poisson CDF) • "neg-bin" (moment-matched marginal Negative Binomial CDF)

Value

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

Examples

```
# Sample some data:
y = rpois(n = 500, lambda = 5)

# Empirical CDF version:
g_np = g_cdf(y, distribution = 'np')

# Poisson version:
g_pois = g_cdf(y, distribution = 'pois')

# Negative binomial version:
g_negbin = g_cdf(y, distribution = 'neg-bin')

# Plot together:
t = 1:max(y) # grid
plot(t, g_np(t), type='l')
lines(t, g_pois(t), lty = 2)
lines(t, g_negbin(t), lty = 3)
```

---

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.

Examples

```
# Sample some data:
y = rpois(n = 500, lambda = 5)

# Empirical CDF transformation:
g_np = g_cdf(y, distribution = 'np')

# Grid for approximation:
```

```
t_grid = seq(1, max(y), length.out = 100)

# Approximate inverse:
g_inv = g_inv_approx(g = g_np, t_grid = t_grid)

# Check the approximation:
plot(t_grid, g_inv(g_np(t_grid)), type='p')
lines(t_grid, t_grid)
```

`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

`init_lm_gprior` *Initialize linear regression parameters assuming a g-prior*

Description

Initialize the parameters for a linear regression model assuming a g-prior for the coefficients.

Usage

```
init_lm_gprior(y, X, X_test = NULL)
```

Arguments

`y`	`n` x `1` vector of data
`X`	`n` x `p` matrix of predictors
`X_test`	`n0` x `p` matrix of predictors at test points (default is NULL)

Value

a named list `params` containing at least

1. `mu`: vector of conditional means (fitted values)
2. `sigma`: the conditional standard deviation
3. `coefficients`: a named list of parameters that determine `mu`

Additionally, if X_test is not NULL, then the list includes an element `mu_test`, the vector of conditional means at the test points

Note

The parameters in `coefficients` are:

- `beta`: the `p` x `1` vector of regression coefficients components of `beta`

Examples

```
# Simulate data for illustration:
sim_dat = simulate_nb_lm(n = 100, p = 5)
y = sim_dat$y; X = sim_dat$X

# Initialize:
params = init_lm_gprior(y = y, X = X)
names(params)
names(params$coefficients)
```

`lm_star` *Fitting frequentist STAR linear model via EM algorithm*

Description

Compute the MLEs and log-likelihood for the STAR linear model. The regression coefficients are estimated using least squares within an EM algorithm.

Usage

```
lm_star(
  formula,
  data = NULL,
  transformation = "np",
  y_max = Inf,
  sd_init = 10,
  tol = 10^-10,
  max_iters = 1000
)
```

Arguments

`formula`	an object of class "`formula`" (see `lm` for details on model specification)
`data`	an optional data frame, list or environment (or object coercible by as.data.frame to a data frame) containing the variables in the model; like `lm`, if not found in data, the variables are taken from `environment(formula)`
`transformation`	transformation to use for the latent data; must be one of • "identity" (identity transformation) • "log" (log transformation) • "sqrt" (square root transformation) • "np" (nonparametric transformation estimated from empirical CDF) • "pois" (transformation for moment-matched marginal Poisson CDF) • "neg-bin" (transformation for moment-matched marginal Negative Binomial CDF) • "box-cox" (box-cox transformation with learned parameter)
`y_max`	a fixed and known upper bound for all observations; default is `Inf`
`sd_init`	add random noise for EM algorithm initialization scaled by `sd_init` times the Gaussian MLE standard deviation; default is 10
`tol`	tolerance for stopping the EM algorithm; default is 10^-10;
`max_iters`	maximum number of EM iterations before stopping; default is 1000

Details

Standard function calls including `coefficients`, `fitted`, and `residuals` apply. Fitted values are the expectation at the MLEs, and as such are not necessarily count-valued.

Value

an object of `class` "lmstar", which is a list with the following elements:

- `coefficients` the MLEs of the coefficients
- `fitted.values` the fitted values at the MLEs
- `g.hat` a function containing the (known or estimated) transformation
- `ginv.hat` a function containing the inverse of the transformation
- `sigma.hat` the MLE of the standard deviation
- `mu.hat` the MLE of the conditional mean (on the transformed scale)
- `z.hat` the estimated latent data (on the transformed scale) at the MLEs
- `residuals` the Dunn-Smyth residuals (randomized)
- `residuals_rep` the Dunn-Smyth residuals (randomized) for 10 replicates
- `logLik` the log-likelihood at the MLEs
- `logLik0` the log-likelihood at the MLEs for the *unrounded* initialization
- `lambda` the Box-Cox nonlinear parameter
- and other parameters that (1) track the parameters across EM iterations and (2) record the model specifications

Note

Infinite latent data values may occur when the transformed Gaussian model is highly inadequate. In that case, the function returns the *indices* of the data points with infinite latent values, which are significant outliers under the model. Deletion of these indices and re-running the model is one option, but care must be taken to ensure that (i) it is appropriate to treat these observations as outliers and (ii) the model is adequate for the remaining data points.

References

Kowal, D. R., & Wu, B. (2021). Semiparametric count data regression for self-reported mental health. *Biometrics*. doi:10.1111/biom.13617

Examples

```
# Simulate data with count-valued response y:
sim_dat = simulate_nb_lm(n = 100, p = 3)
y = sim_dat$y; X = sim_dat$X

# Fit model
fit_em = lm_star(y~X)

# Fitted coefficients:
coef(fit_em)
# Fitted values:
y_hat = fitted(fit_em)
plot(y_hat, y);
```

```
# Residuals:
plot(residuals(fit_em))
qqnorm(residuals(fit_em)); qqline(residuals(fit_em))
```

`plot_coef` *Plot the estimated regression coefficients and credible intervals*

Description

Plot the estimated regression coefficients and credible intervals for the linear effects in up to two models.

Usage

```
plot_coef(
  post_coefficients_1,
  post_coefficients_2 = NULL,
  alpha = 0.05,
  labels = NULL
)
```

Arguments

`post_coefficients_1`	`Nsims` x `p` matrix of simulations from the posterior distribution of the `p` coefficients, where `Nsims` is the number of simulations
`post_coefficients_2`	`Nsims` x `p` matrix of simulations from the posterior distribution of the `p` coefficients from another model
`alpha`	confidence level for the credible intervals
`labels`	`p` dimensional string of labels for the coefficient names

Value

A plot of regression coefficients and credible intervals for 1-2 models

plot_fitted *Plot the fitted values and the data*

Description

Plot the fitted values, plus pointwise credible intervals, against the data. For simulations, one may use the true values in place of the data.

Usage

```
plot_fitted(y, post_y, y_hat = NULL, alpha = 0.05, ...)
```

Arguments

y	n x 1 vector of data
post_y	Nsims x n matrix of simulated fitted values, where Nsims is the number of simulations
y_hat	n x 1 vector of fitted values; if NULL, use the pointwise sample mean colMeans(post_y)
alpha	confidence level for the credible intervals
...	other arguments for plotting

Value

A plot with the fitted values and the credible intervals against the data

plot_pmf *Plot the empirical and model-based probability mass functions*

Description

Plot the empirical probability mass function, i.e., the proportion of data values y that equal j for each j=0,1,..., together with the model-based estimate of the probability mass function based on the posterior predictive distribution.

Usage

```
plot_pmf(y, post.pred, error.bars = FALSE, alpha = 0.05)
```

Arguments

y	n x 1 vector of data
post.pred	nsave draws from the posterior predictive distribution of y
error.bars	logical; if TRUE, include errors bars on the model-based PMF
alpha	confidence level for the credible intervals

Value

A plot of the empirical PMF of y along with a PMF estimate from the model posterior predictive distribution

---

`predict.lmstar` *Predict method for response in STAR linear model*

Description

Outputs predicted values based on an lmstar fit and optionally prediction intervals based on the the (plug-in) predictive distribution for the STAR linear model

Usage

```
## S3 method for class 'lmstar'
predict(object, newdata = NULL, interval = FALSE, level = 0.95, N = 1000, ...)
```

Arguments

`object`	Object of class "lmstar" as output by `lm_star`
`newdata`	An optional matrix/data frame in which to look for variables with which to predict. If omitted, the fitted values are used.
`interval`	logical; whether or not to include prediction intervals (default FALSE)
`level`	Level for prediction intervals
`N`	number of Monte Carlo samples from the posterior predictive distribution used to approximate intervals; default is 1000
`...`	Ignored

Details

If interval=TRUE, then `predict.lmstar` uses a Monte Carlo approach to estimating the (plug-in) predictive distribution for the STAR linear model. The algorithm iteratively samples (i) the latent data given the observed data, (ii) the latent predictive data given the latent data from (i), and (iii) (inverse) transforms and rounds the latent predictive data to obtain a draw from the integer-valued predictive distribution.

The appropriate quantiles of these Monte Carlo draws are computed and reported as the prediction interval.

Value

Either a a vector of predictions (if interval=FALSE) or a matrix of predictions and bounds with column names fit, lwr, and upr

Note

The “plug-in" predictive distribution is a crude approximation. Better approaches are available using the Bayesian models, e.g. blm_star, which provide samples from the posterior predictive distribution.

For highly skewed responses, prediction intervals especially at lower levels may not include the predicted value itself, since the mean is often much larger than the median.

Examples

```
# Simulate data with count-valued response y:
x = seq(0, 1, length.out = 100)
y = rpois(n = length(x), lambda = exp(1.5 + 5*(x -.5)^2))

# Estimate model--assume a quadratic effect (better for illustration purposes)
fit = lm_star(y~x+I(x^2), transformation = 'sqrt')

#Compute the predictive draws for the test points (same as observed points here)
#Also compute intervals using plug-in predictive distribution
y_pred = predict(fit, interval=TRUE)

# Plot the results
plot(x, y, ylim = range(y, y_pred), main = 'STAR: Predictions and 95% PI')
lines(x,y_pred[,"fit"], col='black', type='s', lwd=4)
lines(x, y_pred[,"lwr"], col='darkgray', type='s', lwd=4)
lines(x, y_pred[,"upr"], col='darkgray', type='s', lwd=4)
```

---

pvals *Compute coefficient p-values for STAR linear regression using likelihood ratio test*

Description

For a linear regression model within the STAR framework, compute p-values for regression coefficients using a likelihood ratio test. It also computes a p-value for excluding all predictors, akin to a (partial) F test.

Usage

```
pvals(object)
```

Arguments

object	Object of class "lmstar" as output by lm_star

Value

a list of p+1 p-values, one for each predictor as well as the joint p-value excluding all predictors

Examples

```
# Simulate data with count-valued response y:
sim_dat = simulate_nb_lm(n = 100, p = 2)
y = sim_dat$y; X = sim_dat$X

# Select a transformation:
transformation = 'np'

#Estimate model
fit = lm_star(y~X, transformation = transformation)

#Compute p-values
pvals(fit)
```

`randomForest_star` *Fit Random Forest STAR with EM algorithm*

Description

Compute the MLEs and log-likelihood for the Random Forest STAR model. The STAR model requires a *transformation* and an *estimation function* for the conditional mean given observed data. The transformation can be known (e.g., log or sqrt) or unknown (Box-Cox or estimated nonparametrically) for greater flexibility. The estimator in this case is a random forest. Standard function calls including `fitted` and `residuals` apply.

Usage

```
randomForest_star(
  y,
  X,
  X.test = NULL,
  transformation = "np",
  y_max = Inf,
  sd_init = 10,
  tol = 10^-10,
  max_iters = 1000,
  ntree = 500,
  mtry = max(floor(ncol(X)/3), 1),
  nodesize = 5
)
```

Arguments

`y`	n x 1 vector of observed counts
`X`	n x p matrix of predictors
`X.test`	m x p matrix of out-of-sample predictors

`transformation` transformation to use for the latent data; must be one of

- "identity" (identity transformation)
- "log" (log transformation)
- "sqrt" (square root transformation)
- "np" (nonparametric transformation estimated from empirical CDF)
- "pois" (transformation for moment-matched marginal Poisson CDF)
- "neg-bin" (transformation for moment-matched marginal Negative Binomial CDF)
- "box-cox" (box-cox transformation with learned parameter)

`y_max` a fixed and known upper bound for all observations; default is `Inf`

`sd_init` add random noise for EM algorithm initialization scaled by `sd_init` times the Gaussian MLE standard deviation; default is 10

`tol` tolerance for stopping the EM algorithm; default is 10^-10;

`max_iters` maximum number of EM iterations before stopping; default is 1000

`ntree` Number of trees to grow. This should not be set to too small a number, to ensure that every input row gets predicted at least a few times. Default is 500.

`mtry` Number of variables randomly sampled as candidates at each split. Default is p/3.

`nodesize` Minimum size of terminal nodes. Setting this number larger causes smaller trees to be grown (and thus take less time). Default is 5.

Details

STAR defines a count-valued probability model by (1) specifying a Gaussian model for continuous *latent* data and (2) connecting the latent data to the observed data via a *transformation and rounding* operation.

The expectation-maximization (EM) algorithm is used to produce maximum likelihood estimators (MLEs) for the parameters defined in the The fitted values are computed using out-of-bag samples. As a result, the log-likelihood is based on out-of-bag prediction, and it is similarly straightforward to compute out-of-bag squared and absolute errors.

Value

a list with the following elements:

- `fitted.values`: the fitted values at the MLEs based on out-of-bag samples (training)
- `fitted.values.test`: the fitted values at the MLEs (testing)
- `g.hat` a function containing the (known or estimated) transformation
- `sigma.hat` the MLE of the standard deviation
- `mu.hat` the MLE of the conditional mean (on the transformed scale)
- `z.hat` the estimated latent data (on the transformed scale) at the MLEs
- `residuals` the Dunn-Smyth residuals (randomized)
- `residuals_rep` the Dunn-Smyth residuals (randomized) for 10 replicates

- `logLik` the log-likelihood at the MLEs
- `logLik0` the log-likelihood at the MLEs for the *unrounded* initialization
- `lambda` the Box-Cox nonlinear parameter
- `rfObj`: the object returned by randomForest() at the MLEs
- and other parameters that (1) track the parameters across EM iterations and (2) record the model specifications

Note

Since the random forest produces random predictions, the EM algorithm will never converge exactly.

Infinite latent data values may occur when the transformed Gaussian model is highly inadequate. In that case, the function returns the *indices* of the data points with infinite latent values, which are significant outliers under the model. Deletion of these indices and re-running the model is one option, but care must be taken to ensure that (i) it is appropriate to treat these observations as outliers and (ii) the model is adequate for the remaining data points.

References

Kowal, D. R., & Wu, B. (2021). Semiparametric count data regression for self-reported mental health. *Biometrics*. doi:10.1111/biom.13617

Examples

```
# Simulate data with count-valued response y:
sim_dat = simulate_nb_friedman(n = 100, p = 10)
y = sim_dat$y; X = sim_dat$X

# EM algorithm for STAR (using the log-link)
fit_em = randomForest_star(y = y, X = X,
                 transformation = 'log',
                 max_iters = 100)

# Fitted values (out-of-bag)
y_hat = fitted(fit_em)
plot(y_hat, y);

# Residuals:
plot(residuals(fit_em))
qqnorm(residuals(fit_em)); qqline(residuals(fit_em))

# Log-likelihood at MLEs (out-of-bag):
fit_em$logLik
```

roaches *Data on the efficacy of a pest management system at reducing the number of roaches in urban apartments.*

Description

Data on the efficacy of a pest management system at reducing the number of roaches in urban apartments.

Usage

```
roaches
```

Format

'roaches' A data frame with 262 obs. of 6 variables:

y Number of roaches caught

roach1 Pretreatment number of roaches

treatment Treatment indicator

senior Indicator for only elderly residents in building

exposure2 Number of days for which the roach traps were used

Source

Gelman and Hill (2007); package 'rstanarm'

round_floor *Rounding function*

Description

Define the rounding operator associated with the floor function. The function also returns zero whenever the input is negative and caps the value at `y_max`, where `y_max` is a known upper bound on the data `y` (if specified).

Usage

```
round_floor(z, y_max = Inf)
```

Arguments

`z`	the real-valued input(s)
`y_max`	a fixed and known upper bound for all observations; default is `Inf`

Value

The count-valued output(s) from the rounding function.

Examples

```
# Floor function:
round_floor(1.5)
round_floor(0.5)

# Special treatmeant of negative numbers:
round_floor(-1)
```

`sample_lm_gprior` *Sample the linear regression parameters assuming a g-prior*

Description

Sample the parameters for a linear regression model assuming a g-prior for the coefficients.

Usage

```
sample_lm_gprior(y, X, params, psi = NULL, XtX = NULL, X_test = NULL)
```

Arguments

`y`	n x 1 vector of data
`X`	n x p matrix of predictors
`params`	the named list of parameters containing 1. `mu`: vector of conditional means (fitted values) 2. `sigma`: the conditional standard deviation 3. `coefficients`: a named list of parameters that determine `mu`
`psi`	the prior variance for the g-prior
`XtX`	the p x p matrix of `crossprod(X)` (one-time cost); if NULL, compute within the function
`X_test`	matrix of predictors at test points (default is NULL)

Value

The updated named list `params` with draws from the full conditional distributions of `sigma` and `coefficients` (along with updated `mu` and `mu_test` if applicable).

Note

The parameters in `coefficients` are:

- `beta`: the p x 1 vector of regression coefficients components of `beta`

Examples

```
# Simulate data for illustration:
sim_dat = simulate_nb_lm(n = 100, p = 5)
y = sim_dat$y; X = sim_dat$X
# Initialize:
params = init_lm_gprior(y = y, X = X)
# Sample:
params = sample_lm_gprior(y = y, X = X, params = params)
names(params)
names(params$coefficients)
```

---

simBaS *Compute Simultaneous Band Scores (SimBaS)*

Description

Compute simultaneous band scores (SimBaS) from Meyer et al. (2015, Biometrics). SimBaS uses MC(MC) simulations of a function of interest to compute the minimum alpha such that the joint credible bands at the alpha level do not include zero. This quantity is computed for each grid point (or observation point) in the domain of the function.

Usage

```
simBaS(sampFuns)
```

Arguments

sampFuns	Nsims x m matrix of Nsims MCMC samples and m points along the curve

Value

m x 1 vector of simBaS

Note

The input needs not be curves: the simBaS may be computed for vectors to achieve a multiplicity adjustment.

The minimum of the returned value, PsimBaS_t, over the domain t is the Global Bayesian P-Value (GBPV) for testing whether the function is zero everywhere.

`simulate_nb_friedman` *Simulate count data from Friedman's nonlinear regression*

Description

Simulate data from a negative-binomial distribution with nonlinear mean function.

Usage

```
simulate_nb_friedman(
  n = 100,
  p = 10,
  r_nb = 1,
  b_int = log(1.5),
  b_sig = log(5),
  sigma_true = sqrt(2 * log(1)),
  seed = NULL
)
```

Arguments

`n`	number of observations
`p`	number of predictors
`r_nb`	the dispersion parameter of the Negative Binomial dispersion; smaller values imply greater overdispersion, while larger values approximate the Poisson distribution.
`b_int`	intercept; default is log(1.5).
`b_sig`	regression coefficients for true signals; default is log(5.0).
`sigma_true`	standard deviation of the Gaussian innovation; default is zero.
`seed`	optional integer to set the seed for reproducible simulation; default is NULL which results in a different dataset after each run

Details

The log-expected counts are modeled using the Friedman (1991) nonlinear function with interactions, possibly with additional Gaussian noise (on the log-scale). We assume that half of the predictors are associated with the response, i.e., true signals. For sufficiently large dispersion parameter `r_nb`, the distribution will approximate a Poisson distribution. Here, the predictor variables are simulated from independent uniform distributions.

Value

A named list with the simulated count response `y`, the simulated design matrix `X`, and the true expected counts `Ey`.

Note

Specifying `sigma_true = sqrt(2*log(1 + a))` implies that the expected counts are inflated by `100*a`% (relative to `exp(X*beta)`), in addition to providing additional overdispersion.

Examples

```
# Simulate and plot the count data:
sim_dat = simulate_nb_friedman(n = 100, p = 10);
plot(sim_dat$y)
```

`simulate_nb_lm` *Simulate count data from a linear regression*

Description

Simulate data from a negative-binomial distribution with linear mean function.

Usage

```
simulate_nb_lm(
  n = 100,
  p = 10,
  r_nb = 1,
  b_int = log(1.5),
  b_sig = log(2),
  sigma_true = sqrt(2 * log(1)),
  ar1 = 0,
  intercept = FALSE,
  seed = NULL
)
```

Arguments

`n`	number of observations
`p`	number of predictors (including the intercept)
`r_nb`	the dispersion parameter of the Negative Binomial dispersion; smaller values imply greater overdispersion, while larger values approximate the Poisson distribution.
`b_int`	intercept; default is log(1.5), which implies the expected count is 1.5 when all predictors are zero
`b_sig`	regression coefficients for true signals; default is log(2.0), which implies a twofold increase in the expected counts for a one unit increase in x
`sigma_true`	standard deviation of the Gaussian innovation; default is zero.
`ar1`	the autoregressive coefficient among the columns of the X matrix; default is zero.

intercept	a Boolean indicating whether an intercept column should be included in the returned design matrix; default is FALSE
seed	optional integer to set the seed for reproducible simulation; default is NULL which results in a different dataset after each run

Details

The log-expected counts are modeled as a linear function of covariates, possibly with additional Gaussian noise (on the log-scale). We assume that half of the predictors are associated with the response, i.e., true signals. For sufficiently large dispersion parameter r_nb, the distribution will approximate a Poisson distribution. Here, the predictor variables are simulated from independent standard normal distributions.

Value

A named list with the simulated count response y, the simulated design matrix X (possibly including the intercept), the true expected counts Ey, and the true regression coefficients beta_true.

Note

Specifying sigma_true = sqrt(2*log(1 + a)) implies that the expected counts are inflated by 100*a% (relative to exp(X*beta)), in addition to providing additional overdispersion.

Examples

```
# Simulate and plot the count data:
sim_dat = simulate_nb_lm(n = 100, p = 10);
plot(sim_dat$y)
```

---

spline_star *Estimation for Bayesian STAR spline regression*

Description

Compute samples from the predictive distributions of a STAR spline regression model using either a Gibbs sampling approach or exact Monte Carlo sampling (default is Gibbs sampling which scales better for large n)

Usage

```
spline_star(
  y,
  tau = NULL,
  transformation = "np",
  y_max = Inf,
  psi = NULL,
  approx_Fz = FALSE,
```

```
  approx_Fy = FALSE,
  nsave = 1000,
  use_MCMC = TRUE,
  nburn = 1000,
  nskip = 0,
  verbose = TRUE,
  method_sigma = "mle"
)
```

Arguments

y	n x 1 vector of observed counts
tau	n x 1 vector of observation points; if NULL, assume equally-spaced on [0,1]
transformation	transformation to use for the latent data; must be one of • "identity" (identity transformation) • "log" (log transformation) • "sqrt" (square root transformation) • "bnp" (Bayesian nonparametric transformation using the Bayesian bootstrap) • "np" (nonparametric transformation estimated from empirical CDF) • "pois" (transformation for moment-matched marginal Poisson CDF) • "neg-bin" (transformation for moment-matched marginal Negative Binomial CDF)
y_max	a fixed and known upper bound for all observations; default is Inf
psi	prior variance (1/smoothing parameter); if NULL, update in MCMC
approx_Fz	logical; in BNP transformation, apply a (fast and stable) normal approximation for the marginal CDF of the latent data
approx_Fy	logical; in BNP transformation, approximate the marginal CDF of y using the empirical CDF
nsave	number of MCMC iterations to save (or number of Monte Carlo simulations)
use_MCMC	logical; whether to run Gibbs sampler or Monte Carlo (default is TRUE)
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
method_sigma	method to estimate the latent data standard deviation (only applicable if use_MCMC=FALSE); must be one of • "mle" use the MLE from the STAR EM algorithm (default) • "mmle" use the marginal MLE (Note: slower!)

Details

STAR defines a count-valued probability model by (1) specifying a Gaussian model for continuous *latent* data and (2) connecting the latent data to the observed data via a *transformation and rounding* operation. Here, the continuous latent data model is a spline regression.

There are several options for the transformation. First, the transformation can belong to the *Box-Cox* family, which includes the known transformations 'identity', 'log', and 'sqrt'. Second, the transformation can be estimated (before model fitting) using the empirical distribution of the data `y`. Options in this case include the empirical cumulative distribution function (CDF), which is fully nonparametric ('np'), or the parametric alternatives based on Poisson ('pois') or Negative-Binomial ('neg-bin') distributions. For the parametric distributions, the parameters of the distribution are estimated using moments (means and variances) of `y`. The distribution-based transformations approximately preserve the mean and variance of the count data `y` on the latent data scale, which lends interpretability to the model parameters. Lastly, the transformation can be modeled using the Bayesian bootstrap ('bnp'), which is a Bayesian nonparametric model and incorporates the uncertainty about the transformation into posterior and predictive inference.

Value

a list with the following elements:

- `post_ytilde`: `nsave` x `n` samples from the posterior predictive distribution at the observation points `tau`
- `marg_like`: the marginal likelihood (only if `use_MCMC=FALSE`; otherwise NULL)

Note

For the 'bnp' transformation (without the `Fy` approximation), there are numerical stability issues when `psi` is modeled as unknown. In this case, it is better to fix `psi` at some positive number.

Examples

```
# Simulate some data:
n = 100
tau = seq(0,1, length.out = n)
y = round_floor(exp(1 + rnorm(n)/4 + poly(tau, 4)%*%rnorm(n=4, sd = 4:1)))

# Sample from the predictive distribution of a STAR spline model:
fit = spline_star(y = y, tau = tau)

# Compute 90% prediction intervals:
pi_y = t(apply(fit$post_ytilde, 2, quantile, c(0.05, .95)))

# Plot the results: intervals, median, and smoothed mean
plot(tau, y, ylim = range(pi_y, y))
polygon(c(tau, rev(tau)),c(pi_y[,2], rev(pi_y[,1])),col='gray', border=NA)
lines(tau, apply(fit$post_ytilde, 2, median), lwd=5, col ='black')
lines(tau, smooth.spline(tau, apply(fit$post_ytilde, 2, mean))$y, lwd=5, col='blue')
lines(tau, y, type='p')
```

`warpDLM` *Posterior Inference for warpDLM model with latent structural DLM*

Description

This function outputs posterior quantities and forecasts from a univariate warpDLM model. Currently two latent DLM specifications are supported: local level and the local linear trend.

Usage

```
warpDLM(
  y,
  type = c("level", "trend"),
  transformation = c("np", "identity", "log", "sqrt", "pois", "neg-bin"),
  y_max = Inf,
  R0 = 10,
  nsave = 5000,
  nburn = 5000,
  nskip = 1,
  n.ahead = 1
)
```

Arguments

`y`	the count-valued time series
`type`	the type of latent DLM (must be either level or trend)
`transformation`	transformation to use for the latent process (default is np); must be one of • "identity" (identity transformation) • "log" (log transformation) • "sqrt" (square root transformation) • "np" (nonparametric transformation estimated from empirical CDF) • "pois" (transformation for moment-matched marginal Poisson CDF) • "neg-bin" (transformation for moment-matched marginal Negative Binomial CDF)
`y_max`	a fixed and known upper bound for all observations; default is `Inf`
`R0`	the variance for the initial state theta_0; default is 10
`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
`n.ahead`	number of steps to forecast ahead

Value

A list with the following elements:

- `V_post`: posterior draws of the observation variance
- `W_post`: posterior draws of the state update variance(s)
- `fc_post`: draws from the forecast distribution (of length n.ahead)
- `post_pred`: draws from the posterior predictive distribution of `y`
- `g_func`: transformation function
- `g_inv_func`: inverse transformation function
- `KFAS_mod`: the final KFAS model representing the latent DLM

Index