Package: BKP (via r-universe)

June 7, 2026

Title Beta Kernel Process Modeling

Version 0.2.3.9000

Maintainer Jiangyan Zhao <zhaojy2017@126.com>

Description Implements the Beta Kernel Process (BKP) for nonparametric modeling of spatially varying binomial probabilities, together with its extension, the Dirichlet Kernel Process (DKP), for categorical or multinomial data. The package provides functions for model fitting, predictive inference with uncertainty quantification, posterior simulation, and visualization in one-and two-dimensional input spaces. Multiple kernel functions (Gaussian, Matern 5/2, and Matern 3/2) are supported, with hyperparameters optimized through multi-start gradient-based search. For more details, see Zhao, Qing, and Xu (2025) <doi:10.48550/arXiv.2508.10447>.

License GPL (>= 3)

URL https://github.com/Jiangyan-Zhao/BKP

BugReports https://github.com/Jiangyan-Zhao/BKP/issues

Encoding UTF-8

NeedsCompilation yes

Roxygen list(markdown = TRUE)

Depends R (>= 3.5.0)

Imports dirmult, ggplot2, gridExtra, lattice, optimx, rlang, tgp, Rcpp

Suggests knitr, mlbench, testthat (>= 3.0.0)

Config/testthat/edition 3

LinkingTo Rcpp, RcppArmadillo, nloptr

Config/roxygen2/version 8.0.0

Config/pak/sysreqs cmake

Repository https://jiangyan-zhao.r-universe.dev

Date/Publication 2026-06-07 14:04:06 UTC

RemoteUrl https://github.com/jiangyan-zhao/bkp

RemoteRef HEAD

RemoteSha 0c032ec735ecf8b8abd3bd5c7ecaf29f3fab4ebd

Contents

`BKP-package` *Beta Kernel Process Modeling*

Description

The **BKP** package provides tools for Bayesian nonparametric modeling of binary/binomial or categorical/multinomial response data using the Beta Kernel Process (BKP) and its extension, the Dirichlet Kernel Process (DKP). These methods estimate latent probability surfaces through localized kernel smoothing under a Bayesian framework.

The package offers functionality for model fitting, posterior predictive inference with uncertainty quantification, simulation of posterior draws, and visualization in both one- and two-dimensional input spaces. It also supports flexible prior specification and hyperparameter tuning.

Main Functions

Core functionality is organized as follows:

`fit_BKP`, `fit_DKP` Fit a BKP or DKP model to (multi)binomial response data.

`predict.BKP`, `predict.DKP` Perform posterior predictive inference at new input locations, including predictive means, variances, and credible intervals. When observations correspond to single trials (binary or categorical responses), predicted class labels are returned automatically.

`simulate.BKP`, `simulate.DKP` Generate simulated responses from the posterior predictive distribution of a fitted model.

`plot.BKP`, `plot.DKP` Visualize model predictions and associated uncertainty in one- and two-dimensional input spaces; for inputs with more than two dimensions, users can select one or two dimensions to display via the `dims` argument.

`summary.BKP`, `summary.DKP`, `print.BKP`, `print.DKP` Summarize or print the details of a fitted BKP or DKP model.

References

Zhao J, Qing K, Xu J (2025). *BKP: An R Package for Beta Kernel Process Modeling*. arXiv. https://doi.org/10.48550/arXiv.2508.10447.

Rolland P, Kavis A, Singla A, Cevher V (2019). *Efficient learning of smooth probability functions from Bernoulli tests with guarantees*. In Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA, volume 97 of Proceedings of Machine Learning Research, pp. 5459-5467. PMLR.

MacKenzie CA, Trafalis TB, Barker K (2014). *A Bayesian Beta Kernel Model for Binary Classification and Online Learning Problems*. Statistical Analysis and Data Mining: The ASA Data Science Journal, 7(6), 434-449.

Goetschalckx R, Poupart P, Hoey J (2011). *Continuous Correlated Beta Processes*. In Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two, IJCAI'11, p. 1269-1274. AAAI Press.

`fit_BKP` *Fit a Beta Kernel Process (BKP) Model*

Description

Fits a Beta Kernel Process (BKP) model to binary or binomial response data using local kernel smoothing. The method constructs a flexible latent probability surface by updating Beta priors with kernel-weighted observations.

Usage

```
fit_BKP(
  X,
  y,
  m,
  Xbounds = NULL,
  prior = c("noninformative", "fixed", "adaptive"),
  r0 = 2,
  p0 = mean(y/m),
  kernel = c("gaussian", "matern52", "matern32", "wendland"),
  loss = c("brier", "log_loss"),
  n_multi_start = NULL,
  theta = NULL,
```

```
  isotropic = TRUE
)
```

Arguments

X	A numeric input matrix of size $n \times d$, where each row corresponds to a covariate vector.
y	A numeric vector of observed successes (length n).
m	A numeric vector of total binomial trials (length n), corresponding to each y.
Xbounds	Optional $d \times 2$ matrix specifying the lower and upper bounds of each input dimension. Used to normalize inputs to $[0, 1]^d$. If NULL, inputs are assumed to be pre-normalized, and default bounds $[0, 1]^d$ are applied.
prior	Type of prior: "noninformative" (default), "fixed", or "adaptive".
r0	Global prior precision (used when prior = "fixed" or "adaptive").
p0	Global prior mean (used when prior = "fixed"). Default is mean(y/m).
kernel	Kernel function for local weighting: "gaussian" (default), "matern52", "matern32", or "wendland".
loss	Loss function for kernel hyperparameter tuning: "brier" (default) or "log_loss".
n_multi_start	Number of initial points used in multi-start optimization of the kernel lengthscale parameters. If NULL, the default is $10p$, where $p = 1$ for isotropic kernels and $p = d$ for anisotropic kernels.
theta	Optional. A positive scalar or numeric vector of length d specifying kernel lengthscale parameters directly. If NULL (default), lengthscales are optimized using multi-start derivative-free local optimization to minimize the specified loss.
isotropic	Logical. If TRUE (default), optimize/use a single shared lengthscale across dimensions. If FALSE, use separate per-dimension lengthscales.

Value

A list of class "BKP" containing the fitted BKP model, including:

theta_opt Optimized kernel hyperparameters (lengthscales).

kernel Kernel function used, as a string.

isotropic Logical flag indicating whether a shared lengthscale (TRUE) or per-dimension lengthscales (FALSE) was used.

loss Loss function used for hyperparameter tuning.

loss_min Loss value at the selected/provided kernel parameters.

X Original input matrix ($n \times d$).

Xnorm Normalized input matrix scaled to $[0, 1]^d$.

Xbounds Normalization bounds for each input dimension ($d \times 2$).

y Observed success counts.

m Observed binomial trial counts.

prior Type of prior used.

`r0` Prior precision parameter.

`p0` Prior mean (for fixed priors).

`alpha0` Prior Beta shape parameter $\alpha_0(\mathbf{x})$.

`beta0` Prior Beta shape parameter $\beta_0(\mathbf{x})$.

`alpha_n` Posterior shape parameter $\alpha_n(\mathbf{x})$.

`beta_n` Posterior shape parameter $\beta_n(\mathbf{x})$.

References

Zhao J, Qing K, Xu J (2025). *BKP: An R Package for Beta Kernel Process Modeling*. arXiv. https://doi.org/10.48550/arXiv.2508.10447.

See Also

`fit_DKP` for modeling multinomial responses via the Dirichlet Kernel Process. `predict.BKP`, `plot.BKP`, `simulate.BKP`, and `summary.BKP` for prediction, visualization, posterior simulation, and summarization of a fitted BKP model.

Examples

```
#-------------------------- 1D Example ---------------------------
set.seed(123)

# Define true success probability function
true_pi_fun <- function(x) {
  (1 + exp(-x^2) * cos(10 * (1 - exp(-x)) / (1 + exp(-x)))) / 2
}

n <- 30
Xbounds <- matrix(c(-2,2), nrow=1)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(100, n, replace = TRUE)
y <- rbinom(n, size = m, prob = true_pi)

# Fit BKP model
model1 <- fit_BKP(X, y, m, Xbounds=Xbounds)
print(model1)


#-------------------------- 2D Example ---------------------------
set.seed(123)

# Define 2D latent function and probability transformation
true_pi_fun <- function(X) {
  if(is.null(nrow(X))) X <- matrix(X, nrow=1)
  m <- 8.6928
  s <- 2.4269
  x1 <- 4*X[,1]- 2
  x2 <- 4*X[,2]- 2
```

```
  a <- 1 + (x1 + x2 + 1)^2 *
    (19- 14*x1 + 3*x1^2- 14*x2 + 6*x1*x2 + 3*x2^2)
  b <- 30 + (2*x1- 3*x2)^2 *
    (18- 32*x1 + 12*x1^2 + 48*x2- 36*x1*x2 + 27*x2^2)
  f <- log(a*b)
  f <- (f- m)/s
  return(pnorm(f))  # Transform to probability
}

n <- 100
Xbounds <- matrix(c(0, 0, 1, 1), nrow = 2)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(100, n, replace = TRUE)
y <- rbinom(n, size = m, prob = true_pi)

# Fit BKP model
model2 <- fit_BKP(X, y, m, Xbounds=Xbounds)
print(model2)
```

`fit_DKP` *Fit a Dirichlet Kernel Process (DKP) Model*

Description

Fits a DKP model for categorical or multinomial response data by locally smoothing observed counts to estimate latent Dirichlet parameters. The model constructs flexible latent probability surfaces by updating Dirichlet priors using kernel-weighted observations.

Usage

```
fit_DKP(
  X,
  Y,
  Xbounds = NULL,
  prior = c("noninformative", "fixed", "adaptive"),
  r0 = 2,
  p0 = colMeans(Y/rowSums(Y)),
  kernel = c("gaussian", "matern52", "matern32", "wendland"),
  loss = c("brier", "log_loss"),
  n_multi_start = NULL,
  theta = NULL,
  isotropic = TRUE
)
```

Arguments

X	A numeric input matrix of size $n \times d$, where each row corresponds to a covariate vector.
Y	Numeric matrix of observed multinomial counts, with dimension $n \times q$.
Xbounds	Optional $d \times 2$ matrix specifying the lower and upper bounds of each input dimension. Used to normalize inputs to $[0,1]^d$. If NULL, inputs are assumed to be pre-normalized, and default bounds $[0,1]^d$ are applied.
prior	Type of prior: "noninformative" (default), "fixed", or "adaptive".
r0	Global prior precision (used when prior = "fixed" or "adaptive").
p0	Global prior mean vector (used only when prior = "fixed"). Defaults to the empirical class proportions colMeans(Y / rowSums(Y)). Must have length equal to the number of categories q.
kernel	Kernel function for local weighting: "gaussian" (default), "matern52", "matern32", or "wendland".
loss	Loss function for kernel hyperparameter tuning: "brier" (default) or "log_loss".
n_multi_start	Number of initial points used in multi-start optimization of the kernel lengthscale parameters. If NULL, the default is $10p$, where $p = 1$ for isotropic kernels and $p = d$ for anisotropic kernels.
theta	Optional. A positive scalar or numeric vector of length d specifying kernel lengthscale parameters directly. If NULL (default), lengthscales are optimized using multi-start derivative-free local optimization to minimize the specified loss.
isotropic	Logical. If TRUE (default), optimize/use a single shared lengthscale across dimensions. If FALSE, use separate per-dimension lengthscales.

Value

A list of class "DKP" representing the fitted DKP model, with the following components:

theta_opt Optimized kernel hyperparameters (lengthscales).

kernel Kernel function used, as a string.

isotropic Logical flag indicating whether a shared lengthscale (TRUE) or per-dimension lengthscales (FALSE) was used.

loss Loss function used for hyperparameter tuning.

loss_min Minimum loss value achieved during kernel hyperparameter optimization. Set to NA if theta is user-specified.

X Original (unnormalized) input matrix of size n × d.

Xnorm Normalized input matrix scaled to $[0,1]^d$.

Xbounds Matrix specifying normalization bounds for each input dimension.

Y Observed multinomial counts of size n × q.

prior Type of prior used.

r0 Prior precision parameter.

p0 Prior mean (for fixed priors).

alpha0 Prior Dirichlet parameters at each input location (scalar or matrix).

alpha_n Posterior Dirichlet parameters after kernel smoothing.

References

Zhao J, Qing K, Xu J (2025). *BKP: An R Package for Beta Kernel Process Modeling*. arXiv. https://doi.org/10.48550/arXiv.2508.10447.

See Also

`fit_BKP` for modeling binomial responses via the Beta Kernel Process. `predict.DKP`, `plot.DKP`, `simulate.DKP` for prediction, visualization, and posterior simulation from a fitted DKP model. `summary.DKP`, `print.DKP` for inspecting model summaries.

Examples

```
#-------------------------- 1D Example ----------------------------
set.seed(123)

# Define true class probability function (3-class)
true_pi_fun <- function(X) {
  p1 <- 1/(1+exp(-3*X))
  p2 <- (1 + exp(-X^2) * cos(10 * (1 - exp(-X)) / (1 + exp(-X)))) / 2
  return(matrix(c(p1/2, p2/2, 1 - (p1+p2)/2), nrow = length(p1)))
}

n <- 30
Xbounds <- matrix(c(-2, 2), nrow = 1)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(150, n, replace = TRUE)

# Generate multinomial responses
Y <- t(sapply(1:n, function(i) rmultinom(1, size = m[i], prob = true_pi[i, ])))

# Fit DKP model
model1 <- fit_DKP(X, Y, Xbounds = Xbounds)
print(model1)


#-------------------------- 2D Example ----------------------------
set.seed(123)

# Define latent function and transform to 3-class probabilities
true_pi_fun <- function(X) {
  if (is.null(nrow(X))) X <- matrix(X, nrow = 1)
  m <- 8.6928; s <- 2.4269
  x1 <- 4 * X[,1] - 2
  x2 <- 4 * X[,2] - 2
  a <- 1 + (x1 + x2 + 1)^2 *
    (19 - 14*x1 + 3*x1^2 - 14*x2 + 6*x1*x2 + 3*x2^2)
  b <- 30 + (2*x1 - 3*x2)^2 *
    (18 - 32*x1 + 12*x1^2 + 48*x2 - 36*x1*x2 + 27*x2^2)
  f <- (log(a*b)- m)/s
  p1 <- pnorm(f) # Transform to probability
  p2 <- sin(pi * X[,1]) * sin(pi * X[,2])
```

```
  return(matrix(c(p1/2, p2/2, 1 - (p1+p2)/2), nrow = length(p1)))
}

n <- 100
Xbounds <- matrix(c(0, 0, 1, 1), nrow = 2)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(150, n, replace = TRUE)

# Generate multinomial responses
Y <- t(sapply(1:n, function(i) rmultinom(1, size = m[i], prob = true_pi[i, ])))

# Fit DKP model
model2 <- fit_DKP(X, Y, Xbounds = Xbounds)
print(model2)
```

`fit_TwinBKP` *Fit a Twin Beta Kernel Process (TwinBKP) Model*

Description

Fits a TwinBKP model for binary or binomial response data. The workflow follows `fit_BKP()`, but kernel hyperparameter tuning is accelerated by first selecting `g` representative support points from the full dataset using the `Twining` package, then optimizing the kernel lengthscales on these `g` points only. The final posterior update still uses all `n` observations.

Usage

```
fit_TwinBKP(
  X,
  y,
  m,
  Xbounds = NULL,
  prior = c("noninformative", "fixed", "adaptive"),
  r0 = 2,
  p0 = mean(y/m),
  kernel = c("gaussian", "matern52", "matern32", "wendland"),
  loss = c("brier", "log_loss"),
  n_multi_start = NULL,
  theta = NULL,
  isotropic = TRUE,
  g_nums = NULL
)
```

Arguments

`X`	A numeric input matrix of size $n \times d$, where each row corresponds to a covariate vector.

y	A numeric vector of observed successes (length n).
m	A numeric vector of total binomial trials (length n), corresponding to each y.
Xbounds	Optional $d \times 2$ matrix specifying the lower and upper bounds of each input dimension. Used to normalize inputs to $[0,1]^d$. If NULL, inputs are assumed to be pre-normalized, and default bounds $[0,1]^d$ are applied.
prior	Type of prior: "noninformative" (default), "fixed", or "adaptive".
r0	Global prior precision (used when prior = "fixed" or "adaptive").
p0	Global prior mean (used when prior = "fixed"). Default is mean(y/m).
kernel	Kernel function for local weighting: "gaussian" (default), "matern52", "matern32", or "wendland".
loss	Loss function for kernel hyperparameter tuning: "brier" (default) or "log_loss".
n_multi_start	Number of initial points used in multi-start optimization of the kernel lengthscale parameters. If NULL, the default is $10p$, where $p = 1$ for isotropic kernels and $p = d$ for anisotropic kernels.
theta	Optional. A positive scalar or numeric vector of length d specifying kernel lengthscale parameters directly. If NULL (default), lengthscales are optimized using multi-start derivative-free local optimization to minimize the specified loss.
isotropic	Logical. If TRUE (default), optimize/use a single shared lengthscale across dimensions. If FALSE, use separate per-dimension lengthscales.
g_nums	Positive integer. Number of global support points selected by the Twining algorithm for hyperparameter tuning. If NULL (default), it is set adaptively as $\min\{50d, \max(10d, \sqrt{n})\}$, where d is input dimensionality and n is sample size.

Value

A list of class "TwinBKP" with the same structure as "BKP", plus:

g_nums Number of support points used for tuning.

tune_idx Row indices of the selected support points.

alpha0_global, beta0_global Global-stage prior parameters on support points.

alpha_n_global, beta_n_global Global-stage posterior parameters on support points.

mean_global, var_global Global-stage posterior mean/variance on support points; used directly by fitted.TwinBKP() and summary.TwinBKP() without recomputation.

See Also

fit_BKP

Examples

```
set.seed(123)
true_pi_fun <- function(x) {
  (1 + exp(-x^2) * cos(10 * (1 - exp(-x)) / (1 + exp(-x)))) / 2
}
```

```
n <- 10000
Xbounds <- matrix(c(-2, 2), nrow = 1)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(100, n, replace = TRUE)
y <- rbinom(n, size = m, prob = true_pi)
model <- fit_TwinBKP(X, y, m, Xbounds = Xbounds, g_nums = 100)
print(model)
```

`fit_TwinDKP` *Fit a Twin Dirichlet Kernel Process (TwinDKP) Model*

Description

Fits a TwinDKP model for categorical or multinomial response data. Similar to `fit_TwinBKP()`, the method first selects `g_nums` representative support points using Twining, then optimizes kernel hyperparameters on these support points only.

Usage

```
fit_TwinDKP(
  X,
  Y,
  Xbounds = NULL,
  prior = c("noninformative", "fixed", "adaptive"),
  r0 = 2,
  p0 = colMeans(Y/rowSums(Y)),
  kernel = c("gaussian", "matern52", "matern32", "wendland"),
  loss = c("brier", "log_loss"),
  n_multi_start = NULL,
  theta = NULL,
  isotropic = TRUE,
  g_nums = NULL
)
```

Arguments

`X`	A numeric input matrix of size $n \times d$, where each row corresponds to a covariate vector.
`Y`	Numeric matrix of observed multinomial counts, with dimension $n \times q$.
`Xbounds`	Optional $d \times 2$ matrix specifying the lower and upper bounds of each input dimension. Used to normalize inputs to $[0, 1]^d$. If `NULL`, inputs are assumed to be pre-normalized, and default bounds $[0, 1]^d$ are applied.
`prior`	Type of prior: `"noninformative"` (default), `"fixed"`, or `"adaptive"`.
`r0`	Global prior precision (used when `prior = "fixed"` or `"adaptive"`).

p0	Global prior mean vector (used only when prior = "fixed"). Defaults to the empirical class proportions colMeans(Y / rowSums(Y)). Must have length equal to the number of categories q.
kernel	Kernel function for local weighting: "gaussian" (default), "matern52", "matern32", or "wendland".
loss	Loss function for kernel hyperparameter tuning: "brier" (default) or "log_loss".
n_multi_start	Number of initial points used in multi-start optimization of the kernel lengthscale parameters. If NULL, the default is $10p$, where $p = 1$ for isotropic kernels and $p = d$ for anisotropic kernels.
theta	Optional. A positive scalar or numeric vector of length d specifying kernel lengthscale parameters directly. If NULL (default), lengthscales are optimized using multi-start derivative-free local optimization to minimize the specified loss.
isotropic	Logical. If TRUE (default), optimize/use a single shared lengthscale across dimensions. If FALSE, use separate per-dimension lengthscales.
g_nums	Positive integer. Number of global support points selected by the Twining algorithm for hyperparameter tuning. If NULL (default), it is set adaptively as $\min\{50d, \max(10d, \sqrt{n})\}$, where d is input dimensionality and n is sample size.

Value

A list of class "TwinDKP" with the same structure as "DKP", plus:

g_nums Number of support points used for tuning.

tune_idx Row indices of the selected support points.

alpha0_global Global-stage prior Dirichlet parameters on support points.

alpha_n_global Global-stage posterior Dirichlet parameters on support points.

mean_global, var_global Global-stage posterior mean/variance on support points; used directly by fitted.TwinDKP() and summary.TwinDKP() without recomputation.

See Also

fit_DKP, predict.DKP

---

fitted *Extract BKP or DKP Model Fitted Values*

Description

Compute the posterior fitted values from a fitted BKP or DKP object. For a BKP object, this returns the posterior mean probability of the positive class. For a DKP object, this returns the posterior mean probabilities for each class.

Usage

```
## S3 method for class 'BKP'
fitted(object, ...)

## S3 method for class 'DKP'
fitted(object, ...)

## S3 method for class 'TwinBKP'
fitted(object, ...)

## S3 method for class 'TwinDKP'
fitted(object, ...)
```

Arguments

`object`	An object of class `BKP` or `DKP`, typically the result of a call to `fit_BKP` or `fit_DKP`.
`...`	Additional arguments (currently unused).

Details

For a `BKP` model, the fitted values correspond to the posterior mean probability of the positive class, computed from the Beta Kernel Process. For a `DKP` model, the fitted values correspond to the posterior mean probabilities for each class, derived from the posterior Dirichlet distribution of the class probabilities.

For `TwinBKP`, fitted values are computed on the global support set selected during `fit_TwinBKP()`. Specifically, this method computes posterior Beta parameters on `object$Xnorm_global` and returns the posterior mean $\alpha_n/(\alpha_n + \beta_n)$ for those support points.

This is intentionally different from `BKP`, where fitted values are naturally available on all training points. In TwinBKP, full-sample behavior is prediction-oriented and should be obtained via `predict.TwinBKP()`.

For `TwinDKP`, fitted values are reported on the global support points selected during `fit_TwinDKP()`. The method computes posterior mean class probabilities on `object$Xnorm_global`.

Value

A numeric vector (for `BKP`) or a numeric matrix (for `DKP`) containing posterior mean estimates at the training inputs.

References

Zhao J, Qing K, Xu J (2025). *BKP: An R Package for Beta Kernel Process Modeling*. arXiv. https://doi.org/10.48550/arXiv.2508.10447.

Examples

```
# ------------------------- BKP --------------------------
set.seed(123)
```

```
# Define true success probability function
true_pi_fun <- function(x) {
  (1 + exp(-x^2) * cos(10 * (1 - exp(-x)) / (1 + exp(-x)))) / 2
}

n <- 30
Xbounds <- matrix(c(-2, 2), nrow = 1)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(100, n, replace = TRUE)
y <- rbinom(n, size = m, prob = true_pi)

# Fit BKP model
model <- fit_BKP(X, y, m, Xbounds = Xbounds)

# Extract fitted values
fitted(model)

# -------------------------- DKP ---------------------------
#' set.seed(123)

# Define true class probability function (3-class)
true_pi_fun <- function(X) {
  p1 <- 1/(1+exp(-3*X))
  p2 <- (1 + exp(-X^2) * cos(10 * (1 - exp(-X)) / (1 + exp(-X)))) / 2
  return(matrix(c(p1/2, p2/2, 1 - (p1+p2)/2), nrow = length(p1)))
}

n <- 30
Xbounds <- matrix(c(-2, 2), nrow = 1)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(150, n, replace = TRUE)

# Generate multinomial responses
Y <- t(sapply(1:n, function(i) rmultinom(1, size = m[i], prob = true_pi[i, ])))

# Fit DKP model
model <- fit_DKP(X, Y, Xbounds = Xbounds)

# Extract fitted values
fitted(model)
```

---

get_prior *Construct Prior Parameters for BKP/DKP Models*

Description

Computes prior parameters for the Beta Kernel Process (BKP, for binary outcomes) or Dirichlet Kernel Process (DKP, for multi-class outcomes). Supports `prior = "noninformative"`, `"fixed"`, and `"adaptive"` strategies.

Usage

```
get_prior(
  prior = c("noninformative", "fixed", "adaptive"),
  model = c("BKP", "DKP"),
  r0 = 2,
  p0 = NULL,
  y = NULL,
  m = NULL,
  Y = NULL,
  K = NULL
)
```

Arguments

`prior`	Type of prior: `"noninformative"` (default), `"fixed"`, or `"adaptive"`.
`model`	A character string specifying the model type: `"BKP"` (binary outcome) or `"DKP"` (multi-class outcome).
`r0`	Global prior precision (used when `prior = "fixed"` or `"adaptive"`).
`p0`	For BKP, a scalar in `(0,1)` specifying the prior mean of success probability when `prior = "fixed"`. For DKP, a numeric vector of length equal to the number of classes specifying the global prior mean, which must sum to 1.
`y`	A numeric vector of observed successes (length `n`).
`m`	A numeric vector of total binomial trials (length `n`), corresponding to each `y`.
`Y`	A numeric matrix of observed class counts (`n × q`), required only when `model = "DKP"`, where `n` is the number of observations and `q` the number of classes.
`K`	A precomputed kernel matrix, typically obtained from `kernel_matrix`. Can be rectangular (`m × n`), where `n` is the number of observed points and `m` the number of prediction locations.

Details

- `prior = "noninformative"`: flat prior; all parameters set to 1.
- `prior = "fixed"`:
  - BKP: uniform Beta prior `Beta(r0 * p0, r0 * (1 - p0))` across locations.
  - DKP: all rows of `alpha0` set to `r0 * p0`.
- `prior = "adaptive"`:
  - BKP: prior mean estimated at each location via kernel smoothing of observed proportions `y/m`, with precision `r0`.
  - DKP: prior parameters computed by kernel-weighted smoothing of observed class frequencies in `Y`, scaled by `r0`.

Value

- If `model = "BKP"`: a list with

  `alpha0` Vector of prior alpha parameters for the Beta distribution, length `n`.

  `beta0` Vector of prior beta parameters for the Beta distribution, length `n`.
- If `model = "DKP"`: a matrix `alpha0` of prior Dirichlet parameters at each input location (`n` × `q`).

References

Zhao J, Qing K, Xu J (2025). *BKP: An R Package for Beta Kernel Process Modeling*. arXiv. https://doi.org/10.48550/arXiv.2508.10447

See Also

`fit_BKP` for fitting Beta Kernel Process models, `fit_DKP` for fitting Dirichlet Kernel Process models, `predict.BKP` and `predict.DKP` for making predictions, `kernel_matrix` for computing kernel matrices used in prior construction.

Examples

```
# -------------------------- BKP ---------------------------
set.seed(123)
n <- 10
X <- matrix(runif(n * 2), ncol = 2)
y <- rbinom(n, size = 5, prob = 0.6)
m <- rep(5, n)
K <- kernel_matrix(X)
prior_bkp <- get_prior(
  model = "BKP", prior = "adaptive", r0 = 2, y = y, m = m, K = K
)

# -------------------------- DKP ---------------------------
set.seed(123)
n <- 15; q <- 3
X <- matrix(runif(n * 2), ncol = 2)
true_pi <- t(apply(X, 1, function(x) {
  raw <- c(
    exp(-sum((x - 0.2)^2)),
    exp(-sum((x - 0.5)^2)),
    exp(-sum((x - 0.8)^2))
  )
  raw / sum(raw)
}))
m <- sample(10:20, n, replace = TRUE)
Y <- t(sapply(1:n, function(i) rmultinom(1, size = m[i], prob = true_pi[i, ])))
K <- kernel_matrix(X, theta = 0.2, kernel = "gaussian")
prior_dkp <- get_prior(
  model = "DKP", prior = "adaptive", r0 = 2, Y = Y, K = K
)
```

`kernel_matrix` *Compute Kernel Matrix Between Input Locations*

Description

Computes the kernel matrix between two sets of input locations using a specified kernel function. Supports both isotropic and anisotropic lengthscales. Available kernels include the Gaussian, Matérn 5/2, Matérn 3/2, and Wendland compactly supported kernel.

Usage

```
kernel_matrix(
  X,
  Xprime = NULL,
  theta = 0.1,
  kernel = c("gaussian", "matern52", "matern32", "wendland"),
  isotropic = TRUE
)
```

Arguments

`X`	A numeric matrix (or vector) of input locations with shape $n \times d$.
`Xprime`	An optional numeric matrix of input locations with shape $m \times d$. If `NULL` (default), it is set to `X`, resulting in a symmetric matrix.
`theta`	A positive numeric value or vector specifying the kernel lengthscale(s). If `isotropic = TRUE` (default), this must be a scalar shared by all input dimensions. If `isotropic = FALSE`, this can be a scalar (broadcasted) or a vector of length `d` for per-dimension scaling.
`kernel`	A character string specifying the kernel function. Must be one of `"gaussian"`, `"matern32"`, `"matern52"`, or `"wendland"`.
`isotropic`	Logical. If `TRUE` (default), use a single shared lengthscale across dimensions. If `FALSE`, use per-dimension lengthscales.

Details

Let $\mathbf{x}$ and $\mathbf{x}'$ denote two input points. The scaled distance is defined as

$$r = \left\| \frac{\mathbf{x} - \mathbf{x}'}{\boldsymbol{\theta}} \right\|_2 .$$

The available kernels are defined as:

- **Gaussian:**
$$k(\mathbf{x}, \mathbf{x}') = \exp(-r^2)$$
- **Matérn 5/2:**
$$k(\mathbf{x}, \mathbf{x}') = \left(1 + \sqrt{5}r + \frac{5}{3}r^2\right) \exp(-\sqrt{5}r)$$

- **Matérn 3/2:**

$$k(\mathbf{x}, \mathbf{x}') = \left(1 + \sqrt{3}r\right) \exp(-\sqrt{3}r)$$

- **Wendland:**

$$k(\mathbf{x}, \mathbf{x}') = (qr + 1) \max(0, 1 - r)^q, \quad q = \lfloor d/2 \rfloor + 3$$

The function performs consistency checks on input dimensions and automatically broadcasts `theta` when it is a scalar.

Value

A numeric matrix of size $n \times m$, where each element K_{ij} gives the kernel similarity between input X_i and X'_j.

References

Zhao J, Qing K, Xu J (2025). *BKP: An R Package for Beta Kernel Process Modeling*. arXiv. https://doi.org/10.48550/arXiv.2508.10447.

Rasmussen, C. E., & Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press.

Examples

```
# Basic usage with default Xprime = X
X <- matrix(runif(20), ncol = 2)
K1 <- kernel_matrix(X, theta = 0.2, kernel = "gaussian")

# Anisotropic lengthscales with Matérn 5/2
K2 <- kernel_matrix(X, theta = c(0.1, 0.3), kernel = "matern52", isotropic = FALSE)

# Isotropic Matérn 3/2
K3 <- kernel_matrix(X, theta = 1, kernel = "matern32")

# Use Xprime different from X
Xprime <- matrix(runif(10), ncol = 2)
K4 <- kernel_matrix(X, Xprime, theta = 0.2, kernel = "gaussian")
```

---

loss_fun *Loss Function for BKP and DKP Models*

Description

Computes the loss for fitting BKP (binary) or DKP (multi-class) models. Supports Brier score (mean squared error) and log-loss (cross-entropy) under different prior specifications.

Usage

```
loss_fun(
  gamma,
  Xnorm,
  y = NULL,
  m = NULL,
  Y = NULL,
  model = c("BKP", "DKP"),
  prior = c("noninformative", "fixed", "adaptive"),
  r0 = 2,
  p0 = NULL,
  loss = c("brier", "log_loss"),
  kernel = c("gaussian", "matern52", "matern32", "wendland"),
  isotropic = TRUE
)
```

Arguments

gamma	A numeric vector of log10-transformed kernel hyperparameters.
Xnorm	A numeric matrix of normalized input features (`[0,1]^d`).
y	A numeric vector of observed successes (length `n`).
m	A numeric vector of total binomial trials (length `n`), corresponding to each `y`.
Y	A numeric matrix of observed class counts (`n` × `q`), required only when `model = "DKP"`, where `n` is the number of observations and `q` the number of classes.
model	A character string specifying the model type: `"BKP"` (binary outcome) or `"DKP"` (multi-class outcome).
prior	Type of prior: `"noninformative"` (default), `"fixed"`, or `"adaptive"`.
r0	Global prior precision (used when `prior = "fixed"` or `"adaptive"`).
p0	For BKP, a scalar in `(0,1)` specifying the prior mean of success probability when `prior = "fixed"`. For DKP, a numeric vector of length equal to the number of classes specifying the global prior mean, which must sum to 1.
loss	Loss function for kernel hyperparameter tuning: `"brier"` (default) or `"log_loss"`.
kernel	Kernel function for local weighting: `"gaussian"` (default), `"matern52"`, `"matern32"`, or `"wendland"`.
isotropic	Logical. If `TRUE` (default), optimize/use a single shared lengthscale across dimensions. If `FALSE`, use separate per-dimension lengthscales.

Value

A single numeric value representing the total loss (to be minimized). The value corresponds to either the Brier score (squared error) or the log-loss (cross-entropy).

References

Zhao J, Qing K, Xu J (2025). *BKP: An R Package for Beta Kernel Process Modeling*. arXiv. https://doi.org/10.48550/arXiv.2508.10447.

See Also

fit_BKP for fitting BKP models, fit_DKP for fitting DKP models, get_prior for constructing prior parameters, kernel_matrix for computing kernel matrices.

Examples

```
# -------------------------- BKP --------------------------
set.seed(123)
n <- 10
Xnorm <- matrix(runif(2 * n), ncol = 2)
m <- rep(10, n)
y <- rbinom(n, size = m, prob = runif(n))
loss_fun(gamma = 0, Xnorm = Xnorm, y = y, m = m, model = "BKP")

# -------------------------- DKP --------------------------
set.seed(123)
n <- 10
q <- 3
Xnorm <- matrix(runif(2 * n), ncol = 2)
Y <- matrix(rmultinom(n, size = 10, prob = rep(1/q, q)), nrow = n, byrow = TRUE)
loss_fun(gamma = 0, Xnorm = Xnorm, Y = Y, model = "DKP")
```

---

parameter *Extract Model Parameters from a Fitted BKP or DKP Model*

Description

Retrieve the key model parameters from a fitted BKP or DKP object. For a BKP model, this typically includes the optimized kernel hyperparameters and posterior Beta parameters. For a DKP model, this includes the kernel hyperparameters and posterior Dirichlet parameters.

Usage

```
parameter(object, ...)

## S3 method for class 'BKP'
parameter(object, ...)

## S3 method for class 'DKP'
parameter(object, ...)
```

Arguments

object	An object of class BKP or DKP, typically the result of a call to fit_BKP or fit_DKP.
...	Additional arguments (currently unused).

Value

A named list containing:

- `theta`: Estimated kernel hyperparameters.
- `alpha_n`: Posterior Dirichlet/Beta α parameters.
- `beta_n`: (BKP only) Posterior Beta β parameters.

References

Zhao J, Qing K, Xu J (2025). *BKP: An R Package for Beta Kernel Process Modeling*. arXiv. https://doi.org/10.48550/arXiv.2508.10447.

See Also

`fit_BKP` for fitting BKP models, `fit_DKP` for fitting DKP models.

Examples

```
# -------------------------- BKP --------------------------
set.seed(123)

# Define true success probability function
true_pi_fun <- function(x) {
  (1 + exp(-x^2) * cos(10 * (1 - exp(-x)) / (1 + exp(-x)))) / 2
}

n <- 30
Xbounds <- matrix(c(-2, 2), nrow = 1)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(100, n, replace = TRUE)
y <- rbinom(n, size = m, prob = true_pi)

# Fit BKP model
model <- fit_BKP(X, y, m, Xbounds = Xbounds)

# Extract posterior and kernel parameters
parameter(model)

# -------------------------- DKP --------------------------
#' set.seed(123)

# Define true class probability function (3-class)
true_pi_fun <- function(X) {
  p1 <- 1/(1+exp(-3*X))
  p2 <- (1 + exp(-X^2) * cos(10 * (1 - exp(-X)) / (1 + exp(-X)))) / 2
  return(matrix(c(p1/2, p2/2, 1 - (p1+p2)/2), nrow = length(p1)))
}

n <- 30
Xbounds <- matrix(c(-2, 2), nrow = 1)
```

```
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(150, n, replace = TRUE)

# Generate multinomial responses
Y <- t(sapply(1:n, function(i) rmultinom(1, size = m[i], prob = true_pi[i, ])))

# Fit DKP model
model <- fit_DKP(X, Y, Xbounds = Xbounds)

# Extract posterior quantiles
parameter(model)
```

---

plot *Plot Fitted BKP or DKP Models*

Description

Visualizes fitted BKP (binary) or DKP (multi-class) models according to the input dimensionality. For 1D inputs, it shows predicted class probabilities with credible intervals and observed data. For 2D inputs, it generates contour plots of posterior summaries. For higher-dimensional inputs, users must specify which dimensions to plot.

Usage

```
## S3 method for class 'BKP'
plot(
  x,
  only_mean = FALSE,
  n_grid = 80,
  dims = NULL,
  engine = c("base", "ggplot"),
  ...
)

## S3 method for class 'DKP'
plot(
  x,
  only_mean = FALSE,
  n_grid = 80,
  dims = NULL,
  engine = c("base", "ggplot"),
  ...
)

## S3 method for class 'TwinBKP'
plot(
```

```
  x,
  only_mean = FALSE,
  n_grid = 80,
  dims = NULL,
  engine = c("base", "ggplot"),
  l_nums = NULL,
  v_nums = NULL,
  ...
)

## S3 method for class 'TwinDKP'
plot(
  x,
  only_mean = FALSE,
  n_grid = 80,
  dims = NULL,
  engine = c("base", "ggplot"),
  l_nums = NULL,
  v_nums = NULL,
  ...
)
```

Arguments

x	An object of class "BKP" or "DKP", typically returned by fit_BKP or fit_DKP.
only_mean	Logical. If TRUE, only the predicted mean surface is plotted for 2D inputs (applies to both BKP and DKP models for mean visualization). Default is FALSE.
n_grid	Positive integer specifying the number of grid points per dimension for constructing the prediction grid. Larger values produce smoother and more detailed surfaces, but increase computation time. Default is 80.
dims	Integer vector indicating which input dimensions to plot. Must have length 1 (for 1D) or 2 (for 2D). If NULL (default), all dimensions are used when their number is <= 2.
engine	Character string specifying the plotting backend. Either "base" or "ggplot". The default "base" uses the package's original base/lattice plotting implementation, whereas "ggplot" uses **ggplot2**-based graphics when available. This argument applies to both 1D and 2D plots.
...	Additional arguments passed to internal plotting routines (currently unused).
l_nums	Optional local neighbor count passed to predict.TwinDKP().
v_nums	Optional validation size passed to predict.TwinDKP().

Details

The plotting behavior depends on the dimensionality of the input covariates:

- **1D inputs:**

  - For BKP (binary/binomial data), the function plots the posterior mean curve with a shaded 95% credible interval, overlaid with the observed proportions (y/m).
  - For DKP (categorical/multinomial data), it plots one curve per class, each with a shaded credible interval and the observed class frequencies.
  - For classification tasks, an optional curve of the maximum posterior class probability can be displayed to visualize decision confidence.
- **2D inputs:**
  - For both BKP and DKP models, the function generates contour plots over a 2D prediction grid.
  - Users can choose to plot only the predictive mean surface (only_mean = TRUE) or a set of four summary plots (only_mean = FALSE):
    1. Predictive mean
    2. 97.5th percentile (upper bound of 95% credible interval)
    3. Predictive variance
    4. 2.5th percentile (lower bound of 95% credible interval)
  - For DKP, these surfaces are generated separately for each class.
  - For classification tasks, predictive class probabilities can also be visualized as the maximum posterior probability surface.
- **Input dimensions greater than 2:**
  - The function does not automatically support visualization and will terminate with an error.
  - Users must specify which dimensions to visualize via the dims argument (length 1 or 2).

plot.TwinBKP() follows the same high-level workflow as plot.BKP(), but all predictive quantities are produced through predict.TwinBKP() on a constructed grid.

For 1D: a dense line grid is generated, then mean and credible interval are plotted with observed proportions.

For 2D: a lattice/ggplot surface grid is generated, then mean/upper/lower/ variance panels are visualized from prediction outputs.

plot.TwinDKP() follows the same workflow as plot.DKP(), but all predictive quantities are computed via predict.TwinDKP() on a constructed grid.

For 1D: draw class-wise mean and credible interval curves with observed proportions (or class indicators for single-label classification).

For 2D: draw predicted class map + max probability for classification, or class-wise mean/upper/variance/lower panels for multinomial probability data.

Value

This function is called for its side effects and does not return a value. It produces plots visualizing the predicted probabilities, credible intervals, and posterior summaries.

References

Zhao J, Qing K, Xu J (2025). *BKP: An R Package for Beta Kernel Process Modeling*. arXiv. https://doi.org/10.48550/arXiv.2508.10447.

See Also

fit_BKP and fit_DKP for fitting BKP and DKP models, respectively; predict.BKP and predict.DKP for generating predictions from fitted BKP and DKP models.

Examples

```
# ============================================================ #
# ======================== BKP Examples ======================= #
# ============================================================ #

#-------------------------- 1D Example ---------------------------
set.seed(123)

# Define true success probability function
true_pi_fun <- function(x) {
  (1 + exp(-x^2) * cos(10 * (1 - exp(-x)) / (1 + exp(-x)))) / 2
}

n <- 30
Xbounds <- matrix(c(-2,2), nrow=1)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(100, n, replace = TRUE)
y <- rbinom(n, size = m, prob = true_pi)

# Fit BKP model
model1 <- fit_BKP(X, y, m, Xbounds=Xbounds)

# Plot results
plot(model1)


#-------------------------- 2D Example ---------------------------
set.seed(123)

# Define 2D latent function and probability transformation
true_pi_fun <- function(X) {
  if(is.null(nrow(X))) X <- matrix(X, nrow=1)
  m <- 8.6928
  s <- 2.4269
  x1 <- 4*X[,1]- 2
  x2 <- 4*X[,2]- 2
  a <- 1 + (x1 + x2 + 1)^2 *
    (19- 14*x1 + 3*x1^2- 14*x2 + 6*x1*x2 + 3*x2^2)
  b <- 30 + (2*x1- 3*x2)^2 *
    (18- 32*x1 + 12*x1^2 + 48*x2- 36*x1*x2 + 27*x2^2)
  f <- log(a*b)
  f <- (f- m)/s
  return(pnorm(f))  # Transform to probability
}

n <- 100
```

```
Xbounds <- matrix(c(0, 0, 1, 1), nrow = 2)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(100, n, replace = TRUE)
y <- rbinom(n, size = m, prob = true_pi)

# Fit BKP model
model2 <- fit_BKP(X, y, m, Xbounds=Xbounds)

# Plot results
plot(model2, n_grid = 50)

# ============================================================ #
# ======================== DKP Examples ======================= #
# ============================================================ #

#-------------------------- 1D Example --------------------------
set.seed(123)

# Define true class probability function (3-class)
true_pi_fun <- function(X) {
  p1 <- 1/(1+exp(-3*X))
  p2 <- (1 + exp(-X^2) * cos(10 * (1 - exp(-X)) / (1 + exp(-X)))) / 2
  return(matrix(c(p1/2, p2/2, 1 - (p1+p2)/2), nrow = length(p1)))
}

n <- 30
Xbounds <- matrix(c(-2, 2), nrow = 1)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(150, n, replace = TRUE)

# Generate multinomial responses
Y <- t(sapply(1:n, function(i) rmultinom(1, size = m[i], prob = true_pi[i, ])))

# Fit DKP model
model1 <- fit_DKP(X, Y, Xbounds = Xbounds)

# Plot results
plot(model1)


#-------------------------- 2D Example --------------------------
set.seed(123)

# Define latent function and transform to 3-class probabilities
true_pi_fun <- function(X) {
  if (is.null(nrow(X))) X <- matrix(X, nrow = 1)
  m <- 8.6928; s <- 2.4269
  x1 <- 4 * X[,1] - 2
  x2 <- 4 * X[,2] - 2
  a <- 1 + (x1 + x2 + 1)^2 *
    (19 - 14*x1 + 3*x1^2 - 14*x2 + 6*x1*x2 + 3*x2^2)
```

```
  b <- 30 + (2*x1 - 3*x2)^2 *
    (18 - 32*x1 + 12*x1^2 + 48*x2 - 36*x1*x2 + 27*x2^2)
  f <- (log(a*b)- m)/s
  p1 <- pnorm(f) # Transform to probability
  p2 <- sin(pi * X[,1]) * sin(pi * X[,2])
  return(matrix(c(p1/2, p2/2, 1 - (p1+p2)/2), nrow = length(p1)))
}

n <- 100
Xbounds <- matrix(c(0, 0, 1, 1), nrow = 2)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(150, n, replace = TRUE)

# Generate multinomial responses
Y <- t(sapply(1:n, function(i) rmultinom(1, size = m[i], prob = true_pi[i, ])))

# Fit DKP model
model2 <- fit_DKP(X, Y, Xbounds = Xbounds)

# Plot results
plot(model2, n_grid = 50)
```

`predict` *Posterior Prediction for BKP or DKP Models*

Description

Generates posterior summaries from a fitted Beta Kernel Process (BKP) or Dirichlet Kernel Process (DKP) model at new input locations. For BKP models, summaries can be returned either for the latent success probability or for a future success count under the Beta-Binomial posterior predictive distribution. For DKP models, summaries are returned for the class probability vector.

Usage

```
## S3 method for class 'BKP'
predict(
  object,
  Xnew = NULL,
  CI_level = 0.95,
  threshold = 0.5,
  type = c("probability", "count"),
  Mnew = NULL,
  ...
)

## S3 method for class 'DKP'
predict(
```

```
  object,
  Xnew = NULL,
  CI_level = 0.95,
  type = c("probability", "count"),
  Mnew = NULL,
  ...
)
```

Arguments

`object`	An object of class `"BKP"` or `"DKP"`, typically returned by `fit_BKP` or `fit_DKP`.
`Xnew`	A numeric vector or matrix of new input locations at which to generate predictions. If `NULL`, predictions are returned for the training data.
`CI_level`	Numeric between 0 and 1 specifying the credible level for posterior intervals (default `0.95` for 95% credible interval).
`threshold`	Numeric between 0 and 1 specifying the classification threshold for binary predictions based on posterior mean (used only for BKP; default is `0.5`).
`type`	Character string specifying the prediction target for BKP models. The default `"probability"` returns posterior summaries for the latent success probability $\pi(x)$. The option `"count"` returns posterior predictive summaries for a future success count under the Beta-Binomial distribution. This argument is ignored for DKP models unless otherwise implemented.
`Mnew`	Positive integer trial size used only for BKP prediction when `type = "count"`. It can be either a scalar, applied to all prediction points, or a vector with the same length as the number of prediction points. If `Xnew = NULL` and `Mnew = NULL`, the training trial sizes `object$m` are used.
`...`	Additional arguments passed to generic `predict` methods (currently not used; included for S3 method consistency).

Value

A list containing posterior or posterior predictive summaries:

`X` The original training input locations.

`Xnew` The new input locations for prediction. If `NULL`, predictions are returned at the training input locations.

`alpha_n`, `beta_n` Posterior shape parameters:

- BKP: Vectors `alpha_n` and `beta_n` of Beta posterior shape parameters.
- DKP: Matrix `alpha_n` of Dirichlet posterior concentration parameters, with rows corresponding to input locations and columns to classes.

`mean` Mean of the prediction target:

- BKP with `type = "probability"`: posterior mean of the latent success probability.
- BKP with `type = "count"`: posterior predictive mean of the future success count.
- DKP: matrix of posterior mean class probabilities.

`variance` Variance of the prediction target:

- BKP with `type = "probability"`: posterior variance of the latent success probability.
- BKP with `type = "count"`: posterior predictive variance of the future success count.
- DKP: matrix of posterior variances for class probabilities.

`lower` Lower bound of the credible interval:

- BKP with `type = "probability"`: lower Beta posterior quantile for the latent success probability.
- BKP with `type = "count"`: lower Beta-Binomial posterior predictive quantile for the future success count.
- DKP: matrix of lower posterior quantiles for class probabilities.

`upper` Upper bound of the credible interval:

- BKP with `type = "probability"`: upper Beta posterior quantile for the latent success probability.
- BKP with `type = "count"`: upper Beta-Binomial posterior predictive quantile for the future success count.
- DKP: matrix of upper posterior quantiles for class probabilities.

`class` Predicted label, where applicable:

- BKP: binary class label based on posterior mean and `threshold`, only for binary data with `m = 1` and `type = "probability"`.
- DKP: predicted class label with the highest posterior mean probability.

`CI_level` The specified credible interval level.

`type` The prediction target used for BKP prediction.

`Mnew` The trial sizes used for count prediction, returned only when `type = "count"`.

References

Zhao J, Qing K, Xu J (2025). *BKP: An R Package for Beta Kernel Process Modeling*. arXiv. https://doi.org/10.48550/arXiv.2508.10447.

See Also

`fit_BKP` and `fit_DKP` for model fitting; `plot.BKP` and `plot.DKP` for visualization of fitted models.

Examples

```
# ============================================================ #
# ========================= BKP Examples ===================== #
# ============================================================ #

#-------------------------- 1D Example --------------------------
set.seed(123)

# Define true success probability function
true_pi_fun <- function(x) {
  (1 + exp(-x^2) * cos(10 * (1 - exp(-x)) / (1 + exp(-x)))) / 2
}
```

```
n <- 30
Xbounds <- matrix(c(-2,2), nrow=1)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(100, n, replace = TRUE)
y <- rbinom(n, size = m, prob = true_pi)

# Fit BKP model
model1 <- fit_BKP(X, y, m, Xbounds=Xbounds)

# Prediction on training data
predict(model1)

# Prediction on new data
Xnew <- matrix(seq(-2, 2, length = 10), ncol=1) #new data points
predict(model1, Xnew = Xnew)

# Posterior predictive summaries for future success counts
Mnew <- sample(100, nrow(Xnew), replace = TRUE)
predict(model1, Xnew = Xnew, type = "count", Mnew = Mnew)

#-------------------------- 2D Example ---------------------------
set.seed(123)

# Define 2D latent function and probability transformation
true_pi_fun <- function(X) {
  if(is.null(nrow(X))) X <- matrix(X, nrow=1)
  m <- 8.6928
  s <- 2.4269
  x1 <- 4*X[,1]- 2
  x2 <- 4*X[,2]- 2
  a <- 1 + (x1 + x2 + 1)^2 *
    (19- 14*x1 + 3*x1^2- 14*x2 + 6*x1*x2 + 3*x2^2)
  b <- 30 + (2*x1- 3*x2)^2 *
    (18- 32*x1 + 12*x1^2 + 48*x2- 36*x1*x2 + 27*x2^2)
  f <- log(a*b)
  f <- (f- m)/s
  return(pnorm(f))  # Transform to probability
}

n <- 100
Xbounds <- matrix(c(0, 0, 1, 1), nrow = 2)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(100, n, replace = TRUE)
y <- rbinom(n, size = m, prob = true_pi)

# Fit BKP model
model2 <- fit_BKP(X, y, m, Xbounds=Xbounds)

# Prediction on training data
predict(model2)
```

```
# Prediction on new data
x1 <- seq(Xbounds[1,1], Xbounds[1,2], length.out = 10)
x2 <- seq(Xbounds[2,1], Xbounds[2,2], length.out = 10)
Xnew <- expand.grid(x1 = x1, x2 = x2)
predict(model2, Xnew = Xnew)

# ============================================================ #
# ======================== DKP Examples ======================= #
# ============================================================ #

#-------------------------- 1D Example --------------------------
set.seed(123)

# Define true class probability function (3-class)
true_pi_fun <- function(X) {
  p1 <- 1/(1+exp(-3*X))
  p2 <- (1 + exp(-X^2) * cos(10 * (1 - exp(-X)) / (1 + exp(-X)))) / 2
  return(matrix(c(p1/2, p2/2, 1 - (p1+p2)/2), nrow = length(p1)))
}

n <- 30
Xbounds <- matrix(c(-2, 2), nrow = 1)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(150, n, replace = TRUE)

# Generate multinomial responses
Y <- t(sapply(1:n, function(i) rmultinom(1, size = m[i], prob = true_pi[i, ])))

# Fit DKP model
model1 <- fit_DKP(X, Y, Xbounds = Xbounds)

# Prediction on training data
predict(model1)

# Prediction on new data
Xnew = matrix(seq(-2, 2, length = 10), ncol=1) #new data points
predict(model1, Xnew)


#-------------------------- 2D Example --------------------------
set.seed(123)

# Define latent function and transform to 3-class probabilities
true_pi_fun <- function(X) {
  if (is.null(nrow(X))) X <- matrix(X, nrow = 1)
  m <- 8.6928; s <- 2.4269
  x1 <- 4 * X[,1] - 2
  x2 <- 4 * X[,2] - 2
  a <- 1 + (x1 + x2 + 1)^2 *
    (19 - 14*x1 + 3*x1^2 - 14*x2 + 6*x1*x2 + 3*x2^2)
  b <- 30 + (2*x1 - 3*x2)^2 *
    (18 - 32*x1 + 12*x1^2 + 48*x2 - 36*x1*x2 + 27*x2^2)
```

```
  f <- (log(a*b)- m)/s
  p1 <- pnorm(f) # Transform to probability
  p2 <- sin(pi * X[,1]) * sin(pi * X[,2])
  return(matrix(c(p1/2, p2/2, 1 - (p1+p2)/2), nrow = length(p1)))
}

n <- 100
Xbounds <- matrix(c(0, 0, 1, 1), nrow = 2)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(150, n, replace = TRUE)

# Generate multinomial responses
Y <- t(sapply(1:n, function(i) rmultinom(1, size = m[i], prob = true_pi[i, ])))

# Fit DKP model
model2 <- fit_DKP(X, Y, Xbounds = Xbounds)

# Prediction on training data
predict(model2)

# Prediction on new data
x1 <- seq(Xbounds[1,1], Xbounds[1,2], length.out = 10)
x2 <- seq(Xbounds[2,1], Xbounds[2,2], length.out = 10)
Xnew <- expand.grid(x1 = x1, x2 = x2)
predict(model2, Xnew)
```

`predict.TwinBKP` *Predict from a Fitted TwinBKP Model*

Description

Prediction method for TwinBKP. For each prediction point, the function runs:

1. **Local point selection**: use a kd-tree to find `l_nums` nearest neighbors from the training set as local points.
2. **Wendland kernel hyperparameter**: compute the coverage radius $\hat{\theta}_l$ using Equation (14) as the single local-kernel hyperparameter.
3. **Validation set**: randomly sample `v_nums = 2 * g_nums` points from non-global points.
4. **Mixing weight** λ: minimize the loss of the mixed kernel $\lambda K_g + (1-\lambda)K_l$ on the validation set to obtain the optimal λ.
5. **Posterior prediction**: perform BKP posterior updating with the mixed kernel and return posterior mean/variance/confidence interval.

Usage

```
## S3 method for class 'TwinBKP'
predict(
  object,
  Xnew,
  l_nums = NULL,
  v_nums = NULL,
  CI_level = 0.95,
  threshold = 0.5,
  return_type = c("probability", "count"),
  n_trials = NULL,
  ...
)
```

Arguments

object	A "TwinBKP" object returned by fit_TwinBKP().
Xnew	Prediction input matrix (unnormalized), with dimension n_new x d.
l_nums	Positive integer. Number of local neighbors per prediction point. If NULL (default), it is set to max(25, 3 * d), where d is the input dimensionality.
v_nums	Positive integer. Validation set size. If NULL (default), it is set to 2 * object$g_nums.
CI_level	Numeric confidence level. Default is 0.95.
threshold	Classification threshold (only used for classification with m = 1). Default is 0.5.
return_type	Character string specifying prediction scale: "probability" (default) or "count" (Beta-Binomial success-count prediction).
n_trials	Positive integer total trial count used only when return_type = "count". If NULL, an error is thrown in count mode.
...	Unused.

Value

A list of class "predict_TwinBKP" containing:

X Original training input matrix.

mean Posterior predictive mean matrix of size n_new x 1.

variance Posterior predictive variance matrix of size n_new x 1.

lower, upper Lower/upper confidence interval matrices of size n_new x 1.

alpha_n, beta_n Posterior Beta shape parameters at prediction points.

lambda Mixing weight λ for each prediction point (length n_new).

theta_l Local kernel hyperparameter for each prediction point (length n_new).

local_idx List of selected local training-row indices for each prediction point.

Xnew Original prediction coordinates.

Xnew_norm Normalized prediction coordinates.

See Also

fit_TwinBKP

Examples

```
# ============================================================ #
# ====================== TwinBKP Examples ==================== #
# ============================================================ #

#-------------------------- 1D Example ---------------------------
set.seed(123)

# Define true success probability function
true_pi_fun <- function(x) {
  (1 + exp(-x^2) * cos(10 * (1 - exp(-x)) / (1 + exp(-x)))) / 2
}

n <- 100
Xbounds <- matrix(c(-2, 2), nrow = 1)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(100, n, replace = TRUE)
y <- rbinom(n, size = m, prob = true_pi)

# Fit TwinBKP model (global stage only)
model1 <- fit_TwinBKP(X, y, m, Xbounds = Xbounds, g_nums = 10)

# Prediction on new data
Xnew <- matrix(seq(-2, 2, length.out = 10), ncol = 1)
predict(model1, Xnew = Xnew, l_nums = 10)


#-------------------------- 2D Example ---------------------------
set.seed(123)

# Define 2D latent function and probability transformation
true_pi_fun <- function(X) {
  if (is.null(nrow(X))) X <- matrix(X, nrow = 1)
  m <- 8.6928
  s <- 2.4269
  x1 <- 4 * X[, 1] - 2
  x2 <- 4 * X[, 2] - 2
  a <- 1 + (x1 + x2 + 1)^2 *
    (19 - 14 * x1 + 3 * x1^2 - 14 * x2 + 6 * x1 * x2 + 3 * x2^2)
  b <- 30 + (2 * x1 - 3 * x2)^2 *
    (18 - 32 * x1 + 12 * x1^2 + 48 * x2 - 36 * x1 * x2 + 27 * x2^2)
  f <- log(a * b)
  f <- (f - m) / s
  pnorm(f)
}

n <- 50
```

```
Xbounds <- matrix(c(0, 0, 1, 1), nrow = 2)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(100, n, replace = TRUE)
y <- rbinom(n, size = m, prob = true_pi)

# Fit TwinBKP model
model2 <- fit_TwinBKP(X, y, m, Xbounds = Xbounds, g_nums = 12)

# Prediction on new data
x1 <- seq(Xbounds[1, 1], Xbounds[1, 2], length.out = 8)
x2 <- seq(Xbounds[2, 1], Xbounds[2, 2], length.out = 8)
Xnew <- expand.grid(x1 = x1, x2 = x2)
predict(model2, Xnew = Xnew, l_nums = 12)
```

`predict.TwinDKP` *Predict from a Fitted TwinDKP Model*

Description

Prediction method for TwinDKP. For each prediction point, the function runs:

1. Local point selection via kNN with `l_nums` neighbors.
2. Local Wendland kernel construction with hyperparameter `theta_l`.
3. Validation subset sampling with size `v_nums`.
4. Mixing-weight optimization for $\lambda K_g + (1 - \lambda) K_l$.
5. Posterior prediction for class probabilities and uncertainty.

Usage

```
## S3 method for class 'TwinDKP'
predict(
  object,
  Xnew = NULL,
  l_nums = NULL,
  v_nums = NULL,
  CI_level = 0.95,
  return_type = c("probability", "count"),
  n_trials = NULL,
  ...
)
```

Arguments

object	A "TwinDKP" object returned by fit_TwinDKP().
Xnew	Prediction input matrix (unnormalized), with dimension n_new x d. If NULL, use training X.
l_nums	Positive integer. Number of local neighbors per prediction point. If NULL (default), set to max(25, 3 * d).
v_nums	Positive integer. Validation set size. If NULL (default), set to 2 * object$g_nums.
CI_level	Numeric confidence level in (0,1). Default is 0.95.
return_type	Character string specifying prediction scale: "probability" (default) or "count" (Beta-Binomial marginal success-count prediction per class).
n_trials	Positive integer total trial count used only when return_type = "count". If NULL, an error is thrown in count mode.
...	Unused.

Value

A list of class "predict_TwinDKP" containing:

X Original training input matrix.

Xnew Prediction input matrix used.

Xnew_norm Normalized prediction matrix.

alpha_n Posterior Dirichlet parameters at Xnew (n_new x q).

mean Posterior mean class probabilities (n_new x q).

variance Posterior variances (n_new x q).

lower, upper Marginal credible interval bounds (n_new x q).

lambda Optimized mixing weights (length n_new).

theta_l Local kernel hyperparameter (length n_new).

local_idx List of local training indices for each point.

CI_level Credible interval level.

l_nums, v_nums, g_nums Effective tuning sizes used.

class Predicted class labels when rowSums(Y)=1.

See Also

fit_TwinDKP, predict.DKP

print *Print Methods for BKP and DKP Objects*

Description

Provides formatted console output for fitted BKP/DKP model objects, their summaries, predictions, and simulations. The following specialized methods are supported:

- `print.BKP`, `print.DKP` – display fitted model objects.
- `print.summary_BKP`, `print.summary_DKP` – display model summaries.
- `print.predict_BKP`, `print.predict_DKP` – display posterior predictive results.
- `print.simulate_BKP`, `print.simulate_DKP` – display posterior simulations.

Usage

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

## S3 method for class 'summary_BKP'
print(x, ...)

## S3 method for class 'predict_BKP'
print(x, ...)

## S3 method for class 'simulate_BKP'
print(x, ...)

## S3 method for class 'DKP'
print(x, ...)

## S3 method for class 'summary_DKP'
print(x, ...)

## S3 method for class 'predict_DKP'
print(x, ...)

## S3 method for class 'simulate_DKP'
print(x, ...)

## S3 method for class 'TwinBKP'
print(x, ...)

## S3 method for class 'summary_TwinBKP'
print(x, ...)

## S3 method for class 'simulate_TwinBKP'
```

```
print(x, ...)

## S3 method for class 'TwinDKP'
print(x, ...)

## S3 method for class 'summary_TwinDKP'
print(x, ...)

## S3 method for class 'predict_TwinDKP'
print(x, ...)

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

Arguments

x	An object of class "BKP" or "DKP", or a derived object such as summary, predict, or simulate.
...	Additional arguments passed to the generic print method (currently unused; included for S3 consistency).

Value

Invisibly returns the input object. Called for the side effect of printing human-readable summaries to the console.

References

Zhao J, Qing K, Xu J (2025). *BKP: An R Package for Beta Kernel Process Modeling*. arXiv. https://doi.org/10.48550/arXiv.2508.10447.

See Also

fit_BKP, fit_DKP for model fitting; summary.BKP, summary.DKP for model summaries; predict.BKP, predict.DKP for posterior prediction; simulate.BKP, simulate.DKP for posterior simulations.

Examples

```
# ============================================================ #
# ========================= BKP Examples ===================== #
# ============================================================ #

#-------------------------- 1D Example --------------------------
set.seed(123)

# Define true success probability function
true_pi_fun <- function(x) {
  (1 + exp(-x^2) * cos(10 * (1 - exp(-x)) / (1 + exp(-x)))) / 2
}
```

```
n <- 30
Xbounds <- matrix(c(-2,2), nrow=1)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(100, n, replace = TRUE)
y <- rbinom(n, size = m, prob = true_pi)

# Fit BKP model
model1 <- fit_BKP(X, y, m, Xbounds=Xbounds)
print(model1)                    # fitted object
print(summary(model1))           # summary
print(predict(model1))           # predictions
print(simulate(model1, nsim=3))  # posterior simulations

#-------------------------- 2D Example --------------------------
set.seed(123)

# Define 2D latent function and probability transformation
true_pi_fun <- function(X) {
  if(is.null(nrow(X))) X <- matrix(X, nrow=1)
  m <- 8.6928
  s <- 2.4269
  x1 <- 4*X[,1]- 2
  x2 <- 4*X[,2]- 2
  a <- 1 + (x1 + x2 + 1)^2 *
    (19- 14*x1 + 3*x1^2- 14*x2 + 6*x1*x2 + 3*x2^2)
  b <- 30 + (2*x1- 3*x2)^2 *
    (18- 32*x1 + 12*x1^2 + 48*x2- 36*x1*x2 + 27*x2^2)
  f <- log(a*b)
  f <- (f- m)/s
  return(pnorm(f))  # Transform to probability
}

n <- 100
Xbounds <- matrix(c(0, 0, 1, 1), nrow = 2)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(100, n, replace = TRUE)
y <- rbinom(n, size = m, prob = true_pi)

# Fit BKP model
model2 <- fit_BKP(X, y, m, Xbounds=Xbounds)
print(model2)                    # fitted object
print(summary(model2))           # summary
print(predict(model2))           # predictions
print(simulate(model2, nsim=3))  # posterior simulations

# ============================================================ #
# ======================== DKP Examples ======================== #
# ============================================================ #

#-------------------------- 1D Example --------------------------
set.seed(123)
```

```
# Define true class probability function (3-class)
true_pi_fun <- function(X) {
  p1 <- 1/(1+exp(-3*X))
  p2 <- (1 + exp(-X^2) * cos(10 * (1 - exp(-X)) / (1 + exp(-X)))) / 2
  return(matrix(c(p1/2, p2/2, 1 - (p1+p2)/2), nrow = length(p1)))
}

n <- 30
Xbounds <- matrix(c(-2, 2), nrow = 1)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(150, n, replace = TRUE)

# Generate multinomial responses
Y <- t(sapply(1:n, function(i) rmultinom(1, size = m[i], prob = true_pi[i, ])))

# Fit DKP model
model1 <- fit_DKP(X, Y, Xbounds = Xbounds)
print(model1)                    # fitted object
print(summary(model1))           # summary
print(predict(model1))           # predictions
print(simulate(model1, nsim=3))  # posterior simulations


#-------------------------- 2D Example ---------------------------
set.seed(123)

# Define latent function and transform to 3-class probabilities
true_pi_fun <- function(X) {
  if (is.null(nrow(X))) X <- matrix(X, nrow = 1)
  m <- 8.6928; s <- 2.4269
  x1 <- 4 * X[,1] - 2
  x2 <- 4 * X[,2] - 2
  a <- 1 + (x1 + x2 + 1)^2 *
    (19 - 14*x1 + 3*x1^2 - 14*x2 + 6*x1*x2 + 3*x2^2)
  b <- 30 + (2*x1 - 3*x2)^2 *
    (18 - 32*x1 + 12*x1^2 + 48*x2 - 36*x1*x2 + 27*x2^2)
  f <- (log(a*b)- m)/s
  p1 <- pnorm(f) # Transform to probability
  p2 <- sin(pi * X[,1]) * sin(pi * X[,2])
  return(matrix(c(p1/2, p2/2, 1 - (p1+p2)/2), nrow = length(p1)))
}

n <- 100
Xbounds <- matrix(c(0, 0, 1, 1), nrow = 2)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(150, n, replace = TRUE)

# Generate multinomial responses
Y <- t(sapply(1:n, function(i) rmultinom(1, size = m[i], prob = true_pi[i, ])))
```

```
# Fit DKP model
model2 <- fit_DKP(X, Y, Xbounds = Xbounds)
print(model2)                    # fitted object
print(summary(model2))           # summary
print(predict(model2))           # predictions
print(simulate(model2, nsim=3))  # posterior simulations
```

---

quantile *Posterior Quantiles from a Fitted BKP or DKP Model*

Description

Compute posterior quantiles from a fitted BKP or DKP model. For a BKP object, this returns the posterior quantiles of the positive class probability. For a DKP object, this returns posterior quantiles for each class probability.

Usage

```
## S3 method for class 'BKP'
quantile(x, probs = c(0.025, 0.5, 0.975), ...)

## S3 method for class 'DKP'
quantile(x, probs = c(0.025, 0.5, 0.975), ...)

## S3 method for class 'TwinBKP'
quantile(
  x,
  probs = c(0.025, 0.5, 0.975),
  Xnew = NULL,
  n_grid = 1000,
  l_nums = NULL,
  v_nums = NULL,
  ...
)

## S3 method for class 'TwinDKP'
quantile(
  x,
  probs = c(0.025, 0.5, 0.975),
  Xnew = NULL,
  n_grid = 1000,
  l_nums = NULL,
  v_nums = NULL,
  ...
)
```

Arguments

x	An object of class BKP or DKP, typically the result of a call to fit_BKP or fit_DKP.
probs	Numeric vector of probabilities specifying which posterior quantiles to return. Defaults to c(0.025, 0.5, 0.975).
...	Additional arguments (currently unused).
Xnew	Optional prediction points for quantile evaluation.
n_grid	Number of generated points when Xnew = NULL. Default is 1000.
l_nums	Optional local neighbor count passed to predict.TwinDKP().
v_nums	Optional validation size passed to predict.TwinDKP().

Details

For a BKP model, posterior quantiles are computed from the Beta Kernel Process for the positive class probability. For a DKP model, posterior quantiles for each class are approximated using the Beta approximation of the marginal distributions of the posterior Dirichlet distribution.

For TwinBKP, posterior quantiles are computed by first calling predict.TwinBKP() at Xnew, then applying qbeta() to returned alpha_n and beta_n. If Xnew = NULL, the method generates n_grid points in Xbounds via Latin hypercube sampling and then performs prediction.

For TwinDKP, posterior quantiles are prediction-driven. The method first calls predict.TwinDKP() at Xnew and then computes class-wise marginal quantiles via Beta approximation: for class j, use qbeta(probs, alpha_{ij}, sum(alpha_i)-alpha_{ij}).

If Xnew = NULL, n_grid points are generated in x$Xbounds by Latin hypercube sampling.

Value

For BKP: a numeric vector (if length(probs) = 1) or a numeric matrix (if length(probs) > 1) of posterior quantiles. Rows correspond to observations, and columns correspond to the requested probabilities.

For DKP: a numeric matrix (if length(probs) = 1) or a 3D array (if length(probs) > 1) of posterior quantiles. Dimensions correspond to observations × classes × probabilities.

See Also

fit_BKP, fit_DKP for model fitting.

Examples

```
# -------------------------- BKP --------------------------
set.seed(123)

# Define true success probability function
true_pi_fun <- function(x) {
  (1 + exp(-x^2) * cos(10 * (1 - exp(-x)) / (1 + exp(-x)))) / 2
}
```

```
n <- 30
Xbounds <- matrix(c(-2, 2), nrow = 1)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(100, n, replace = TRUE)
y <- rbinom(n, size = m, prob = true_pi)

# Fit BKP model
model <- fit_BKP(X, y, m, Xbounds = Xbounds)

# Extract posterior quantiles
quantile(model)

# -------------------------- DKP --------------------------
#' set.seed(123)

# Define true class probability function (3-class)
true_pi_fun <- function(X) {
  p1 <- 1/(1+exp(-3*X))
  p2 <- (1 + exp(-X^2) * cos(10 * (1 - exp(-X)) / (1 + exp(-X)))) / 2
  return(matrix(c(p1/2, p2/2, 1 - (p1+p2)/2), nrow = length(p1)))
}

n <- 30
Xbounds <- matrix(c(-2, 2), nrow = 1)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(100, n, replace = TRUE)

# Generate multinomial responses
Y <- t(sapply(1:n, function(i) rmultinom(1, size = m[i], prob = true_pi[i, ])))

# Fit DKP model
model <- fit_DKP(X, Y, Xbounds = Xbounds)

# Extract posterior quantiles
quantile(model)
```

`simulate` *Simulate from a Fitted BKP or DKP Model*

Description

Generates random draws from the posterior predictive distribution of a fitted BKP or DKP model at specified input locations.

For BKP models, posterior samples are generated from Beta distributions characterizing success probabilities. Optionally, binary class labels can be derived by applying a user-specified classification threshold.

For DKP models, posterior samples are generated from Dirichlet distributions characterizing class probabilities. If training responses are single-label (i.e., one-hot encoded), class labels may additionally be assigned using the maximum a posteriori (MAP) rule.

Usage

```
## S3 method for class 'BKP'
simulate(object, nsim = 1, seed = NULL, Xnew = NULL, threshold = NULL, ...)

## S3 method for class 'DKP'
simulate(object, nsim = 1, seed = NULL, Xnew = NULL, ...)

## S3 method for class 'TwinBKP'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  Xnew = NULL,
  n_grid = 1000,
  l_nums = NULL,
  v_nums = NULL,
  threshold = NULL,
  ...
)

## S3 method for class 'TwinDKP'
simulate(
  object,
  nsim = 1,
  seed = NULL,
  Xnew = NULL,
  n_grid = 1000,
  l_nums = NULL,
  v_nums = NULL,
  ...
)
```

Arguments

`object`	An object of class `"BKP"` or `"DKP"`, typically returned by `fit_BKP` or `fit_DKP`.
`nsim`	Number of posterior samples to generate (default = 1).
`seed`	Optional integer seed for reproducibility.
`Xnew`	A numeric matrix or vector of new input locations at which simulations are generated.
`threshold`	Classification threshold for binary decisions (BKP only). When specified, posterior draws exceeding the threshold are classified as 1, and those below as 0. The default is `NULL`.
`...`	Additional arguments (currently unused).

n_grid	Number of generated points when Xnew = NULL. Default is 1000.
l_nums	Optional local neighbor count passed to predict.TwinDKP().
v_nums	Optional validation size passed to predict.TwinDKP().

Details

For TwinBKP, posterior simulation is prediction-driven:

1. Build/receive Xnew.
2. Call predict.TwinBKP() to obtain alpha_n, beta_n.
3. Draw nsim samples from Beta(alpha_n, beta_n) at each point.

If Xnew = NULL, n_grid points are generated in Xbounds (default 1000) and used for simulation.

For TwinDKP, posterior simulation is prediction-driven:

1. Build/receive Xnew.
2. Call predict.TwinDKP() to obtain alpha_n.
3. Draw nsim samples from class-wise Dirichlet posteriors.

If Xnew = NULL, n_grid points are generated in Xbounds (default 1000) and used for simulation.

Value

A list with the following components:

samples For **BKP**: A numeric matrix of size nrow(Xnew) × nsim, where each column corresponds to one posterior draw of success probabilities.
For **DKP**: A numeric array of dimension nsim × q × nrow(Xnew), containing simulated class probabilities from Dirichlet posteriors, where q is the number of classes.

mean For **BKP**: A numeric vector of posterior mean success probabilities at each Xnew.
For **DKP**: A numeric matrix of dimension nrow(Xnew) × q, containing posterior mean class probabilities.

class For **BKP**: An integer matrix of dimension nrow(Xnew) × nsim, indicating simulated binary class labels (0/1), returned when threshold is specified.
For **DKP**: An integer matrix of dimension nrow(Xnew) × nsim, where each entry corresponds to a MAP-predicted class label, returned only when training data is single-label.

X The training input matrix used to fit the BKP/DKP model.

Xnew The new input locations at which simulations are generated.

threshold The classification threshold used for generating binary class labels (if provided).

References

Zhao J, Qing K, Xu J (2025). *BKP: An R Package for Beta Kernel Process Modeling*. arXiv. https://doi.org/10.48550/arXiv.2508.10447.

See Also

fit_BKP, fit_DKP for model fitting; predict.BKP, predict.DKP for posterior prediction.

Examples

```
## -------------------- BKP --------------------
set.seed(123)

# Define true success probability function
true_pi_fun <- function(x) {
  (1 + exp(-x^2) * cos(10 * (1 - exp(-x)) / (1 + exp(-x)))) / 2
}

n <- 30
Xbounds <- matrix(c(-2,2), nrow=1)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(100, n, replace = TRUE)
y <- rbinom(n, size = m, prob = true_pi)

# Fit BKP model
model <- fit_BKP(X, y, m, Xbounds=Xbounds)

# Simulate 5 posterior draws of success probabilities
Xnew <- matrix(seq(-2, 2, length.out = 5), ncol = 1)
simulate(model, Xnew = Xnew, nsim = 5)

# Simulate binary classifications (threshold = 0.5)
simulate(model, Xnew = Xnew, nsim = 5, threshold = 0.5)

## -------------------- DKP --------------------
set.seed(123)

# Define true class probability function (3-class)
true_pi_fun <- function(X) {
  p1 <- 1/(1+exp(-3*X))
  p2 <- (1 + exp(-X^2) * cos(10 * (1 - exp(-X)) / (1 + exp(-X)))) / 2
  return(matrix(c(p1/2, p2/2, 1 - (p1+p2)/2), nrow = length(p1)))
}

n <- 30
Xbounds <- matrix(c(-2, 2), nrow = 1)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(150, n, replace = TRUE)

# Generate multinomial responses
Y <- t(sapply(1:n, function(i) rmultinom(1, size = m[i], prob = true_pi[i, ])))

# Fit DKP model
model <- fit_DKP(X, Y, Xbounds = Xbounds)

# Simulate 5 draws from posterior Dirichlet distributions at new point
Xnew <- matrix(seq(-2, 2, length.out = 5), ncol = 1)
simulate(model, Xnew = Xnew, nsim = 5)
```

summary *Summary of a Fitted BKP or DKP Model*

Description

Provides a structured summary of a fitted Beta Kernel Process (BKP) or Dirichlet Kernel Process (DKP) model. This function reports the model configuration, prior specification, kernel settings, and key posterior quantities, giving users a concise overview of the fitting results.

Usage

```
## S3 method for class 'BKP'
summary(object, ...)

## S3 method for class 'DKP'
summary(object, ...)

## S3 method for class 'TwinBKP'
summary(object, ...)

## S3 method for class 'TwinDKP'
summary(object, ...)
```

Arguments

object	An object of class "BKP" (from fit_BKP) or "DKP" (from fit_DKP).
...	Additional arguments passed to the generic summary method (currently not used).

Details

For TwinBKP, summary() reports global-stage information and posterior summaries on the global support points selected by Twining. This mirrors the role of summary.BKP, while respecting that TwinBKP fits global hyperparameters first and performs full prediction behavior at predict.TwinBKP() stage.

For TwinDKP, summary() focuses on global-stage tuning information and posterior summaries computed on Twining support points. Full prediction on arbitrary inputs should be obtained via predict.TwinDKP().

Value

A list containing key summaries of the fitted model:

n_obs Number of training observations.

input_dim Input dimensionality (number of columns in X).

kernel Kernel type used in the model.

theta_opt Estimated kernel hyperparameters.

loss Loss function type used in the model.

`loss_min` Minimum value of the loss function achieved.

`prior` Prior type used (e.g., "noninformative", "fixed", "adaptive").

`r0` Prior precision parameter.

`p0` Prior mean parameter.

`post_mean` Posterior mean estimates. For BKP: posterior mean success probabilities at training points. For DKP: posterior mean class probabilities ($n_{\text{obs}} \times q$).

`post_var` Posterior variance estimates. For BKP: variance of success probabilities. For DKP: variance for each class probability.

`n_class` (Only for DKP) Number of classes in the response.

References

Zhao J, Qing K, Xu J (2025). *BKP: An R Package for Beta Kernel Process Modeling*. arXiv. https://doi.org/10.48550/arXiv.2508.10447.

See Also

`fit_BKP`, `fit_DKP` for model fitting.

Examples

```
# ============================================================ #
# ======================== BKP Examples ====================== #
# ============================================================ #

#-------------------------- 1D Example --------------------------
set.seed(123)

# Define true success probability function
true_pi_fun <- function(x) {
  (1 + exp(-x^2) * cos(10 * (1 - exp(-x)) / (1 + exp(-x)))) / 2
}

n <- 30
Xbounds <- matrix(c(-2,2), nrow=1)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(100, n, replace = TRUE)
y <- rbinom(n, size = m, prob = true_pi)

# Fit BKP model
model1 <- fit_BKP(X, y, m, Xbounds=Xbounds)
summary(model1)


#-------------------------- 2D Example --------------------------
set.seed(123)

# Define 2D latent function and probability transformation
true_pi_fun <- function(X) {
```

```
  if(is.null(nrow(X))) X <- matrix(X, nrow=1)
  m <- 8.6928
  s <- 2.4269
  x1 <- 4*X[,1]- 2
  x2 <- 4*X[,2]- 2
  a <- 1 + (x1 + x2 + 1)^2 *
    (19- 14*x1 + 3*x1^2- 14*x2 + 6*x1*x2 + 3*x2^2)
  b <- 30 + (2*x1- 3*x2)^2 *
    (18- 32*x1 + 12*x1^2 + 48*x2- 36*x1*x2 + 27*x2^2)
  f <- log(a*b)
  f <- (f- m)/s
  return(pnorm(f))  # Transform to probability
}

n <- 100
Xbounds <- matrix(c(0, 0, 1, 1), nrow = 2)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(100, n, replace = TRUE)
y <- rbinom(n, size = m, prob = true_pi)

# Fit BKP model
model2 <- fit_BKP(X, y, m, Xbounds=Xbounds)
summary(model2)

# ============================================================ #
# ======================== DKP Examples ======================= #
# ============================================================ #

#-------------------------- 1D Example ---------------------------
set.seed(123)

# Define true class probability function (3-class)
true_pi_fun <- function(X) {
  p1 <- 1/(1+exp(-3*X))
  p2 <- (1 + exp(-X^2) * cos(10 * (1 - exp(-X)) / (1 + exp(-X)))) / 2
  return(matrix(c(p1/2, p2/2, 1 - (p1+p2)/2), nrow = length(p1)))
}

n <- 30
Xbounds <- matrix(c(-2, 2), nrow = 1)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(150, n, replace = TRUE)

# Generate multinomial responses
Y <- t(sapply(1:n, function(i) rmultinom(1, size = m[i], prob = true_pi[i, ])))

# Fit DKP model
model1 <- fit_DKP(X, Y, Xbounds = Xbounds)
summary(model1)
```

```
#-------------------------- 2D Example ---------------------------
set.seed(123)

# Define latent function and transform to 3-class probabilities
true_pi_fun <- function(X) {
  if (is.null(nrow(X))) X <- matrix(X, nrow = 1)
  m <- 8.6928; s <- 2.4269
  x1 <- 4 * X[,1] - 2
  x2 <- 4 * X[,2] - 2
  a <- 1 + (x1 + x2 + 1)^2 *
    (19 - 14*x1 + 3*x1^2 - 14*x2 + 6*x1*x2 + 3*x2^2)
  b <- 30 + (2*x1 - 3*x2)^2 *
    (18 - 32*x1 + 12*x1^2 + 48*x2 - 36*x1*x2 + 27*x2^2)
  f <- (log(a*b)- m)/s
  p1 <- pnorm(f) # Transform to probability
  p2 <- sin(pi * X[,1]) * sin(pi * X[,2])
  return(matrix(c(p1/2, p2/2, 1 - (p1+p2)/2), nrow = length(p1)))
}

n <- 100
Xbounds <- matrix(c(0, 0, 1, 1), nrow = 2)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
m <- sample(150, n, replace = TRUE)

# Generate multinomial responses
Y <- t(sapply(1:n, function(i) rmultinom(1, size = m[i], prob = true_pi[i, ])))

# Fit DKP model
model2 <- fit_DKP(X, Y, Xbounds = Xbounds)
summary(model2)
```

Index