Package: BKP (via r-universe)

October 4, 2026

Title Beta Kernel Process Modeling

Version 0.3.1.9000

Maintainer Jiangyan Zhao <zhaojy2017@126.com>

Description Implements the Beta Kernel Process (BKP) for nonparametric modeling of covariate-dependent binomial probabilities, and the Dirichlet Kernel Process (DKP) for categorical or multinomial response data. Scalable global-local approximations are provided through TwinBKP and TwinDKP, using twinning-selected global subsets and local nearest-neighbour updates. Functions are included for model fitting, predictive inference with uncertainty quantification, posterior simulation, and visualization in one- and two-dimensional input spaces. Gaussian, Matern 5/2, Matern 3/2, and Wendland kernels are supported, with hyperparameters selected by multi-start derivative-free optimization. For more details, see Zhao, Qing, and Xu (2025) <doi:10.48550/arXiv.2508.10447>.

License GPL (>= 3)

Copyright See inst/COPYRIGHTS

URL https://github.com/Jiangyan-Zhao/BKP, https://jiangyan-zhao.github.io/BKP-website, https://github.com/Jiangyan-Zhao/BKP-paper

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

Encoding UTF-8

NeedsCompilation yes

Roxygen list(markdown = TRUE)

Depends R (>= 4.1.0)

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

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 2026-10-04 11:41:59 UTC

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

RemoteRef HEAD

RemoteSha ac575244e9541b4b11237f1b5f259451f6299f8e

Contents

BKP-package . . . 2
fit_BKP . . . 4
fit_DKP . . . 7
fit_TwinBKP . . . 11
fit_TwinDKP . . . 16
fitted . . . 21
get_prior . . . 24
kernel_matrix . . . 27
loss_fun . . . 29
parameter . . . 31
plot . . . 34
predict . . . 40
print . . . 46
quantile . . . 49
simulate . . . 52
summary . . . 56

Index **59**

`BKP-package` *Beta Kernel Process Modeling*

Description

The **BKP** package provides tools for Bayesian nonparametric modeling of binary/binomial and categorical/multinomial response data using the Beta Kernel Process (BKP), the Dirichlet Kernel Process (DKP), and their scalable global-local approximations, TwinBKP and TwinDKP. These methods estimate latent probability surfaces by combining kernel-based smoothing with conjugate posterior updates.

The package supports model fitting, posterior prediction with uncertainty quantification, posterior simulation, and visualization in one- and two-dimensional input spaces. It also provides flexible prior specification, multiple kernel choices, hyperparameter tuning, and twinning-based global-local computation for scalable BKP and DKP modeling.

Main Functions

Core functionality is organized as follows:

Model fitting Use `fit_BKP` and `fit_DKP` for full BKP and DKP models, and `fit_TwinBKP` and `fit_TwinDKP` for scalable global-local approximations using twinning-selected global subsets and local nearest-neighbour updates.

Prediction Use `predict` with fitted `BKP`, `DKP`, `TwinBKP`, or `TwinDKP` objects for posterior inference at new input locations, including posterior means, variances, and credible intervals. Decision labels, when needed, can be obtained from posterior means or method-specific outputs.

Simulation Use `simulate` with fitted model objects to generate posterior draws of latent success probabilities or class-probability vectors.

Visualization Use `plot` with fitted model objects to visualize 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. TwinBKP and TwinDKP plots can optionally highlight the selected global subset.

Summaries and printing Use `summary` and `print` to summarize or display fitted model objects and associated results.

Extraction Use `fitted`, `parameter`, and `quantile` to extract fitted posterior means, model parameters, and posterior quantiles.

Utilities Use `kernel_matrix` to construct kernel matrices, `get_prior` to construct BKP or DKP prior parameters, and `loss_fun` to evaluate leave-one-out tuning losses.

References

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

Vakayil A, Joseph VR (2024). *A Global-Local Approximation Framework for Large-Scale Gaussian Process Modeling*. Technometrics, 66(2), 295–305. doi:10.1080/00401706.2023.2296451.

Rolland P, Kavis A, Immer 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. https://proceedings.mlr.press/v97/rolland19a.html.

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. doi:10.1002/sam.11241.

Goetschalckx R, Poupart P, Hoey J (2011). *Continuous Correlated Beta Processes*. In Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, IJCAI-11, p. 1269–1274. AAAI Press. https://www.ijcai.org/Proceedings/11/Papers/215.pdf.

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

Description

Fit a Beta Kernel Process (BKP) model for binary or binomial response data. The model estimates a covariate-dependent success probability surface by combining kernel-based smoothing with conjugate Beta posterior updates.

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"),
  isotropic = TRUE,
  theta = NULL,
  loss = c("brier", "log_loss"),
  ess = c("none", "shepard"),
  n_multi_start = NULL,
  n_threads = 1
)
```

Arguments

`X`	A numeric input matrix or data frame of size $n \times d$, where each row is an input or covariate vector.
`y`	A numeric vector of observed success counts of length `n`.
`m`	A numeric vector of total binomial trial counts of length `n`. Each element of `y` must be between 0 and the corresponding element of `m`.
`Xbounds`	Optional $d \times 2$ numeric matrix giving the lower and upper bounds of each input dimension. When supplied, `X` is normalized to $[0, 1]^d$ before fitting. If `NULL`, `X` is assumed to be already normalized to $[0, 1]^d$.
`prior`	Type of prior specification. Options are `"noninformative"` (default), `"fixed"`, and `"adaptive"`.
`r0`	Positive scalar prior precision used when `prior = "fixed"` or `prior = "adaptive"`. Default is `2`.
`p0`	Prior mean used when `prior = "fixed"`. It must be a scalar in $(0, 1)$. The default is the empirical mean `mean(y / m)`.

`kernel` Kernel function used for local weighting. Options are `"gaussian"` (default), `"matern52"`, `"matern32"`, and `"wendland"`.

`isotropic` Logical. If `TRUE` (default), use a single shared lengthscale across input dimensions. If `FALSE`, use separate per-dimension lengthscales.

`theta` Optional positive kernel lengthscale parameter. When `isotropic = TRUE`, this must be a scalar. When `isotropic = FALSE`, this can be either a scalar, which is broadcast to all dimensions, or a numeric vector of length `d`. If `NULL`, the lengthscale parameters are selected by minimizing the specified leave-one-out loss.

`loss` Leave-one-out loss used for kernel hyperparameter tuning. Options are `"brier"` (default) and `"log_loss"`.

`ess` Effective-sample-size calibration for the kernel-weighted data contribution. Use `"none"` (default) for the standard BKP posterior update. Use `"shepard"` to rescale the kernel-weighted data contribution so that its effective trial size is $\rho(\mathbf{x})m_S(\mathbf{x})$, where $m_S(\mathbf{x})$ is a Shepard interpolation of the observed trial sizes on the normalized input scale and $\rho(\mathbf{x}) = \max_i k(\mathbf{x}, \mathbf{x}_i)$. This calibration preserves the kernel-weighted empirical proportion and changes only the data precision, not the prior parameters.

`n_multi_start` Number of initial points used in multi-start derivative-free optimization of the kernel lengthscale parameters. If `NULL`, the default is $10p$, where $p = 1$ for isotropic kernels and $p = d$ for anisotropic kernels.

`n_threads` Number of OpenMP threads used for multi-start optimization. Default is `1`. This argument only affects fitting when `theta = NULL`.

Details

Inputs are normalized to $[0, 1]^d$ using `Xbounds`. For a location $\mathbf{x}$, BKP computes kernel weights $k(\mathbf{x}, \mathbf{x}_i)$ between $\mathbf{x}$ and the training inputs. These weights are used to update a local Beta prior with kernel-weighted binomial counts:

$$\alpha_n(\mathbf{x}) = \alpha_0(\mathbf{x}) + \sum_i k(\mathbf{x}, \mathbf{x}_i) y_i,$$

$$\beta_n(\mathbf{x}) = \beta_0(\mathbf{x}) + \sum_i k(\mathbf{x}, \mathbf{x}_i)(m_i - y_i).$$

If `theta = NULL`, the kernel lengthscale parameters are selected by leave-one-out cross-validation using the specified `loss`. Optimization is performed over log-transformed lengthscales using a multi-start derivative-free search. If `theta` is supplied, optimization is skipped and the model is fitted using the supplied lengthscale parameter.

If `ess = "shepard"`, only the kernel-weighted data contribution is rescaled to match the local effective-sample-size target; the prior parameters are not rescaled. Shepard calibration requires unique training input locations on the normalized input scale.

The returned object stores posterior parameters evaluated at the training inputs. Posterior inference at new inputs is performed by `predict.BKP`.

Value

A list of class "BKP" containing the fitted model, with the following components:

theta_opt Optimized or user-specified kernel lengthscale parameter(s).

kernel Kernel function used.

isotropic Logical flag indicating whether a shared lengthscale or per-dimension lengthscales were used.

loss Loss function used for hyperparameter tuning.

loss_min Loss value at the selected or user-specified lengthscale parameter(s).

X Original input matrix.

Xnorm Input matrix normalized to $[0, 1]^d$.

Xbounds Normalization bounds for each input dimension.

y Observed success counts, stored as a one-column matrix.

m Observed binomial trial counts, stored as a one-column matrix.

prior Prior specification used.

r0 Prior precision parameter.

p0 Prior mean used when prior = "fixed".

alpha0 Prior Beta α shape parameters evaluated at the training inputs.

beta0 Prior Beta β shape parameters evaluated at the training inputs.

alpha_n Posterior Beta α shape parameters evaluated at the training inputs.

beta_n Posterior Beta β shape parameters evaluated at the training inputs.

ess Effective-sample-size calibration method used.

ess_info ESS calibration diagnostics, or NULL when ess = "none".

References

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

See Also

fit_DKP for Dirichlet Kernel Process modeling of categorical or multinomial responses; fit_TwinBKP for the scalable global-local TwinBKP approximation; predict.BKP, plot.BKP, simulate.BKP, and summary.BKP for downstream methods.

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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model1 <- fit_BKP(X, y, m, Xbounds = Xbounds, theta = 0.04)
print(model1)


#-------------------------- 2D Example ---------------------------
# 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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model2 <- fit_BKP(X, y, m, Xbounds = Xbounds, theta = 0.08)
print(model2)
```

---

fit_DKP *Fit a Dirichlet Kernel Process Model*

Description

Fit a Dirichlet Kernel Process (DKP) model for categorical or multinomial count response data. The model estimates covariate-dependent class-probability surfaces by combining kernel-based smoothing with conjugate Dirichlet posterior updates.

Usage

```
fit_DKP(
  X,
  Y,
  Xbounds = NULL,
  prior = c("noninformative", "fixed", "adaptive"),
  r0 = 2,
  p0 = NULL,
  kernel = c("gaussian", "matern52", "matern32", "wendland"),
  isotropic = TRUE,
  theta = NULL,
  loss = c("brier", "log_loss"),
  ess = c("none", "shepard"),
  n_multi_start = NULL,
  n_threads = 1
)
```

Arguments

X	A numeric input matrix or data frame of size $n \times d$, where each row is an input or covariate vector.
Y	A numeric matrix or data frame of observed multinomial counts, with dimension $n \times q$. Each row corresponds to one input location and each column corresponds to one class. Entries must be nonnegative, and each row must have a positive row sum. Row sums represent the multinomial trial sizes.
Xbounds	Optional $d \times 2$ numeric matrix giving the lower and upper bounds of each input dimension. When supplied, X is normalized to $[0,1]^d$ before fitting. If NULL, X is assumed to be already normalized to $[0,1]^d$.
prior	Type of prior specification. Options are "noninformative" (default), "fixed", and "adaptive".
r0	Positive scalar prior precision used when prior = "fixed" or prior = "adaptive". Default is 2.
p0	Prior class-probability vector used when prior = "fixed". It must be a nonnegative finite numeric vector of length q and sum to one. If NULL, it is set to the empirical class-proportion vector colMeans(Y / rowSums(Y)).
kernel	Kernel function used for local weighting. Options are "gaussian" (default), "matern52", "matern32", and "wendland".
isotropic	Logical. If TRUE (default), use a single shared lengthscale across input dimensions. If FALSE, use separate per-dimension lengthscales.

`theta` Optional positive kernel lengthscale parameter. When `isotropic = TRUE`, this must be a scalar. When `isotropic = FALSE`, this can be either a scalar, which is broadcast to all dimensions, or a numeric vector of length `d`. If `NULL`, the lengthscale parameters are selected by minimizing the specified leave-one-out loss.

`loss` Leave-one-out loss used for kernel hyperparameter tuning. Options are `"brier"` (default) and `"log_loss"`.

`ess` Effective-sample-size calibration for the kernel-weighted class-count contribution. Use `"none"` (default) for the standard DKP posterior update. Use `"shepard"` to rescale the kernel-weighted class-count contribution so that its effective trial size is $\rho(\mathbf{x})m_S(\mathbf{x})$, where $m_S(\mathbf{x})$ is a Shepard interpolation of the observed row sums `rowSums(Y)` on the normalized input scale and $\rho(\mathbf{x}) = \max_i k(\mathbf{x}, \mathbf{x}_i)$. This calibration preserves the kernel-weighted empirical class proportions and changes only the data precision, not the prior parameters.

`n_multi_start` Number of initial points used in multi-start derivative-free optimization of the kernel lengthscale parameters. If `NULL`, the default is $10p$, where $p = 1$ for isotropic kernels and $p = d$ for anisotropic kernels.

`n_threads` Number of OpenMP threads used for multi-start optimization. Default is `1`. This argument only affects fitting when `theta = NULL`.

Details

Inputs are normalized to $[0, 1]^d$ using `Xbounds`. For a location $\mathbf{x}$, DKP computes kernel weights $k(\mathbf{x}, \mathbf{x}_i)$ between $\mathbf{x}$ and the training inputs. These weights are used to update a local Dirichlet prior with kernel-weighted multinomial counts:

$$\alpha_{n,s}(\mathbf{x}) = \alpha_{0,s}(\mathbf{x}) + \sum_i k(\mathbf{x}, \mathbf{x}_i) Y_{i,s}, \qquad s = 1, \ldots, q.$$

Equivalently,

$$\boldsymbol{\alpha}_n(\mathbf{x}) = \boldsymbol{\alpha}_0(\mathbf{x}) + \sum_i k(\mathbf{x}, \mathbf{x}_i) \mathbf{Y}_i.$$

The posterior class-probability vector at $\mathbf{x}$ follows a Dirichlet distribution with concentration vector $\boldsymbol{\alpha}_n(\mathbf{x})$.

If `theta = NULL`, the kernel lengthscale parameters are selected by leave-one-out cross-validation using the specified `loss`. Optimization is performed over log-transformed lengthscales using a multi-start derivative-free search. If `theta` is supplied, optimization is skipped and the model is fitted using the supplied lengthscale parameter.

If `ess = "shepard"`, only the kernel-weighted class-count contribution is rescaled to match the local effective-sample-size target; the prior parameters are not rescaled. Shepard calibration requires unique training input locations on the normalized input scale.

The returned object stores posterior Dirichlet concentration parameters evaluated at the training inputs. Posterior inference at new inputs is performed by `predict.DKP`.

Value

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

`theta_opt` Optimized or user-specified kernel lengthscale parameter(s).

`kernel` Kernel function used.

`isotropic` Logical flag indicating whether a shared lengthscale or per-dimension lengthscales were used.

`loss` Loss function used for hyperparameter tuning.

`loss_min` Loss value at the selected or user-specified lengthscale parameter(s).

`X` Original input matrix.

`Xnorm` Input matrix normalized to $[0, 1]^d$.

`Xbounds` Normalization bounds for each input dimension.

`Y` Observed multinomial count matrix.

`prior` Prior specification used.

`r0` Prior precision parameter.

`p0` Prior class-probability vector used when `prior = "fixed"`.

`alpha0` Prior Dirichlet concentration parameters evaluated at the training inputs.

`alpha_n` Posterior Dirichlet concentration parameters evaluated at the training inputs.

`ess` Effective-sample-size calibration method used.

`ess_info` ESS calibration diagnostics, or `NULL` when `ess = "none"`.

References

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

See Also

`fit_BKP` for Beta Kernel Process modeling of binary or binomial responses; `fit_TwinDKP` for the scalable global-local TwinDKP approximation; `predict.DKP`, `plot.DKP`, `simulate.DKP`, and `summary.DKP` for downstream methods.

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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model1 <- fit_DKP(X, Y, Xbounds = Xbounds, theta = 0.04)
print(model1)


#------------------------- 2D Example --------------------------
# 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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model2 <- fit_DKP(X, Y, Xbounds = Xbounds, theta = 0.08)
print(model2)
```

`fit_TwinBKP` *Fit a Twin Beta Kernel Process Model*

Description

Fit a Twin Beta Kernel Process (TwinBKP) model for binary or binomial response data. TwinBKP is a scalable global-local approximation to the full `fit_BKP` model. It uses a twinning-selected

global subset to capture the broad input-response structure and local nearest-neighbour updates to refine posterior inference near each target location.

Usage

```
fit_TwinBKP(
  X,
  y,
  m,
  Xbounds = NULL,
  prior = c("noninformative", "fixed", "adaptive"),
  r0 = 2,
  p0 = mean(y/m),
  global_kernel = c("gaussian", "matern52", "matern32", "wendland"),
  local_kernel = c("wendland"),
  loss = c("brier", "log_loss"),
  n_multi_start = NULL,
  isotropic = TRUE,
  n_threads = 1,
  theta_g = NULL,
  theta_l = NULL,
  g = NULL,
  l = NULL,
  twins = 5,
  store_kernel = FALSE
)
```

Arguments

`X`	A numeric input matrix or data frame of size $n \times d$, where each row is an input or covariate vector.
`y`	A numeric vector of observed success counts of length `n`.
`m`	A numeric vector of total binomial trial counts of length `n`. Each element of `y` must be between 0 and the corresponding element of `m`.
`Xbounds`	Optional $d \times 2$ numeric matrix giving the lower and upper bounds of each input dimension. When supplied, `X` is normalized to $[0,1]^d$ before fitting. If `NULL`, `X` is assumed to be already normalized to $[0,1]^d$.
`prior`	Type of prior specification. Options are `"noninformative"` (default), `"fixed"`, and `"adaptive"`.
`r0`	Positive scalar prior precision used when `prior = "fixed"` or `prior = "adaptive"`. Default is `2`.
`p0`	Prior mean used when `prior = "fixed"`. It must be a scalar in $(0,1)$. The default is the empirical mean `mean(y / m)`.
`global_kernel`	Kernel function used for the global subset contribution. Options are `"gaussian"` (default), `"matern52"`, `"matern32"`, and `"wendland"`.

`local_kernel` Kernel function used for the local-neighbour contribution. Currently only `"wendland"` is supported, corresponding to the compactly supported local kernel used by the TwinBKP approximation.

`loss` Leave-one-out loss used for kernel hyperparameter tuning. Options are `"brier"` (default) and `"log_loss"`.

`n_multi_start` Number of initial points used in multi-start optimization of the global kernel lengthscale parameter(s). If `NULL`, the default from `fit_BKP` is used on the selected global subset.

`isotropic` Logical. If `TRUE` (default), use a single shared lengthscale across input dimensions. If `FALSE`, use separate per-dimension lengthscales.

`n_threads` Number of OpenMP threads used for global-subset hyperparameter optimization when `theta_g = NULL`. Default is `1`.

`theta_g` Optional positive global kernel lengthscale parameter. When `isotropic = TRUE`, this must be a scalar. When `isotropic = FALSE`, this can be either a scalar, which is broadcast to all dimensions, or a numeric vector of length `d`. If `NULL`, the global lengthscale parameter is selected by fitting a BKP model on the selected global subset.

`theta_l` Optional positive scalar specifying the local Wendland kernel range. If `NULL`, it is set to the empirical covering radius of the global subset on the normalized input scale.

`g` Target global subset size. If `NULL`, the default is $\min\{n-1, 50d, \max(\lfloor\sqrt{n}\rfloor, 10d)\}$.

`l` Number of local non-global neighbours used at each training or prediction location. If `NULL`, the default is $\min\{n - |G|, \max(25, 3d)\}$ after the global subset has been selected.

`twins` Number of Twinning runs used to identify the global subset. Larger values may improve the selected global subset at additional computational cost. Default is `5`.

`store_kernel` Logical. If `TRUE`, store dense diagnostic kernel matrices `K`, `K_global`, and `K_local`. This option is intended for testing and diagnostics only. The default `FALSE` avoids dense $n \times n$ kernel storage and preserves the scalable memory behavior of TwinBKP.

Details

TwinBKP first normalizes the input matrix to $[0, 1]^d$. The global subset is selected by `twin_select_global_rcpp()` using the augmented representation `cbind(Xnorm, y / m)`, so the selected points represent both the normalized input distribution and the empirical response surface.

Given the selected global subset G, TwinBKP uses the union of G and a location-specific set of `l` nearest non-global neighbours for posterior aggregation. The global contribution uses `global_kernel` and `theta_g`; the local contribution uses the compactly supported `local_kernel` and `theta_l`. Local neighbours are found with a kd-tree over the non-global training points via **nanoflann**.

If `theta_g = NULL`, the global lengthscale parameter is selected by leave-one-out cross-validation on the selected global subset, using the specified `loss`. If `theta_g` is supplied, global tuning is skipped and the supplied value is used.

By default, TwinBKP aggregates posterior pseudo-counts row-wise and does not store dense $n \times n$ kernel matrices. Fitting posterior aggregation costs $O(n(g+l))$. Prediction at t new input points costs $O(t(\log n + g + l))$ for fixed input dimension. When `store_kernel = TRUE`, dense diagnostic matrices are additionally stored and memory use increases to $O(n^2)$.

Effective-sample-size calibration is currently available for full BKP and DKP models. TwinBKP uses the uncalibrated global-local posterior update to preserve the intended scalable approximation.

Value

A list of class `"TwinBKP"` containing the fitted TwinBKP model, with the following components:

`theta_opt` Alias for `theta_g`, retained for consistency with `fit_BKP`.

`theta_g` Optimized or user-specified global kernel lengthscale parameter(s).

`theta_l` Local kernel range parameter used for the nearest-neighbour local component.

`kernel` Alias for `global_kernel`, retained for consistency with `fit_BKP`.

`global_kernel` Kernel function used for the global subset contribution.

`local_kernel` Kernel function used for the local-neighbour contribution.

`isotropic` Logical flag indicating whether the global kernel uses one shared lengthscale or per-dimension lengthscales.

`loss` Loss function used for global lengthscale tuning.

`loss_min` Loss value at the selected or user-specified global lengthscale parameter(s).

`X` Original training input matrix.

`Xnorm` Training input matrix normalized to $[0,1]^d$.

`Xbounds` Normalization bounds for each input dimension.

`y` Observed success counts, stored as a one-column matrix.

`m` Observed binomial trial counts, stored as a one-column matrix.

`prior` Prior specification used in the TwinBKP posterior update.

`r0` Prior precision parameter.

`p0` Prior mean used when `prior = "fixed"`.

`alpha0` Prior Beta α shape parameters evaluated at the training inputs.

`beta0` Prior Beta β shape parameters evaluated at the training inputs.

`alpha_n` Posterior Beta α shape parameters evaluated at the training inputs.

`beta_n` Posterior Beta β shape parameters evaluated at the training inputs.

`global_indices` One-based indices of the selected global subset.

`control` A list of fitting controls and realized Twinning settings, including `g_target`, `g`, `l`, `r`, `twins`, `u1`, `leaf_size`, `n_multi_start`, `n_threads`, and `store_kernel`.

`diagnostics` A list of diagnostic objects. It contains `twin_info` from the C++ Twinning routine and, when `store_kernel = TRUE`, dense matrices `K`, `K_global`, and `K_local`. When `store_kernel = FALSE`, these kernel matrices are `NULL`.

References

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

Vakayil A, Joseph VR (2022). Data Twinning. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 15(5), 598–610. doi:10.1002/sam.11574.

Vakayil A, Joseph VR (2024). A Global-Local Approximation Framework for Large-Scale Gaussian Process Modeling. *Technometrics*, 66(2), 295–305. doi:10.1080/00401706.2023.2296451.

Blanco JL, PK Rai (2014). nanoflann: a C++ header-only fork of FLANN, a library for nearest neighbor (NN) with kd-trees. https://github.com/jlblancoc/nanoflann

See Also

fit_BKP for the full BKP model, fit_DKP for multinomial responses, fit_TwinDKP for the multinomial TwinDKP analogue, and predict.TwinBKP, plot.TwinBKP, simulate.TwinBKP, and summary.TwinBKP for downstream methods.

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 <- 200
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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model1 <- fit_TwinBKP(
     X, y, m,
     Xbounds = Xbounds,
     theta_g = 0.04,
     g = 20,
     twins = 1,
     n_threads = 1
   )

#------------------------- 2D Example --------------------------
# 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 <- 200
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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model2 <- fit_TwinBKP(
     X, y, m,
     Xbounds = Xbounds,
     theta_g = 0.08,
     g = 20,
     twins = 1,
     n_threads = 1
   )
```

`fit_TwinDKP` *Fit a Twin Dirichlet Kernel Process Model*

Description

Fit a Twin Dirichlet Kernel Process (TwinDKP) model for categorical or multinomial count response data. TwinDKP is a scalable global-local approximation to the full `fit_DKP` model. It uses a twinning-selected global subset to capture the broad input-response structure and local nearest-neighbour updates to refine posterior inference near each target location.

Usage

```
fit_TwinDKP(
  X,
  Y,
  Xbounds = NULL,
  prior = c("noninformative", "fixed", "adaptive"),
  r0 = 2,
```

```
  p0 = NULL,
  global_kernel = c("gaussian", "matern52", "matern32", "wendland"),
  local_kernel = c("wendland"),
  loss = c("brier", "log_loss"),
  n_multi_start = NULL,
  isotropic = TRUE,
  n_threads = 1,
  theta_g = NULL,
  theta_l = NULL,
  g = NULL,
  l = NULL,
  twins = 5,
  store_kernel = FALSE
)
```

Arguments

X	A numeric input matrix or data frame of size $n \times d$, where each row is an input or covariate vector.
Y	A numeric matrix or data frame of observed multinomial counts, with dimension $n \times q$. Each row corresponds to one input location and each column corresponds to one class. Entries must be nonnegative, and each row must have a positive row sum. Row sums represent the multinomial trial sizes.
Xbounds	Optional $d \times 2$ numeric matrix giving the lower and upper bounds of each input dimension. When supplied, X is normalized to $[0,1]^d$ before fitting. If NULL, X is assumed to be already normalized to $[0,1]^d$.
prior	Type of prior specification. Options are "noninformative" (default), "fixed", and "adaptive".
r0	Positive scalar prior precision used when prior = "fixed" or prior = "adaptive". Default is 2.
p0	Prior class-probability vector used when prior = "fixed". It must be a nonnegative finite numeric vector of length q and sum to one. If NULL, it is set to the empirical class-proportion vector colMeans(Y / rowSums(Y)).
global_kernel	Kernel function used for the global subset contribution. Options are "gaussian" (default), "matern52", "matern32", and "wendland".
local_kernel	Kernel function used for the local-neighbour contribution. Currently only "wendland" is supported, corresponding to the compactly supported local kernel used by the TwinDKP approximation.
loss	Leave-one-out loss used for kernel hyperparameter tuning. Options are "brier" (default) and "log_loss".
n_multi_start	Number of initial points used in multi-start optimization of the global kernel lengthscale parameter(s). If NULL, the default from fit_DKP is used on the selected global subset.
isotropic	Logical. If TRUE (default), use a single shared lengthscale across input dimensions. If FALSE, use separate per-dimension lengthscales.

n_threads	Number of OpenMP threads used for global-subset hyperparameter optimization when theta_g = NULL. Default is 1.
theta_g	Optional positive global kernel lengthscale parameter. When isotropic = TRUE, this must be a scalar. When isotropic = FALSE, this can be either a scalar, which is broadcast to all dimensions, or a numeric vector of length d. If NULL, the global lengthscale parameter is selected by fitting a DKP model on the selected global subset.
theta_l	Optional positive scalar specifying the local Wendland kernel range. If NULL, it is set to the empirical covering radius of the global subset on the normalized input scale.
g	Target global subset size. If NULL, the default is $\min\{n-1, 50d, \max(\lfloor\sqrt{n}\rfloor, 10d)\}$.
l	Number of local non-global neighbours used at each training or prediction location. If NULL, the default is $\min\{n - \lvert G\rvert, \max(25, 3d)\}$ after the global subset has been selected.
twins	Number of Twinning runs used to identify the global subset. Larger values may improve the selected global subset at additional computational cost. Default is 5.
store_kernel	Logical. If TRUE, store dense diagnostic kernel matrices K, K_global, and K_local. This option is intended for testing and diagnostics only. The default FALSE avoids dense $n \times n$ kernel storage and preserves the scalable memory behavior of TwinDKP.

Details

TwinDKP first normalizes the input matrix to $[0, 1]^d$. The global subset is selected by twin_select_global_rcpp() using the augmented representation cbind(Xnorm, Y / rowSums(Y)), so the selected points represent both the normalized input distribution and the empirical class-probability surface.

Given the selected global subset G, TwinDKP uses the union of G and a location-specific set of l nearest non-global neighbours for posterior aggregation. The global contribution uses global_kernel and theta_g; the local contribution uses the compactly supported local_kernel and theta_l. Local neighbours are found with a kd-tree over the non-global training points via **nanoflann**.

If theta_g = NULL, the global lengthscale parameter is selected by leave-one-out cross-validation on the selected global subset, using the specified loss. If theta_g is supplied, global tuning is skipped and the supplied value is used.

By default, TwinDKP aggregates posterior pseudo-counts row-wise and does not store dense $n \times n$ kernel matrices. Fitting posterior aggregation costs $O(n(g + l))$. Prediction at t new input points costs $O(t(\log n + g + l))$ for fixed input dimension. When store_kernel = TRUE, dense diagnostic matrices are additionally stored and memory use increases to $O(n^2)$.

Effective-sample-size calibration is currently available for full BKP and DKP models. TwinDKP uses the uncalibrated global-local posterior update to preserve the intended scalable approximation.

Value

A list of class "TwinDKP" containing the fitted TwinDKP model, with the following components:

theta_opt Alias for theta_g, retained for consistency with fit_DKP.

`theta_g` Optimized or user-specified global kernel lengthscale parameter(s).

`theta_l` Local kernel range parameter used for the nearest-neighbour local component.

`kernel` Alias for `global_kernel`, retained for consistency with `fit_DKP`.

`global_kernel` Kernel function used for the global subset contribution.

`local_kernel` Kernel function used for the local-neighbour contribution.

`isotropic` Logical flag indicating whether the global kernel uses one shared lengthscale or per-dimension lengthscales.

`loss` Loss function used for global lengthscale tuning.

`loss_min` Loss value at the selected or user-specified global lengthscale parameter(s).

`X` Original training input matrix.

`Xnorm` Training input matrix normalized to $[0, 1]^d$.

`Xbounds` Normalization bounds for each input dimension.

`Y` Observed multinomial count matrix.

`prior` Prior specification used in the TwinDKP posterior update.

`r0` Prior precision parameter.

`p0` Prior class-probability vector used when `prior = "fixed"`.

`alpha0` Prior Dirichlet concentration parameters evaluated at the training inputs.

`alpha_n` Posterior Dirichlet concentration parameters evaluated at the training inputs.

`prob` Posterior mean class-probability matrix, computed as normalized rows of `alpha_n`.

`global_indices` One-based indices of the selected global subset.

`control` A list of fitting controls and realized Twinning settings, including `g_target`, `g`, `l`, `r`, `twins`, `u1`, `leaf_size`, `n_multi_start`, `n_threads`, and `store_kernel`.

`diagnostics` A list of diagnostic objects. It contains `twin_info` from the C++ Twinning routine and, when `store_kernel = TRUE`, dense matrices `K`, `K_global`, and `K_local`. When `store_kernel = FALSE`, these kernel matrices are `NULL`.

References

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

Vakayil A, Joseph VR (2022). Data Twinning. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 15(5), 598–610. doi:10.1002/sam.11574.

Vakayil A, Joseph VR (2024). A Global-Local Approximation Framework for Large-Scale Gaussian Process Modeling. *Technometrics*, 66(2), 295–305. doi:10.1080/00401706.2023.2296451.

Blanco JL, PK Rai (2014). nanoflann: a C++ header-only fork of FLANN, a library for nearest neighbor (NN) with kd-trees. https://github.com/jlblancoc/nanoflann

See Also

`fit_DKP` for the full DKP model, `fit_TwinBKP` for the binomial TwinBKP analogue, `fit_BKP` for binary or binomial responses, and `predict.TwinDKP`, `plot.TwinDKP`, `simulate.TwinDKP`, and `summary.TwinDKP` for downstream methods.

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 <- 200
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 TwinDKP model
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model1 <- fit_TwinDKP(
     X, Y,
     Xbounds = Xbounds,
     theta_g = 0.04,
     g = 20,
     twins = 1,
     n_threads = 1
   )
print(model1)


#-------------------------- 2D Example ---------------------------
# 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 <- 200
```

```
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 TwinDKP model
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model2 <- fit_TwinDKP(
     X, Y,
     Xbounds = Xbounds,
     theta_g = 0.08,
     g = 20,
     twins = 1,
     n_threads = 1
   )
print(model2)
```

`fitted` *Extract Fitted Posterior Means from BKP Package Model Objects*

Description

Extract posterior fitted values from fitted `BKP`, `DKP`, `TwinBKP`, or `TwinDKP` model objects. For `BKP` and `TwinBKP` objects, this returns the posterior mean success probability at each training input. For `DKP` and `TwinDKP` objects, this returns the posterior mean class-probability vector at each training input.

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	A fitted model object of class BKP, DKP, TwinBKP, or TwinDKP, typically returned by fit_BKP, fit_DKP, fit_TwinBKP, or fit_TwinDKP.
...	Additional arguments. Currently unused.

Details

The fitted() method extracts posterior means already stored in the fitted model object. For BKP and TwinBKP, the fitted value at a training input is computed from the corresponding Beta posterior shape parameters. For DKP and TwinDKP, fitted values are obtained by normalizing the corresponding Dirichlet posterior concentration parameters across classes.

Value

For BKP and TwinBKP objects, a numeric vector containing posterior mean success probabilities at the training inputs. For DKP and TwinDKP objects, a numeric matrix containing posterior mean class probabilities at the training inputs, with one row per training input and one column per class.

References

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

See Also

fit_BKP, fit_DKP, fit_TwinBKP, and fit_TwinDKP for model fitting; predict for posterior prediction at new input locations.

Examples

```
# -------------------------- BKP and TwinBKP ---------------------------
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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model <- fit_BKP(X, y, m, Xbounds = Xbounds, theta = 0.04)
```

```
# Extract fitted values
fitted(model)

## Not run:
# Larger TwinBKP example
n <- 1000
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 using the default global lengthscale tuning
model <- fit_TwinBKP(
     X, y, m,
     Xbounds = Xbounds
   )

# Extract fitted values
fitted(model)

## End(Not run)

# -------------------------- DKP and TwinDKP --------------------------
# 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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model <- fit_DKP(X, Y, Xbounds = Xbounds, theta = 0.04)

# Extract fitted values
fitted(model)

## Not run:
# Larger TwinDKP example
n <- 1000
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 TwinDKP model using the default global lengthscale tuning
model <- fit_TwinDKP(
     X, Y,
     Xbounds = Xbounds
   )

# Extract fitted values
fitted(model)

## End(Not run)
```

`get_prior` *Construct Prior Parameters for BKP and DKP Models*

Description

Construct prior shape parameters for Beta Kernel Process (BKP) and Dirichlet Kernel Process (DKP) models. The function supports `prior = "noninformative"`, `"fixed"`, and `"adaptive"` prior specifications.

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 specification. Options are `"noninformative"` (default), `"fixed"`, and `"adaptive"`.
`model`	A character string specifying the model type: `"BKP"` (binary outcome) or `"DKP"` (multi-class outcome).
`r0`	Positive scalar prior precision used when `prior = "fixed"` or `prior = "adaptive"`. Default is `2`.

`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 success counts of length `n`.

`m` A numeric vector of total binomial trial counts of length `n`. Each element of `y` must be between 0 and the corresponding element of `m`.

`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` Optional precomputed kernel matrix, typically obtained from `kernel_matrix`. For adaptive priors, `K` is required and must have one column per observed training input. The number of rows of `K` determines the number of locations at which prior parameters are evaluated. For noninformative and fixed priors, if `K = NULL`, the output length is inferred from `y`/`m` for BKP or from `Y` for DKP when available; otherwise a single prior row is returned.

Details

The noninformative prior sets all Beta or Dirichlet shape parameters to 1.

The fixed prior uses a common prior distribution at every evaluated location. For BKP, this is a Beta prior with shape parameters `r0 * p0` and `r0 * (1 - p0)`. For DKP, this is a Dirichlet prior with concentration vector `r0 * p0`.

The adaptive prior estimates location-specific prior means from the observed data. BKP uses kernel smoothing of the observed proportions `y / m`, while DKP uses kernel smoothing of the row-normalized class counts in `Y`. In both cases, the adaptive prior precision is scaled by the local kernel mass.

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. doi: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, theta = 0.2, kernel = "gaussian")

# Adaptive BKP prior evaluated at the rows of K
prior_bkp <- get_prior(
  model = "BKP",
  prior = "adaptive",
  r0 = 2,
  y = y,
  m = m,
  K = K
)

# Fixed BKP prior; with K = NULL, output length is inferred from y/m
prior_bkp_fixed <- get_prior(
  model = "BKP",
  prior = "fixed",
  r0 = 2,
  p0 = 0.5,
  y = y,
  m = m
)

# -------------------------- DKP --------------------------
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(seq_len(n), function(i) {
  rmultinom(1, size = m[i], prob = true_pi[i, ])
}))
K <- kernel_matrix(X, theta = 0.2, kernel = "gaussian")

# Adaptive DKP prior evaluated at the rows of K
prior_dkp <- get_prior(
```

```
  model = "DKP",
  prior = "adaptive",
  r0 = 2,
  Y = Y,
  K = K
)

# Fixed DKP prior; with K = NULL, output rows are inferred from Y
prior_dkp_fixed <- get_prior(
  model = "DKP",
  prior = "fixed",
  r0 = 2,
  p0 = rep(1 / q, q),
  Y = Y
)
```

`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"`, `"matern52"`, `"matern32"`, 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 Euclidean distance is

$$h(\mathbf{x}, \mathbf{x}'; \boldsymbol{\theta}) = \left\| \frac{\mathbf{x} - \mathbf{x}'}{\boldsymbol{\theta}} \right\|_2 .$$

For isotropic kernels, $\boldsymbol{\theta}$ is a scalar shared by all input dimensions. For anisotropic kernels, $\boldsymbol{\theta}$ is a vector of dimension-specific lengthscales.

The available kernels are

- **Gaussian:**
$$k(\mathbf{x}, \mathbf{x}') = \exp(-h^2).$$
- **Matérn 5/2:**
$$k(\mathbf{x}, \mathbf{x}') = \left(1 + \sqrt{5}h + \frac{5}{3}h^2\right) \exp(-\sqrt{5}h).$$
- **Matérn 3/2:**
$$k(\mathbf{x}, \mathbf{x}') = \left(1 + \sqrt{3}h\right) \exp(-\sqrt{3}h).$$
- **Wendland:**
$$k(\mathbf{x}, \mathbf{x}') = (\zeta h + 1) \max(0, 1 - h)^{\zeta}, \qquad \zeta = \lfloor d/2 \rfloor + 3.$$

The returned matrix has one row for each row of `X` and one column for each row of `Xprime`. If `Xprime = NULL`, the function returns the symmetric kernel matrix for `X`.

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. doi:10.48550/arXiv.2508.10447.

Rasmussen, C. E., & Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press. https://gaussianprocess.org/gpml/.

Wendland, H. (1995). Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Advances in Computational Mathematics*, 4(1), 389–396. doi:10.1007/BF02123482.

Examples

```
set.seed(123)

X <- matrix(runif(20), ncol = 2)

# Compute kernel matrices for all implemented kernels
kernels <- c("gaussian", "matern52", "matern32", "wendland")
K_list <- lapply(kernels, function(k) {
  kernel_matrix(X, theta = 0.2, kernel = k)
```

```
})
names(K_list) <- kernels

# Inspect part of the Gaussian and Wendland kernel matrices
K_list$gaussian[1:3, 1:3]
K_list$wendland[1:3, 1:3]

# Anisotropic lengthscales
K_aniso <- kernel_matrix(
  X,
  theta = c(0.1, 0.3),
  kernel = "matern52",
  isotropic = FALSE
)

# Cross-kernel matrix between two input sets
Xprime <- matrix(runif(10), ncol = 2)
K_cross <- kernel_matrix(X, Xprime, theta = 0.2, kernel = "gaussian")
dim(K_cross)
```

`loss_fun` *Leave-One-Out Loss for BKP and DKP Models*

Description

Computes the leave-one-out cross-validation loss used for kernel hyperparameter tuning in BKP and DKP models. The input `gamma` represents log10-transformed kernel lengthscale parameters, with `theta = 10^gamma`. The function supports Brier score and log-loss under noninformative, fixed, and adaptive 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,
  ess = c("none", "shepard")
)
```

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 success counts of length n.
m	A numeric vector of total binomial trial counts of length n. Each element of y must be between 0 and the corresponding element of m.
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 specification. Options are "noninformative" (default), "fixed", and "adaptive".
r0	Positive scalar prior precision used when prior = "fixed" or prior = "adaptive". Default is 2.
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	Leave-one-out loss used for kernel hyperparameter tuning. Options are "brier" (default) and "log_loss".
kernel	Kernel function used for local weighting. Options are "gaussian" (default), "matern52", "matern32", and "wendland".
isotropic	Logical. If TRUE (default), use a single shared lengthscale across input dimensions. If FALSE, use separate per-dimension lengthscales.
ess	Effective-sample-size calibration for leave-one-out data contributions. Use "none" (default) for the standard LOOCV loss. Use "shepard" to apply the same data-precision rescaling as in model fitting, with the Shepard target computed in leave-one-out form.

Details

The kernel lengthscale parameter is represented on the log10 scale: theta = 10^gamma. The function first computes the kernel matrix using kernel_matrix and then sets its diagonal entries to zero, so that each fitted value is computed with the corresponding observation excluded. This implements leave-one-out cross-validation without repeatedly refitting the model.

For model = "BKP", the loss compares the leave-one-out posterior mean success probability with the empirical proportion y / m. For model = "DKP", the loss compares the leave-one-out posterior mean class-probability vector with the empirical class-proportion vector Y / rowSums(Y).

If ess = "shepard", the same data-precision rescaling used in model fitting is applied in leave-one-out form. This option requires unique training input locations on the normalized input scale.

Value

A single numeric value giving the average leave-one-out loss to be minimized. For BKP, the Brier score and log-loss are averaged over observations. For DKP, class-wise losses are summed within each observation and then averaged over observations.

References

Zhao J, Qing K, Xu J (2025). *BKP: An R Package for Beta Kernel Process Modeling*. arXiv. doi: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)
pi_true <- runif(n)
y <- rbinom(n, size = m, prob = pi_true)
loss_bkp <- loss_fun(gamma = log10(0.2), Xnorm = Xnorm, y = y, m = m)

# -------------------------- DKP --------------------------
n <- 10
q <- 3
Xnorm <- matrix(runif(2 * n), ncol = 2)
m <- rep(10, n)
p0 <- rep(1 / q, q)
Y <- matrix(rmultinom(n, size = 10, prob = rep(1/q, q)), nrow = n, byrow = TRUE)
loss_dkp <- loss_fun(gamma = log10(0.2), Xnorm = Xnorm, Y = Y, model = "DKP")
```

---

parameter *Extract Model Parameters from Fitted BKP Package Models*

Description

Extract key fitted parameters from `BKP`, `DKP`, `TwinBKP`, and `TwinDKP` model objects. For full BKP and DKP models, this includes the fitted kernel lengthscale parameter(s) and posterior shape or concentration parameters. For TwinBKP and TwinDKP models, the returned list also includes global-local approximation parameters, selected global subset indices, and fitting controls.

Usage

```
parameter(object, ...)

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

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

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

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

Arguments

object	A fitted model object of class BKP, DKP, TwinBKP, or TwinDKP, typically returned by fit_BKP, fit_DKP, fit_TwinBKP, or fit_TwinDKP.
...	Additional arguments. Currently unused.

Value

A named list containing fitted model parameters. Common entries are:

- theta: Fitted kernel lengthscale parameter(s). For TwinBKP and TwinDKP, this is an alias for the global lengthscale theta_g.
- alpha_n: Posterior Beta α shape parameters for BKP and TwinBKP, or posterior Dirichlet concentration parameters for DKP and TwinDKP.
- beta_n: Posterior Beta β shape parameters, returned for BKP and TwinBKP objects.

For TwinBKP and TwinDKP objects, the returned list also includes theta_g, theta_l, global_kernel, local_kernel, global_indices, and control.

References

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

See Also

fit_BKP, fit_DKP, fit_TwinBKP, and fit_TwinDKP for model fitting; fitted for posterior mean extraction.

Examples

```
# -------------------------- BKP and TwinBKP --------------------------
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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model <- fit_BKP(X, y, m, Xbounds = Xbounds, theta = 0.04)

# Extract posterior parameters
parameter(model)

## Not run:
# Larger TwinBKP example
n <- 1000
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 using the default global lengthscale tuning
model <- fit_TwinBKP(
     X, y, m,
     Xbounds = Xbounds
   )

# Extract posterior and kernel parameters
parameter(model)

## End(Not run)

# -------------------------- DKP and TwinDKP ---------------------------
# 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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model <- fit_DKP(X, Y, Xbounds = Xbounds, theta = 0.04)
```

```
# Extract model parameters
parameter(model)

## Not run:
# Larger TwinDKP example
n <- 1000
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 TwinDKP model using the default global lengthscale tuning
model <- fit_TwinDKP(
     X, Y,
     Xbounds = Xbounds
   )

# Extract posterior and kernel parameters
parameter(model)

## End(Not run)
```

---

plot *Plot Fitted BKP Package Model Objects*

Description

Visualize fitted BKP, DKP, TwinBKP, and TwinDKP model objects according to the selected input dimension(s). For one-dimensional plots, the methods show posterior mean probability curves, credible intervals, and observed proportions or class frequencies. For two-dimensional plots, the methods generate posterior summary surfaces over a prediction grid. For higher-dimensional inputs, users must specify one or two dimensions through dims.

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"),
  show_global = TRUE,
  ...
)

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

Arguments

x	A fitted model object of class "BKP", "DKP", "TwinBKP", or "TwinDKP", typically returned by fit_BKP, fit_DKP, fit_TwinBKP, or fit_TwinDKP.
only_mean	Logical. If TRUE, only the predicted mean surface is plotted for 2D inputs. This applies to BKP, DKP, TwinBKP, and TwinDKP models. 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).
show_global	Logical. For TwinBKP and TwinDKP objects, if TRUE, highlight the selected global subset used by the twinning-based approximation. Default is TRUE.

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 BKP, DKP, TwinBKP, and TwinDKP models, the function generates contour plots over a 2D prediction grid.
  - Users can choose to plot only the Posterior Mean surface (only_mean = TRUE) or a set of four summary plots (only_mean = FALSE):
    1. Posterior Mean
    2. 97.5th percentile (upper bound of 95% credible interval)
    3. Posterior 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).

For TwinBKP and TwinDKP objects, the plotting behavior is analogous to their full-model counterparts, with optional highlighting of the selected global subset.

Value

Invisibly returns NULL. The function is called for its side effect of producing plots.

References

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

See Also

fit_BKP, fit_DKP, fit_TwinBKP, and fit_TwinDKP for model fitting; predict.BKP, predict.DKP, predict.TwinBKP, and predict.TwinDKP for generating predictions from fitted models.

Examples

```
#-------------------------- BKP and TwinBKP ---------------------------
#-------------------------- 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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model1 <- fit_BKP(X, y, m, Xbounds = Xbounds, theta = 0.04)

# Plot results
plot(model1)

## Not run:
# Larger TwinBKP example
n <- 1000
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 using the default global lengthscale tuning
model1 <- fit_TwinBKP(X, y, m, Xbounds = Xbounds)

# Plot results
plot(model1)

## End(Not run)

#-------------------------- 2D Example ---------------------------
# 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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model2 <- fit_BKP(X, y, m, Xbounds = Xbounds, theta = 0.08)

# Plot results
plot(model2)

## Not run:
# Larger TwinBKP example
n <- 1000
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 using the default global lengthscale tuning
model2 <- fit_TwinBKP(X, y, m, Xbounds = Xbounds)

# Plot results
plot(model2)

## End(Not run)

#-------------------------- DKP and TwinDKP ---------------------------
#-------------------------- 1D Example ---------------------------
# 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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model1 <- fit_DKP(X, Y, Xbounds = Xbounds, theta = 0.04)

# Plot results
plot(model1)

## Not run:
# Larger TwinDKP example
n <- 1000
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 TwinDKP model using the default global lengthscale tuning
model1 <- fit_TwinDKP(X, Y, Xbounds = Xbounds)

# Plot results
plot(model1)

## End(Not run)

#-------------------------- 2D Example ---------------------------
# 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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model2 <- fit_DKP(X, Y, Xbounds = Xbounds, theta = 0.08)

# Plot results
plot(model2)

## Not run:
# Larger TwinDKP example
n <- 1000
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 TwinDKP model using the default global lengthscale tuning
model2 <- fit_TwinDKP(X, Y, Xbounds = Xbounds)

# Plot results
plot(model2)

## End(Not run)
```

`predict` *Posterior Prediction for BKP Package Model Objects*

Description

Generate posterior summaries from fitted `BKP`, `DKP`, `TwinBKP`, and `TwinDKP` model objects at training or new input locations. For BKP and TwinBKP 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 and TwinDKP models, summaries can be returned either for the latent class probability vector or for future class counts using marginal Beta-Binomial posterior predictive distributions.

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,
  ...
)

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

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

Arguments

`object`	A fitted model object of class `"BKP"`, `"DKP"`, `"TwinBKP"`, or `"TwinDKP"`, typically returned by `fit_BKP`, `fit_DKP`, `fit_TwinBKP`, or `fit_TwinDKP`.
`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 for BKP and TwinBKP models. Default is `0.5`.
`type`	Character string specifying the prediction target. The default `"probability"` returns posterior summaries for latent success probabilities (BKP and TwinBKP)

or latent class probabilities (DKP). The option `"count"` returns posterior predictive summaries for future counts under the Beta-Binomial distribution (BKP and TwinBKP) or marginal Beta-Binomial distributions for each multinomial class (DKP).

`Mnew` Positive integer trial size used 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$m` (BKP and TwinBKP) or `rowSums(object$Y)` (DKP) 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 input locations at which predictions are returned. If `Xnew = NULL`, predictions are returned at the training input locations.

`alpha_n` Posterior shape or concentration parameters:

- For `BKP` and `TwinBKP`, a vector of posterior Beta α shape parameters.
- For `DKP` and `TwinDKP`, a matrix of posterior Dirichlet concentration parameters, with rows corresponding to prediction locations and columns corresponding to classes.

`beta_n` Posterior Beta β shape parameters, returned for `BKP` and `TwinBKP` objects.

`mean` Mean of the prediction target:

- For `BKP` and `TwinBKP` with `type = "probability"`, the posterior mean of the latent success probability.
- For `BKP` and `TwinBKP` with `type = "count"`, the posterior predictive mean of a future success count.
- For `DKP` and `TwinDKP` with `type = "probability"`, a matrix of posterior mean class probabilities.
- For `DKP` and `TwinDKP` with `type = "count"`, a matrix of marginal posterior predictive mean class counts.

`variance` Variance of the prediction target:

- For `BKP` and `TwinBKP` with `type = "probability"`, the posterior variance of the latent success probability.
- For `BKP` and `TwinBKP` with `type = "count"`, the posterior predictive variance of a future success count.
- For `DKP` and `TwinDKP` with `type = "probability"`, a matrix of marginal posterior variances for class probabilities.
- For `DKP` and `TwinDKP` with `type = "count"`, a matrix of marginal posterior predictive variances for class counts.

`lower` Lower bound of the credible or predictive interval:

- For `BKP` and `TwinBKP` with `type = "probability"`, the lower Beta posterior quantile for the latent success probability.

- For BKP and TwinBKP with type = "count", the lower Beta-Binomial posterior predictive quantile for a future success count.
- For DKP and TwinDKP with type = "probability", a matrix of lower marginal posterior quantiles for class probabilities.
- For DKP and TwinDKP with type = "count", a matrix of lower marginal Beta-Binomial posterior predictive quantiles for class counts.

upper Upper bound of the credible or predictive interval:

- For BKP and TwinBKP with type = "probability", the upper Beta posterior quantile for the latent success probability.
- For BKP and TwinBKP with type = "count", the upper Beta-Binomial posterior predictive quantile for a future success count.
- For DKP and TwinDKP with type = "probability", a matrix of upper marginal posterior quantiles for class probabilities.
- For DKP and TwinDKP with type = "count", a matrix of upper marginal Beta-Binomial posterior predictive quantiles for class counts.

class Predicted class label, returned only when applicable:

- For BKP and TwinBKP, a binary class label based on the posterior mean and threshold, returned only for binary classification data with m = 1 and type = "probability".
- For DKP and TwinDKP, the class with the largest posterior mean probability, returned only for one-hot classification data and type = "probability".

threshold The classification threshold, returned for BKP and TwinBKP classification predictions.

CI_level The specified credible interval level.

type The prediction target, either "probability" or "count".

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

ess Effective-sample-size calibration method inherited from the fitted model, returned for BKP and DKP objects.

ess_info ESS diagnostics for BKP and DKP predictions when available. TwinBKP and TwinDKP use uncalibrated global-local posterior updates and do not return ESS diagnostics.

References

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

See Also

fit_BKP, fit_DKP, fit_TwinBKP, and fit_TwinDKP for model fitting; plot.BKP, plot.DKP, plot.TwinBKP, and plot.TwinDKP for visualization.

Examples

```
#------------------------- BKP and TwinBKP --------------------------
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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model <- fit_BKP(X, y, m, Xbounds = Xbounds, theta = 0.04)

# Prediction on training data
predict(model)

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

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

## Not run:
# Larger TwinBKP example
n <- 1000
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 using the default global lengthscale tuning
model <- fit_TwinBKP(
     X, y, m,
     Xbounds = Xbounds
   )

# Prediction on training data
predict(model)

# Prediction on new data
predict(model, Xnew = Xnew)

# Posterior predictive summaries for future success counts
predict(model, Xnew = Xnew, type = "count", Mnew = Mnew)

## End(Not run)

#------------------------- DKP and TwinDKP ---------------------------
```

```
# 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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model <- fit_DKP(X, Y, Xbounds = Xbounds, theta = 0.04)

# Prediction on training data
predict(model)

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

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

## Not run:
# Larger TwinDKP example
n <- 1000
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 TwinDKP model using the default global lengthscale tuning
model <- fit_TwinDKP(
     X, Y,
     Xbounds = Xbounds
   )

# Prediction on training data
predict(model)

# Prediction on new data
predict(model, Xnew)
```

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

## End(Not run)
```

`print` *Print Methods for BKP Package Objects*

Description

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

- `print.BKP`, `print.DKP`, `print.TwinBKP`, and `print.TwinDKP` display fitted model objects.
- `print.summary_BKP`, `print.summary_DKP`, `print.summary_TwinBKP`, and `print.summary_TwinDKP` display model summaries.
- `print.predict_BKP`, `print.predict_DKP`, `print.predict_TwinBKP`, and `print.predict_TwinDKP` display posterior predictive results.
- `print.simulate_BKP`, `print.simulate_DKP`, `print.simulate_TwinBKP`, and `print.simulate_TwinDKP` 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 'predict_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 'predict_TwinDKP'
print(x, ...)

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

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

Arguments

x	An object of class "BKP", "DKP", "TwinBKP", or "TwinDKP", or a derived object such as a summary, prediction, or simulation object returned by the corresponding S3 methods.
...	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. doi:10.48550/arXiv.2508.10447.

See Also

fit_BKP, fit_DKP, fit_TwinBKP, and fit_TwinDKP for model fitting; summary.BKP, summary.DKP, summary.TwinBKP, and summary.TwinDKP for model summaries; predict.BKP, predict.DKP,

predict.TwinBKP, and predict.TwinDKP for posterior prediction; simulate.BKP, simulate.DKP, simulate.TwinBKP, and simulate.TwinDKP for posterior simulations.

Examples

```
#------------------------- BKP and TwinBKP --------------------------
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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model <- fit_BKP(X, y, m, Xbounds = Xbounds, theta = 0.04)
print(model)                    # fitted object
print(summary(model))           # summary
print(predict(model))           # predictions
print(simulate(model, nsim = 3))  # posterior simulations

## Not run:
# Larger TwinBKP example
n <- 1000
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 using the default global lengthscale tuning
model <- fit_TwinBKP(
     X, y, m,
     Xbounds = Xbounds
   )
print(model)                    # fitted object
print(summary(model))           # summary
print(predict(model))           # predictions
print(simulate(model, nsim = 3))  # posterior simulations

## End(Not run)

#------------------------- DKP and TwinDKP --------------------------
# 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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model <- fit_DKP(X, Y, Xbounds = Xbounds, theta = 0.04)
print(model)                    # fitted object
print(summary(model))           # summary
print(predict(model))           # predictions
print(simulate(model, nsim = 3))  # posterior simulations

## Not run:
# Larger TwinDKP example
n <- 1000
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 TwinDKP model using the default global lengthscale tuning
model <- fit_TwinDKP(
     X, Y,
     Xbounds = Xbounds
   )
print(model)                    # fitted object
print(summary(model))           # summary
print(predict(model))           # predictions
print(simulate(model, nsim = 3))  # posterior simulations

## End(Not run)
```

---

quantile *Posterior Quantiles from Fitted BKP Package Models*

Description

Compute posterior quantiles from fitted BKP, DKP, TwinBKP, and TwinDKP model objects. For BKP and TwinBKP objects, this returns posterior quantiles of the latent success probability. For DKP and TwinDKP objects, this returns marginal 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), ...)

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

Arguments

x	A fitted model object of class "BKP", "DKP", "TwinBKP", or "TwinDKP", typically returned by fit_BKP, fit_DKP, fit_TwinBKP, or fit_TwinDKP.
probs	Numeric vector of probabilities specifying which posterior quantiles to return. Defaults to c(0.025, 0.5, 0.975).
...	Additional arguments (currently unused).

Details

For BKP and TwinBKP models, posterior quantiles are computed from the corresponding Beta posterior for the latent success probability. For DKP and TwinDKP models, marginal posterior quantiles for each class probability are computed from the Beta marginal distributions of the posterior Dirichlet distribution. These are marginal class-wise quantiles, not joint Dirichlet credible regions.

Value

For BKP and TwinBKP: 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 and TwinDKP: a numeric matrix if length(probs) = 1, or a three-dimensional array if length(probs) > 1, of marginal posterior quantiles for class probabilities. Dimensions correspond to observations $\times$ classes $\times$ probabilities.

References

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

See Also

fit_BKP, fit_DKP, fit_TwinBKP, and fit_TwinDKP for model fitting; predict.BKP, predict.DKP, predict.TwinBKP, and predict.TwinDKP for posterior prediction.

Examples

```
# ------------------------- BKP and TwinBKP --------------------------
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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model <- fit_BKP(X, y, m, Xbounds = Xbounds, theta = 0.04)

# Extract posterior quantiles
quantile(model)

## Not run:
# Larger TwinBKP example
n <- 1000
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 using the default global lengthscale tuning
model <- fit_TwinBKP(
     X, y, m,
     Xbounds = Xbounds
   )

# Extract posterior quantiles
quantile(model)

## End(Not run)

# ------------------------- DKP and TwinDKP --------------------------
# 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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model <- fit_DKP(X, Y, Xbounds = Xbounds, theta = 0.04)

# Extract posterior quantiles
quantile(model)

## Not run:
# Larger TwinDKP example
n <- 1000
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 TwinDKP model using the default global lengthscale tuning
model <- fit_TwinDKP(
     X, Y,
     Xbounds = Xbounds
   )

# Extract posterior quantiles
quantile(model)

## End(Not run)
```

`simulate` *Simulate from Fitted BKP Package Models*

Description

Generate random posterior draws from fitted `BKP`, `DKP`, `TwinBKP`, and `TwinDKP` model objects at training or new input locations.

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

For DKP and TwinDKP models, posterior samples are generated from Dirichlet distributions for latent class-probability vectors. If the training responses are single-label, that is, one-hot encoded, class labels are additionally assigned using the maximum posterior simulated probability.

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, threshold = NULL, ...)

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

Arguments

object	A fitted model object of class "BKP", "DKP", "TwinBKP", or "TwinDKP", typically returned by fit_BKP, fit_DKP, fit_TwinBKP, or fit_TwinDKP.
nsim	Number of posterior samples to generate. The default is 1.
seed	Optional integer seed for reproducibility.
Xnew	A numeric vector or matrix of new input locations at which posterior simulations are generated. If NULL, simulations are generated at the training input locations.
threshold	Classification threshold for binary decisions, used for BKP and TwinBKP models. When specified, posterior draws exceeding the threshold are classified as 1, and those below or equal to the threshold are classified as 0. The default is NULL.
...	Additional arguments passed to the corresponding predict method when Xnew is supplied.

Value

A list containing posterior simulations:

samples For BKP and TwinBKP, a numeric matrix with one row per simulation location and one column per posterior draw. Each entry is a simulated latent success probability.
For DKP and TwinDKP, a numeric array with dimensions simulation locations $\times$ classes $\times$ posterior draws. Each slice contains simulated latent class probabilities from the posterior Dirichlet distribution.

mean Posterior mean at the simulation locations. For BKP and TwinBKP, this is a numeric vector of success-probability means. For DKP and TwinDKP, this is a matrix of class-probability means.

`class` Simulated class labels when applicable. For `BKP` and `TwinBKP`, this is returned when `threshold` is supplied. For `DKP` and `TwinDKP`, this is returned when the training response is one-hot encoded.

`X` The original training input locations.

`Xnew` The new input locations used for simulation. If `NULL`, simulations are returned at the training input locations.

`threshold` The binary classification threshold, returned for `BKP` and `TwinBKP` simulations when supplied.

`ess` Effective-sample-size calibration method inherited from the fitted model, returned for `BKP` and `DKP` objects.

`ess_info` ESS diagnostics for `BKP` and `DKP` simulations when available. `TwinBKP` and `TwinDKP` do not support ESS calibration and do not return this component.

References

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

See Also

`fit_BKP`, `fit_DKP`, `fit_TwinBKP`, and `fit_TwinDKP` for model fitting; `predict.BKP`, `predict.DKP`, `predict.TwinBKP`, and `predict.TwinDKP` for posterior prediction.

Examples

```
# -------------------------- BKP and TwinBKP --------------------------
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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model <- fit_BKP(X, y, m, Xbounds = Xbounds, theta = 0.04)

# 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)

## Not run:
# Larger TwinBKP example
n <- 1000
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 using the default global lengthscale tuning
model <- fit_TwinBKP(
     X, y, m,
     Xbounds = Xbounds
   )

# Simulate 5 posterior draws of success probabilities
simulate(model, Xnew = Xnew, nsim = 5)

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

## End(Not run)

# -------------------------- DKP and TwinDKP ---------------------------
# 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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model <- fit_DKP(X, Y, Xbounds = Xbounds, theta = 0.04)

# 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)

## Not run:
# Larger TwinDKP example
n <- 1000
```

```
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 TwinDKP model using the default global lengthscale tuning
model <- fit_TwinDKP(
     X, Y,
     Xbounds = Xbounds
   )

# Simulate 5 draws from posterior Dirichlet distributions at new point
simulate(model, Xnew = Xnew, nsim = 5)

## End(Not run)
```

---

summary *Summarize Fitted BKP Package Models*

Description

Provides a structured summary of fitted BKP, DKP, TwinBKP, and TwinDKP model objects. The summary reports model dimensions, kernel settings, optimized hyperparameters, prior specification, loss information, approximation settings when applicable, and posterior summaries at the training input locations.

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	A fitted model object of class "BKP", "DKP", "TwinBKP", or "TwinDKP", typically returned by fit_BKP, fit_DKP, fit_TwinBKP, or fit_TwinDKP.
...	Additional arguments passed to the generic summary method (currently unused).

Value

A list containing model configuration, prior information, kernel hyperparameters, loss information, and posterior summaries at the training inputs.

For `BKP` and `TwinBKP`, posterior summaries are returned as vectors of posterior means and variances for the latent success probabilities.

For `DKP` and `TwinDKP`, posterior summaries are returned as matrices of marginal posterior means and variances for class probabilities.

For `TwinBKP` and `TwinDKP`, the list additionally includes global-local approximation fields such as `global_kernel`, `local_kernel`, `theta_g`, `theta_l`, `global_size`, `global_target`, `local_size`, and `twins`.

References

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

See Also

`fit_BKP`, `fit_DKP`, `fit_TwinBKP`, and `fit_TwinDKP` for model fitting; `predict.BKP`, `predict.DKP`, `predict.TwinBKP`, and `predict.TwinDKP` for posterior prediction.

Examples

```
# -------------------------- BKP and TwinBKP --------------------------
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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model <- fit_BKP(X, y, m, Xbounds = Xbounds, theta = 0.04)
summary(model)

## Not run:
# Larger TwinBKP example
n <- 1000
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 using the default global lengthscale tuning
model <- fit_TwinBKP(
     X, y, m,
     Xbounds = Xbounds
   )
summary(model)

## End(Not run)

# ------------------------- DKP and TwinDKP --------------------------
# 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
# A fixed theta is used here only to keep the example fast and reproducible.
# In practice, omit theta to select it by leave-one-out cross-validation.
model <- fit_DKP(X, Y, Xbounds = Xbounds, theta = 0.04)
summary(model)

## Not run:
# Larger TwinDKP example
n <- 1000
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 TwinDKP model using the default global lengthscale tuning
model <- fit_TwinDKP(
     X, Y,
     Xbounds = Xbounds
   )
summary(model)

## End(Not run)
```

Index

∗ **BKP**
  fitted, 21
  parameter, 31
  plot, 34
  predict, 40
  print, 46
  quantile, 49
  simulate, 52
  summary, 56
∗ **DKP**
  fitted, 21
  parameter, 31
  plot, 34
  predict, 40
  print, 46
  quantile, 49
  simulate, 52
  summary, 56
∗ **TwinBKP**
  fitted, 21
  parameter, 31
  plot, 34
  predict, 40
  print, 46
  quantile, 49
  simulate, 52
  summary, 56
∗ **TwinDKP**
  fitted, 21
  parameter, 31
  plot, 34
  predict, 40
  print, 46
  quantile, 49
  simulate, 52
  summary, 56

BKP-package, 2

fit_BKP, *3*, 4, *10, 11*, *13–15*, *19*, *22*, *25*, *31*, *32*, *35, 36*, *41*, *43*, *47*, *50, 51*, *53, 54*, *56, 57*
fit_DKP, *3*, *6*, 7, *15–19*, *22*, *25*, *31, 32*, *35, 36*, *41*, *43*, *47*, *50, 51*, *53, 54*, *56, 57*
fit_TwinBKP, *3*, *6*, 11, *19*, *22*, *32*, *35, 36*, *41*, *43*, *47*, *50, 51*, *53, 54*, *56, 57*
fit_TwinDKP, *3*, *10*, *15*, 16, *22*, *32*, *35, 36*, *41*, *43*, *47*, *50, 51*, *53, 54*, *56, 57*
fitted, *3*, 21, *32*

get_prior, *3*, 24, *31*

kernel_matrix, *3*, *25*, 27, *30, 31*

loss_fun, *3*, 29

parameter, *3*, 31
plot, *3*, 34
plot.BKP, *6*, *43*
plot.DKP, *10*, *43*
plot.TwinBKP, *15*, *43*
plot.TwinDKP, *19*, *43*
predict, *3*, *22*, 40
predict.BKP, *5, 6*, *25*, *36*, *47*, *51*, *54*, *57*
predict.DKP, *9, 10*, *25*, *36*, *47*, *51*, *54*, *57*
predict.TwinBKP, *15*, *36*, *48*, *51*, *54*, *57*
predict.TwinDKP, *19*, *36*, *48*, *51*, *54*, *57*
print, *3*, 46

quantile, *3*, 49

simulate, *3*, 52
simulate.BKP, *6*, *48*
simulate.DKP, *10*, *48*
simulate.TwinBKP, *15*, *48*
simulate.TwinDKP, *19*, *48*
summary, *3*, 56
summary.BKP, *6*, *47*
summary.DKP, *10*, *47*
summary.TwinBKP, *15*, *47*
summary.TwinDKP, *19*, *47*