Package 'BKP'

August 19, 2025

Title Beta Kernel Process Modeling

Version 0.1.1

Maintainer Jiangyan Zhao <zhaojy2017@126.com>

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

License GPL-3

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

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

Encoding UTF-8

NeedsCompilation no

RoxygenNote 7.3.2

Depends R (>= 3.5.0)

Imports gridExtra, lattice, optimx, tgp

Suggests knitr, mlbench, rmarkdown, rticles, testthat (>= 3.0.0), tinytex

Config/testthat/edition 3

BuildVignettes true

VignetteBuilder knitr

Author Jiangyan Zhao [cre, aut],
Kunhai Qing [aut],
Jin Xu [aut]

Repository CRAN

Date/Publication 2025-08-19 08:40:30 UTC

Contents

`BKP-package` *Beta Kernel Process Modeling*

Description

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

The package includes functionality for model fitting, probabilistic prediction with uncertainty quantification, posterior simulation, and visualization in both one- and two-dimensional input spaces. It also supports hyperparameter tuning and flexible prior specification.

Main Functions

Core functionality is organized into the following groups:

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

`predict.BKP`, `predict.DKP` Perform posterior predictive inference at new input locations, including predictive means, variances, and credible intervals. Classification labels are returned automatically when observations represent single trials (i.e., binary outcomes).

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

`plot.BKP`, `plot.DKP` Visualize model predictions and uncertainty bands in 1D and 2D input spaces.

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

References

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

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

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

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

`fit.BKP` *Fit a Beta Kernel Process (BKP) Model*

Description

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

Usage

```
fit.BKP(
  X,
  y,
  m,
  Xbounds = NULL,
  prior = c("noninformative", "fixed", "adaptive"),
  r0 = 2,
  p0 = 0.5,
  kernel = c("gaussian", "matern52", "matern32"),
  loss = c("brier", "log_loss"),
  n_multi_start = NULL,
  theta = NULL
)
```

Arguments

`X`	A numeric input matrix of size $n \times d$, where each row represents a covariate vector.
`y`	A numeric vector of observed successes (length `n`).
`m`	A numeric vector of total binomial trials (length `n`), corresponding to each `y`.

Xbounds	Optional $d \times 2$ matrix specifying the lower and upper bounds of each input dimension. Used to normalize inputs to $[0,1]^d$. If Xbounds is NULL, the input is assumed to have already been normalized, and the default bounds are set to $[0,1]^d$.
prior	Type of prior to use. One of "noninformative", "fixed", or "adaptive".
r0	Global prior precision (only used when prior = "fixed" or "adaptive").
p0	Global prior mean (only used when prior = "fixed").
kernel	Kernel function for local weighting. Choose from "gaussian", "matern52", or "matern32".
loss	Loss function for kernel hyperparameter tuning. One of "brier" (default) or "log_loss".
n_multi_start	Number of random initializations for multi-start optimization. Default is 10 × d.
theta	Optional. A positive scalar or a numeric vector of length equal to the input dimension d. If specified, these values will be used directly as the kernel length-scale parameters, bypassing the internal optimization procedure. If NULL (default), the kernel parameters are optimized via (multi-start) L-BFGS-B to minimize the chosen loss function.

Value

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

theta_opt Optimized kernel hyperparameters (lengthscales).

kernel Kernel function used, as a string.

loss Loss function used for hyperparameter tuning.

loss_min Minimum loss value achieved during optimization.

loss_min Minimum loss value achieved during kernel hyperparameter optimization. If theta was manually specified by the user, this value is set to NA.

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

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

Xbounds Matrix specifying normalization bounds for each input dimension.

y Observed success counts.

m Observed binomial trial counts.

prior Type of prior used.

r0 Prior precision parameter.

p0 Prior mean (for fixed priors).

alpha0 Prior shape parameter $\alpha_0(\mathbf{x})$, either a scalar or vector.

beta0 Prior shape parameter $\beta_0(\mathbf{x})$, either a scalar or vector.

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

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

References

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

See Also

`fit.DKP` for modeling multinomial responses using the Dirichlet Kernel Process. `predict.BKP`, `plot.BKP`, `simulate.BKP` for making predictions, visualizing results, and generating simulations from a fitted BKP model. `summary.BKP`, `print.BKP` for inspecting model details.

Examples

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

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

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

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


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

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

n <- 100
```

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

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

`fit.DKP` *Fit a Dirichlet Kernel Process (DKP) Model*

Description

Fits a DKP model for categorical or multinomial response data by locally smoothing observed counts to estimate latent Dirichlet parameters.

Usage

```
fit.DKP(
  X,
  Y,
  Xbounds = NULL,
  prior = c("noninformative", "fixed", "adaptive"),
  r0 = 2,
  p0 = NULL,
  kernel = c("gaussian", "matern52", "matern32"),
  loss = c("brier", "log_loss"),
  n_multi_start = NULL,
  theta = NULL
)
```

Arguments

`X`	A numeric input matrix of size $n \times d$, where each row represents a covariate vector.
`Y`	Matrix of observed multinomial counts, with dimension $n \times q$.
`Xbounds`	Optional $d \times 2$ matrix specifying the lower and upper bounds of each input dimension. Used to normalize inputs to $[0, 1]^d$. If `Xbounds` is `NULL`, the input is assumed to have already been normalized, and the default bounds are set to $[0, 1]^d$.
`prior`	Type of prior to use. One of `"noninformative"`, `"fixed"`, or `"adaptive"`.
`r0`	Global prior precision (only used when `prior = "fixed"` or `"adaptive"`).
`p0`	Global prior mean vector (only used when `prior = "fixed"`). Must be of length q.

`kernel`	Kernel function for local weighting. Choose from `"gaussian"`, `"matern52"`, or `"matern32"`.
`loss`	Loss function for kernel hyperparameter tuning. One of `"brier"` (default) or `"log_loss"`.
`n_multi_start`	Number of random initializations for multi-start optimization. Default is `10 × d`.
`theta`	Optional. A positive scalar or a numeric vector of length equal to the input dimension `d`. If specified, these values will be used directly as the kernel length-scale parameters, bypassing the internal optimization procedure. If `NULL` (default), the kernel parameters are optimized via (multi-start) L-BFGS-B to minimize the chosen loss function.

Value

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

`theta_opt` Optimized kernel hyperparameters (lengthscales).

`kernel` Kernel function used, as a string.

`loss` Loss function used for hyperparameter tuning.

`loss_min` Minimum loss value achieved during kernel hyperparameter optimization. If `theta` was manually specified by the user, this value is set to `NA`.

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

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

`Xbounds` Matrix specifying normalization bounds for each input dimension.

`Y` Observed multinomial counts of size `n × q`.

`prior` Type of prior used.

`r0` Prior precision parameter.

`p0` Prior mean (for fixed priors).

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

`alpha_n` Posterior Dirichlet parameters after kernel smoothing.

References

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

See Also

`fit.BKP` for modeling binomial responses using the Beta Kernel Process. `predict.DKP`, `plot.DKP`, `simulate.DKP` for making predictions, visualizing results, and generating simulations from a fitted DKP model. `summary.DKP`, `print.DKP` for inspecting fitted model summaries.

Examples

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

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

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
model1 <- fit.DKP(X, Y, Xbounds = Xbounds)
print(model1)


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

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

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)

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

# Fit DKP model
```

```
model2 <- fit.DKP(X, Y, Xbounds = Xbounds)
print(model2)
```

`get_prior` *Construct Prior Parameters for the BKP Model*

Description

Computes the prior Beta distribution parameters `alpha0` and `beta0` at each input location, based on the chosen prior specification. Supports noninformative, fixed, and data-adaptive prior strategies.

Usage

```
get_prior(
  prior = c("noninformative", "fixed", "adaptive"),
  r0 = 2,
  p0 = 0.5,
  y = NULL,
  m = NULL,
  K = NULL
)
```

Arguments

`prior`	Character string specifying the type of prior to use. One of `"noninformative"`, `"fixed"`, or `"adaptive"`.
`r0`	Positive scalar indicating the global precision parameter. Used when `prior` is `"fixed"` or `"adaptive"`.
`p0`	Prior mean for the success probability (in (0,1)). Used only when `prior = "fixed"`.
`y`	Numeric vector of observed successes, of length `n`.
`m`	Numeric vector of total binomial trials, of length `n`.
`K`	A precomputed kernel matrix of size `n × n`, typically obtained from `kernel_matrix`.

Details

- For `prior = "noninformative"`, all prior parameters are set to 1 (noninformative prior).
- For `prior = "fixed"`, all locations share the same Beta prior: `Beta(r0 * p0, r0 * (1 - p0))`.
- For `prior = "adaptive"`, the prior mean at each location is computed by kernel smoothing the observed proportions `y/m`, and precision `r0` is distributed accordingly.

Value

A list with two numeric vectors:

`alpha0` Prior alpha parameters of the Beta distribution, length `n`.

`beta0` Prior beta parameters of the Beta distribution, length `n`.

References

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

See Also

get_prior_dkp, fit.BKP, predict.BKP, kernel_matrix

Examples

```
# Simulated data
set.seed(123)
n <- 10
X <- matrix(runif(n * 2), ncol = 2)
y <- rbinom(n, size = 5, prob = 0.6)
m <- rep(5, n)

# Example kernel matrix (Gaussian)
K <- kernel_matrix(X)

# Construct adaptive prior
prior <- get_prior(prior = "adaptive", r0 = 2, y = y, m = m, K = K)
```

---

get_prior_dkp *Construct Prior Parameters for the DKP Model*

Description

Computes prior Dirichlet distribution parameters `alpha0` at each input location for the Dirichlet Kernel Process model, based on the specified prior type: noninformative, fixed, or adaptive.

Usage

```
get_prior_dkp(
  prior = c("noninformative", "fixed", "adaptive"),
  r0 = 2,
  p0 = NULL,
  Y = NULL,
  K = NULL
)
```

Arguments

`prior`	Character string specifying the type of prior to use. One of `"noninformative"`, `"fixed"`, or `"adaptive"`.
`r0`	Positive scalar indicating the global precision parameter. Used when `prior` is `"fixed"` or `"adaptive"`.

p0	Numeric vector specifying the global prior mean for each class (must sum to 1). Only used when prior = "fixed". Should be of length equal to the number of classes.
Y	Numeric matrix of observed class counts of size n × q, where n is the number of observations and q the number of classes.
K	A precomputed kernel matrix of size n × n, typically obtained from kernel_matrix.

Details

- When prior = "noninformative", all entries in alpha0 are set to 1 (flat Dirichlet).
- When prior = "fixed", all rows of alpha0 are set to r0 * p0.
- When prior = "adaptive", each row of alpha0 is computed by kernel-weighted smoothing of the observed relative frequencies in Y, scaled by r0.

Value

A list containing:

alpha0 A numeric matrix of prior Dirichlet parameters at each input location; dimension n × q.

References

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

See Also

get_prior, fit.DKP, predict.DKP, kernel_matrix

Examples

```
# Simulated multi-class data
set.seed(123)
n <- 15           # number of training points
q <- 3            # number of classes
X <- matrix(runif(n * 2), ncol = 2)

# Simulate class probabilities and draw multinomial counts
true_pi <- t(apply(X, 1, function(x) {
  raw <- c(
    exp(-sum((x - 0.2)^2)),
    exp(-sum((x - 0.5)^2)),
    exp(-sum((x - 0.8)^2))
  )
  raw / sum(raw)
}))
m <- sample(10:20, n, replace = TRUE)
Y <- t(sapply(1:n, function(i) rmultinom(1, size = m[i], prob = true_pi[i, ])))

# Compute kernel matrix (Gaussian)
```

```
K <- kernel_matrix(X, theta = rep(0.2, 2), kernel = "gaussian")

# Construct adaptive prior
prior_dkp <- get_prior_dkp(prior = "adaptive", r0 = 2, Y = Y, K = K)
```

`kernel_matrix` *Compute Kernel Matrix Between Input Locations*

Description

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

Usage

```
kernel_matrix(
  X,
  Xprime = NULL,
  theta = 0.1,
  kernel = c("gaussian", "matern52", "matern32"),
  anisotropic = 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 a scalar, the same lengthscale is applied to all input dimensions. If a vector, it must be of length `d`, corresponding to anisotropic scaling.
`kernel`	A character string specifying the kernel function. Must be one of `"gaussian"`, `"matern32"`, or `"matern52"`.
`anisotropic`	Logical. If `TRUE` (default), `theta` is interpreted as a vector of per-dimension lengthscales. If `FALSE`, `theta` is treated as a scalar.

Details

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

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

The available kernels are defined as:

- **Gaussian:**

$$k(\mathbf{x}, \mathbf{x}') = \exp(-r^2)$$

- **Matérn 5/2:**

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

- **Matérn 3/2:**

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

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

Value

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

References

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

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

Examples

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

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

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

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

loss_fun *Loss Function for Fitting the BKP Model*

Description

Computes the loss used to fit the BKP model. Supports the Brier score (mean squared error) and negative log-loss (cross-entropy), under different prior specifications.

Usage

```
loss_fun(
  gamma,
  Xnorm,
  y,
  m,
  prior = c("noninformative", "fixed", "adaptive"),
  r0 = 2,
  p0 = 0.5,
  loss = c("brier", "log_loss"),
  kernel = c("gaussian", "matern52", "matern32")
)
```

Arguments

gamma	A numeric vector of log-transformed kernel hyperparameters.
Xnorm	A numeric matrix of normalized inputs (each column scaled to [0,1]).
y	A numeric vector of observed successes (length n).
m	A numeric vector of total binomial trials (length n), corresponding to each y.
prior	Type of prior to use. One of "noninformative", "fixed", or "adaptive".
r0	Global prior precision (only used when prior = "fixed" or "adaptive").
p0	Global prior mean (only used when prior = "fixed").
loss	Loss function for kernel hyperparameter tuning. One of "brier" (default) or "log_loss".
kernel	Kernel function for local weighting. Choose from "gaussian", "matern52", or "matern32".

Value

A single numeric value representing the total loss (to be minimized).

References

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

See Also

loss_fun_dkp, fit.BKP, get_prior, kernel_matrix

Examples

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

---

loss_fun_dkp *Loss Function for Fitting the DKP Model*

Description

Computes the loss used to fit the DKP model. Supports the Brier score (mean squared error) and negative log-loss (cross-entropy), under different prior specifications.

Usage

```
loss_fun_dkp(
  gamma,
  Xnorm,
  Y,
  prior = c("noninformative", "fixed", "adaptive"),
  r0 = 2,
  p0 = NULL,
  loss = c("brier", "log_loss"),
  kernel = c("gaussian", "matern52", "matern32")
)
```

Arguments

gamma	A numeric vector of log-transformed kernel hyperparameters.
Xnorm	A numeric matrix of normalized inputs (each column scaled to [0,1]).
Y	Matrix of observed multinomial counts, with dimension $n \times q$.
prior	Type of prior to use. One of "noninformative", "fixed", or "adaptive".
r0	Global prior precision (only used when prior = "fixed" or "adaptive").
p0	Global prior mean vector (only used when prior = "fixed"). Must be of length q.
loss	Loss function for kernel hyperparameter tuning. One of "brier" (default) or "log_loss".
kernel	Kernel function for local weighting. Choose from "gaussian", "matern52", or "matern32".

Value

A single numeric value representing the total loss (to be minimized).

References

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

See Also

loss_fun, fit.DKP, get_prior_dkp, kernel_matrix

Examples

```
set.seed(123)
n = 10
Xnorm = matrix(runif(2 * n), ncol = 2)
m = rep(10, n)
y = rbinom(n, size = m, prob = runif(n))
Y = cbind(y, m-y)
loss_fun_dkp(gamma = rep(0, 2), Xnorm = Xnorm, Y = Y)
```

---

plot *Plot Fitted BKP or DKP Models*

Description

Visualizes fitted BKP or DKP models depending on the input dimensionality. For 1-dimensional inputs, it displays predicted class probabilities with credible intervals and observed data. For 2-dimensional inputs, it generates contour plots of posterior summaries.

Usage

```
## S3 method for class 'BKP'
plot(x, only_mean = FALSE, n_grid = 80, ...)

## S3 method for class 'DKP'
plot(x, only_mean = FALSE, n_grid = 80, ...)
```

Arguments

x	An object of class "BKP" or "DKP", typically returned by fit.BKP or fit.DKP.
only_mean	Logical. If TRUE, only the predicted mean surface is plotted for 2D inputs (only applies to BKP models). Default is FALSE.
n_grid	Integer. Number of grid points used in each dimension to construct the prediction grid. A larger value produces a smoother and more detailed decision boundary, but increases computational cost. Default is 80.
...	Additional arguments passed to internal plotting routines (currently unused).

Details

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

- **1D inputs:**
  - For BKP (binary/binomial data), plots the posterior mean curve with a 95% credible band, overlaid with the observed proportions (y/m).
  - For DKP (categorical/multinomial data), plots one curve per class, each with a shaded credible interval and the observed class frequencies.
  - For classification tasks, an optional curve of the maximum posterior class probability can be displayed to visualize decision confidence.
- **2D inputs:**
  - For both models, produces either:
    1. A predictive mean surface (optionally maximum posterior probability for classification), or
    2. A 2-by-2 panel of contour plots showing: predictive mean, predictive 97.5th percentile (upper bound of 95% credible interval), predictive variance, and predictive 2.5th percentile (lower bound).
  - For DKP, these surfaces are generated separately for each class.

For input dimensions greater than two, the function terminates with an error message.

Value

This function does not return a value. It is called for its side effects, producing plots that visualize the model predictions and uncertainty.

References

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

See Also

fit.BKP, predict.BKP, fit.DKP, predict.DKP

Examples

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

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

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

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

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

# Plot results
plot(model1)


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

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

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

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

# Plot results
plot(model2, n_grid = 50)

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

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

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

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
model1 <- fit.DKP(X, Y, Xbounds = Xbounds)

# Plot results
plot(model1)


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

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

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)

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

# Fit DKP model
model2 <- fit.DKP(X, Y, Xbounds = Xbounds)
```

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

---

predict *Predict Method for BKP or DKP Models*

Description

Generates posterior predictive summaries from a fitted BKP or DKP model at new input locations.

Usage

```
## S3 method for class 'BKP'
predict(object, Xnew, CI_level = 0.95, threshold = 0.5, ...)

## S3 method for class 'DKP'
predict(object, Xnew, CI_level = 0.95, ...)
```

Arguments

`object`	An object of class `"BKP"` or `"DKP"`, typically returned by `fit.BKP` or `fit.DKP`.
`Xnew`	A numeric matrix (or vector) of new input locations where predictions are desired.
`CI_level`	Credible level for prediction intervals (default is `0.95`, corresponding to 95% CI).
`threshold`	Classification threshold for binary prediction based on posterior mean (used only for BKP; default is `0.5`).
`...`	Additional arguments passed to generic `predict` methods (currently not used; included for S3 method consistency).

Value

A list with the following components:

`Xnew` The new input locations.

`mean` BKP: Posterior mean of the success probability at each location. DKP: A matrix of posterior mean class probabilities (rows = inputs, columns = classes).

`variance` BKP: Posterior variance of the success probability. DKP: A matrix of posterior variances for each class.

`lower` BKP: Lower bound of the prediction interval (e.g., 2.5th percentile for 95% CI). DKP: A matrix of lower bounds for each class (e.g., 2.5th percentile).

`upper` BKP: Upper bound of the prediction interval (e.g., 97.5th percentile for 95% CI). DKP: A matrix of upper bounds for each class (e.g., 97.5th percentile).

`class` BKP: Predicted binary label (0 or 1), based on posterior mean and threshold, if `m = 1`. DKP: Predicted class label (i.e., the class with the highest posterior mean probability).

References

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

See Also

`fit.BKP` for fitting Beta Kernel Process models. `fit.DKP` for fitting Dirichlet Kernel Process models. `plot.BKP` for visualizing fitted BKP models. `plot.DKP` for visualizing fitted DKP models.

Examples

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

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

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

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

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

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


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

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

```
  f <- log(a*b)
  f <- (f- m)/s
  return(pnorm(f))  # Transform to probability
}

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

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

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

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

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

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

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
model1 <- fit.DKP(X, Y, Xbounds = Xbounds)

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


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

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

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)

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

# Fit DKP model
model2 <- fit.DKP(X, Y, Xbounds = Xbounds)
print(model2)

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

`print` *Print Summary of a Fitted BKP or DKP Model*

Description

Displays a concise summary of a fitted BKP or DKP model. The output includes key characteristics such as sample size, input dimensionality, kernel type, loss function, optimized kernel hyperparameters, and minimum loss.

Usage

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

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

Arguments

x	An object of class "BKP" (from fit.BKP) or "DKP" (from fit.DKP).
...	Additional arguments passed to the generic print method (currently not used).

Value

Invisibly returns the input object (of class "BKP" or "DKP"). The function is called for its side effect of printing a summary to the console.

References

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

See Also

fit.BKP, fit.DKP, summary.BKP, summary.DKP.

Examples

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

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

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

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

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


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

# Define 2D latent function and probability transformation
```

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

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

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

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

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

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

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
model1 <- fit.DKP(X, Y, Xbounds = Xbounds)
print(model1)
```

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

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

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)

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

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

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

Description

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

For BKP models, posterior samples are drawn from Beta distributions representing success probabilities, with optional binary class labels determined by a threshold.

For DKP models, posterior samples are drawn from Dirichlet distributions representing class probabilities, with optional class labels determined by the maximum a posteriori (MAP) rule if training responses are one-hot encoded.

Usage

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

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

Arguments

object	An object of class "BKP" or "DKP", typically returned by fit.BKP or fit.DKP.
Xnew	A numeric matrix or vector of new input locations for simulation.
n_sim	Number of posterior samples to generate (default = 1).
threshold	Classification threshold for binary output (only used for BKP). If specified, the output will include binary class labels with values above the threshold classified as 1 (default is NULL).
seed	Optional integer seed for reproducibility.
...	Additional arguments (currently unused).

Value

A list with the following components:

sims For **BKP**: A numeric matrix of dimension nrow(Xnew) × n_sim, containing simulated success probabilities.
For **DKP**: A numeric array of dimension n_sim × q × nrow(Xnew), containing simulated class probabilities from Dirichlet posteriors, where q is the number of classes.

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

class For **BKP**: A binary matrix of dimension nrow(Xnew) × n_sim indicating simulated class labels (0/1), returned if threshold is specified.
For **DKP**: A numeric matrix of dimension nrow(Xnew) × n_sim containing MAP-predicted class labels, returned only when training data is single-label (i.e., each row of Y sums to 1).

References

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

See Also

fit.BKP, fit.DKP, predict.BKP, predict.DKP

Examples

```
## -------------------- BKP Simulation 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
model <- fit.BKP(X, y, m, Xbounds=Xbounds)

# Simulate 5 posterior draws of success probabilities
Xnew <- matrix(seq(-2, 2, length.out = 100), ncol = 1)
simulate(model, Xnew, n_sim = 5)

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

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

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

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

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

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

# Simulate 5 draws from posterior Dirichlet distributions at new point
Xnew <- matrix(seq(-2, 2, length.out = 100), ncol = 1)
simulate(model, Xnew = Xnew, n_sim = 5)
```

---

summary *Summary of a Fitted BKP or DKP Model*

Description

Provides a summary of a fitted Beta Kernel Process (BKP) or Dirichlet Kernel Process (DKP) model. Currently, this function acts as a wrapper for `print.BKP` or `print.DKP`, delivering a concise overview of key model characteristics and fitting results.

Usage

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

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

Arguments

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

Value

Invisibly returns the input object (of class `"BKP"` or `"DKP"`). Called for side effects: prints a concise summary of the fitted model.

References

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

See Also

`fit.BKP`, `fit.DKP`, `print.BKP`, `print.DKP`.

Examples

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

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

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

n <- 30
Xbounds <- matrix(c(-2,2), nrow=1)
X <- tgp::lhs(n = n, rect = Xbounds)
true_pi <- true_pi_fun(X)
```

```
m <- sample(100, n, replace = TRUE)
y <- rbinom(n, size = m, prob = true_pi)

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


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

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

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

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

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

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

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

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
model1 <- fit.DKP(X, Y, Xbounds = Xbounds)
summary(model1)


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

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

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)

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

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

Index