BKP: An R Package for Beta Kernel Process Modeling

Jiangyan Zhao
School of Statistics
East China Normal University

Kunhai Qing
School of Statistics
East China Normal University

Jin Xu
School of Statistics
and
Key Laboratory of Advanced Theory and Application in Statistics and Data Science - MOE
East China Normal University

Abstract

We present **BKP**, a user-friendly and extensible R package that implements the Beta Kernel Process (BKP) – a fully nonparametric and computationally efficient framework for modeling spatially varying binomial probabilities. The BKP model combines localized kernel-weighted likelihoods with conjugate beta priors, resulting in closed-form posterior inference without requiring latent variable augmentation or intensive MCMC sampling. The package supports binary and aggregated binomial responses, allows flexible choices of kernel functions and prior specification, and provides loss-based kernel hyperparameter tuning procedures. In addition, BKP extends naturally to the Dirichlet Kernel Process (DKP) for modeling spatially varying multinomial or compositional data. To our knowledge, this is the first publicly available R package for implementing BKP-based methods. We illustrate the use of **BKP** through several synthetic and real-world datasets, highlighting its interpretability, accuracy, and scalability. The package aims to facilitate practical application and future methodological development of kernel-based beta modeling in statistics and machine learning.

Keywords: Beta kernel process, Dirichlet kernel process, nonparametric Bayesian modeling, binomial and multinomial data, R.

1. Introduction

Estimating a continuous probability function from binary or binomial observations is a fundamental task in modern statistics and machine learning (Hastie *et al.* 2009; Rolland *et al.* 2019; Murphy 2022). Such problems arise in various domains where the goal is to infer a latent probability surface from individual Bernoulli outcomes or aggregated binomial counts over a continuous input space. Representative applications include binary classification (MacKenzie *et al.* 2014; Wen *et al.* 2025), probability calibration (Sung *et al.* 2020; Dimitriadis *et al.* 2022), relative abundance modeling (Martin *et al.* 2020), and longitudinal analysis of patient-

reported outcomes (Najera-Zuloaga *et al.* 2018, 2019).

Classical approaches such as logistic regression and generalized additive models provide computationally efficient tools for binomial regression, but their parametric form often limits their ability to capture complex nonlinear patterns (Hastie *et al.* 2009). Nonparametric models like Gaussian Process (GP) classifiers provide greater modeling flexibility and principled uncertainty quantification (Rasmussen and Williams 2006), yet the non-Gaussian likelihood of binary responses necessitates approximate inference, leading to substantial computational cost and implementation complexity (Nickisch and Rasmussen 2008).

The **BKP** package implements the *Beta Kernel Process* (BKP), a scalable and interpretable nonparametric model for estimating binomial probability surfaces. Developed in R (R Core Team 2025) and available on the Comprehensive R Archive Network (CRAN) at `https://cran.r-project.org/package=BKP`, **BKP** combines localized kernel-weighted likelihoods with conjugate beta priors, yielding closed-form posterior inference without latent variables or numerical approximation. It supports both binary and aggregated binomial data, and delivers full posterior uncertainty quantification while maintaining linear scalability and computational efficiency (Goetschalckx *et al.* 2011; MacKenzie *et al.* 2014; Rolland *et al.* 2019).

Compared to GP-based methods, BKP eliminates the need for sampling or variational inference and provides interpretable, locally updated estimates. This makes it suitable for applications demanding transparency, real-time feedback, or adaptive decision-making. BKP has been extended to the multinomial setting via the *Dirichlet Kernel Process* (DKP), enabling nonparametric modeling of categorical proportions across multiple groups. Despite its practical appeal, the theoretical properties of BKP remain underexplored. Mussi *et al.* (2024) identified the lack of formal guarantees for BKP-based decision processes as an open problem, emphasizing the need for further methodological development.

While GP models are supported by mature software libraries, such as **DiceKriging** (Roustant *et al.* 2012), **GPfit** (MacDonald *et al.* 2015), and **gplite** (Piironen 2022) in R, and **GPyTorch** (Gardner *et al.* 2018), **GPflow** (Matthews *et al.* 2017), and **GPy** (GPy since 2012) in Python. So far, there are about 100 GP-related packages on CRAN (as seen by running `packagefinder::findPackage("Gaussian process", display = "browser")`). In contrast, software for kernel-based modeling of binomial and multinomial data remains scarce. The **BKP** package fills this gap by offering a natively, unified and extensible framework for nonparametric modeling of binary/binomial, categorical/multinomial data. It provides multiple options for prior specification, kernel functions, and loss functions, enabling flexible model customization. These features make **BKP** applicable to a wide range of problems in biomedicine, ecology, social science, and industrial statistics.

The remainder of the paper is organized as follows. Section 2 introduces the statistical foundation of the BKP model, including prior specification strategies, kernel functions, loss-based hyperparameter tuning, and the DKP extension. Section 3 describes the structure and main functionalities of the **BKP** package, including model fitting, prediction, and simulation for both BKP and DKP. Section 4 presents illustrative examples to demonstrate the use of BKP and DKP in binary classification and compositional modeling tasks. Section 5 concludes with a discussion of current limitations and potential extensions.

2. Statistical Foundation

2.1. Beta Kernel Process

Let $\boldsymbol{x} = (x_1, x_2, \ldots, x_d) \in \mathcal{X} \subset \mathbb{R}^d$ denote a d-dimensional input. Suppose the success probability surface $\pi(\boldsymbol{x}) \in [0,1]$ is unknown. At each location $\boldsymbol{x}$, the observed data is modeled as

$$y(\boldsymbol{x}) \sim \text{Binomial}(m(\boldsymbol{x}), \pi(\boldsymbol{x})),$$

where $y(\boldsymbol{x})$ is the number of successes out of $m(\boldsymbol{x})$ independent trials. The full dataset comprises n observations $\mathcal{D}_n = \{(\boldsymbol{x}_i, y_i, m_i)\}_{i=1}^n$, where we write $y_i = y(\boldsymbol{x}_i)$ and $m_i = m(\boldsymbol{x}_i)$ for brevity.

In line with the Bayesian paradigm, assign a Beta prior to the unknown probability function as

$$\pi(\boldsymbol{x}) \sim \text{Beta}(\alpha_0(\boldsymbol{x}), \beta_0(\boldsymbol{x})),$$

where $\alpha_0(\boldsymbol{x}) > 0$ and $\beta_0(\boldsymbol{x}) > 0$ are spatially varying shape parameters. Details on prior specification are discussed in Section 2.2.

Let $k : \mathcal{X} \times \mathcal{X} \to [0,1]$ denote a user-defined kernel function measuring the similarity between input locations. By the kernel-based Bayesian updating strategy (Goetschalckx *et al.* 2011; Rolland *et al.* 2019), the BKP model constructs a closed-form posterior distribution for $\pi(\boldsymbol{x})$ as

$$\begin{aligned} \pi(\boldsymbol{x}) \mid \mathcal{D}_n &\sim \text{Beta}\left(\alpha_n(\boldsymbol{x}), \beta_n(\boldsymbol{x})\right), \\ \alpha_n(\boldsymbol{x}) &= \alpha_0(\boldsymbol{x}) + \sum_{i=1}^n k(\boldsymbol{x}, \boldsymbol{x}_i) y_i = \alpha_0(\boldsymbol{x}) + \boldsymbol{k}^\top(\boldsymbol{x})\boldsymbol{y}, \\ \beta_n(\boldsymbol{x}) &= \beta_0(\boldsymbol{x}) + \sum_{i=1}^n k(\boldsymbol{x}, \boldsymbol{x}_i)(m_i - y_i) = \beta_0(\boldsymbol{x}) + \boldsymbol{k}^\top(\boldsymbol{x})(\boldsymbol{m} - \boldsymbol{y}), \end{aligned} \tag{1}$$

where $\boldsymbol{k}(\boldsymbol{x}) = [k(\boldsymbol{x}, \boldsymbol{x}_1), \ldots, k(\boldsymbol{x}, \boldsymbol{x}_n)]^\top$ is the vector of kernel weights and $\boldsymbol{y} = (y_1, \ldots, y_n)^\top$.

Remark 1. *While the BKP model leverages the conjugacy of the Beta-Binomial pair, it differs from the traditional Bayesian paradigm in the sense that the posterior update is induced from a kernel-weighted local likelihood defined by*

$$\widetilde{L}(\pi(\boldsymbol{x}); \mathcal{D}_n) \propto \prod_{i=1}^n \left\{\pi(\boldsymbol{x})^{y_i}(1-\pi(\boldsymbol{x}))^{m_i - y_i}\right\}^{k(\boldsymbol{x}, \boldsymbol{x}_i)} = \pi(\boldsymbol{x})^{\boldsymbol{k}^\top(\boldsymbol{x})\boldsymbol{y}}\{1-\pi(\boldsymbol{x})\}^{\boldsymbol{k}^\top(\boldsymbol{x})(\boldsymbol{m}-\boldsymbol{y})}.$$

This approach mimics the local likelihood method of Fan et al. *(1998), where data are reweighted based on distance to the target point in the input space. And, the choice of kernel parameters is driven by empirical risk minimization rather than posterior inference. Thus, BKP is best interpreted as a nonparametric, Bayesian-inspired smoothing framework.*

Remark 2. *One prominent advantage of the closed-form updating scheme is its light burden of computational complexity. Fitting the BKP model involves $\mathcal{O}(n^2)$ operations for computing the kernel matrix, in contrast to the $\mathcal{O}(n^3)$ operations typically required by Gaussian process regression. While, evaluating the posterior at a new location requires only $\mathcal{O}(n)$ kernel computations.*

Based on the resulting posterior distribution (1), the posterior mean

$$\widehat{\pi}_n(\boldsymbol{x}) = \mathbb{E}[\pi(\boldsymbol{x}) \mid \mathcal{D}_n] = \frac{\alpha_n(\boldsymbol{x})}{\alpha_n(\boldsymbol{x}) + \beta_n(\boldsymbol{x})} \tag{2}$$

serves as a smooth estimator of the latent success probability. The corresponding posterior variance

$$\sigma_n^2(\boldsymbol{x}) = \mathrm{Var}[\pi(\boldsymbol{x}) \mid \mathcal{D}_n] = \frac{\widehat{\pi}_n(\boldsymbol{x})\{1 - \widehat{\pi}_n(\boldsymbol{x})\}}{\alpha_n(\boldsymbol{x}) + \beta_n(\boldsymbol{x}) + 1} \tag{3}$$

provides a local measure of epistemic uncertainty. These posterior summaries can be used to visualize prediction quality across the input space, particularly highlighting regions with sparse data coverage. See Section 4 for illustrations.

For binary classification, the posterior mean can be thresholded to produce hard predictions through

$$\widehat{y}(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \widehat{\pi}_n(\boldsymbol{x}) > \pi_0, \\ 0 & \text{otherwise,} \end{cases} \tag{4}$$

where $\pi_0 \in (0, 1)$ is a user-specified threshold, typically set to be 0.5.

2.2. Prior Specification

We provide three strategies for prior specification as follows.

- **Non-informative prior:** A default and widely used choice is the uniform prior, which sets $\alpha_0(\boldsymbol{x}) = \beta_0(\boldsymbol{x}) \equiv 1$ for all $\boldsymbol{x}$. It is appropriate when no prior knowledge is available.
- **Informative prior with fixed mean:** When prior information about the overall success probability $p_0 \in (0, 1)$ is available, an informative prior can be constructed by setting

  $$\alpha_0(\boldsymbol{x}) = r_0 p_0, \qquad \beta_0(\boldsymbol{x}) = r_0(1 - p_0),$$

  where $r_0 > 0$ is a scalar precision parameter controlling the strength of the prior. Larger value of r_0 represents greater prior certainty. It contains the previous non-informative prior as a special case with $r_0 = 2$ and $p_0 = 0.5$.
- **Data-adaptive informative prior:** To accommodate spatial variation in the data, **BKP** supports a locally adaptive prior in which both the prior mean and prior strength vary across the input space. Specifically, at each location $\boldsymbol{x}$, the prior mean $p(\boldsymbol{x})$ and prior precision $r(\boldsymbol{x})$ are estimated using kernel-weighted averages through

  $$p(\boldsymbol{x}) = \sum_{i=1}^{n} w_i(\boldsymbol{x}) \cdot \frac{y_i}{m_i}, \qquad r(\boldsymbol{x}) = r_0 \sum_{i=1}^{n} k(\boldsymbol{x}, \boldsymbol{x}_i),$$

  where $r_0 > 0$ is a global precision parameter and $w_i(\boldsymbol{x})$ are normalized kernel weights defined by $w_i(\boldsymbol{x}) = k(\boldsymbol{x}, \boldsymbol{x}_i)/\sum_{j=1}^{n} k(\boldsymbol{x}, \boldsymbol{x}_j)$. Then, the prior parameters at $\boldsymbol{x}$ are given by

  $$\alpha_0(\boldsymbol{x}) = r(\boldsymbol{x})p(\boldsymbol{x}), \qquad \beta_0(\boldsymbol{x}) = r(\boldsymbol{x})\{1 - p(\boldsymbol{x})\}.$$

  This data-adaptive formulation allows the prior to dynamically respond to the local sampling density. In well-sampled regions, it becomes more concentrated, while in sparse regions, it remains diffuse. Such adaptivity improves calibration in under-sampled areas and reduces overfitting where data are dense, thereby enhancing inference robustness under spatial heterogeneity. This strategy is consistent with the principle of local likelihood modeling (Fan and Gijbels 1996).

Table 1: Kernel functions implemented in **BKP**.

Kernel Type	**Function** $k(h)$
Gaussian	$k(h) = \exp(-h^2)$
Matérn $\nu = 5/2$	$k(h) = \left(1 + \sqrt{5}h + \frac{5}{3}h^2\right)\exp(-\sqrt{5}h)$
Matérn $\nu = 3/2$	$k(h) = \left(1 + \sqrt{3}h\right)\exp(-\sqrt{3}h)$

We offer some practical guideline to choose the global precision parameter. In classification when each input location receives a single categorical observation, we recommend choosing a relatively small value, such as $0.01 \leq r_0 \leq 0.1$, to prevent the prior domination and keep the posterior inference primarily data-driven. We illustrate this point in Example 1 of Section 4.1. For model fitting when binomial responses are observed at each input location, set r_0 to be about 5–10% of the average number of trials per location for moderate regularization. When strong prior knowledge is available, choose r_0 to be 1–5% of the total sample size. In the absence of strong prior information, a good starting point is $r_0 = 0.01$. Then, use cross-validation or predictive performance to make adjustment. In both situations, one can conduct a sensitivity analysis over several r_0 values (e.g., 0.01, 0.1, 0.5, 1, 2, 5) to evaluate the stability of predictions and decision boundaries.

2.3. Model Selection via Kernel Hyperparameter Tuning

Kernel Functions

Let $h(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\theta})$ denote the scaled Euclidean distance:

$$h(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\theta}) = \sqrt{\sum_{j=1}^{d} \left(\frac{x_j - x'_j}{\theta_j}\right)^2},$$

where $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_d)$ are positive kernel hyperparameters governing the relative importance of each input component. Based on this metric, define the kernel function as $k(\boldsymbol{x}, \boldsymbol{x}') = k(h)$, where the functional form of $k(\cdot)$ determines the kernel type (Rasmussen and Williams 2006).

The **BKP** package implements several widely used kernel functions, namely Gaussian (squared-exponential) and Matérn families, summarized in Table 1.

Loss Functions

Hyperparameter tuning is conducted by minimizing a user-specified loss function evaluated using the leave-one-out cross-validation (LOOCV) method (Montesano and Lopes 2012). For each data point $\boldsymbol{x}_i$, the model is refitted up on the remaining data $\mathcal{D}_n^{-i}$, and the posterior mean $\widehat{\pi}_n^{-i}(\boldsymbol{x}_i)$ is used as the prediction. The LOOCV procedure is known to mitigate overfitting and produce better generalization performance than marginal likelihood-based approaches (Rasmussen and Williams 2006; Vehtari *et al.* 2017).

Currently, **BKP** supports two loss functions: the Brier score (i.e., squared error loss) and the log-loss (i.e., negative log-likelihood or cross-entropy).

Brier Score Let $\widetilde{\pi}_i = y_i/m_i$ denote the empirical success proportion at input location $\boldsymbol{x}_i$. The LOOCV Brier score is defined as

$$\mathrm{BS}(\boldsymbol{\theta}; \mathcal{D}_n) = \frac{1}{n}\sum_{i=1}^{n}\left\{\widehat{\pi}_n^{-i}(\boldsymbol{x}_i) - \widetilde{\pi}_i\right\}^2, \tag{5}$$

which penalizes the squared deviation between the predicted and observed success proportions.

Log-Loss The log-loss (or cross-entropy) is widely used for probabilistic classification. Based on the LOOCV predictions, it is defined as

$$\mathrm{LL}(\boldsymbol{\theta}; \mathcal{D}_n) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log \widehat{\pi}_n^{-i}(\boldsymbol{x}_i) + (m_i - y_i)\log\left\{1 - \widehat{\pi}_n^{-i}(\boldsymbol{x}_i)\right\}\right]. \tag{6}$$

It corresponds to the negative log-likelihood of the binomial model evaluated at the LOOCV predictive probabilities.

Remark 3. *Although both criteria rely on LOOCV estimates, they emphasize different aspects of predictive performance (Gneiting and Raftery 2007; Flores* et al. *2025). The Brier score directly penalizes squared deviations from empirical proportions, making it more robust to poorly calibrated probabilities and better suited for smooth probability estimation under binomial data. In contrast, the log-loss penalizes overconfident mispredictions more heavily and is highly sensitive to the accuracy of the predicted probabilities. In practice, the choice between the two depends on whether calibration quality or robustness is prioritized in the evaluation (DRatings 2023).*

Hyperparameter Optimization

To improve optimization stability and reduce irregularities near the boundary of the parameter space, we adopt the reparameterization strategy proposed by MacDonald *et al.* (2015). Specifically, we transform the kernel scale parameters via $\gamma_j = \log_{10}(\theta_j)$ for $j = 1, \ldots, d$, and denote $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_d)$. For example, the Gaussian kernel becomes

$$k(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\gamma}) = \exp\left\{-\sum_{j=1}^{d}\left(\frac{x_j - x_j'}{10^{\gamma_j}}\right)^2\right\}. \tag{7}$$

Although gradient-based optimizers such as L-BFGS-B (Byrd *et al.* 1995) are computationally efficient, their performance can be sensitive to initialization in non-convex settings. To improve robustness, we adopt a multi-start strategy based on a Latin Hypercube design (LHD) of $n_0 = 10d$ initial runs within a carefully constructed region (Loeppky *et al.* 2009).

By MacDonald *et al.* (2015), the Gaussian kernel value $k(\boldsymbol{x}, \boldsymbol{x}'; \boldsymbol{\gamma})$ typically lies within the interval $[0.0067, 0.9999]$, approximately $[\exp(-5), \exp(-10^{-4})]$, to ensure stable numerical behavior. Assuming a space-filling design of size $n = 10d$ over $[0, 1]^d$ and isotropic smoothness

(i.e., $\gamma_j = \gamma_0$ for all j),the minimum pairwise distance along each coordinate is roughly $|x_{ik} - x_{jk}| \approx 0.1$. This leads to the following search region for $\boldsymbol{\gamma}$:

$$\Omega_0 = \left[\frac{\log_{10} d - \log_{10} 500}{2},\ \frac{\log_{10} d + 2}{2}\right]^d. \tag{8}$$

For the mat'{e}rn kernels, we adopt the same search region.

The initial values $\boldsymbol{\gamma}^{(1)}, \ldots, \boldsymbol{\gamma}^{(n_0)} \subset \Omega_0$ are drawn using LHD. For each initial value, solve

$$\widehat{\boldsymbol{\gamma}}^{(i)} = \underset{\boldsymbol{\gamma} \in \Omega}{\operatorname{argmin}}\ L(\boldsymbol{\gamma}; \mathcal{D}_n), \quad i = 1, \ldots, n_0,$$

where $L(\cdot)$ is the chosen loss function from (5) or (6), and $\Omega = [-10, 10]^d$ is a broader search region than Ω_0. The final estimate is chosen as the best local minimum through

$$\widehat{\boldsymbol{\gamma}} = \underset{1 \le i \le n_0}{\operatorname{argmin}}\ L(\widehat{\boldsymbol{\gamma}}^{(i)}; \mathcal{D}_n).$$

The procedure is summarized below:

1. Generate $10d$ initial values $\boldsymbol{\gamma}$ in Ω_0 using a space-filling LHD.
2. Run the L-BFGS-B algorithm from each initial value.
3. Select the solution that produces the lowest loss function.

Remark 4. *The implementation of LOOCV necessitates refitting the model n times, which can be computationally intensive. However, the closed-form posterior updates in the BKP model mitigate this cost by reducing the computational complexity to $\mathcal{O}(n^2)$. This represents a substantial improvement over the typical $\mathcal{O}(n^3)$ complexity of Gaussian process models with binomial likelihoods, as demonstrated in Example 3.*

2.4. Extension to Dirichlet Kernel Process

The BKP model can be naturally extended to handle multi-class responses via the Dirichlet Kernel Process (DKP) on replacing the binomial likelihood with a multinomial model and the Beta prior with a Dirichlet prior (MacKenzie *et al.* 2014).

Let the response at input $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^d$ be $\boldsymbol{y}(\boldsymbol{x}) = (y_1(\boldsymbol{x}), \ldots, y_q(\boldsymbol{x}))$, where $y_s(\boldsymbol{x})$ denotes the count of class s out of $m(\boldsymbol{x}) = \sum_{s=1}^q y_s(\boldsymbol{x})$ total trials. Assume

$$\boldsymbol{y}(\boldsymbol{x}) \sim \text{Multinomial}(m(\boldsymbol{x}), \boldsymbol{\pi}(\boldsymbol{x})),$$

with class probabilities $\boldsymbol{\pi}(\boldsymbol{x}) = (\pi_1(\boldsymbol{x}), \ldots, \pi_q(\boldsymbol{x}))$ satisfying $\sum_{s=1}^q \pi_s(\boldsymbol{x}) = 1$.

A Dirichlet prior is imposed on $\boldsymbol{\pi}(\boldsymbol{x})$ as

$$\boldsymbol{\pi}(\boldsymbol{x}) \sim \text{Dirichlet}(\boldsymbol{\alpha}_0(\boldsymbol{x})),$$

where $\boldsymbol{\alpha}_0(\boldsymbol{x}) = (\alpha_{0,1}(\boldsymbol{x}), \ldots, \alpha_{0,q}(\boldsymbol{x}))$ are prior concentration parameters. Given training data $\mathcal{D}_n = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$, define the response matrix as $\boldsymbol{Y} = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n]^\top \in \mathbb{R}^{n \times q}$. Then the kernel-smoothed conjugate posterior becomes

$$\boldsymbol{\pi}(\boldsymbol{x}) \mid \mathcal{D}_n \sim \text{Dirichlet}\left(\boldsymbol{\alpha}_n(\boldsymbol{x})\right), \quad \text{with} \quad \boldsymbol{\alpha}_n(\boldsymbol{x}) = \boldsymbol{\alpha}_0(\boldsymbol{x}) + \boldsymbol{k}^\top(\boldsymbol{x})\boldsymbol{Y}. \tag{9}$$

Table 2: Main functions provided in the **BKP** package.

Function	Description
`fit.BKP()`	Fit a Beta Kernel Process (BKP) model for binomial or binary data.
`predict.BKP()`	Make predictions using a fitted BKP model.
`simulate.BKP()`	Generate posterior samples from fitted BKP models.
`plot.BKP()`	Visualization for BKP model results.
`fit.DKP()`	Fit a Dirichlet Kernel Process (DKP) model for multinomial data.
`predict.DKP()`	Make predictions using a fitted DKP model.
`simulate.DKP()`	Generate posterior samples from fitted DKP models.
`plot.DKP()`	Visualization for DKP model results.

The posterior means and variances of the class probabilities are

$$\widehat{\pi}_{n,s}(\boldsymbol{x}) = \frac{\alpha_{n,s}(\boldsymbol{x})}{\sum_{s'=1}^{q} \alpha_{n,s'}(\boldsymbol{x})}, \quad \sigma^2_{n,s}(\boldsymbol{x}) = \frac{\widehat{\pi}_{n,s}(\boldsymbol{x})\{1-\widehat{\pi}_{n,s}(\boldsymbol{x})\}}{\sum_{s'=1}^{q} \alpha_{n,s'}(\boldsymbol{x}) + 1}, \quad s = 1, \ldots, q.$$

For classification tasks, labels are assigned by the maximum a posteriori (MAP) decision rule through

$$\widehat{y}(\boldsymbol{x}) = \underset{s\in\{1,\ldots,q\}}{\operatorname{argmax}}\ \widehat{\pi}_{n,s}(\boldsymbol{x}). \tag{10}$$

Prior choices (e.g., non-informative, fixed informative, or locally adaptive) follow similarly from the BKP framework in Section 2.2 by treating component-wise specification of prior class proportions.

Kernel hyperparameters are tuned by minimizing LOOCV-based loss functions as in Section 2.3, where the multi-class Brier score and log-loss are defined as:

$$\mathrm{BS}_{\mathrm{multi}}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\sum_{s=1}^{q}\left\{\widehat{\pi}^{-i}_{n,s}(\boldsymbol{x}_i) - \frac{y_{i,s}}{m_i}\right\}^2, \tag{11}$$

$$\mathrm{LL}_{\mathrm{multi}}(\boldsymbol{\theta}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{s=1}^{q} y_{i,s}\log\widehat{\pi}^{-i}_{n,s}(\boldsymbol{x}_i). \tag{12}$$

These multi-class loss functions reduce to (5) and (6) when $q = 2$.

3. BKP Package

The **BKP** package provides the implementation of the Beta Kernel Process (BKP) and its extension to the Dirichlet Kernel Process (DKP). The package is designed with usability and extensibility in mind, offering a modular structure with separate functions for model fitting, prediction, simulation, kernel construction, prior specification, and hyperparameter tuning. Users can specify the kernel type and prior form to adapt to various application scenarios. The core functionality of the package is summarized in Table 2.

To install the package, one can install the stable version from CRAN using `install.packages("BKP")` or the development version from GitHub via `pak::pak("Jiangyan-Zhao/BKP")`.

3.1. BKP Model

We first present the BKP model in detail. The DKP model will be described in Section 3.2, with a focus on differences from the BKP formulation.

The function `fit.BKP()` fits the BKP model (1) for binomial or binary response data. Its main arguments are:

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

The first group of arguments specifies the core data inputs: `X` is an $n \times d$ input matrix, `y` is the vector of observed successes, and `m` is the corresponding vector of total trials. The `Xbounds` argument is used to constrain or confine `X` to the unit hypercube $[0,1]^d$, facilitating kernel hyperparameter optimization. When `Xbounds = NULL` (default), the inputs are assumed to be already standardized in $[0,1]^d$.

The second group defines the prior distribution for the latent success probability surface. The `prior` argument controls the prior type, with available options: `"noninformative"` (default), `"fixed"`, and `"adaptive"`, corresponding to the formulations in Section 2.2. The parameters `r0` and `p0` specify the prior strength and prior mean for the latter two options.

The third group specifies the loss function and optimization settings. The `loss` argument chooses the criterion for kernel hyperparameter estimation: `"brier"` (default), based on the Brier score in (5), and `"log_loss"`, based on the log loss in (6). The `kernel` argument selects the kernel function from `"gaussian"` (default), `"matern32"`, and `"matern52"` in Table 1. The `n_multi_start` argument determines the number of random initial points in the multi-start optimization, with the default setting (`NULL`) being $10d$. Alternatively, the `theta` argument allows users to supply a fixed positive kernel length scale, either as a scalar (applied to all dimensions) or a vector of length d. When `theta` is provided, the multi-start optimization is skipped and the specified value is used directly. The optimization, when invoked, is carried out using the `multistart()` function from the **optimx** package (Nash and Varadhan 2011; Nash 2014).

Remark 5. *The current version is implemented entirely in R. Computational efficiency can be further improved by implementing additional components in C++ via* ***Rcpp*** *(Eddelbuettel and François 2011), and by enabling options that leverage parallel computing architectures, for example, to accelerate multi-start optimization routines used in hyperparameter tuning.*

`fit.BKP()` returns an object of class 'BKP', which contains: the estimated model hyperparameters $\boldsymbol{\theta}$ (`theta_opt`); the minimum achieved loss value (`loss_min`); the estimated posterior parameters $\alpha_n(\boldsymbol{x})$ and $\beta_n(\boldsymbol{x})$ (stored as `alpha_n` and `beta_n`); and other input arguments.

The resulting 'BKP' object can be directly used as the `object` argument in the functions `predict()`, `simulate()`, and `summary()`, or as the `x` argument in the functions `plot()` and `print()`.

Assume `BKPmodel` is an object of class 'BKP'. The function call:

```
predict(BKPmodel, Xnew, CI_level = 0.95, threshold = 0.5, ...)
```

returns the predictive mean of the success probability $\widehat{\pi}_n(\boldsymbol{x})$, the predictive variance $\sigma_n^2(\boldsymbol{x})$, and the lower and upper bounds of the `CI_level` credible interval for each $\boldsymbol{x}$ in `Xnew`. If `m = 1`, the function also returns the predicted binary label (0 or 1), determined by comparing the posterior mean to the specified `threshold`, after the classification rule in (4).

The function call:

```
simulate(BKPmodel, Xnew, n_sim = 1, threshold = NULL, ...)
```

generates random samples from the posterior predictive distribution of a fitted BKP model at new input locations. This function draws `n_sim` samples from the posterior Beta distribution at each row of `Xnew`, representing the distribution over success probabilities learned by the BKP model. If `threshold` is specified and `m = 1`, binary class labels (0 or 1) are generated for each simulated value by comparing it to the threshold. The optional argument `seed` allows reproducibility of the simulations. These samples can be used in various decision-making contexts, such as Bayesian optimization via Thompson sampling (Garnett 2023).

The `plot()`, `summary()`, and `print()` methods provide graphical, numerical, and textual summaries of a fitted '`BKP`' object, respectively. Plotting is supported for $d \leq 2$.

For one-dimensional inputs ($d = 1$), the `plot()` method displays the posterior mean of the predicted success probability as a line plot. A shaded region represents the 95% credible interval, and the observed proportions ($\widetilde{\pi}_i = y_i/m_i$) are shown as points overlaid on the plot. When used for classification, a horizontal dashed line is added to indicate the classification threshold.

For two-dimensional inputs ($d = 2$), the `plot()` method returns a 2×2 grid of contour plots illustrating the posterior mean, posterior variance, and the 2.5% and 97.5% quantiles of the predictive distribution, providing a comprehensive view of the model's predictions and associated uncertainty across the input space. When used for classification, only the posterior mean and posterior variance surfaces are displayed. When the argument `only_mean = TRUE` is specified, only the contour plot of the predictive mean surface is shown. This option is useful when a simplified visualization of the predictive mean is preferred.

3.2. DKP Model

The function `fit.DKP()` fits the DKP model (9), which generalizes the BKP model to multinomial response data with $q > 2$ classes. The overall structure and interface of `fit.DKP()` closely follow those of `fit.BKP()`, including support for prior specification, loss-based kernel hyperparameter optimization, and multi-start routines. The response input `Y` should be an $n \times q$ matrix of observed counts, where each row corresponds to an observation and each column to a category. The argument `m` is omitted in this setting, as the total count is implicitly defined by the row sums of `Y`. If `prior = "fixed"`, the user must provide a fixed prior mean vector p_0 of length q.

The function returns an object of class '`DKP`', which contains analogous elements to the '`BKP`' class, including the optimized kernel parameters, posterior parameters $\boldsymbol{\alpha}_n(\boldsymbol{x})$ (one vector per $\boldsymbol{x}$), and the input settings.

Prediction, simulation, and plotting methods are also extended from the BKP setting. The method `predict()` returns the same components as for class '`BKP`', except it produces per-class outputs. If all row sums of `Y` are 1, the function returns the predicted multi-class label

following the classification MAP rule in (10). The `simulate()` method generates draws from the posterior predictive Dirichlet distribution, optionally producing sampled class labels via Thompson sampling. Plotting for $d = 1$ displays one curve per class, while plotting for $d = 2$ generates a panel of contour plots for the predictive mean or other quantities of each class.

4. Examples using BKP

4.1. BKP Model

This subsection presents detailed illustrative examples based on the BKP model. Examples for the DKP model are provided in Section 4.2.

Example 1 Let $x \in [-2, 2]$, and suppose the true Bernoulli probability function is given by

$$\pi_1(x) = \frac{1}{1 + e^{-3x}}, \tag{13}$$

which is referred as the function `true_pi_fun` in the code below. We aim to fit the BKP model based on seven input locations that are uniformly distributed over $[-2, 2]$, with each location associated with a binomial observation having a maximum trial count of 100. The input locations are generated using the `lhs()` function from the R package **tgp** (Gramacy and Taddy 2010). The following R code illustrates how to simulate the data and fit the BKP model using the `fit.BKP()` function.

```
R> n <- 7
R> Xbounds <- matrix(c(-2,2), nrow = 1)
R> X <- lhs(n = n, rect = Xbounds)
R> true_pi <- true_pi_fun(X)
R> m <- sample(100, n, replace = TRUE)
R> y <- rbinom(n, size = m, prob = true_pi)
R> BKP_model_1D_1 <- fit.BKP(X, y, m, Xbounds = Xbounds)
```

The estimates of the parameters of the fitted BKP model can be displayed using the `print()` function:

```
R> print(BKP_model_1D_1)
```

```
------------------------------------------------
          Beta Kernel Process (BKP) Model
------------------------------------------------
Number of observations (n):  7
Input dimensionality (d):    1
Kernel type:                 gaussian
Loss function used:          brier
Optimized kernel parameters: 0.1748
Minimum achieved loss:       0.01165
```

```
Kernel parameters were obtained by optimization.

Prior specification:
  Noninformative prior: Beta(1,1).
-------------------------------------------------
```

The `BKP_model_1D_1` object can be used for visualization and prediction over a grid of input values via the `plot()` and `simulate()` methods:

```
R> plot(BKP_model_1D_1)
R> Xnew = matrix(seq(-2, 2, length = 100), ncol = 1)
R> sim <- simulate(BKP_model_1D_1, Xnew = Xnew, n_sim = 3)
```

Figure 1 presents two views of the model output. Panel (a) shows the posterior mean estimate of the probability function $\pi(x)$ (blue), along with a 95% credible interval (gray band), observed proportions (red dots), and the true underlying probability function (black). Panel (b) presents three posterior sample curves generated using the `simulate()` method, illustrating the variability of the estimated probability surface.

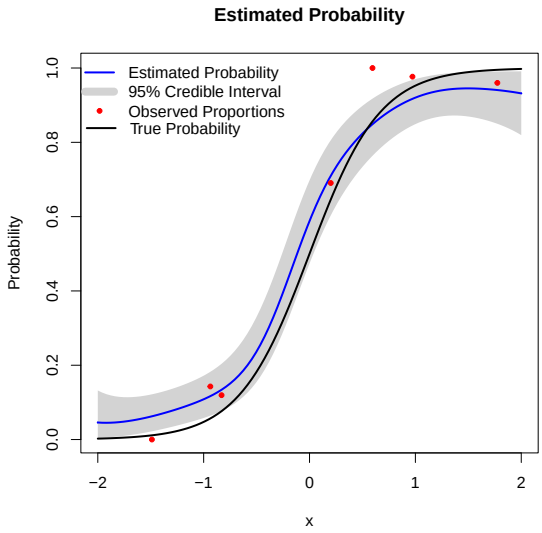
We continue with the same probability function of this example to show the impact of the global precision parameters, r_0, in classification tasks. However, the sample size is changed from 7 to 20 for classification. The following R code generates the label of response and fit the BKP model for classification with r_0 being 0.01 and 2.

```
R> # Fit BKP model with r0 = 0.01
R> BKP_model_1D_1_class_1 <- fit.BKP(
+    X, y, m, Xbounds = Xbounds,
+    prior = "fixed", r0 = 0.01, loss = "log_loss")
R> # Fit BKP model with r0 = 2
R> BKP_model_1D_1_class_2 <- fit.BKP(
+    X, y, m, Xbounds = Xbounds,
+    prior = "fixed", r0 = 2, loss = "log_loss")
```

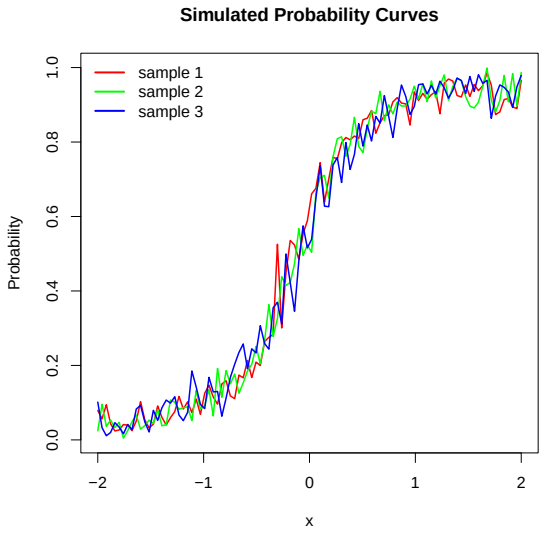
Figure 2a shows a sigmoidal curve with steep slope around zero (in solid line) and a narrow 95% credible interval (in grey band) under $r_0 = 0.01$, indicating a decisive and confident classification boundary. In contrast, Figure 2b displays a sine-shape curve with a much wider credible interval, reflecting greater uncertainty and less effective separation. This demonstrates the preference of a small value of r_0 value for classification task.

Example 2 The first example is essentially a generalized linear model with a smooth logit link, and thus poses limited modeling complexity. To demonstrate the capability of the BKP model in handling more challenging classification structures, we consider a second example with a highly nonlinear underlying probability surface. Define the true Bernoulli probability as

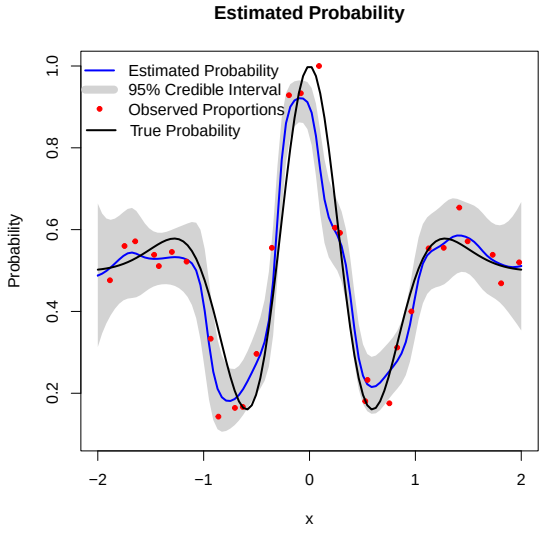
$$\pi_2(x) = \frac{1}{2}\left[1 + e^{-x^2}\cos\left(10\frac{1-e^{-x}}{1+e^{-x}}\right)\right], \tag{14}$$



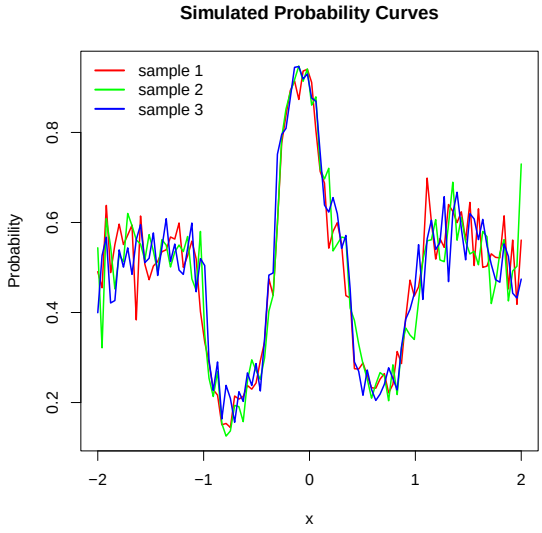
(a) Posterior mean and 95% CI of Example 1



(b) Posterior samples of Example 1



(c) Posterior mean and 95% CI of Example 2

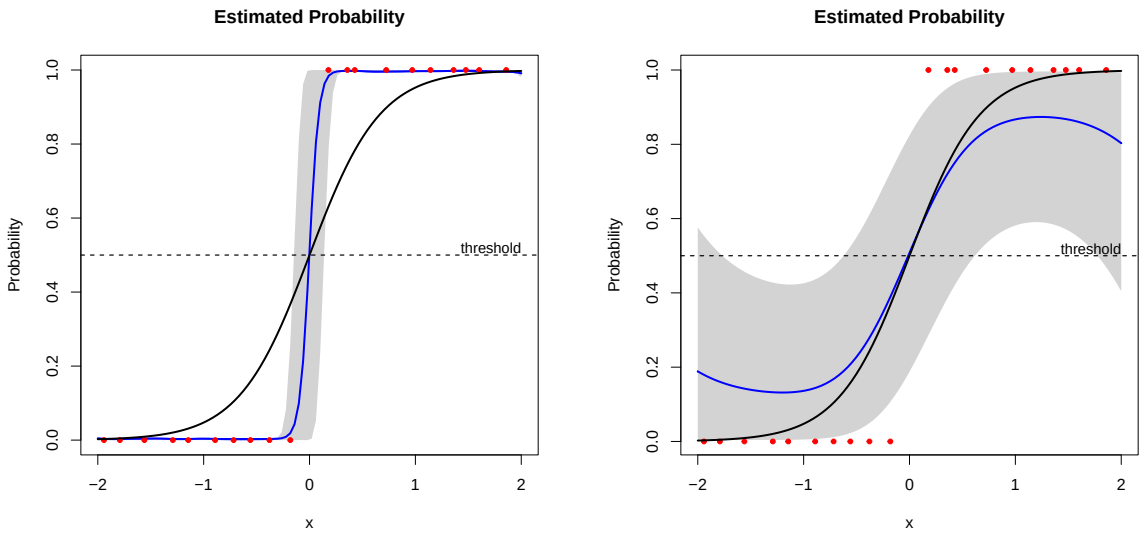


(d) Posterior samples of Example 2

Figure 1: Posterior inference and simulation results from the fitted BKP models

where $x \in [-2, 2]$ (Goetschalckx *et al.* 2011). This example involves rapid local oscillations and strong nonlinearity, making it substantially more difficult to fit than Example 1. Here, we increase the number of locations to 30.

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



(a) $r_0 = 0.01$ (b) $r_0 = 2$

Figure 2: Posterior mean and 95% CI of Example 1 (classification task) under $r_0 = 0.01$ and 2

```
R> y <- rbinom(n, size = m, prob = true_pi)
R> BKP_model_1D_2 <- fit.BKP(X, y, m, Xbounds = Xbounds)
```

The results are shown in panels (c) and (d) of Figure 1. Panel (c) demonstrates that the BKP model accurately recovers the highly nonlinear pattern, while panel (d) illustrates posterior variability through three representative realizations of $\pi(x)$.

Example 3 We now consider a two-dimensional test function to further illustrate the modeling capabilities of the **BKP** package. Let $\boldsymbol{x} \in [0,1]^2$, and define the latent surface using a re-scaled version of the Goldstein–Price function (Picheny *et al.* 2013):

$$
\begin{aligned}
f(\boldsymbol{x}) =& \frac{\log[\{1 + a(\boldsymbol{x})\}\{30 + b(\boldsymbol{x})\} - 8.6928]}{2.4269}, \quad \text{with} \\
a(\boldsymbol{x}) =& (4x_1 + 4x_2 - 3)^2 \times \\
& \{75 - 56\,(x_1 + x_2) + 3\,(4x_1 - 2)^2 + 6\,(4x_1 - 2)\,(4x_2 - 2) + 3\,(4x_2 - 2)^2\}, \\
b(\boldsymbol{x}) =& (8x_1 - 12x_2 + 2)^2 \times \\
& \{-14 - 128x_1 + 12\,(4x_1 - 2)^2 + 192x_2 - 36\,(4x_1 - 2)\,(4x_2 - 2) + 27\,(4x_2 - 2)^2\}.
\end{aligned}
$$

The true Bernoulli probability surface is then defined by

$$\pi_3(\boldsymbol{x}) = \Phi\{f(\boldsymbol{x})\}, \tag{15}$$

where $\Phi(\cdot)$ is the cumulative distribution function of the standard normal distribution. This formulation produces a smooth yet highly non-linear response surface, providing a challenging test scenario for probabilistic modeling.

To construct the training data, we generate a LHD of size 100 over $[0,1]^2$. Each location is associated with a binomial observation whose number of trials is randomly drawn from $\{1,\ldots,100\}$. The following R code demonstrates the data simulation and model fitting using the `fit.BKP()` function:

```
R> n <- 100
R> Xbounds <- matrix(c(0, 0, 1, 1), nrow = 2)
R> X <- lhs(n = n, rect = Xbounds)
R> true_pi <- true_pi_fun(X)
R> m <- sample(100, n, replace = TRUE)
R> y <- rbinom(n, size = m, prob = true_pi)
R> BKP_model_2D <- fit.BKP(X, y, m, Xbounds=Xbounds)
R> print(BKP_model_2D)

-------------------------------------------------
           Beta Kernel Process (BKP) Model
-------------------------------------------------
Number of observations (n):  100
Input dimensionality (d):    2
Kernel type:                 gaussian
Loss function used:          brier
Optimized kernel parameters: 0.1112, 0.0680
Minimum achieved loss:       0.01041
Kernel parameters were obtained by optimization.

Prior specification:
  Noninformative prior: Beta(1,1).
-------------------------------------------------
```

Figure 3 is obtained by the following code, which is the heatmaps with contour lines for the true probability and the estimated.

```
R> plot(BKP_model_2D)
```

The results of model prediction are as follows:

```
R> Xnew <- lhs(n = 10, rect = Xbounds)
R> predict(BKP_model_2D, Xnew)

$Xnew
            [,1]       [,2]
 [1,] 0.21839993 0.10462329
 [2,] 0.94012023 0.61081728
 [3,] 0.67819130 0.80874492
 [4,] 0.15911245 0.91202275
 [5,] 0.78384305 0.25559477
```

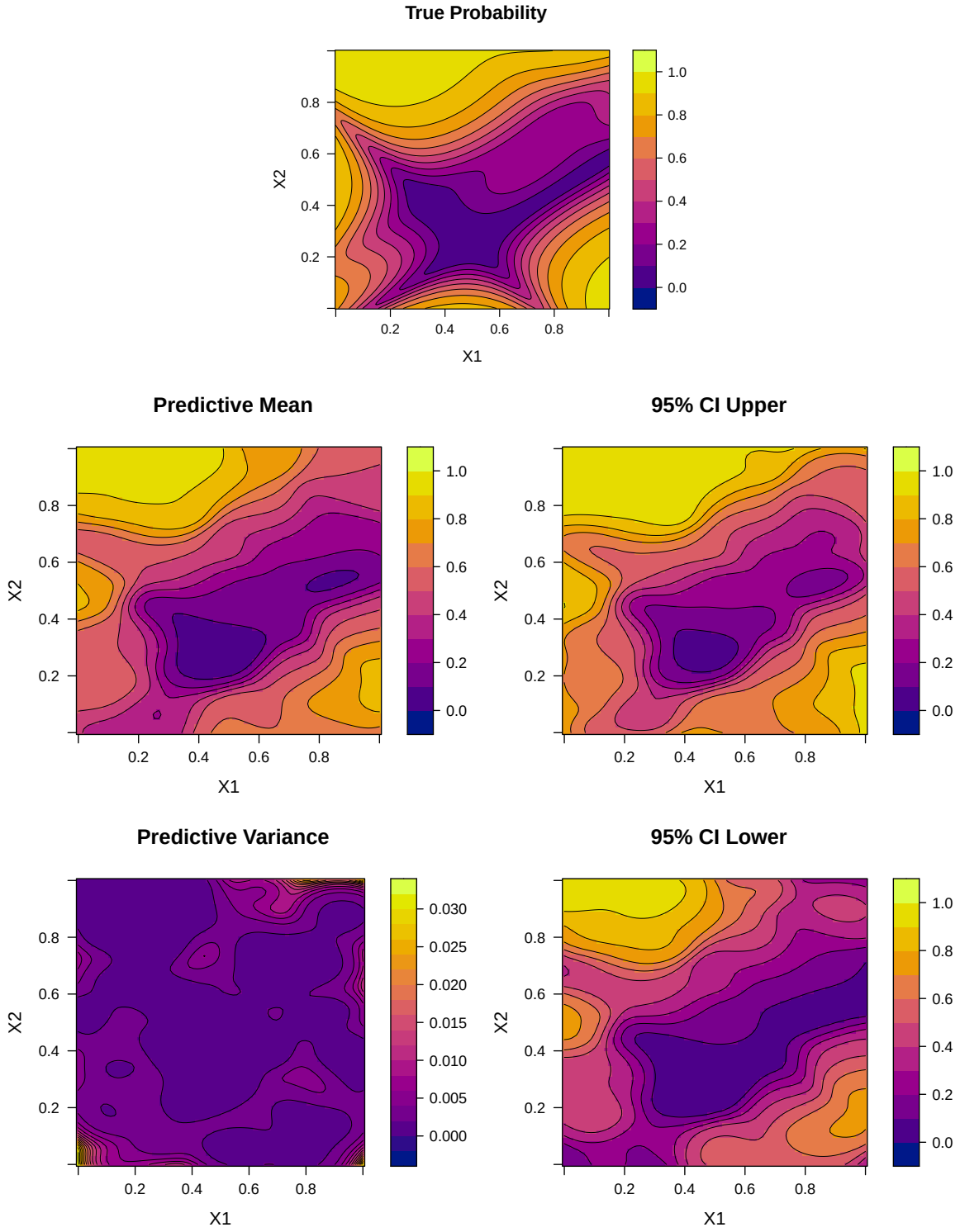


Figure 3: Comparison of the true probability surface and BKP-based posterior summaries of Example 3

```
 [6,] 0.41851057 0.52146871
 [7,] 0.30241571 0.08882792
 [8,] 0.88767862 0.33088498
 [9,] 0.08770149 0.48006939
[10,] 0.51597110 0.71151567

$mean
 [1] 0.3581193 0.1591143 0.5132974 0.9589202 0.4911282 0.2664934
 [7] 0.3105955 0.6079035 0.7455542 0.4478995

$variance
 [1] 0.0029847470 0.0041082794 0.0018284629 0.0003391657 0.0032395988
 [6] 0.0007819776 0.0022007004 0.0019417134 0.0021114379 0.0028286298

$lower
 [1] 0.25492486 0.05577439 0.42943483 0.91600288 0.38011034 0.21352972
 [7] 0.22263662 0.52002256 0.65058913 0.34506661

$upper
 [1] 0.4683872 0.3030328 0.5967884 0.9869524 0.6025878 0.3230099
 [7] 0.4060059 0.6924258 0.8300536 0.5530176

$CI_level
[1] 0.95
```

We continue with Example 3 to demonstrate the scalability of the BKP model in comparison to the logistic Gaussian process (LGP) model (Rasmussen and Williams 2006). Let the sample size range from 200 to 5000. Two scenarios for hyperparameter determination are considered, namely, i) one scenario with fixed value of `theta = 1`, and ii) the other scenario with hyperparameters optimized (through `n_multi_start = 1` or `n_multi_start = NULL`).

The LGP model was implemented using the `gp_fit()` and `gp_optim()` functions from the **gplite** package (Piironen 2022). Notably, the **gplite** package employs a multi-start optimization strategy that is only triggered upon optimization failure; therefore, the additional computational cost of multiple restarts was not included in our timing measurements.

All experiments were conducted on a workstation equipped with an Intel(R) Xeon(R) W-2235 CPU @ 3.80GHz (12 cores) and 16 GB RAM. The computation times reported here are averages over 20 independent repetitions to mitigate the effects of variability due to random initialization and computational fluctuations.

The average computation times for both models under these scenarios are presented in Figure 4. The observed computational costs align well with the theoretical complexity predictions, confirming the expected $\mathcal{O}(n^2)$ scalability for BKP and $\mathcal{O}(n^3)$ for the LGP model. Additionally, the computational cost of the multi-start optimization approach for BKP is roughly proportional to the number of restarts, which was set to $10d$. For the current setting, this corresponds to approximately 20 times the cost of the single-start method, reflecting the increased complexity incurred by multiple initializations. This trade-off between computational expense and potentially improved optimization robustness should be carefully considered when select-

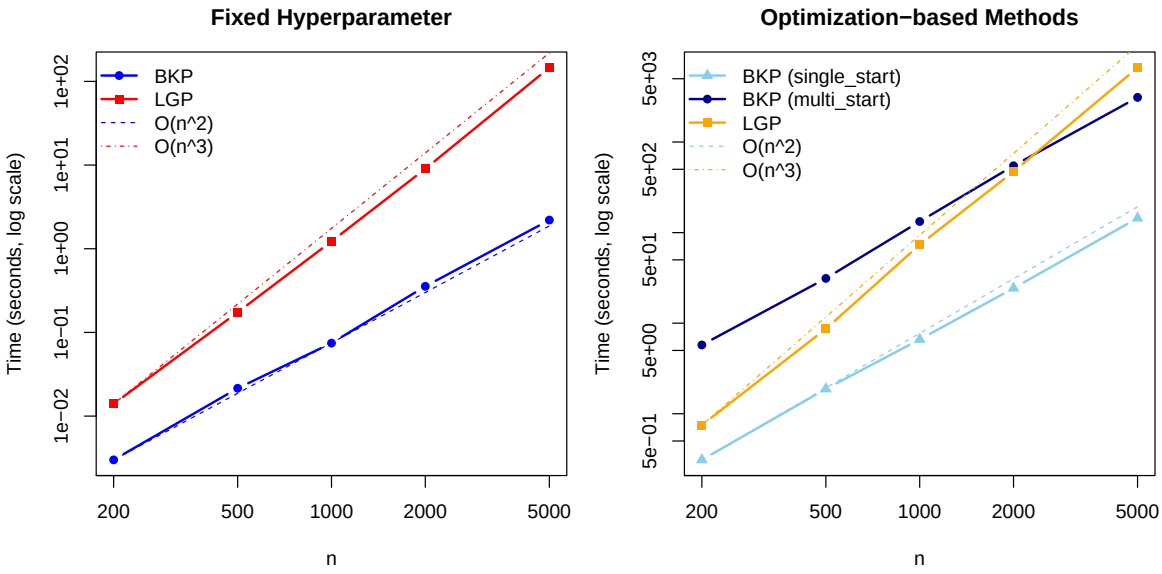


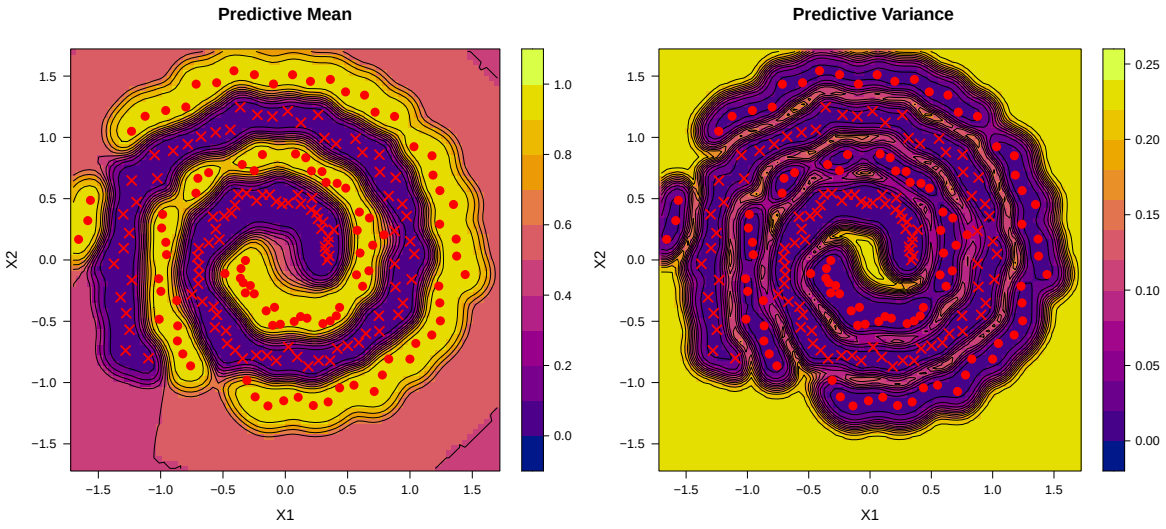
Figure 4: Comparison of computation times (in log scale) between BKP and LGP methods: (a) fixed hyperparameter; (b) optimization-based methods

ing an appropriate strategy.

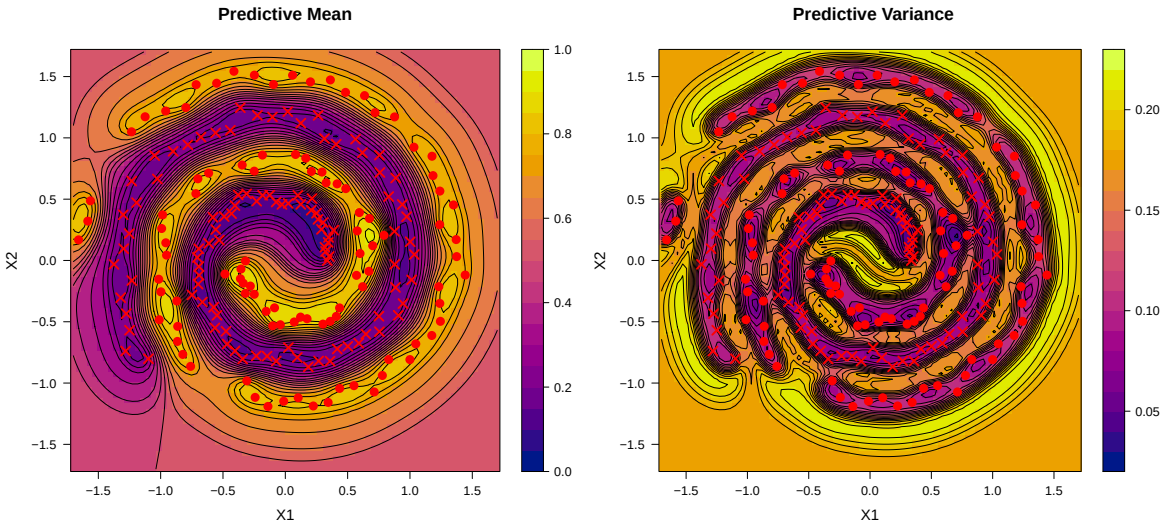
Example 4 We next consider a binary classification task using the *Two Spirals* dataset (Chalup and Wiklendt 2007), a well-known benchmark consisting of two intertwined spirals in a bounded two-dimensional input space. This dataset is particularly challenging due to the complex, non-linearly separable class structure.

We generate $n = 250$ observations using the `mlbench.spirals()` function from the R package **mlbench** (Leisch and Dimitriadou 2024), with two complete rotations and additive Gaussian noise of standard deviation `sd = 0.05`. The inputs $\boldsymbol{x}$ are constrained to the domain $[-1.7, 1.7]^2$, and the binary class labels are encoded as 0 and 1. We fit the BKP model using a fixed prior specification with `r_0 = 0.1` and `p_0 = 0.5`.

```
R> n <- 250
R> n_train <- 200
R> n_test <- 50
R> data <- mlbench.spirals(n, cycles = 2, sd = 0.05)
R> X_train <- data$x[1:n_train, ]
R> y_train <- as.numeric(data$classes[1:n_train]) - 1 # Convert to 0/1 for BKP
R> X_test <- data$x[(n_train + 1):n, ]
R> y_test <- as.numeric(data$classes[(n_train + 1):n]) - 1
R> m <- rep(1, n_train)
R> Xbounds <- rbind(c(-1.7, 1.7), c(-1.7, 1.7))
R> BKP_model_Class <- fit.BKP(
+    X_train, y_train, m, Xbounds = Xbounds,
+    prior = "fixed", r0 = 0.1, loss = "log_loss")
R> prediction <- predict(BKP_model_Class, X_test)
```



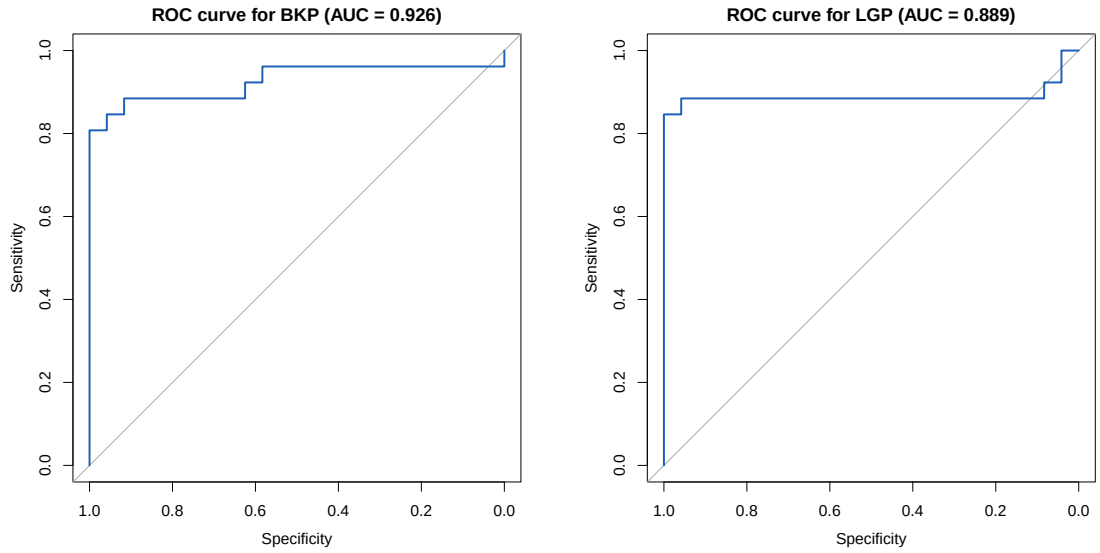
(a) BKP



(b) LGP

Figure 5: Predictions of the two spirals by BKP and LGP models in terms of posterior predictive mean (left panel) and predictive variance (right panel) of class probabilities, where color shading represents the estimated success probability, circles and crosses indicate training observations from the two classes

Figure 5 presents the predictive mean and variance surfaces of the class probabilities. The upper left panel shows that the BKP model accurately captures the intricate spiral structure, with smoothly varying predicted probabilities that delineate the nonlinear decision boundary. The upper right panel displays the associated predictive uncertainty, which is highest near the boundary regions between the spirals. Circles and crosses indicate training observations from the two respective classes. These results illustrate the capacity of BKP to flexibly model complex classification boundaries while providing coherent uncertainty quantification.



(a) BKP (b) LGP

Figure 6: ROC curves and their respective AUCs for BKP and LGP models

We compare the prediction performance with the LGP model implemented by R package **gplite**.

```
R> gp <- gp_init(cf = cf_sexp(), lik = lik_bernoulli())
R> gp <- gp_optim(gp, X_train, y_train, method = method_full(),
+                 approx = approx_ep(), verbose = FALSE)
R> prediction_gp <- gp_pred(gp, as.matrix(X_test), transform = TRUE)
```

The lower panel of Figure 5 indicates that the LGP model encounters difficulty in capturing the intricate geometry of the Two Spirals dataset. Although it roughly follows the spiral structure, its predictive mean exhibits a more irregular and less sharply defined decision boundary compared to the BKP model. This pattern suggests that the LGP model may be more prone to overfitting or less capable of generalizing effectively. The corresponding predictive variance plots reinforce this interpretation when viewed separately in regions with and without training data. In regions where data are available, the LGP model exhibits elevated variance (brighter regions) across a broader portion of the input space indicating greater uncertainty in areas critical for classification. In contrast, the BKP model maintains lower variance concentrated more closely around the decision boundary, reflecting higher confident predictions. Interestingly, in regions without training data, the LGP model's uncertainty is not necessarily the highest; instead, it sometimes shows comparatively lower variance than the BKP model. This suggests that the LGP model may over-smooth or fail to express appropriate uncertainty far from observed data, whereas BKP better captures uncertainty behavior in unobserved regions. The receiver operating characteristic (ROC) curves and the corresponding area under the curve (AUC) values in Figure 6 further corroborate this conclusion: BKP achieves an AUC of 0.926, compared to 0.889 for LGP, reflecting superior discriminatory performance and a better balance between sensitivity and specificity.

4.2. DKP Model

Example 5 Consider a one-dimensional three-class classification problem. The input is defined on the interval $x \in [-2, 2]$, and the true class probability vector is given by

$$\boldsymbol{\pi}(x) = \left[\frac{\pi_1(x)}{2}, \frac{\pi_2(x)}{2}, 1 - \frac{\pi_1(x)}{2} - \frac{\pi_2(x)}{2}\right]^\top,$$

where $\pi_1(x)$ and $\pi_2(x)$ are smooth functions defined in (13) and (14), respectively.

We generate $n = 30$ input locations using Latin hypercube sampling over the interval $[-2, 2]$. At each location, the response is a multinomial vector with probability $\boldsymbol{\pi}(x)$ and a random total count sampled from $\{1, \ldots, 150\}$. The DKP model is then fitted using the `fit.DKP()` function and visualized with the `plot()` method:

```
R> n <- 30
R> Xbounds <- matrix(c(-2, 2), nrow = 1)
R> X <- lhs(n = n, rect = Xbounds)
R> true_pi <- true_pi_fun(X)
R> m <- sample(150, n, replace = TRUE)
R> Y <- t(sapply(1:n, function(i) rmultinom(1, size = m[i], prob = true_pi[i, ])))
R> DKP_model_1D <- fit.DKP(X, Y, Xbounds = Xbounds)
R> plot(DKP_model_1D)
```

Figure 7 shows that the DKP model accurately recovers the true class probability functions and provides meaningful uncertainty quantification. The predictive mean curves align with the true underlying structure, and the shaded bands reflect posterior uncertainty.

Example 6 Consider a two-dimensional three-class classification problem. Let $\boldsymbol{x} = [x_1, x_2]^\top \in [0, 1]^2$, and define the true class probability function as

$$\boldsymbol{\pi}(\boldsymbol{x}) = \left[\frac{\pi_3(\boldsymbol{x})}{2}, \frac{\pi_4(\boldsymbol{x})}{2}, 1 - \frac{\pi_3(\boldsymbol{x})}{2} - \frac{\pi_4(\boldsymbol{x})}{2}\right]^\top,$$

where $\pi_3(\boldsymbol{x})$ is defined in (15), and

$$\pi_4(\boldsymbol{x}) = \sin(\pi x_1)\cos\{\pi(x_2 - 0.5)\}.$$

We generate $n = 100$ input points using Latin hypercube sampling over $[0, 1]^2$. At each location, the response is a multinomial vector with probability $\boldsymbol{\pi}(\boldsymbol{x})$ and a total count randomly sampled from $\{1, \ldots, 150\}$. We fit the DKP model using `fit.DKP()` and visualize the results with the `plot()` method:

```
R> n <- 100
R> Xbounds <- matrix(c(0, 0, 1, 1), nrow = 2)
R> X <- lhs(n = n, rect = Xbounds)
R> true_pi <- true_pi_fun(X)
R> m <- sample(150, n, replace = TRUE)
R> Y <- t(sapply(1:n, function(i) rmultinom(1, size = m[i], prob = true_pi[i, ])))
R> DKP_model_2D <- fit.DKP(X, Y, Xbounds=Xbounds)
```

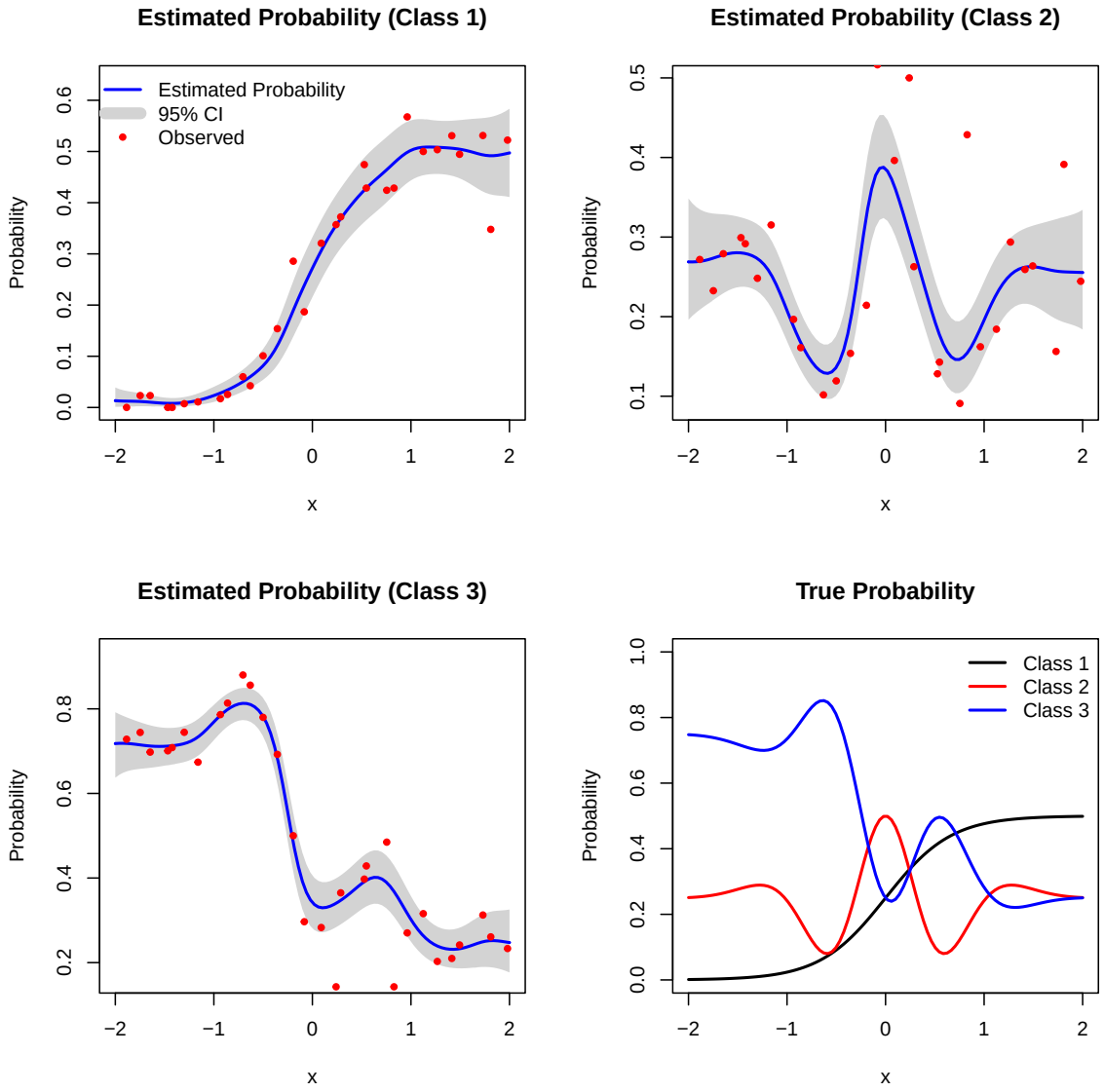


Figure 7: Posterior inference from the fitted DKP model for a one-dimensional three-class problem

Figures 8–10 display the ground truth surfaces and corresponding posterior inference for each of the three classes. In each figure, the top panel shows the true class probability surface, while the bottom row presents the posterior mean, variance, and uncertainty bounds produced by the DKP model.

Example 7 Consider another three-class classification task based on the well-known *Iris* dataset available in R. This classic benchmark dataset contains measurements of three iris species—*setosa*, *versicolor*, and *virginica*—each described by four features: sepal length, sepal width, petal length, and petal width. Due to the overlap in feature space, particularly be-

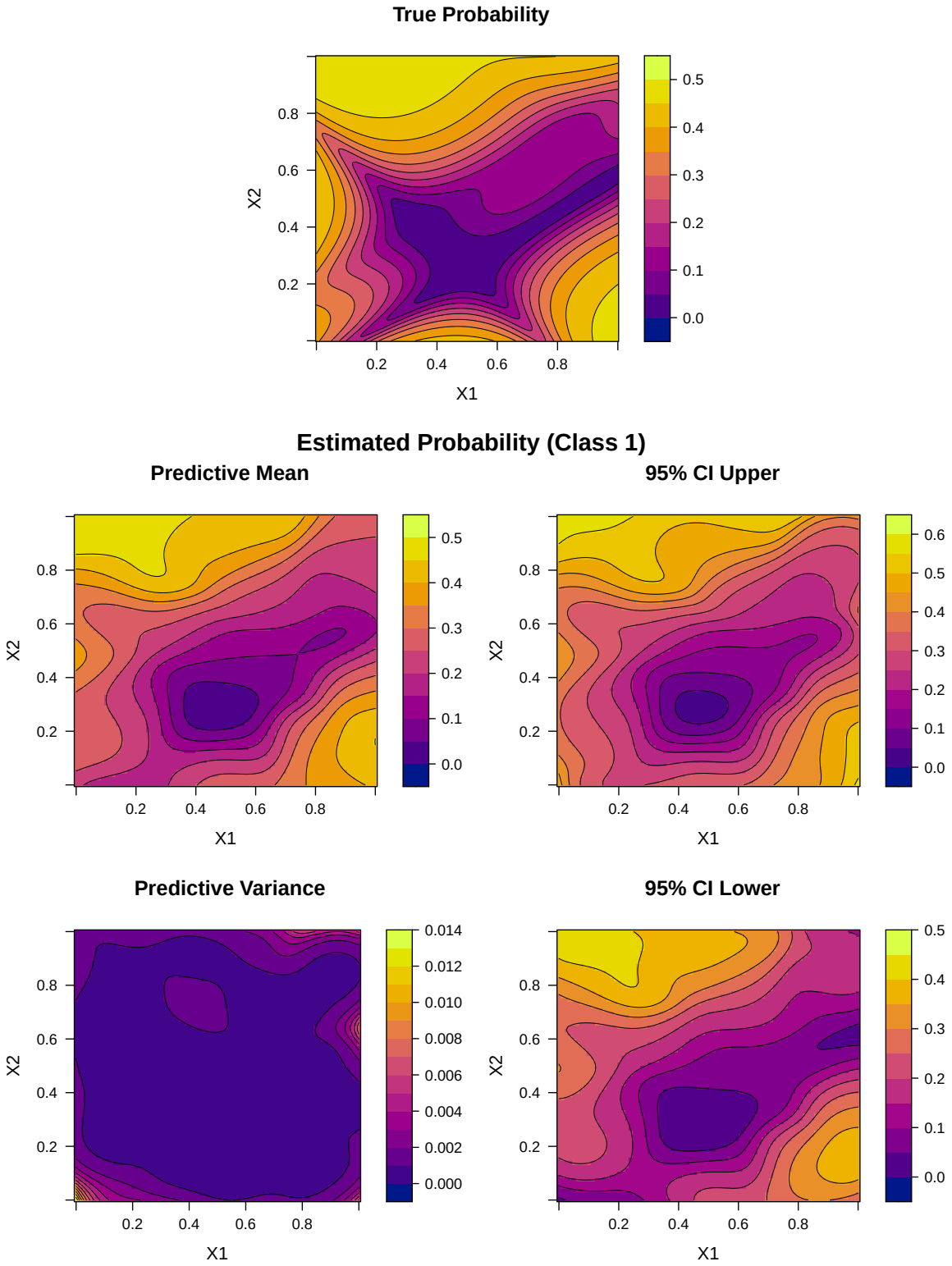


Figure 8: Class 1: true distribution (top) and DKP model posterior summaries (bottom)

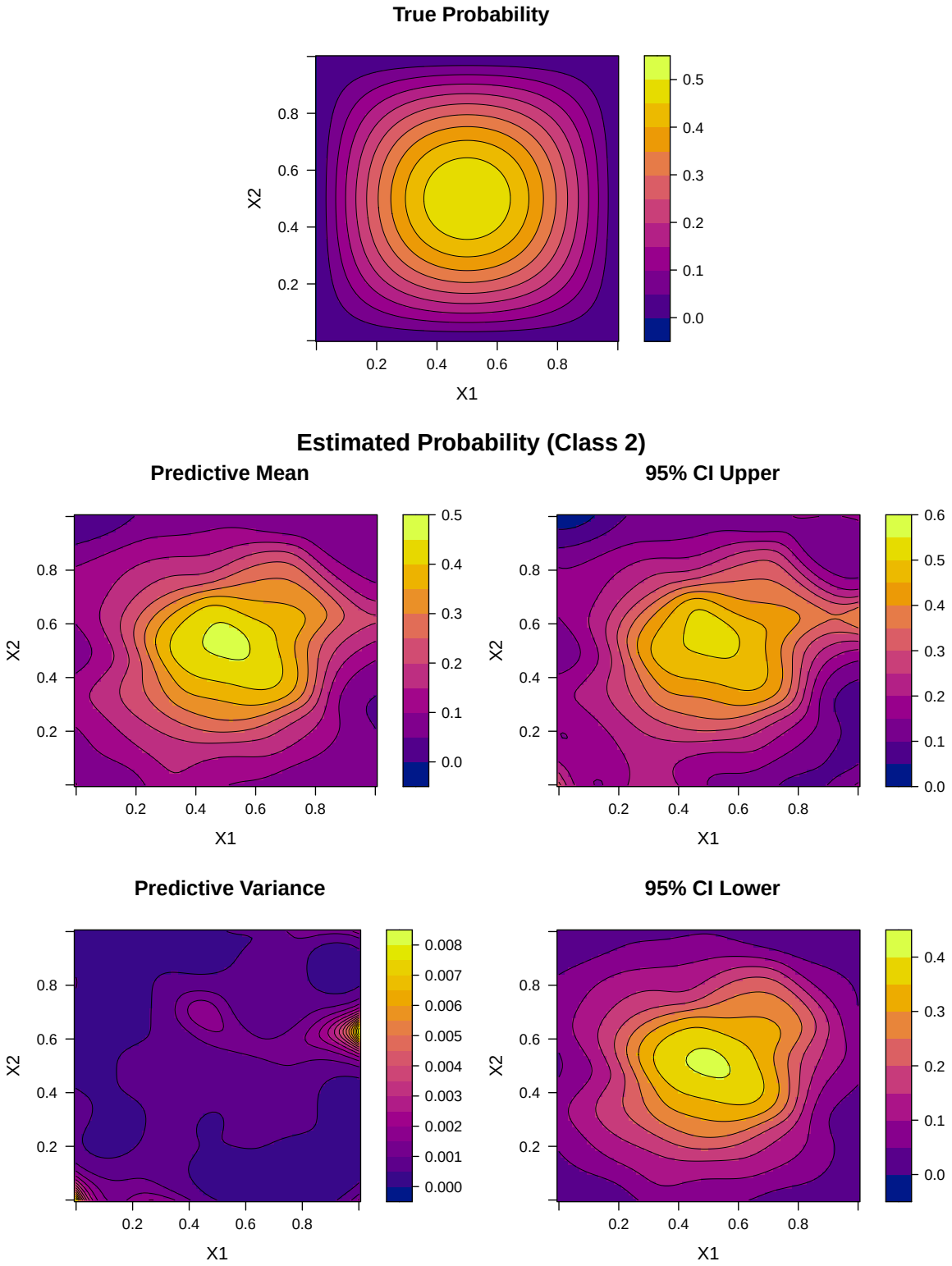


Figure 9: Class 2: true distribution (top) and DKP model posterior summaries (bottom)

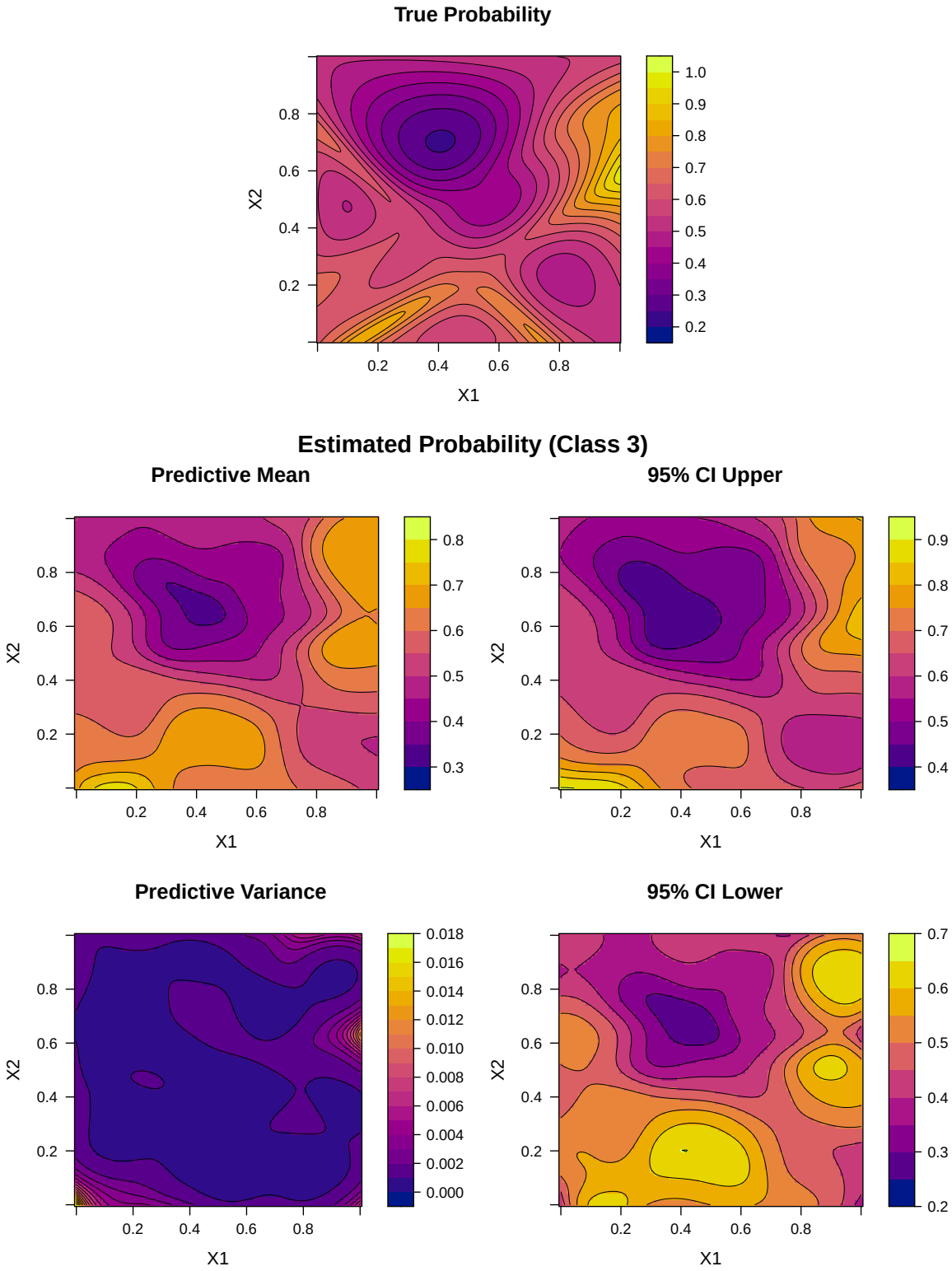


Figure 10: Class 3: true distribution (top) and DKP model posterior summaries (bottom)

tween *versicolor* and *virginica*, the class boundaries are not linearly separable, making this dataset a standard testbed for evaluating multi-class classification algorithms. For visualization purposes, we restrict our analysis to the first two features: sepal length and sepal width. We fit the DKP model using a fixed prior specification with `r0 = 0.1` and `p0 = rep(1/3, 3)`, assuming equal prior probability for each class.

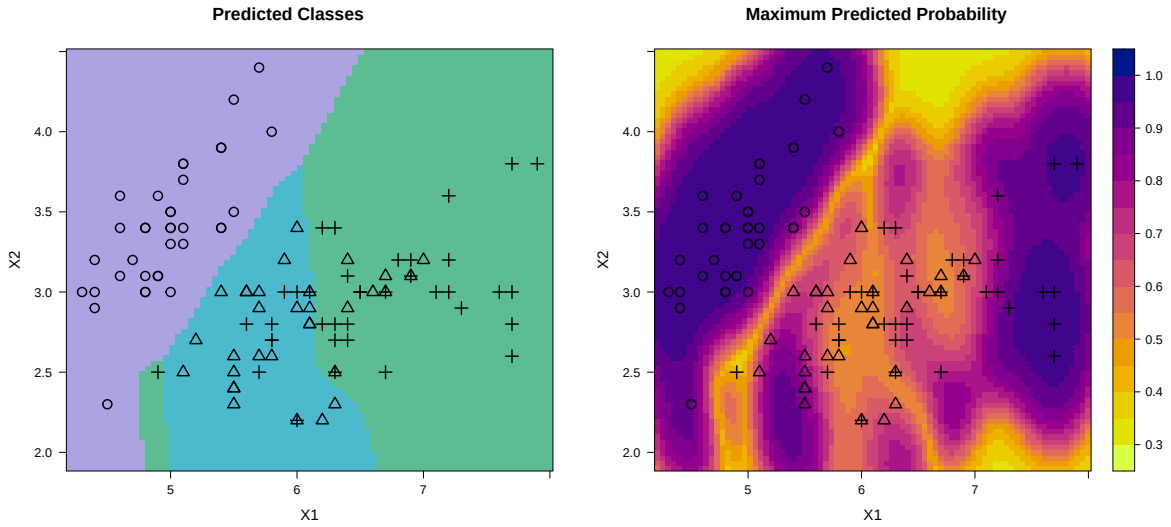
```
R> data(iris)
R> X <- as.matrix(iris[, 1:2])
R> Xbounds <- rbind(c(4.2, 8), c(1.9, 4.5))
R> labels <- iris$Species
R> Y <- model.matrix(~ labels - 1)  # expand factors to a set of dummy variables
R> train_indices <- sample(1:nrow(iris), 0.7 * nrow(iris))
R> X_train <- X[train_indices, ]
R> Y_train <- Y[train_indices, ]
R> DKP_model_Class <- fit.DKP(
+    X_train, Y_train, Xbounds = Xbounds, loss = "log_loss",
+    prior = "fixed", r0 = 0.1, p0 = rep(1/3, 3))
R> X_test <- X[-train_indices, ]
R> Y_test <- Y[-train_indices, ]
R> dkp_pred_probs <- predict(DKP_model_Class, X_test)$mean
```

Figure 11 provides a visual summary of the fitted DKP model. The upper left panel displays the MAP classification boundaries, which clearly separate *setosa* from the other two species. In contrast, the boundary between *versicolor* and *virginica* is more intricate, reflecting the known overlap between these two species in sepal measurements.

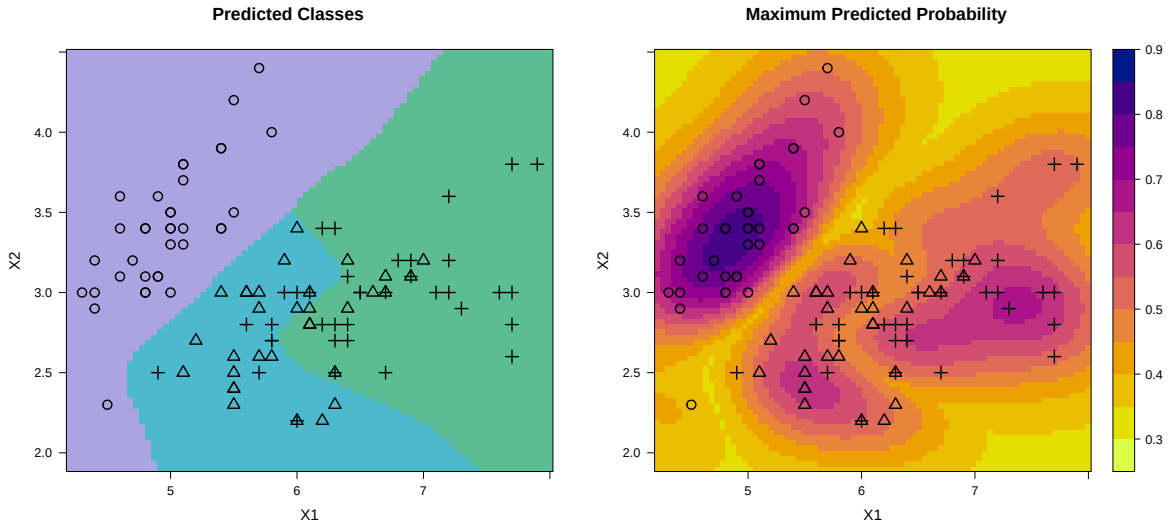
To further illustrate the model's prediction uncertainty, the upper right panel shows a contour map of the maximum predicted probability $\max_j \pi_j(\boldsymbol{x})$ across the input space. This diagnostic highlights regions where the classifier is most confident (values near 1) versus uncertain (values close to 1/3), which effectively identifies decision boundaries and ambiguous areas. Such visualization aids in understanding the reliability of classification decisions and the structure of the learned decision surfaces.

Again, we compare the performance with LGP model, which is implemented by R package **kernlab** (Karatzoglou *et al.* 2004). We adopt the one-vs-rest (OvR) approach for comparison, which works by training a separate binary classifier for each class. Each classifier is tasked with distinguishing its assigned class from all other classes combined.

```
R> iris_data <- data.frame(
+    Sepal.Length = iris$Sepal.Length,
+    Sepal.Width = iris$Sepal.Width,
+    Species = iris$Species
+  )
R> iris_train <- iris_data[train_indices, ]
R> iris_test <- iris_data[-train_indices, ]
R> gausspr_model <- gausspr(Species ~ ., data = iris_train,
+                            kernel = "rbfdot", kpar = "automatic")
R> lgp_pred_probs <- predict(gausspr_model, newdata = iris_test,
+                             type = "probabilities")
```



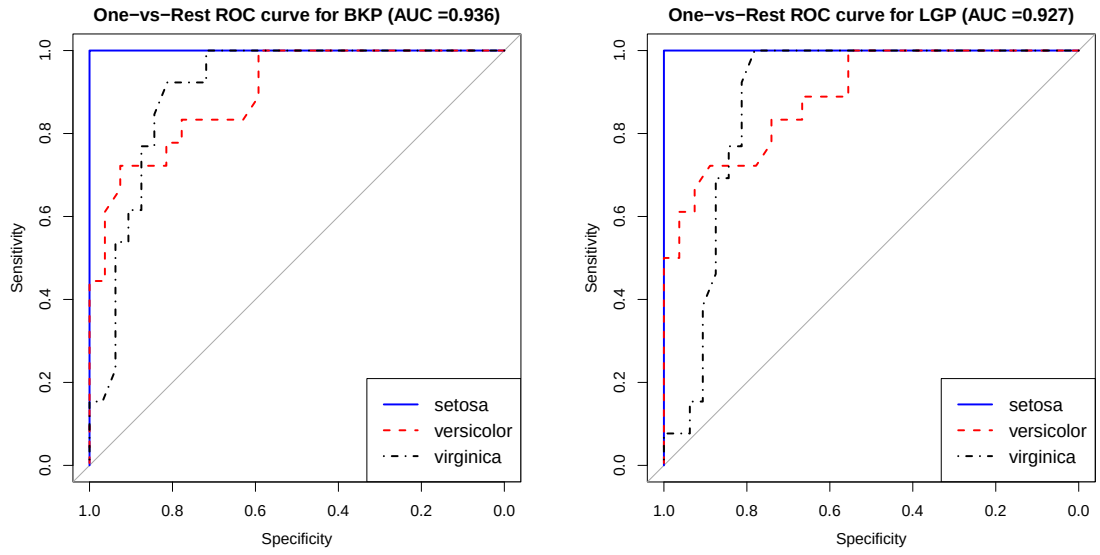
(a) DKP



(b) LGP

Figure 11: Classification and uncertainty visualization after applying the DKP model on the *Iris* dataset. The left panel shows the predicted classification regions based on the MAP decision rule, where each color corresponds to one of the three iris species. The training data points are overlaid using shape-coded markers. The right panel visualizes the model's predictive uncertainty using the maximum predicted class probability. Lower values (light-colored regions) indicate higher classification uncertainty, which usually occurs near the class boundaries.

It is seen that the decision boundaries in the `Predicted Classes'' plot (lower left panel) appear less smooth, particularly in the regions between the two clusters of `triangle' and `plus' data points. Additionally, the`Maximum Predicted Probability'' plot (lower right panel) shows a more fragmented and less uniform confidence landscape. There are pockets of high confidence, but also areas of lower confidence (yellows and



(a) BKP (b) LGP

Figure 12: One-vs-Rest ROC curves and their respective macro-average AUCs for BKP and LGP models

greens) scattered over the region, even within what appear to be the core regions of the classes. This indicates that DKP model is more accurate in prediction and more consistent in uncertainty quantification than the LGP model. The corresponding multiclass ROC curves with macro-average AUC (0.936 vs 0.927) are presented in Figure 12.

5. Summary and discussion

This article has presented **BKP**, a user-friendly R package that implements the Beta Kernel Process (BKP) – a scalable, interpretable, and fully Bayesian nonparametric framework for modeling spatially varying binomial probabilities. The package provides a flexible and modular interface for fitting BKP models to both binary and aggregated binomial data, and extends naturally to the Dirichlet Kernel Process (DKP) for handling multinomial and compositional responses with multiple categories. To our knowledge, **BKP** is the first publicly available software for implementing BKP methodology, thereby filling an important gap in the toolkit for spatially varying binomial and multinomial modeling.

Future development directions include extending the BKP framework to support more complex data structures, such as multivariate responses, functional data, time series, and combinations of qualitative and quantitative covariates. Another promising avenue is to generalize the BKP methodology under alternative likelihoods beyond the binomial family. For example, negative binomial likelihoods are particularly suitable for over-dispersed count data, where the variance exceeds the mean, a common phenomenon in ecological surveys, RNA-seq gene expression counts, and epidemiological incidence data. Geometric likelihoods, as a special case of the negative binomial, naturally model the number of trials until the first success and are useful in reliability analysis, survival studies, and modeling waiting times in event processes. %https://www.datacamp.com/tutorial/negative-binomial-distribution These method-

ological and computational extensions would substantially broaden the applicability of BKP across applied statistics, biostatistics, and machine learning.

At last, we welcome contributions from the community and invite developers to participate in the ongoing maintenance and extension of the package by submitting pull requests via GitHub at https://github.com/Jiangyan-Zhao/BKP/pulls.

References

Byrd RH, Lu P, Nocedal J, Zhu C (1995). "A Limited Memory Algorithm for Bound Constrained Optimization." *SIAM Journal on Scientific Computing*, **16**(5), 1190–1208. doi:10.1137/0916069.

Chalup SK, Wiklendt L (2007). "Variations of the two-spiral task." *Connection Science*, **19**(2), 183–199. doi:10.1080/09540090701398017.

Dimitriadis T, Dümbgen L, Henzi A, Puke M, Ziegel J (2022). "Honest calibration assessment for binary outcome predictions." *Biometrika*, **110**(3), 663–680. doi:10.1093/biomet/asac068.

DRatings (2023). "Log Loss vs. Brier Score." Accessed on August 02, 2025, URL https://www.dratings.com/log-loss-vs-brier-score/.

Eddelbuettel D, François R (2011). "Rcpp: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. doi:10.18637/jss.v040.i08.

Fan J, Farmen M, Gijbels I (1998). "Local Maximum Likelihood Estimation and Inference." *Journal of the Royal Statistical Society Series B: Statistical Methodology*, **60**(3), 591–608. doi:10.1111/1467-9868.00142.

Fan J, Gijbels I (1996). *Local Polynomial Modelling and Its Applications*, volume 66 of *Monographs on Statistics and Applied Probability*. 1st edition. Routledge, Boca Raton, Florida. doi:10.1201/9780203748725.

Flores G, Schiff A, Smith AH, Fukuyama JA, Wilson AC (2025). "A Consequentialist Critique of Binary Classification Evaluation Practices." URL https://arxiv.org/abs/2504.04528.

Gardner J, Pleiss G, Weinberger KQ, Bindel D, Wilson AG (2018). "GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration." In S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, R Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/27e8e17134dd7083b050476733207ea1-Paper.pdf.

Garnett R (2023). *Bayesian Optimization*. Cambridge University Press, Cambridge, United Kingdom. doi:10.1017/9781108348973. https://bayesoptbook.com/.

Gneiting T, Raftery AE (2007). "Strictly Proper Scoring Rules, Prediction, and Estimation." *Journal of the American Statistical Association*, **102**(477), 359–378. doi:10.1198/016214506000001437.

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. URL https://dl.acm.org/doi/10.5555/2283516.2283608.

GPy (since 2012). "GPy: A Gaussian process framework in python." http://github.com/SheffieldML/GPy.

Gramacy RB, Taddy M (2010). "Categorical Inputs, Sensitivity Analysis, Optimization and Importance Tempering with tgp Version 2, an R Package for Treed Gaussian Process Models." *Journal of Statistical Software*, **33**(6), 1–48. URL https://www.jstatsoft.org/v33/i06/.

Hastie T, Tibshirani R, Friedman J (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, volume 2. Springer New York, New York, NY. doi:10.1007/978-0-387-84858-7.

Karatzoglou A, Smola A, Hornik K, Zeileis A (2004). "kernlab – An S4 Package for Kernel Methods in R." *Journal of Statistical Software*, **11**(9), 1–20. doi:10.18637/jss.v011.i09.

Leisch F, Dimitriadou E (2024). *mlbench: Machine Learning Benchmark Problems*. R package version 2.1-6, URL https://CRAN.R-project.org/package=mlbench.

Loeppky JL, Sacks J, Welch WJ (2009). "Choosing the Sample Size of a Computer Experiment: A Practical Guide." *Technometrics*, **51**(4), 366–376. doi:10.1198/TECH.2009.08040.

MacDonald B, Ranjan P, Chipman H (2015). "GPfit: An R Package for Fitting a Gaussian Process Model to Deterministic Simulator Outputs." *Journal of Statistical Software*, **64**(12). doi:10.18637/jss.v064.i12.

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.

Martin BD, Witten D, Willis AD (2020). "Modeling microbial abundances and dysbiosis with beta-binomial regression." *The Annals of Applied Statistics*, **14**(1), 94–115. doi:10.1214/19-AOAS1283.

Matthews AGdG, van der Wilk M, Nickson T, Fujii K, Boukouvalas A, León-Villagrá P, Ghahramani Z, Hensman J (2017). "GPflow: A Gaussian process library using TensorFlow." *Journal of Machine Learning Research*, **18**(40), 1–6. URL http://jmlr.org/papers/v18/16-537.html.

Montesano L, Lopes M (2012). "Active learning of visual descriptors for grasping using non-parametric smoothed beta distributions." *Robotics and Autonomous Systems*, **60**(3), 452–462. doi:10.1016/j.robot.2011.07.013.

Murphy KP (2022). *Probabilistic Machine Learning: An introduction*. Adaptive Computation and Machine Learning Series. The MIT Press, Cambridge, Massachusetts. URL http://probml.github.io/book1.

Mussi M, Drago S, Metelli AM (2024). "Open Problem: Tight Bounds for Bernoulli Rewards in Kernelized Multi-Armed Bandits." In *Workshop on Aligning Reinforcement Learning Experimentalists and Theorists at the International Conference on Machine Learning (ICML)*. ArXiv: 2407.06321, URL https://arxiv.org/abs/2407.06321.

Najera-Zuloaga J, Lee DJ, Arostegui I (2018). "Comparison of beta-binomial regression model approaches to analyze health-related quality of life data." *Statistical Methods in Medical Research*, **27**(10), 2989—009. doi:10.1177/0962280217690413.

Najera-Zuloaga J, Lee DJ, Arostegui I (2019). "A beta-binomial mixed-effects model approach for analysing longitudinal discrete and bounded outcomes." *Biometrical Journal*, **61**(3), 600–615. doi:10.1002/bimj.201700251.

Nash JC (2014). "On Best Practice Optimization Methods in R." *Journal of Statistical Software*, **60**(2), 1–14. doi:10.18637/jss.v060.i02.

Nash JC, Varadhan R (2011). "Unifying Optimization Algorithms to Aid Software System Users: optimx for R." *Journal of Statistical Software*, **43**(9), 1–14. doi:10.18637/jss.v043.i09.

Nickisch H, Rasmussen CE (2008). "Approximations for Binary Gaussian Process Classification." *Journal of Machine Learning Research*, **9**(67), 2035–2078. URL http://jmlr.org/papers/v9/nickisch08a.html.

Picheny V, Wagner T, Ginsbourger D (2013). "A Benchmark of Kriging-based Infill Criteria for Noisy Optimization." *Structural and Multidisciplinary Optimization*, **48**(3), 607–626. doi:10.1007/s00158-013-0919-4.

Piironen J (2022). *gplite: General Purpose Gaussian Process Modelling*. doi:10.32614/CRAN.package.gplite. R package version 0.13.0, URL https://CRAN.R-project.org/package=gplite.

R Core Team (2025). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rasmussen CE, Williams CK (2006). *Gaussian Processes for Machine Learning*. the MIT Press, Cambridge, MA. URL https://gaussianprocess.org/gpml/.

Rolland P, Kavis A, Singla A, Cevher V (2019). "Efficient learning of smooth probability functions from Bernoulli tests with guarantees." In K Chaudhuri, R Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 5459–5467. PMLR. URL https://proceedings.mlr.press/v97/rolland19a.html.

Roustant O, Ginsbourger D, Deville Y (2012). "DiceKriging, DiceOptim: Two R Packages for the Analysis of Computer Experiments by Kriging-Based Metamodeling and Optimization." *Journal of Statistical Software*, **51**(1), 1–55. doi:10.18637/jss.v051.i01.

Sung CL, Hung Y, Rittase W, Zhu C, Wu CFJ (2020). "Calibration for Computer Experiments With Binary Responses and Application to Cell Adhesion Study." *Journal of the American Statistical Association*, **115**(532), 1664–1674. doi:10.1080/01621459.2019.1699419.

Vehtari A, Gelman A, Gabry J (2017). "Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC." *Statistics and Computing*, **27**(5), 1413–1432. doi:10.1007/s11222-016-9696-4.

Wen H, Betken A, Hang H (2025). "Optimal Learning of Kernel Logistic Regression for Complex Classification Scenarios." In *The Thirteenth International Conference on Learning Representations*. URL https://openreview.net/forum?id=WlhVRh2rQ0.

Affiliation:

Jiangyan Zhao
School of Statistics
East China Normal University
3663 North Zhongshan Road,
Shanghai 200062, China
E-mail: jyzhao@sfs.ecnu.edu.cn

Kunhai Qing
School of Statistics
East China Normal University
3663 North Zhongshan Road,
Shanghai 200062, China
E-mail: 51254404017@stu.ecnu.edu.cn

Jin Xu
School of Statistics
and
Key Laboratory of Advanced Theory and Application in Statistics and Data Science - MOE
East China Normal University
3663 North Zhongshan Road,
Shanghai 200062, China
E-mail: jxu@stat.ecnu.edu.cn