Report on the Implementation of Probabilistic Predictive Elicitation

Panagiotis Anastasakis

Multi-Source Probabilistic Inference research group
University of Helsinki

July 15, 2024

Contents

1. Introduction

In this report, I will provide a detailed description of my work as a research assistant in the Multi-Source Probabilistic Inference (MUPI) research group at the University of Helsinki. The topic of my work was prior elicitation for probabilistic Bayesian models. Specifically, my goal was to implement methods for helping people specify better prior distributions for statistical models, closely following the theory of the paper 'Flexible Prior Elicitation via the Prior Predictive Distribution' by Hartmann et al. [1]. My tasks included both some algorithmic work that builds on existing research, as well as implementation efforts to make the algorithms easy to use. The github repository with the code can be found here.

1.1 Overview

The prior distribution plays a crucial role in Bayesian inference. It encapsulates information about an unknown parameter $\boldsymbol{\theta}$, which is then integrated with the probability distribution of new data Y, leading to the posterior distribution. Specifying a suitable prior distribution can be challenging, especially when there is lack of prior information about the parameters of the model. One way to obtain informed priors is by deriving prior information from an expert, which can then be turned into a prior distribution. Most methods for eliciting such information involve asking questions regarding the values of the parameters. This approach, however, has the downside that the expert may be unfamiliar with statistical measures used in probabilistic distributions, thus the elicitation process can lead to unreasonable priors [1].

This issue is addressed in [1], where an alternative methodology is presented for capturing expert information and transforming it into a prior. The main idea is that instead of asking questions about the parameters, the modeller can extract probabilistic judgements from an expert about the target Y, which can then be used to obtain the hyperparameters of the prior distribution. Herein, we will refer to this method as Probabilistic Predictive Elicitation (PPE).

1.2 Structure of the report

This report is structured as follows: in section 2, the core concepts of PPE are presented. In section 3, the implementation tasks are discussed. In section 4, results of simulations are presented. Section 5 provides future directions and concludes the report.

2. Probabilistic Predictive Elicitation

2.1 Prior Predictive Distribution

We start off by presenting the concept of *prior predictive distribution*, which is a fundamental block in the theory of PPE. The prior predictive distribution reflects the expected distribution of the target Y prior to observing any data. Let $Y \in \Omega \subseteq \mathbb{R}$ with $Y|\boldsymbol{\theta} \sim \pi_{Y|\boldsymbol{\theta}}$ being the data probability distribution. Let also $\boldsymbol{\theta} \sim \pi_{\boldsymbol{\theta}}$, $\boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^D$, where $\pi_{\boldsymbol{\theta}}$ belongs to a family of distributions that is indexed by a hyperparameter vector $\boldsymbol{\lambda}$. We get the prior predictive distribution by integrating over the parameter space:

$$\pi_Y(y|\boldsymbol{\lambda}) = \int_\Theta \pi_{Y|\boldsymbol{\theta}}(y|\boldsymbol{\theta})\pi_{\boldsymbol{\theta}}(\boldsymbol{\theta}|\boldsymbol{\lambda})d\boldsymbol{\theta} \tag{1}$$

Given any subset $A \in \Omega$ and a fixed $\boldsymbol{\lambda}$, we define the *prior predictive probability* as:

$$\mathbb{P}_{A|\boldsymbol{\lambda}} = \mathbb{P}(Y \in A|\boldsymbol{\lambda}) = \int_A \pi_Y(y|\boldsymbol{\lambda})dy \tag{2}$$

2.2 Mathematical Formulation of PPE

The approach in [1] for performing PPE can be summarized as follows:

1. Define the parametric generative model for Y, consisting of a probabilistic model $Y|\boldsymbol{\theta} \sim \pi_{Y|\boldsymbol{\theta}}$ conditioned on the parameters $\boldsymbol{\theta}$ and a prior distribution $\boldsymbol{\theta} \sim \pi_{\boldsymbol{\theta}}$ for the parameters. The prior distribution depends on hyperparameters $\boldsymbol{\lambda}$, which define the prior and that we are trying to estimate.

2. Partition the data space Ω into exhaustive and mutually exclusive data categories $\{A_1,...,A_n\}$ such that $\cup_{i=1}^n A_i = \Omega$ and $\cap_{i=1}^n A_i = \emptyset$. For each category A_i, ask the expert what they believe is the probability p_i of the data falling in that category.

3. Model the elicited probabilities from Step 2 as a function of the hyperparameters $\boldsymbol{\lambda}$.

4. Perform optimization of the model from Step 3 to obtain an estimate for $\boldsymbol{\lambda}$ describing the expert opinion best within the chosen parametric family of prior distributions.

5. Evaluate how well the predictions obtained from the optimal prior distribution of Step 4 can describe the elicited expert opinion.

Now, assuming that the probabilistic model for Y has been defined, and we have elicited the probabilistic judgements $\boldsymbol{p} = [p_1,...,p_n]$ from the expert about the partitioning $\boldsymbol{A} = \{A_1,...,A_n\}$, our next task is to model these probabilities as a function of $\boldsymbol{\lambda}$. When doing so, it is important to take into account the inherent uncertainty about the expert's opinion. This is accomplished by assuming that $\boldsymbol{p}$ follows a Dirichlet distribution with base measure given by the prior predictive probabilities $\mathbb{P}_{\boldsymbol{A}|\boldsymbol{\lambda}} = [\mathbb{P}_{A_1|\boldsymbol{\lambda}},...,\mathbb{P}_{A_n|\boldsymbol{\lambda}}]$ and precision parameter α. Hence, for any chosen partition $\boldsymbol{A}$ of size n, the density function of $\boldsymbol{p}$ is:

$$D(\boldsymbol{p}|\alpha,\boldsymbol{\lambda}) = \frac{\Gamma(\alpha)}{\prod_{i=1}^{n}\Gamma(\alpha\cdot\mathbb{P}_{A_i|\boldsymbol{\lambda}})}\prod_{i=1}^{n} p_i^{\alpha\cdot\mathbb{P}_{A_i|\boldsymbol{\lambda}}-1} \tag{3}$$

Using the Dirichlet density, we account for the uncertainty around the expert's input $\boldsymbol{p}$. Additionally, this formulation involves the hyperparameter α, which can be interpreted as a measure of how well the prior predictive probability model is able to represent the probability data provided in the elicitation process. The larger the values of α, the less variance around the expected value $\mathbb{P}_{\boldsymbol{A}|\boldsymbol{\lambda}}$. It can be shown ([1]) that the maximum likelihood estimate $\hat{\alpha}$ of α is

$$\hat{\alpha} \approx \frac{\frac{n-1}{2}}{KL(\mathbb{P}_{\boldsymbol{A}|\boldsymbol{\lambda}}||\boldsymbol{p})}. \tag{4}$$

In section 4, we will denote this approximation also using $\alpha|\boldsymbol{\lambda}$, to emphasize that the formula is directly dependent on the hyperparameters.

We now show how PPE can also be applied for models where Y is modelled as a function of covariates $\boldsymbol{x}$. First, PPE requires that the expert provides probabilistic judgments $\boldsymbol{p}^{(1)},...,\boldsymbol{p}^{(J)}$ for Y, corresponding to the covariate sets $\boldsymbol{x}^{(1)},...,\boldsymbol{x}^{(J)}$ respectively. Here, $\boldsymbol{p}^{(j)} = [p_{j,1},...,p_{j,n_j}]$, where n_j is the partitioning size for covariate set j. Naturally, we require that $\sum_{i_j=1}^{n_j} p_{j,i_j} = 1$.

Now, if we assume that $\boldsymbol{p}^{(1)},...,\boldsymbol{p}^{(J)}$ are pairwise conditionally independent, we can model them using the Dirichlet likelihood:

$$D(\boldsymbol{p}^{(1)},...,\boldsymbol{p}^{(J)}|\boldsymbol{\lambda},\alpha) = \prod_{j=1}^{J} D(\boldsymbol{p}^{(j)}|\boldsymbol{\lambda},\alpha) = \prod_{j=1}^{J}\frac{\Gamma(\alpha)}{\prod_{i_j=1}^{n_j}\Gamma(\alpha\cdot\mathbb{P}_{A_{i_j}|\boldsymbol{\lambda},\boldsymbol{x}_j})}\prod_{i_j=1}^{n_j} p_{i_j}^{\alpha\cdot\mathbb{P}_{A_{i_j}|\boldsymbol{\lambda},\boldsymbol{x}_j}-1} \tag{5}$$

where $\mathbb{P}_{A_{i_j}|\boldsymbol{\lambda},\boldsymbol{x}_j}$ is the prior predictive probability for the set A_{i,j_i} related to covariate set $\boldsymbol{x}_j$. Under the presence of covariates, it can be shown ([1]) that the maximum likelihood estimate $\hat{\alpha}$ of α now is:

$$\hat{\alpha} \approx \frac{\sum_{j=1}^{J}\frac{n_j-1}{2}}{\sum_{j=1}^{J} KL(\mathbb{P}_{\boldsymbol{A}^{(j)}|\boldsymbol{\lambda},\boldsymbol{x}^{(j)}}||\boldsymbol{p}^{(j)})}, \tag{6}$$

where $\mathbb{P}_{\boldsymbol{A}^{(j)}|\boldsymbol{\lambda},\boldsymbol{x}^{(j)}} = [\mathbb{P}_{A_{j,1}|\boldsymbol{\lambda},\boldsymbol{x}^{(j)}},...,\mathbb{P}_{A_{j,n_j}|\boldsymbol{\lambda},\boldsymbol{x}^{(j)}}]$.

2.3 Optimization

Having established the modeling of the expert-elicited probabilities, we now turn our attention to optimization with the goal of determining an optimal set of hyperparameters $\boldsymbol{\lambda}$. We explore two approaches for optimizing the Dirichlet likelihood: gradient-based optimization and gradient-free optimization.

Gradient-based Optimization

Our task is to maximize the Dirichlet likelihood, or equivalently, minimize the negative log-likelihood. In gradient-based approaches, we can compute the log-likelihood gradient with respect to $\boldsymbol{\lambda}$, and perform gradient descent.

Let $LD(\boldsymbol{p}|\alpha,\boldsymbol{\lambda})$ be the Dirichlet log-likelihood of $\boldsymbol{p} \in \mathbb{R}^n$ with a base measure $\mathbb{P}_{\boldsymbol{A}|\boldsymbol{\lambda}}$. Let also $\boldsymbol{\lambda} \in \mathbb{R}^m$. Then, we have that $\mathbb{P}_{\boldsymbol{A}|\boldsymbol{\lambda}}$ is a function such that $\mathbb{P}_{\boldsymbol{A}|\boldsymbol{\lambda}} : \mathbb{R}^m \to \mathbb{R}^n$. The Dirichlet log-likelihood is a function from $\mathbb{R}^n \to \mathbb{R}$. Using this notation, we can easily see that the gradient $\nabla_{\boldsymbol{\lambda}} LD(\boldsymbol{p}|\alpha,\boldsymbol{\lambda})$ can be computed using the chain rule by multiplying the Jacobian of $\mathbb{P}_{\boldsymbol{A}|\boldsymbol{\lambda}}$, $J_{\mathbb{P}_{\boldsymbol{A}|\boldsymbol{\lambda}}}$ (matrix of size $m \times n$, taken with respect to $\boldsymbol{\lambda}$) with the gradient $\nabla_{\mathbb{P}_{\boldsymbol{A}|\boldsymbol{\lambda}}} LD(\boldsymbol{p}|\alpha,\boldsymbol{\lambda})$, taken with respect to $\mathbb{P}_{\boldsymbol{A}|\boldsymbol{\lambda}}$. Thus, we have that

$$\nabla_{\boldsymbol{\lambda}} LD(\boldsymbol{p}|\alpha,\boldsymbol{\lambda}) = J_{\mathbb{P}_{\boldsymbol{A}|\boldsymbol{\lambda}}} \cdot \nabla_{\mathbb{P}_{\boldsymbol{A}|\boldsymbol{\lambda}}} LD(\boldsymbol{p}|\alpha,\boldsymbol{\lambda}). \tag{7}$$

The above is for one partition, but if we have multiple set of covariates ($J > 1$), then we get a sum of log-likelihoods and therefore we need to sum the gradients for all $\boldsymbol{p}^{(1)},...,\boldsymbol{p}^{(J)}$:

$$\nabla_{\boldsymbol{\lambda}} LD(\boldsymbol{p}^{(1)},...,\boldsymbol{p}^{(J)}|\alpha,\boldsymbol{\lambda}) = \sum_{j=1}^{J} J_{\mathbb{P}_{\boldsymbol{A}^{(j)}|\boldsymbol{\lambda},\boldsymbol{x}^{(j)}}} \cdot \nabla_{\mathbb{P}_{\boldsymbol{A}^{(j)}|\boldsymbol{\lambda},\boldsymbol{x}^{(j)}}} LD(\boldsymbol{p}^{(j)}|\alpha,\boldsymbol{\lambda}). \tag{8}$$

Now, since we have a formula for the Dirichlet log-likelihood, we can compute $\nabla_{\mathbb{P}_{\boldsymbol{A}|\boldsymbol{\lambda}}} LD(\boldsymbol{p}|\alpha,\boldsymbol{\lambda})$ in closed form. If we also have the prior predictive distribution (1) in closed form, we can take the derivatives of the prior predictive probabilities (2) for the Jacobian $J_{\mathbb{P}_{\boldsymbol{A}|\boldsymbol{\lambda}}}$, which then gives us the gradient of $LD(\cdot)$. If there is no closed form for (1), then one can still proceed with gradient-based optimization, but the gradients will be stochastic and computed using the reparameterization trick and automatic differentiation.

Gradient-free Optimization

In order to optimize $\boldsymbol{\lambda}$, we can also move beyond gradient-based optimization and employ general-purpose global optimization tools. Methods such as Bayesian Optimization [2], only requiring the ability to evaluate the objective function (3) can be a great alternative, especially in cases where computing the gradient is difficult.

Optimizing the concentration parameter α

In PPE, one can choose to either keep α fixed, or to optimize both α and $\boldsymbol{\lambda}$. Given that α is interpreted as a quantification of uncertainty of how well we can represent the expert's probabilistic judgements with the prior predictive probability, keeping it fixed to an appropriate value can be difficult. If one chooses a value that does not correctly capture the model's capabilities, it may negatively affect the optimization of $\boldsymbol{\lambda}$. If we treat α as an additional hyperparameter to estimate, we can either directly optimize the Dirichlet log-likelihood, or we could opt for alternating optimization between α and $\boldsymbol{\lambda}$, which could be simpler as there are approximate closed-form expressions for α.

3. Implementation

Since the theory behind PPE has been well established in [1], we now turn our attention to its implementation. Previous implementation attempts, including those in the original paper [1] and G. Agiashvilli's thesis [3], were limited to specific models. While these efforts have been valuable in demonstrating the utility of PPE, they have not provided a framework that can be easily adapted to various contexts. A generalized approach to PPE would allow practitioners across different fields to utilize this method more effectively, facilitating a broader range of applications.

Prior implementation work was exclusively conducted using the programming language R. However, we have decided to transition to Python to develop a more generalized approach, with the specific aim of integrating this work with PreliZ [4], a Python package that assists practitioners in choosing prior distributions. Additionally, as discussed in Section 2, PPE ultimately reduces to an optimization problem, which can be addressed in multiple ways. Python offers the flexibility to explore a wide array of methods, potentially leading to improved estimates for the prior distribution hyperparameters.

We now provide a general overview of the tasks to be completed for implementing PPE:

1. Write the backend code that supports optimization in the following cases:
   (a) The prior predictive distribution (1) is available in closed form
   (b) The closed form for (1) is difficult/impossible to obtain:
      - Optimization using stochastic gradients
      - Gradient-free optimization using Bayesian Optimization [2]
2. Run experiments to test the method. Additionally, track the optimization results across different configurations, e.g. optimizing α or keeping it fixed, having more covariate sets to elicit probabilities for, having partitionings with a higher/lower number of intervals etc.
3. Integrate the software with PreliZ.

3.1 Backend Code

Most of the code written for PPE is object-oriented to facilitate easier integration with PreliZ. Firstly, we have constructed a class for the Dirichlet distribution. This class supports the computation of the Dirichlet log-likelihood from (3), (5), and allows for calculating the concentration parameter α using the approximate formulas (4) and (6). Additionally, it includes functionality for calculating the gradient $\nabla_{\mathbb{P}_{A|\lambda}} LD(\boldsymbol{p}|\alpha,\boldsymbol{\lambda})$, which is part of the chain rule for computing the gradient of the Dirichlet log likelihood with respect to the hyperparameters, according to (7) and (8). For these gradient calculations and all subsequent ones, we utilized automatic differentiation with the library "Jax" [5].

Next, we added a class called "PPEProbabilities" that contains methods for processing inputs into probabilities in a form compatible with the optimization software

used for PPE. This class can accept a path to a file containing the expert's probabilistic judgments and the partitions, and, assuming a specific structure for these files, it returns the probabilities along with the partitions. Furthermore, "PPEProbabilities" includes a method for computing the prior predictive probabilities from samples of the prior predictive distribution. This is particularly useful because, in practice, one often does not have a closed form of the prior predictive distribution. Instead, having the probabilistic model and the priors of $\boldsymbol{\theta}$ allows for straightforward sampling from the prior predictive distribution using forward sampling, thereby approximating the probabilities (2). We will revisit this when discussing the optimization of arbitrary probabilistic models.

Optimization in closed-form cases

When the prior predictive distribution is available in closed form, we can perform gradient descent to optimize $\boldsymbol{\lambda}$. For this purpose, we have created a class that provides methods for computing the gradient of the objective function with respect to $\boldsymbol{\lambda}$, with the option to optimize α as well.

To optimize α, we express it as a function of $\mathbb{P}_{\boldsymbol{A}|\boldsymbol{\lambda}}$, denoted as $\hat{\alpha}(\mathbb{P}_{\boldsymbol{A}|\boldsymbol{\lambda}})$, using the approximate formulas, which we then substitute this into the objective function. Using the chain rule, we can derive the gradient and perform gradient descent. Note however that in this specific case, the chain rule is not strictly necessary because we can directly express all components of the Dirichlet likelihood with respect to $\boldsymbol{\lambda}$ and compute the gradients directly. The code supports both ways for computing the gradient.

Stochastic Optimization

As mentioned previously, requiring a closed form for the prior predictive distribution of the target is impractical, as even for simpler models it is often difficult or impossible to obtain. In such cases, we can define the probabilistic model of the target Y in Python using PyMC [6]. PyMC is a library that allows users to build Bayesian models with a simple Python API and fit them using Markov chain Monte Carlo (MCMC) methods. One of the main advantages of using PyMC in the context of PPE is its friendly syntax for defining probabilistic models and sampling from the prior predictive distribution via forward sampling. Additionally, since PreliZ is already compatible with PyMC, this ensures consistency in probabilistic syntax, which is beneficial for the integration.

Using PyMC, we can sample from the prior predictive distribution and approximate the model probabilities through methods from the class "PPEProbabilities". However, we cannot compute the gradients analytically as the prior predictive distribution is not available in closed form. Therefore, we need to resort to other approaches for optimizing $\boldsymbol{\lambda}$. If we proceed with gradient-based optimization, the gradients will be stochastic and computed using the reparameterization trick. The mathematical formulation for stochastic gradients is detailed in [1]. Although comprehensive software supporting the computation of stochastic gradients for any probabilistic model has not yet been developed, there are examples of stochastic gradients for simple models that we also have in closed form. These examples demonstrate that this approach can lead to accurate computation of the gradients and thus, efficient optimization of $\boldsymbol{\lambda}$. The code for these examples was written by Bernardo Williams Moreno.

Gradient-Free Learning with Bayesian Optimization

Moving away from methods that require gradient computation for optimization, we now turn to Bayesian Optimization. In the context of PPE, Bayesian Optimization only requires the ability to evaluate the objective function, which involves computing the prior predictive probabilities, as all other quantities are known. Assuming that the probabilistic model structure is available, PyMC can be used to easily obtain these probabilities, making Bayesian Optimization an excellent alternative for dealing with complex models.

To perform Bayesian Optimization, we have created a class named "Bayesian_ Optimization" that optimizes the hyperparameters. It requires as inputs the probabilistic PyMC model, the expert's probabilistic judgments, and a defined search space for the hyperparameters. Internally, this class computes the Dirichlet log-likelihood by sampling from the prior predictive distribution for each set of hyperparameters and calculating the prior predictive probabilities. For the actual optimization process, we use the "optimize" function from the "Ax" library [7] with its default settings. Similar to the gradient-based approaches, the optimization of α is supported.

3.2 Hyperpriors

We now discuss a possible extension of PPE to account for possible prior knowledge about the hyperparameters. In the general case, with PPE we want to obtain $\boldsymbol{\lambda}$ by solving:

$$\hat{\boldsymbol{\lambda}} = \underset{\boldsymbol{\lambda}}{\operatorname{argmin}}\big(-LD(\boldsymbol{p}^{(1)},...,\boldsymbol{p}^{(J)}|\alpha,\boldsymbol{\lambda})\big). \tag{9}$$

Let's now assume that we have more information about the hyperparameters, provided by the expert. Specifically, let's assume that the expert can give some numerical estimates about $\boldsymbol{\lambda}$. In that case, we can use **hyperpriors** to account for such information. These hyperpriors would affect the specification of the loss to encourage values that are in close range to the estimates of the expert.

More formally, suppose that the expert can give estimates $\boldsymbol{\lambda}^{\star} = \{\lambda_1^{\star},...,\lambda_m^{\star}\}$ for the hyperparameters $\boldsymbol{\lambda} = \{\lambda_1,...,\lambda_m\}$. Suppose also that we use a normal distribution as a hyperprior for each λ_k, $k = 1,...,m$, with mean $\lambda_k^{\star}$ and standard deviation σ_k. Then, we can write the optimization problem as:

$$\hat{\boldsymbol{\lambda}} = \underset{\boldsymbol{\lambda}}{\operatorname{argmin}}\big(-LD(\boldsymbol{p}^{(1)},...,\boldsymbol{p}^{(J)}|\alpha,\boldsymbol{\lambda})) - \sum_{k=1}^{m} log(N(\lambda_k|\lambda_k^{\star},\sigma_k))\big). \tag{10}$$

If we also define a search range for the parameter λ_k that takes the form $[a_k,b_k]$, we can instead use the truncated normal distribution with a_k,b_k as bounds. Then, the objective would become

$$\hat{\boldsymbol{\lambda}} = \underset{\boldsymbol{\lambda}}{\operatorname{argmin}}\big(-LD(\boldsymbol{p}^{(1)},...,\boldsymbol{p}^{(J)}|\alpha,\boldsymbol{\lambda})) - \sum_{k=1}^{m} log(TN(\lambda_k|\lambda_k^{\star},\sigma_k,a_k,b_k))\big). \tag{11}$$

Here, we could also choose σ_k to be half the width of the bounds, thus having $\sigma_k = \frac{a_k+b_k}{2}$.

The formulation (11) however, does not fully utilize the additional information that is provided. It could be the case for instance that there are many covariate sets, so the objective function would be mostly determined by the dirichlet likelihood and the hyperparameter estimates would not be properly accounted for. It could also be so that the expert is either very confident about their estimates, or they are uncertain and only provide a general estimate, if any. To account for these scenarios, we can introduce a confidence level $w_k \in [0,1]$, with 0 representing complete uncertainty and 1 corresponding to a very confident estimate. Then, our optimization problem would take the form

$$\hat{\boldsymbol{\lambda}} = \underset{\boldsymbol{\lambda}}{\operatorname{argmin}}\big(-LD(\boldsymbol{p}^{(1)},...,\boldsymbol{p}^{(J)}|\alpha,\boldsymbol{\lambda})) - \sum_{k=1}^{m}(J\cdot w_k)\cdot log(TN(\lambda_k|\lambda_k^{\star},\sigma_k,a_k,b_k))\big). \tag{12}$$

Here, if $w_k = 0$, then we do not consider any hyperprior for λ_k, while when $w_k = 1$, there will be one likelihood component for each covariate set j ($w_k \cdot J = 1 \cdot J = J$), thus the optimization process will take more into account the provided estimate $\lambda_k^{\star}$. Also, note that we can set $w_k = \frac{1}{J}$ and then we would get the same formulation as in (11).

One additional benefit of using hyperpriors is that they can guard against the problem of unidentifiability in optimization. Unidentifiability occurs when the optimization algorithm cannot distinguish the roles of different hyperparameters in the objective function. By providing prior estimates that cover different regions, the distinct roles of the hyperparameters are better accounted for. This helps the algorithm to navigate the hyperparameter space more efficiently, improving the robustness and reliability of the optimization process. In our software, the option for using hyperpriors is supported with Bayesian Optimization.

3.3 Integration with PreliZ

The end goal of this project is to eventually integrate the produced software with PreliZ, making it accessible to a large audience of modellers who use Python. For the integration, it was suggested that the backend code for performing PPE should be built independently of PreliZ. This approach allows for modular development and easier maintenance. The PreliZ interface would then be used for user input and displaying results. This is an initial plan and may be reconsidered upon further discussions between the developers of PPE and PreliZ to ensure the most efficient and user-friendly implementation.

4. Experiments

4.1 Examples with closed form prior predictive distributions

Starting off, to demonstrate the functionality of PPE we use simple distributions for which we can compute the prior predictive distribution in a closed form. For both examples, gradient descent is used to optimize the hyperparameters.

Gaussian Case

For the first example, we assume $Y \sim N(\theta, \sigma)$, with $\theta \sim N(\mu_1, \sigma_1)$. The hyperparameter vector is then $\boldsymbol{\lambda} = [\mu_1, \sigma, \sigma_1]$. Also, for $A = (a, b]$, we can easily show that

$$\mathbb{P}_{A|\boldsymbol{\lambda}} = \Phi\Big((b-\mu_1)/\sqrt{\sigma^2+\sigma_1^2}\Big) - \Phi\Big((a-\mu_1)/\sqrt{\sigma^2+\sigma_1^2}\Big). \tag{13}$$

We partition the data space using three intervals, $\boldsymbol{A} = \{(-\infty,-2],(-2,3],(3,\infty)\}$ and we assign the following probabilities to each, serving as the expert's judgements:

$$\begin{aligned} p_1 &= \mathbb{P}(Y \in (-\infty,-2]) = 0.2 \\ p_2 &= \mathbb{P}(Y \in (-2,3]) = 0.7 \\ p_3 &= \mathbb{P}(Y \in (3,\infty)) = 0.1. \end{aligned}$$

Given these settings and a constant initial value $\alpha = 6$ for the concentration parameter, the gradient descent algorithm converged to the hyperparameter values $\hat{\mu_1} = 0.07748684, \hat{\sigma} = 1.9856238, \hat{\sigma_1} = 1.9856238$. Note here that the estimates for σ and σ_1 are exactly the same. This is a direct consequence of the unidentifibility problem mentioned in the previous section, caused by the fact that σ, σ_1 are interchangeable in the prior predictive probability formula (13). This issue could be addressed by assigning hyperpriors to σ and σ_1.

Given the estimate $\hat{\boldsymbol{\lambda}} = [\hat{\mu_1}, \hat{\sigma}, \hat{\sigma_1}]$, we get the following prior predictive probabilities:

$$\begin{aligned} \mathbb{P}(Y \in (-\infty,-2]|\hat{\boldsymbol{\lambda}}) &= 0.223 \\ \mathbb{P}(Y \in (-2,3]|\hat{\boldsymbol{\lambda}}) &= 0.621 \\ \mathbb{P}(Y \in [3,\infty)|\hat{\boldsymbol{\lambda}}) &= 0.149. \end{aligned}$$

For these probabilities, the value of $\alpha|\hat{\boldsymbol{\lambda}}$ computed from the MLE formula (4) is 58.4.

We inspect further the convergence of the gradient descent algorithm by plotting the values of the objective function and the $l2$-norm of the gradient across iterations:

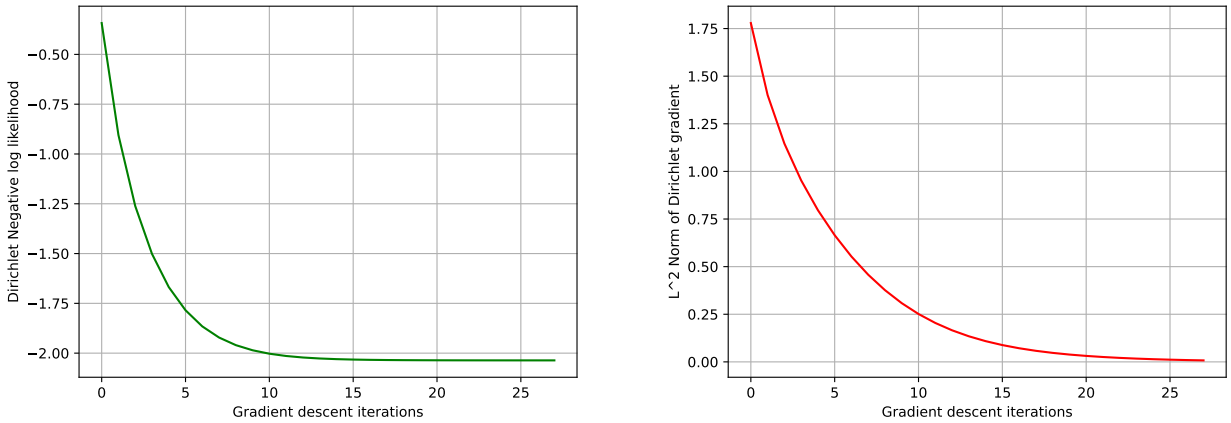

Figure 1: Gradient descent progression for the Gaussian model. In the left figure, the negative Dirichlet log-likelihood is displayed across iterations. The right figure contains the $l2$-norm of the Dirichlet log-likelihood gradient.

Looking at the figures, we can see that gradient descent converges quickly after 25 iterations, with the gradient approaching 0. It is worth noting that for this example, optimizing α along with $\boldsymbol{\lambda}$ did not lead to convergence. One possible justification for this is that the values of α that maximize the Dirichlet likelihood are very different for different sets of prior predictive probabilities, therefore utilizing gradient descent to optimize both can be unstable.

Bernoulli Case

For the second example, we consider a generative model for binary data in the presence of a vector of covariates. The observable variable conditioned on the parameters is distributed according to a Bernoulli model and we take a multivariate Gaussian distribution as the prior distribution for the vector of parameters in the predictor function. This can be formalized as:

$$y|\boldsymbol{\theta} \sim B(\boldsymbol{\Phi}(\boldsymbol{x}^{\mathbf{T}}\boldsymbol{\theta})),$$

where

$$\boldsymbol{\theta} \sim N_D(\mu, \Sigma).$$

This gives us

$$y \sim B(p(\boldsymbol{x}, \boldsymbol{\lambda})),$$

where

$$p(\boldsymbol{x},\boldsymbol{\lambda}) = \Phi\Big(\frac{\boldsymbol{x}^{\mathrm{T}}\boldsymbol{\mu}}{\sqrt{1+\boldsymbol{x}^{\mathrm{T}}\Sigma\boldsymbol{x}}}\Big).$$

Our hyperparameter set is defined as $\boldsymbol{\lambda} = [\mu, \Sigma]$. Since we have binary classification, there is only one partition, $\{\{0\},\{1\}\}$. If we define $A_1 = \{0\}$ and $A_2 = \{1\}$ we get:

$$\mathbb{P}_{A_1|\boldsymbol{\lambda}} = 1 - p(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{\lambda}) \text{ and } \mathbb{P}_{A_2|\boldsymbol{\lambda}} = p(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{\lambda}).$$

For this example we assume $D = 5$ and that the variables are independent with one another, meaning that Σ is a diagonal matrix. Thus, we have 10 hyperparameters to optimize.

We consider 3 covariate sets ($J = 3$) that we have an expert input for as shown in Table 1:

Table 1: Expert probabilities for the Bernoulli model

Covariate set	$P(Y=0)$	$P(Y=1)$
$\boldsymbol{x}^{(1)}$ = [1.3, 0.7, 0.5, -0.7, -0.5]	0.35	0.65
$\boldsymbol{x}^{(2)}$ = [1, 0.5, 0.4, -0.8, 0]	0.3	0.7
$\boldsymbol{x}^{(3)}$ = [0.3, 0.7, 2, -2, 0.2]	0.2	0.8

When keeping α constant at $\alpha = 5$, the algorithm converges to the hyperparameters $\hat{\boldsymbol{\lambda}} = [\hat{\boldsymbol{\mu}}, \mathrm{diag}(\hat{\boldsymbol{\Sigma}})]$, where

$$\hat{\boldsymbol{\mu}} = [0.76, 1.67, 2.58, 1.86, 2.31]$$
$$\mathrm{diag}(\hat{\boldsymbol{\Sigma}}) = [2.94, 2.28, 2.4, 2.68, 2.07].$$

The prior predictive probabilities corresponding to these hyperparameters are in Table 2:

Table 2: Prior predictive probabilities given the optimized hyperparameters $\hat{\boldsymbol{\lambda}}$ for the Bernoulli model

Covariate set	$P(Y=0\|\hat{\boldsymbol{\lambda}})$	$P(Y=1\|\hat{\boldsymbol{\lambda}})$
$\boldsymbol{x}^{(1)}$	0.374	0.626
$\boldsymbol{x}^{(2)}$	0.329	0.671
$\boldsymbol{x}^{(3)}$	0.245	0.755

For those probabilities, the value of $\alpha|\hat{\boldsymbol{\lambda}}$ is 164.5.

Same as before, an attempt to optimize both $\boldsymbol{\lambda}$ and α was made, but again there was no convergence. When adding one more covariate set however, meaning that $J = 4$, with a very small step size (0.002 compared to 0.5 for fixed α), gradient descent did converge and the resulting $\alpha|\hat{\boldsymbol{\lambda}}$ is 88.4.

4.2 Simulations with Bayesian Optimization

The two examples from section 4.1 are indicative of how the performance of PPE can vary depending on the problem settings. We are interested in exploring further the behavior of PPE as we configure these settings and for that purpose, we conduct simulations with more complicated probabilistic structures, where we utilize Bayesian Optimization to obtain hyperparameter estimates. We want to address the following questions:

- How efficient is Bayesian Optimization in the context of PPE as probabilistic models become increasingly complicated?
- How does PPE perform for different number of covariate sets that expert probabilities are elicited for?
- Does how we partition the data affect the quality of the results? More specifically, is a more detailed partitioning preferable?

For the simulations, we consider three families of models: first is the Gaussian family, where we assume that

$$Y \sim N(\mu, \sigma^2),$$

with different possible priors used to define μ and σ.

The second probabilistic model family is that of Bayesian Linear Regression, where again it is assumed that Y is drawn from a Gaussian distribution, but now the mean is estimated as a linear combination of a set of covariates $\boldsymbol{x} = \{x_1, ..., x_n\}$:

$$Y \sim N(b_0 + \prod_{i=1}^{n} b_i \cdot x_i, \sigma^2),$$

where $b_i \sim N(\mu_i, \sigma_i)$.

The third family is that of Logistic Regression, where Y is now binary and a function of covariates $\boldsymbol{x} = \{x_1, ..., x_n\}$. In probabilistic notation, we have:

$$Y \sim B(p(\boldsymbol{x}, \boldsymbol{\theta})),$$

where $p(\boldsymbol{x}, \boldsymbol{\theta}) = \frac{e^{\boldsymbol{x}^T \boldsymbol{\theta}}}{1+e^{\boldsymbol{x}^T \boldsymbol{\theta}}}$, with $\theta_i \sim N(\mu_i, \sigma_i^2)$, $i = 0, ..., n$, with θ_0 being the intercept.

For the Gaussian family and for Bayesian Linear Regression, we consider increasingly detailed partitionings that consist of 2, 5, 10 and 20 intervals. For instance, the partitioning $\{(-\infty, 0], (0, \infty)\}$ of $\mathbb{R}$ consists of two intervals, while the partitioning $\{(-\infty, -10], (-10, 0], (0, 10], (10, \infty)\}$ consists of four. In order to obtain the partitions,

we choose an area $[a,b]$ wide enough to contain Y with high probability. Then, we partition this area, resulting in partitions of the form $\{(-\infty,a],(a,p_1],(p_1,p_2],...,(p_k,b],(b,\infty)\}$. We refer to this area as "inner partitioning" area. When the partition consists of only two intervals, we define it as $\{(-\infty,\frac{a+b}{2}],(\frac{a+b}{2},\infty)\}$. For Logistic Regression, the target is binary therefore the partitioning can only be $\{\{0\},\{1\}\}$.

For Bayesian Linear Regression and Logistic Regression models, we also consider different number J of covariate sets that we elicit probabilities for. Specifically, we perform simulations with 2, 5 and 10 covariate sets.

Now, regarding the choice of α, we have observed that it can have a significant effect in the optimization process. If we decide to keep it fixed, selecting a value that does not represent the model's true capabilities (e.g. too small α, when the model has the capacity of accurately capturing the expert's beliefs) may lead to worse performance. Furthermore, as we mentioned, optimizing α along with $\boldsymbol{\lambda}$ can make the optimization process unstable. To account for these issues, we proceed as follows: for each model, we optimize $\boldsymbol{\lambda}$ for partitions consisting of 2, 5, 10 and 20 intervals, corresponding to 4 optimization procedures. Starting from the partition with 2 intervals, we assume no knowledge of α, therefore we optimize it with the hyperparameters. Using the resulting hyperparameters $\hat{\boldsymbol{\lambda}}$, we compute $\hat{\alpha}|\hat{\boldsymbol{\lambda}}$ from the MLE formula, which we then use as input for optimizing using 5-interval partitioning (therefore α is not optimized now). We then compute again the resulting $\hat{\hat{\alpha}}|\hat{\boldsymbol{\lambda}}$, which we use as input for 10 intervals and we do the same for partitions with 20 intervals. In Logistic Regression where there is only one partitioning, we optimize α for each model.

The reasoning behind this approach is that each previous α estimate is assumed to be an approximation of the model's capacity to match the expert's input, which we then use for the next optimization with more partitions. That way, we do not have to optimize alpha for 5, 10 and 20 intervals and also for each we get a fixed value that represents how well PPE performs with fewer intervals in the partition.

Finally, we simulate the expert's probabilistic judgements by assuming a true set of hyperparameters $\boldsymbol{\lambda}_{\text{true}}$, according to which we draw 20.000 samples from the probabilistic model. Then, for each partition interval A_i, we use these samples to compute the probabilities $\mathbb{P}(Y \in A_i|\boldsymbol{\lambda}_{\text{true}})$ that we use as the expert's input.

Since we use Bayesian Optimization, it is necessary to select bounds for the hyperparameters. For quantities that can only take positive values, we choose the bounds $(0,10]$, whereas for quantities that can take negative values as well we choose $[-10,10]$. Whenever α is optimized, we assign the search space $(0,70]$. Each optimization loop is run for 75 iterations, while for computing the model probabilities at each iteration for a given $\boldsymbol{\lambda}$ we use 1.500 samples. Using more samples can increase the accuracy with which the prior predictive probabilities are calculated, but it would also take more time to run. In the context of PPE, a more efficient algorithm in terms of computational complexity while retaining as much accuracy as possible is of course preferred.

Gaussian Family

For the Gaussian family, we consider two different probabilistic structures:

- Model 1 is $Y \sim N(\mu, \sigma^2)$, where $\mu \sim N(\mu_1, \sigma_1)$ and $\sigma \sim \text{Gamma}(a,b)$. The hyperparameter vector is $\boldsymbol{\lambda} = [\mu_1, \sigma_1, a, b]$. For the simulation, we assume that $\mu_1 = 5, \sigma_1 = 2, a = 2$ and $b = 3$.
- For Model 2, we again have $Y \sim N(\mu, \sigma^2)$, where $\mu \sim N(\mu_1, \sigma_1)$ and $\sigma \sim \text{Gamma}(a,b)$. The difference is that now we also define priors for the parameters of μ's prior, specifically $\mu_1 \sim N(\mu_m, \sigma_m)$ and $\sigma_1 \sim LN(\mu_s, \sigma_s)$, where LN is the log-gaussian distribution. This implies the hyperparameter vector $\boldsymbol{\lambda} = [\mu_m, \sigma_m, \mu_s, \sigma_s, a, b]$. For the simulation, we assume that $\mu_m = 5, \sigma_m = 1, \mu_s = 0.4, \sigma_m = 4, a = 2$ and $b = 3$.

For both models, the inner partitioning area chosen is $[a,b] = [-15, 25]$. To evaluate the models' performance across different partitions, we report the Dirichlet log-likelihood for the optimized hyperparameters $\hat{\boldsymbol{\lambda}}$, the concentration parameter $\alpha|\hat{\boldsymbol{\lambda}}$ and the Wasserstein distance W between the simulated expert's probabilistic judgements and the resulting prior predictive probabilities, denoted as $\boldsymbol{p}$ and $\mathbb{P}_{\boldsymbol{A}|\hat{\boldsymbol{\lambda}}}$ respectively. The Wasserstein distance between two distributions is interpreted as the amount of effort it would take to transform the one distribution onto the other by moving it and reshaping it. Here, it can be viewed as an additional evaluation metric apart from α, quantifying the distance between the optimized prior predictive probabilities and the true ones. The computation was done using the function "wasserstein_distance" from "scipy.stats" [8].

The results of PPE are captured in the following tables:

Table 3: Results from PPE for Model 1.

Number of intervals	Dirichlet log-likelihood	$\alpha\|\hat{\boldsymbol{\lambda}}$	$W(\mathbb{P}_{\boldsymbol{A}\|\hat{\boldsymbol{\lambda}}}, \boldsymbol{p})$
2	1.9	899.7	0.0112
5	18.2	1508.3	0.0003
10	55	940.1	0.0019
20	111.2	3364.9	0.0019

Table 4: Results from PPE for Model 2.

Number of intervals	Dirichlet log-likelihood	$\alpha\|\hat{\boldsymbol{\lambda}}$	$W(\mathbb{P}_{\boldsymbol{A}\|\hat{\boldsymbol{\lambda}}}, \boldsymbol{p})$
2	1.8	62632.6	0.0013
5	-443.9	527.9	0.0075
10	24.8	348.2	0.0075
20	76.5	752.9	0.0046

For both Model 1 and 2, we observe that α is very high, while the Wasserstein distance between the expert probabilities and the prior predictive probabilities remains small. This suggests that PPE has managed to accurately capture the expert input in both cases. The differences across the values of α may be considerably different as the number of intervals changes, but since they are so high we can conclude that the method has been successful in all cases. To better capture the effect of different partitions, we plot the distribution of Y given the resulting hyperparameters $\hat{\boldsymbol{\lambda}}$ for each partitioning, which we compare with the true distribution:

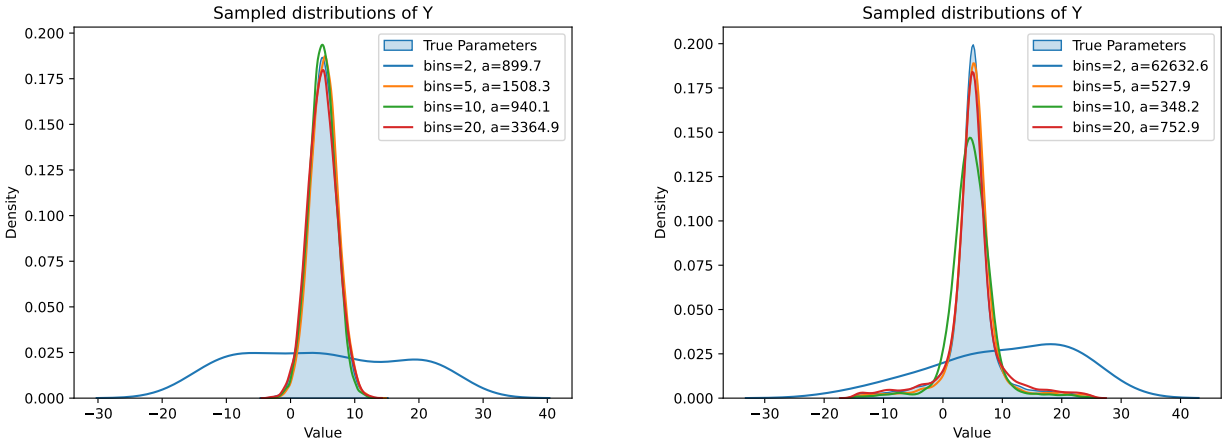

Figure 2: Densities for different number of bins (intervals) for Models 1 and 2.

Here, we can see that as the number of intervals increases, we are able to better capture the true distribution. For the 2-interval partitioning, the resulting density is considerably wider, while as we obtain additional information through increasing number of intervals, we quickly approach the true density of Y.

Bayesian Linear Regression

For the Bayesian Linear Regression model, $Y \sim N(b_0 + \prod_{i=1}^{n} b_i \cdot x_i, \sigma^2)$, $b_i \sim N(\mu_i, \sigma_i)$ with hyperparameters $\boldsymbol{\lambda} = [\sigma, \mu_0, \sigma_0, \mu_1, \sigma_1, ..., \mu_n, \sigma_n]$, we consider three values for n, specifically $n \in 1, 2, 4$, corresponding to three different probabilistic models for Y. The first (Model 1) has 5 hyperparameters, the second (Model 2) has 7 and the third (Model 3) has 11. To test each of the three models, we again have different partitions with 2, 5, 10 and 20 intervals, while we also test how PPE performs with different number of covariate sets $J = 2, 5, 10$ that we simulate the expert's opinion on. For all models, we used the inner partitioning area $[a, b] = [-70, 70]$.

The covariate sets that were used to simulate the expert's probabilistic judgements and the resulting tables are reported in appendices A and B respectively. Here, we plot the densities for each covariate set and each model:

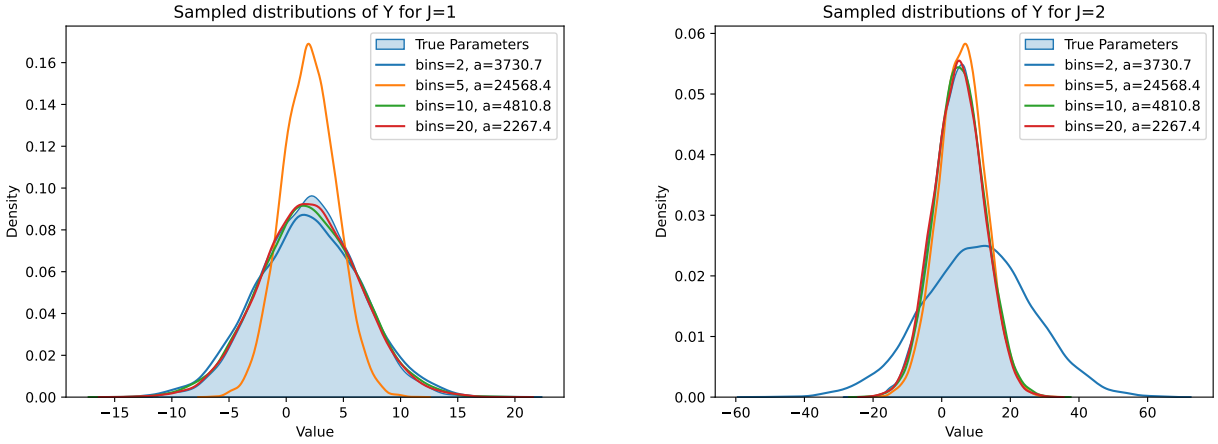

Figure 3: Results of PPE for Bayesian Linear Regression, Model 1 and $J = 2$. Each figure corresponds to one covariate set $j \in \{1, 2\}$.

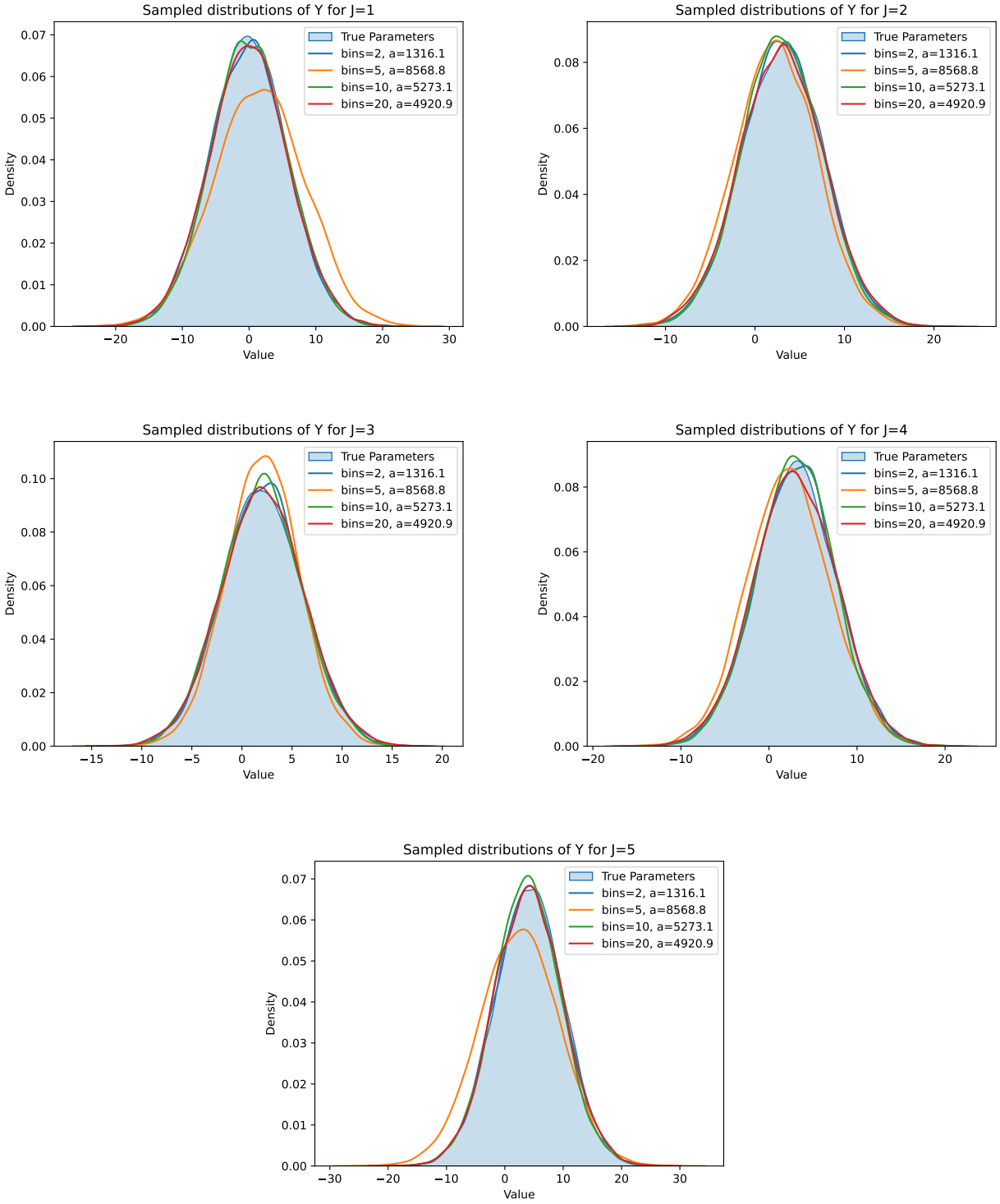

Figure 4: Results of PPE for Bayesian Linear Regression, Model 1 and $J = 5$. Each figure corresponds to one covariate set $j \in \{1, ..., 5\}$.

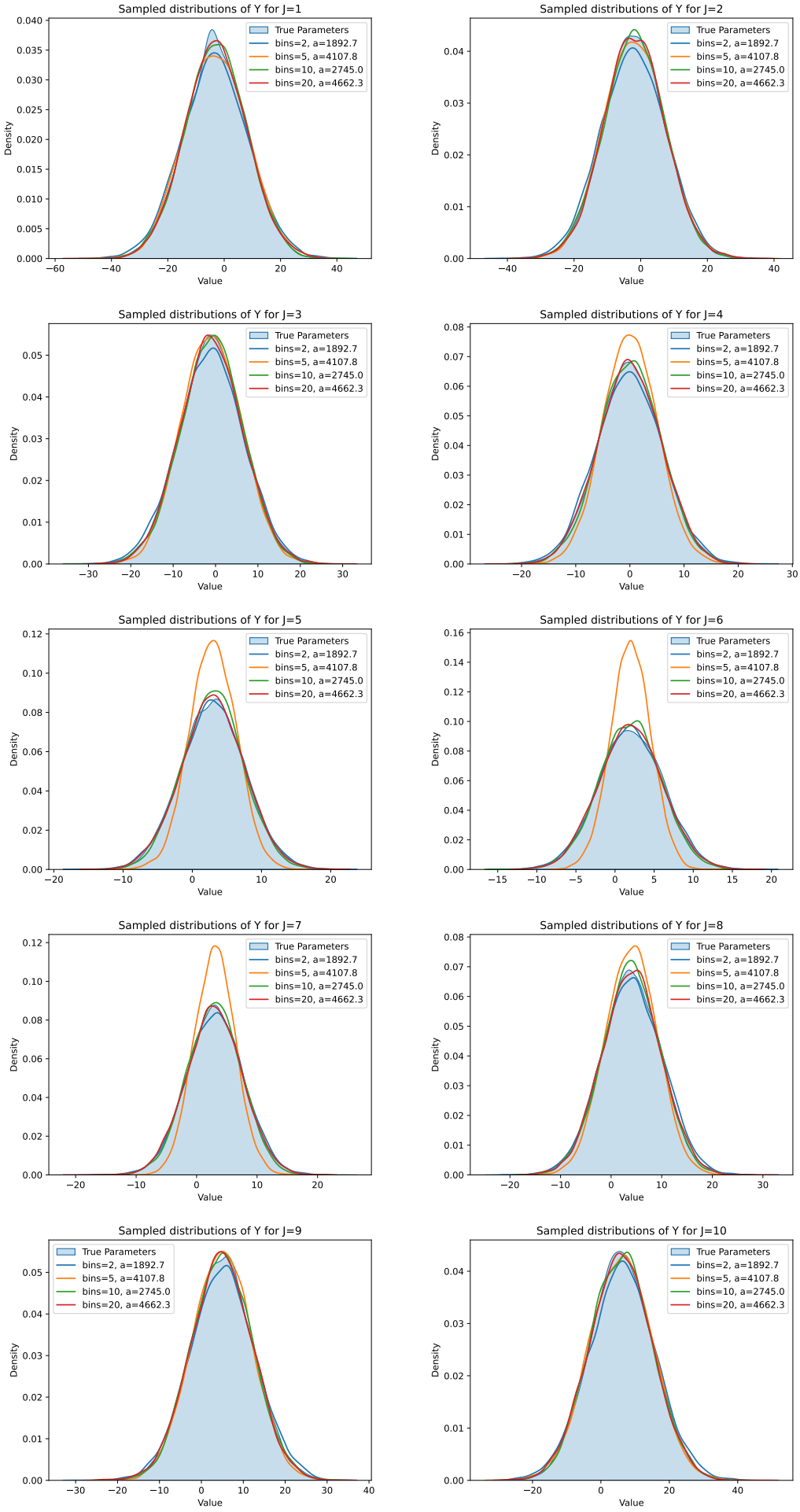

Figure 5: Results of PPE for Bayesian Linear Regression, Model 1 and $J = 10$. Each figure corresponds to one covariate set $j \in \{1, ..., 10\}$.

Looking at the figures for Model 1, we observe that for all J's we have very high values of α, while we can accurately capture the true distribution, especially when the number of intervals in the partitions increases.

We now plot the results for Model 2, where there are more hyperparameters to be optimized compared to the first model:

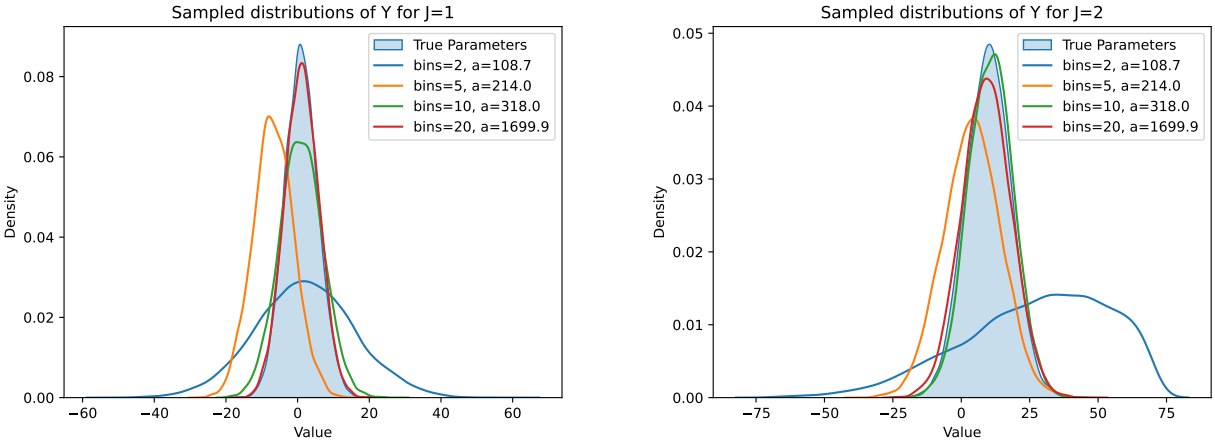

Figure 6: Results of PPE for Bayesian Linear Regression, Model 2 and $J = 2$.

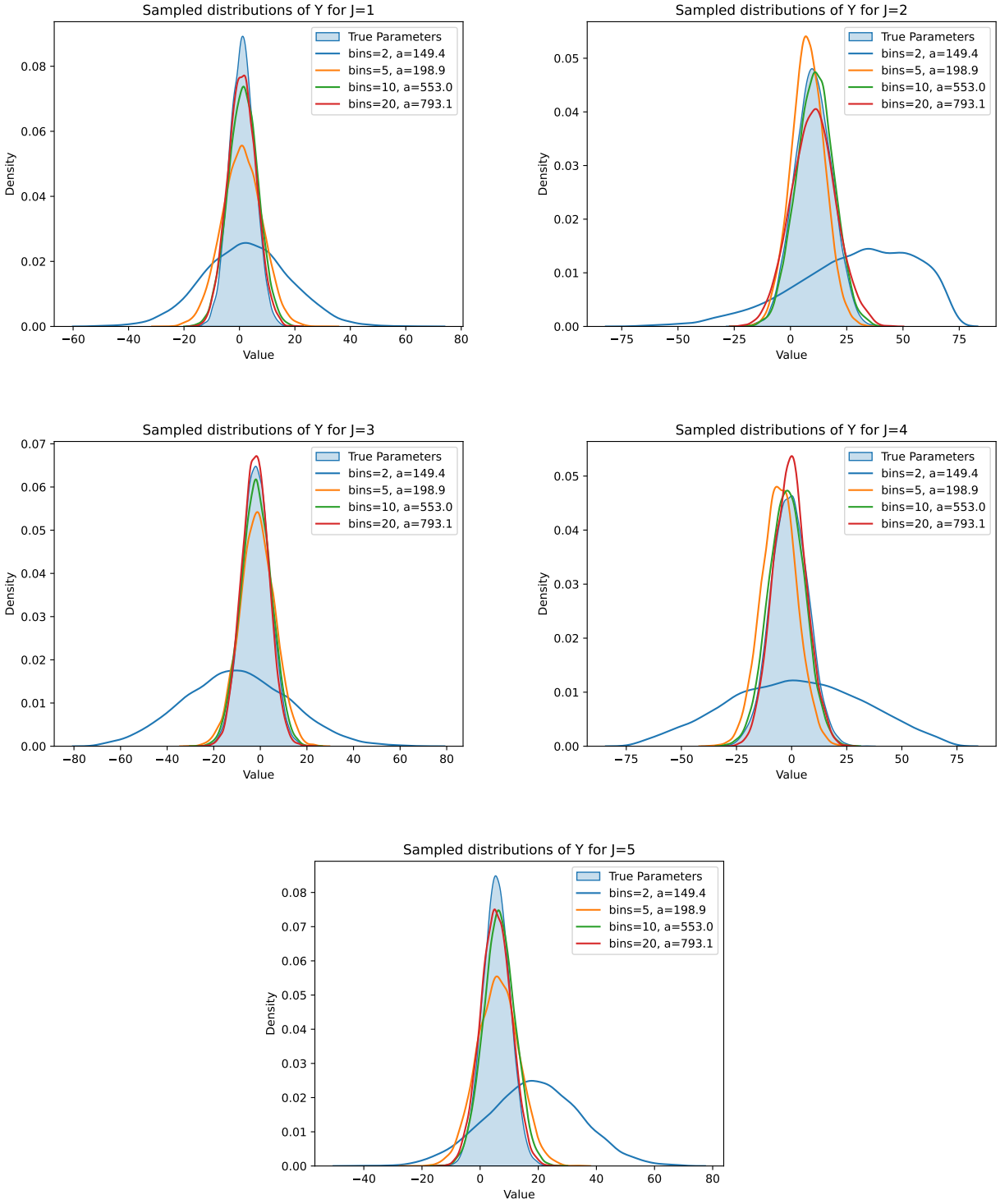

Figure 7: Results of PPE for Bayesian Linear Regression, Model 2 and $J = 5$.

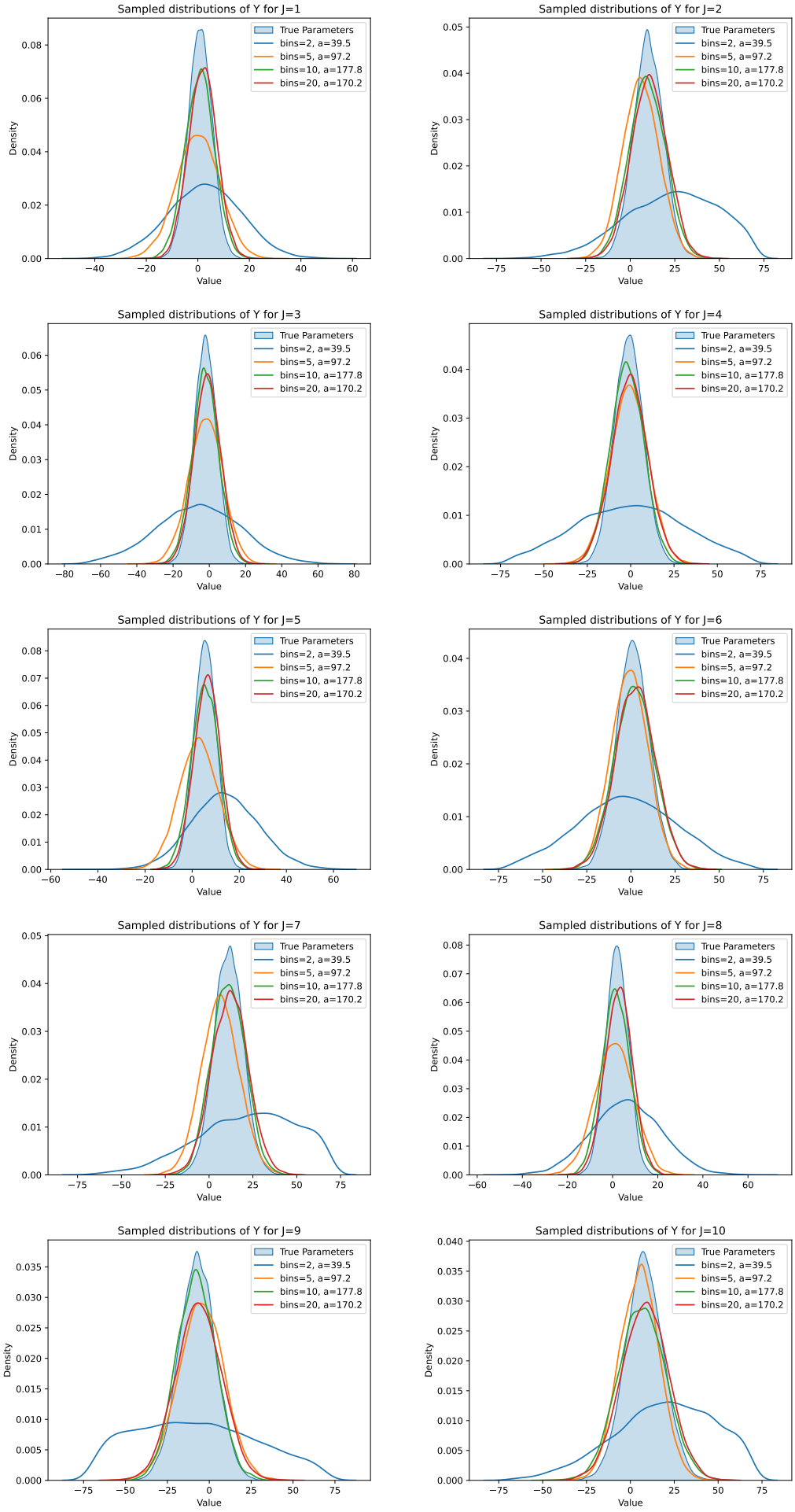

Figure 8: Results of PPE for Bayesian Linear Regression, Model 2 and $J = 10$.

For Model 2, we can see that the values of α, although relatively high, they are considerably smaller than those for Model 1. Furthermore, the density plots suggest that true distribution is not captured as efficiently, especially when we only have 2 intervals in the partitioning. Nonetheless, for 20 intervals, the density curve follows the true one more closely in general and the α's for this more detailed partitioning are higher for $J = 2$, $J = 5$ and almost as high as the one of 10-interval partitioning for $J = 10$.

Regarding the effect of increasing J, we observe that α decreases when more covariate sets are included in the optimization process. For $J = 2$ and $J = 5$, the values remain high, but for $J = 10$ they drop for every partitioning.

Prior to the simulations, a performance deterioration with more complex models was anticipated. The reason is that Bayesian Optimization becomes less efficient as we consider more hyperparameters as it is more difficult to explore all the parameter space. We continue to investigate the effect of increasingly large models by plotting the results for Model 3, which has 11 hyperparameters:

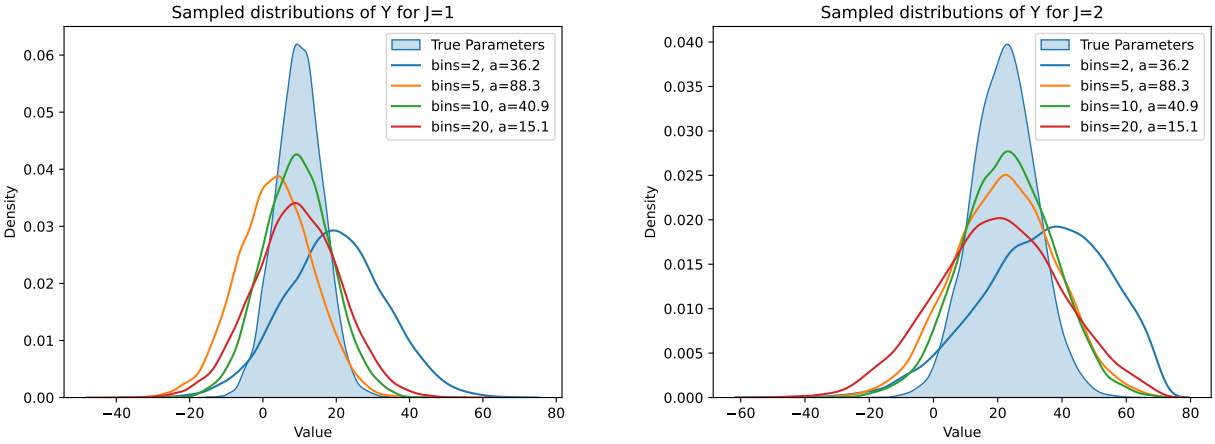

Figure 9: Results of PPE for Bayesian Linear Regression, Model 3 and $J = 2$.

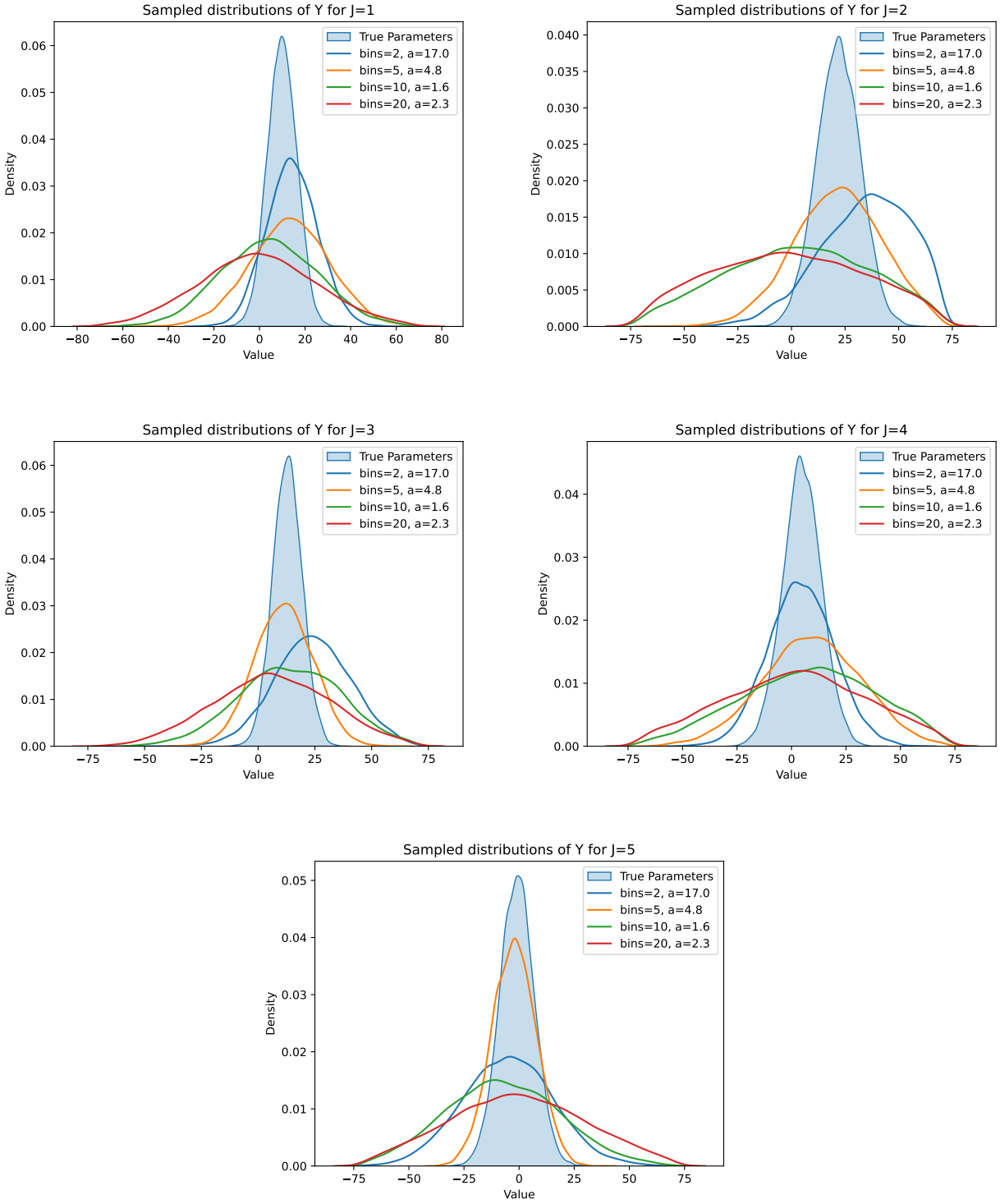

Figure 10: Results of PPE for Bayesian Linear Regression, Model 3 and $J = 5$.

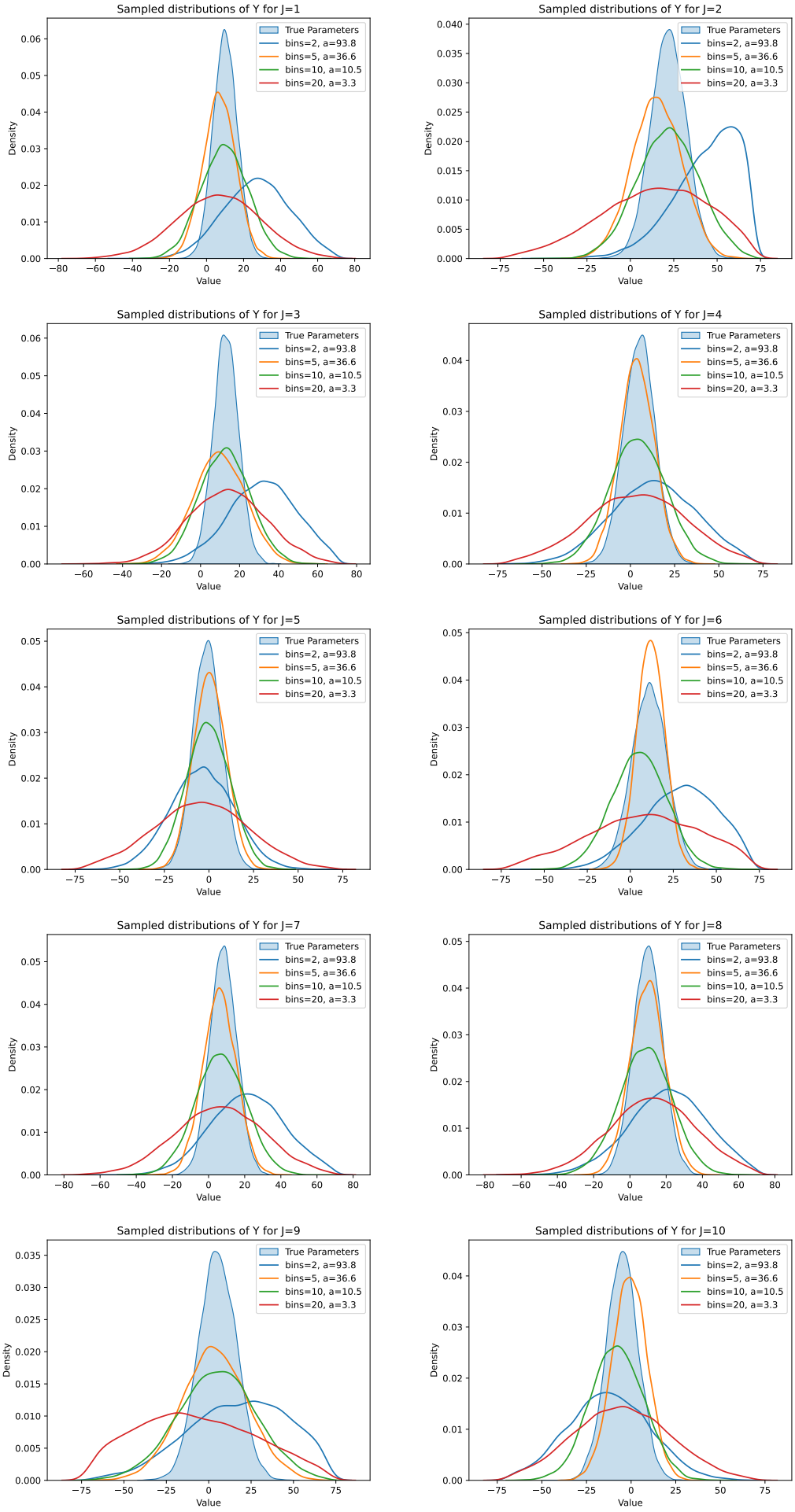

Figure 11: Results of PPE for Bayesian Linear Regression, Model 3 and $J = 10$.

Based on the figures, we observe that PPE becomes less efficient with Model 3, which has 11 hyperparameters. The values of α become smaller as more covariate sets are included in the elicitation process, while having more detailed partitioning in terms of number of intervals does not lead to better results. In fact, both 10 and 20 partitions lead to worse performances, as α's are smaller and the predicted densities are more far off from the true one.

Logistic Regression

For Logistic Regression models, $Y \sim N(p(\boldsymbol{x},\boldsymbol{\theta})), p(\boldsymbol{x},\boldsymbol{\theta}) = \frac{e^{\boldsymbol{x}^T\boldsymbol{\theta}}}{1+e^{\boldsymbol{x}^T\boldsymbol{\theta}}}$, with $\theta_i \sim N(\mu_i, \sigma_i^2)$, $i = 0,...,n$, we consider the same three values for n, which are $n \in 1,2,4$, corresponding to three different probabilistic models for Y. The first (Model 1) has 4 hyperparameters, the second (Model 2) has 6 and the third (Model 3) has 10. Same as with Bayesian Linear Regression, we test each model with $J = 2,5$ and 10 covariate sets. For each of these, we use the same covariate sets as in Bayesian Linear Regression. Given that in the previous simulations we consistently obtained very high values for α, we now increase the upper bound for α from 70 to 2000 thus the search space now is $(0,2000]$.

For each model, we report the Dirichlet log-likelihood, the concentration parameter $\alpha|\hat{\boldsymbol{\lambda}}$ and the Wasserstein distance between the expert probabilities $\boldsymbol{p}$ and the prior predictive probabilities $\mathbb{P}_{\boldsymbol{A}|\hat{\boldsymbol{\lambda}}}$ for the optimized $\hat{\boldsymbol{\lambda}}$. Since there are multiple covariate sets, the distribution of the probabilities is now multivariate, therefore we use the appropriate function "wasserstein_distance_nd" from "scipy.stats" [8].

The results are captured in the following tables:

Table 5: Results for Logistic Regression Model 1 for different number of covariate sets J.

Covariate Sets J	Dirichlet log-likelihood	$\alpha\|\hat{\boldsymbol{\lambda}}$	$W(\mathbb{P}_{\boldsymbol{A}\|\hat{\boldsymbol{\lambda}}}, \boldsymbol{p})$
2	5.7	1652.3	0.0082
5	6.7	65.9	0.0379
10	41.6	172.7	0.0388

Table 6: Results for Logistic Regression Model 2 for different number of covariate sets J.

Covariate Sets J	Dirichlet log-likelihood	$\alpha\|\hat{\boldsymbol{\lambda}}$	$W(\mathbb{P}_{\boldsymbol{A}\|\hat{\boldsymbol{\lambda}}}, \boldsymbol{p})$
2	5.5	495.8	0.0101
5	8.7	703.3	0.0149
10	16.9	104.4	0.0485

Table 7: Results for Logistic Regression Model 3 for different number of covariate sets J.

Covariate Sets J	Dirichlet log-likelihood	$\alpha\|\hat{\boldsymbol{\lambda}}$	$W(\mathbb{P}_{\boldsymbol{A}\|\hat{\boldsymbol{\lambda}}}, \boldsymbol{p})$
2	4.8	30.1	0.0436
5	9.7	38.8	0.0359
10	13	20.8	0.0778

First of all, we observe that the smaller Models 1,2 have higher values for α compared to Model 3, while the Wasserstein distances tend to be smaller. This suggests a trend similar to that of Bayesian Linear Regression, where the efficiency of PPE with Bayesian Optimization was decreasing as the models became more complicated.

In general, however, the results of Logistic Regression seem to not be as stable. For example, in Model 1, we can see very large differences for α between $J = 2$ and $J = 5$ or 10. Also, we are not able to identify a clear pattern regarding the effect of increasing J. In fact, different runs of the Logistic Regression simulation resulted in significantly dissimilar and inconsistent outcomes. One possible justification about this is the fact that α is optimized here along with $\boldsymbol{\lambda}$, contrary to the other two simulations. As we mentioned, this may lead to the optimization being unstable.

5. Conclusions and Future Directions

In this report, we discussed Probabilistic Predictive Elicitation (PPE), a method introduced in [1] that enables modelers to select more appropriate prior distributions by quantifying probabilistic assessments from experts regarding the target variable. Since the theory behind PPE is well-established, our focus was on its implementation. We detailed various optimization approaches for obtaining an optimal set of hyperparameters, including methods that utilize gradient information and more general optimization tools, specifically Bayesian Optimization. Through experiments and simulations, we demonstrated PPE's functionality across different models and setups, particularly emphasizing its application with Bayesian Optimization for more complex models.

Our findings suggest that PPE, when combined with Bayesian Optimization, is capable of providing accurate estimates of the prior's hyperparameters. However, as the dimensionality increases to 10 hyperparameters or more, the method's efficiency is limited under the current experimental setup. It is possible that with additional optimization trials and other modifications to the Bayesian Optimization process, better performance could be achieved.

Regarding the effect of eliciting probabilities for more covariate sets, we found that PPE effectively captures these probabilities in smaller models. However, as more covariate sets are included and the model complexity increases, the performance deteriorates. Additionally, we confirmed that more detailed partitioning is beneficial when performing PPE. In other words, asking the expert to provide as much information as possible in the form of probabilistic judgments for more intervals where the target could lie can lead to a more accurate prior.

Another topic we addressed is the selection of the concentration parameter α. Choosing an appropriate value for α is challenging and can significantly impact the quality of inferences made with PPE. An inappropriate value that does not accurately reflect the model's ability to match the expert's beliefs may lead to poor prior estimates, so fixing it to a random value is not advisable. Instead, one could optimize it alongside the other hyperparameters $\boldsymbol{\lambda}$. The downside of this approach is that it can render the optimization process unstable. In our simulations, we addressed this issue by updating α based on previous estimates for the same model and partitionings with fewer intervals.

Limitations and future Directions

In section 4.2, we reported results from simulations where the expert probabilities were derived from the true distribution we aimed to estimate. However, in real-life situations, the expert's input may not be fully captured by a probabilistic model. For example, a modeler might assume a Gaussian distribution for the target, while the expert's probabilistic judgments might be better represented by a multimodal distribution. Thus, it is crucial to explore the behavior of PPE with Bayesian Optimization in such non-idealized scenarios.

Our simulations revealed significant variability in the results of the Logistic Regression model. While the Gaussian and Bayesian Linear Regression models also exhibited some variability, the key findings, such as the impact of more detailed partitioning and the efficiency of PPE with more complex models, remained consistent. This inconsis-

tency likely arises from the increased stochasticity in estimating both expert and prior predictive probabilities, as well as the use of Bayesian Optimization. Future studies of PPE could explore ways to guarantee more stable and reliable results.

One idea for PPE is to make it interactive, allowing the modeller to input probabilistic judgments and obtain the resulting probabilistic distribution in real-time. This would enable modifications to the input to explore how the output changes, helping select an optimal fit for their analysis. For an interactive environment, PPE must run quickly in the background to minimize waiting time. However, Bayesian Optimization can be time-consuming; in our simulations, each optimization took approximately 60 to 90 seconds per model. If an interactive platform is the ultimate goal for PPE, strategies to reduce running time should be explored.

Finally, although we highlighted the challenge of selecting the concentration parameter α, we did not find a definitive solution for choosing or optimizing it to ensure consistent and efficient hyperparameter estimates. This topic should be looked into as it is central to the implementation of PPE.

Bibliography

[1] Hartmann, M., Agiashvili, G., Bürkner, P., & Klami, A. (2020). Flexible prior elicitation via the prior predictive distribution. In Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence, UAI 2020 (pp. 1129–1138). Association For Uncertainty in Artificial Intelligence (AUAI).

[2] Boender, C. G. E., & Mockus, J. (1991). Bayesian Approach to Global Optimization–Theory and Applications. Mathematics of Computation, 56(194), 878. https://doi.org/10.2307/2008419

[3] Agiashvili, G. (2021). Probabilistic Predictive Elicitation [Master's Thesis, University of Helsinki]. https://helda.helsinki.fi/items/ff4c6c54-25a4-4c66-981b-07e7a979a689

[4] Icazatti, A., Abril-Pla, O., Klami, A., & Martin, O. A. (2023). PreliZ: A tool-box for prior elicitation. Journal of Open Source Software, 8(89), 5499. https://doi.org/10.21105/joss.05499

[5] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne and Qiao Zhang (2018). JAX: composable transformations of Python+NumPy programs, version 0.3.13, http://github.com/google/jax

[6] Abril-Pla O, Andreani V, Carroll C, Dong L, Fonnesbeck CJ, Kochurov M, Kumar R, Lao J, Luhmann CC, Martin OA, Osthege M, Vieira R, Wiecki T, Zinkov R. 2023. PyMC: a modern, and comprehensive probabilistic programming framework in Python. PeerJ Computer Science 9:e1516 https://doi.org/10.7717/peerj-cs.1516

[7] Bakshy, E., Dworkin, L., Karrer, B., Kashin, K., Letham, B., Murthy, A., & Facebook, S. S. (2018). AE: A domain-agnostic platform for adaptive experimentation. In Conference on Neural Information Processing Systems (pp. 1–8).

[8] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne

M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. (2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17(3), 261-272.

Appendix A. Covariate sets used in simulations

For both Bayesian Linear Regression and Logistic Regression, we used the same covariate sets to simulate the expert's probabilistic judgements. For $n = 1$, we used the following:

- $J = 2 \rightarrow x^{(1)} = 0, x^{(2)} = 3.$
- $J = 5 \rightarrow x^{(1)} = -2, x^{(2)} = -1, x^{(3)} = 0, x^{(4)} = 1, x^{(5)} = 2.$
- $J = 10 \rightarrow x^{(1)} = -5, x^{(2)} = -4, x^{(3)} = -3, ..., x^{(8)} = 2, x^{(9)} = 3, x^{(10)} = 4.$

For $n = 2$, we used the first 2, 5 and all 10 of the following vectors for $J = 2, 5, 10$ respectively:

Table 8: Table for the covariate sets used for $n = 2$ for both Bayesian Linear Regression and Logistic Regression

Covariate Sets
$\boldsymbol{x}^{(1)} = [0, 1]$
$\boldsymbol{x}^{(2)} = [3, -2]$
$\boldsymbol{x}^{(3)} = [-1, 2]$
$\boldsymbol{x}^{(4)} = [2, 3]$
$\boldsymbol{x}^{(5)} = [0.5, -1]$
$\boldsymbol{x}^{(6)} = [-4, -1]$
$\boldsymbol{x}^{(7)} = [2, -3]$
$\boldsymbol{x}^{(8)} = [1, 1]$
$\boldsymbol{x}^{(9)} = [0, 5]$
$\boldsymbol{x}^{(10)} = [5, 0]$

Similar for $n = 4$, the following covariate sets were used:

Table 9: Table for the covariate sets used for $n = 4$ for both Bayesian Linear Regression and Logistic Regression

Covariate Sets
$\boldsymbol{x}^{(1)} = [0, 1, 2, -1]$
$\boldsymbol{x}^{(2)} = [3, -2, 2, -2]$
$\boldsymbol{x}^{(3)} = [1, 0, 1, -2]$
$\boldsymbol{x}^{(4)} = [-1, 3, 2, -1]$
$\boldsymbol{x}^{(5)} = [-3, -1, 0, 1]$
$\boldsymbol{x}^{(6)} = [4, 1, 2, 0]$
$\boldsymbol{x}^{(7)} = [0, 2, 2, -1]$
$\boldsymbol{x}^{(8)} = [1, -3, -1, -1]$
$\boldsymbol{x}^{(9)} = [-1, 0, 4, 3]$
$\boldsymbol{x}^{(10)} = [-2, 3, 1, 1]$

Appendix B. Tables for Bayesian Linear Regression Simulation

Table 10: Results from PPE for Model 1 of Bayesian Linear Regression, for J=2. The Wasserstein distance is computed using the function "wasserstein_distance_nd" from "scipy.stats" [8], which supports multi-dimensional vectors, as we have more that one set of probabilities for Y (one for each $j \in \{1,...,J\}$)).

Number of intervals	Dirichlet log-likelihood	$\alpha\|\hat{\boldsymbol{\lambda}}$	$W(\mathbb{P}_{\boldsymbol{A}\|\hat{\boldsymbol{\lambda}}}, \boldsymbol{p})$
2	3.9	3730.7	0.0112
5	51.5	24568.4	0.0003
10	103.8	4810.8	0.0019
20	268.9	2267.4	0.0019

Table 11: Results from PPE for Model 1 of Bayesian Linear Regression, for J=5.

Number of intervals	Dirichlet log-likelihood	$\alpha\|\hat{\boldsymbol{\lambda}}$	$W(\mathbb{P}_{\boldsymbol{A}\|\hat{\boldsymbol{\lambda}}}, \boldsymbol{p})$
2	9.4	1316.1	0.0076
5	138.8	8568.8	0.0003
10	277.7	5273.1	0.0047
20	664.5	4920.9	0.0068

Table 12: Results from PPE for Model 1 of Bayesian Linear Regression, for J=10.

Number of intervals	Dirichlet log-likelihood	$\alpha\|\hat{\boldsymbol{\lambda}}$	$W(\mathbb{P}_{\boldsymbol{A}\|\hat{\boldsymbol{\lambda}}}, \boldsymbol{p})$
2	19.6	1892.7	0.0078
5	267	4107.8	0.0013
10	536.4	2745	0.0076
20	1203.7	4662.3	0.0060

Table 13: Results from PPE for Model 2 of Bayesian Linear Regression, for J=2.

Number of intervals	Dirichlet log-likelihood	$\alpha\|\hat{\boldsymbol{\lambda}}$	$W(\mathbb{P}_{\boldsymbol{A}\|\hat{\boldsymbol{\lambda}}}, \boldsymbol{p})$
2	3.4	108.7	0.0318
5	42.4	214	0.0144
10	87.7	318	0.0329
20	193.6	1699.9	0.0285

Table 14: Results from PPE for Model 2 of Bayesian Linear Regression, for J=5.

Number of intervals	Dirichlet log-likelihood	$\alpha\|\hat{\boldsymbol{\lambda}}$	$W(\mathbb{P}_{\boldsymbol{A}\|\hat{\boldsymbol{\lambda}}}, \boldsymbol{p})$
2	8	149.4	0.0403
5	98	198.9	0.0157
10	203.7	553	0.039
20	477.3	793.1	0.0463

Table 15: Results from PPE for Model 2 of Bayesian Linear Regression, for J=10.

Number of intervals	Dirichlet log-likelihood	$\alpha\|\hat{\boldsymbol{\lambda}}$	$W(\mathbb{P}_{\boldsymbol{A}\|\hat{\boldsymbol{\lambda}}}, \boldsymbol{p})$
2	10.3	39.5	0.0691
5	190.5	97.2	0.0185
10	368.2	177.8	0.0698
20	840.4	170.2	0.0896

Table 16: Results from PPE for Model 3 of Bayesian Linear Regression, for J=2.

Number of intervals	Dirichlet log-likelihood	$\alpha\|\hat{\boldsymbol{\lambda}}$	$W(\mathbb{P}_{\boldsymbol{A}\|\hat{\boldsymbol{\lambda}}}, \boldsymbol{p})$
2	4.4	36.2	0.0306
5	46.4	88.3	0.0218
10	65.2	40.9	0.1867
20	153.2	15.1	0.2275

Table 17: Results from PPE for Model 3 of Bayesian Linear Regression, for J=5.

Number of intervals	Dirichlet log-likelihood	$\alpha\|\hat{\boldsymbol{\lambda}}$	$W(\mathbb{P}_{\boldsymbol{A}\|\hat{\boldsymbol{\lambda}}}, \boldsymbol{p})$
2	8.2	17	0.0831
5	78	4.8	0.2109
10	280	1.6	0.4938
20	602	2.3	0.4

Table 18: Results from PPE for Model 3 of Bayesian Linear Regression, for J=10.

Number of intervals	Dirichlet log-likelihood	$\alpha\|\hat{\boldsymbol{\lambda}}$	$W(\mathbb{P}_{\boldsymbol{A}\|\hat{\boldsymbol{\lambda}}}, \boldsymbol{p})$
2	18	93.8	0.0258
5	154.8	36.6	0.0645
10	333.9	10.5	0.2721
20	1110.9	3.3	0.3504