MANUSCRIPT BI

A PREPRINT

September 27, 2024

1 Title

1.0.1 Abstract

1.1 Introduction

Bayesian modeling has emerged as a vital tool in modern statistics and machine learning, providing a framework for robust inference under uncertainty. Despite its potential, the current ecosystem of Bayesian software presents significant challenges for researchers, particularly due to limited formal training in Bayesian methods and model development, coding languages diversity, scalability, and the specialized nature of many Bayesian frameworks.

One of the key obstacles in Bayesian modeling is the limited formal training many researchers receive in Bayesian methods and model development. Although Bayesian inference has gained traction, the complexity of model formulation, coupled with the need for a deeper understanding of probabilistic frameworks, can be overwhelming. The gap between theoretical knowledge and practical application often results in users defaulting to more simplistic models that may not fully leverage the power of Bayesian analysis. For instance, while accessible frameworks like BRMS exist, they are typically constrained to linear regression models with flat priors. This reflects the predominance of linear regression in many fields, where statistical analysis often relies heavily on techniques that are commonly emphasized in educational settings. As a result, the potential of Bayesian methods remains underutilized in more complex, real-world applications.

Another significant barrier to the adoption of Bayesian tools is the diversity of coding languages used across various frameworks. Researchers often face the challenge of learning multiple programming languages, each associated with a specific Bayesian tool, which can complicate the model development process and create inefficiencies. For example, while Stan remains a gold standard in Bayesian inference due to its highly optimized sampling algorithms and broad support for various models, it requires users to learn and write models in its own specialized syntax (Stan code). For researchers accustomed to working in Python or R, the need to switch to a different programming environment can be a considerable hurdle. Additionally, Stan faces challenges related to scaling, particularly in the parallelization of operations and the utilization of GPU resources, which can limit its efficiency when handling large datasets or complex models. For researchers accustomed to working in Python or R, the need to switch to a different programming environment can be a considerable hurdle. This added cognitive load makes the modeling process more labor-intensive, particularly for those without extensive expertise in multiple programming languages. As a result, users may be discouraged from fully exploring the capabilities of Bayesian methods due to the friction introduced by these language barriers.

Many Bayesian modeling tools are specialized for specific types of models or applications, which can limit their flexibility and broader utility. Although these frameworks often excel in their respective domains, their specialization can introduce barriers for users looking to apply Bayesian inference to a wider array of problems. For example, TensorFlow Probability (TFP) provides a robust framework for probabilistic modeling within the TensorFlow ecosystem. However, a significant challenge arises from the requirement that users frequently need to build their own Bayesian samplers—algorithms such as Metropolis-Hastings or Hamiltonian Monte Carlo, which are critical for generating samples from the posterior distribution. This complexity, combined with the steep learning curve of the TensorFlow infrastructure, can deter researchers who are not already familiar with the system's intricacies.

Similarly, specialized tools like STRAND and BISON highlight the trade-offs inherent in current Bayesian software. STRAND and BISON excels at modeling network, yet its narrow focus can result in limited flexibility when adapting to more general Bayesian modeling needs such as computing network metrics within the same model. And as both are

based on STAN, they also face challenges related to scalability and parallelization, which can limit their efficiency when handling large datasets which in the context of network can arrive at the order of magnitude of the number of nodes.

Given these limitations, there is a clear need for a more user-friendly, scalable, and versatile Bayesian modeling tool that can address the gaps in the current software landscape. Our proposed Python software aims to meet this need by providing a platform that combines the power of Bayesian inference with greater accessibility and efficiency, making it an ideal choice for both novice and experienced users. This software leverages the computational efficiency of JAX by using either TensorFlow Probability (TFP) or NumPyro as its backbone structure, while simplifying the process of model specification and retaining their flexibility and power (including parallelization of operations and the utilization of GPU resources).

1.2 Software Presentation

Our Bayesian Inference (BI) software comes in two versions: one relying on NumPyro and the other on TFP. We have simplified the process of model specification by providing a more user-friendly interface for defining Bayesian models, making it accessible to both novice users and experienced practitioners. Since both NumPyro and TFP rely on JAX, we inherit the advantages of JAX, including speed, parallelization, scalability, and GPU support. The user code remains consistent for running parallelized mathematical operations and utilizing either GPU or CPU. From start to finish, users can employ a class object to: 1) handle data, 2) define models, 3) specify inference algorithms, 4) run inference, 5) compute posterior distributions, 6) compute model predictions, and 7) plot results. Additionally, as one of the major issue in the research area is the lack of training on bayesian modeling, we present 20 different models to cover wide range of research questions. All presented models are accompanied by detailed explanations, making it easier for users to understand the underlying assumptions and apply the models to their specific research questions.

1.2.1 User-Friendly Interface for Defining Bayesian Models

A primary feature of our Bayesian Inference (BI) software is its user-friendly interface designed for defining Bayesian models. We understand that Bayesian modeling can be daunting, especially for those who may lack formal training in the subject. To mitigate this, our interface offers a simplified version of the NumPyro and TFP functions. Typically, to define a parameter in a model, both libraries require three different functions. We have merged these functions into one, along with the sampling functions that are separated in NumPyro and TFP. The code below illustrates the differences between NumPyro, TFP, and BI for declaring a parameter of length 10 that follows a Normal distribution within a model.

```
numpyro.sample("mu", dist.Normal(0, 1)).expand([10])

yield Root(tfd.Sample(tfd.Normal(loc=1.0, scale=1.0), sample_shape=10))

bi.dist.normal("mu", 0, 1, shape = (10,))
```

Additionally, while NumPyro and TFP provide different functions to sample a parameter compared to those used to declare it, we have combined these into the same function. The code below demonstrates the differences between NumPyro, TFP, and BI for sampling a parameter of length 10 that follows a Normal distribution outside of a model.

```
numpyro.sample('samples', dist.Normal(0, 1).expand([10]), rng_key=jax.random.PRNGKey(0))

tfp.distributions.Normal(loc=0., scale=1.).sample(10)

bi.dist.normal("mu", 0, 1, shape = (10,), sample = True)
```

By combining functions to declare parameters within a model and those used to sample parameters, we allow users to easily test their models by first building them with the sampling options turned on. This approach enables users to quickly see the shapes of the objects obtained and better manage and combine model parameters. Finally, we include several custom functions tailored for advanced statistical and network modeling. These functions support specialized tasks such as centered random effects, block modeling, and the computation of network measures. The code below demonstrates the differences between NumPyro, TFP, and BI for declaring a random center effect in a model:

```
# Numpyro version of rendom centered effect-----------------------------------------------------
    a = numpyro.sample("a", dist.Normal(5, 2))
        b = numpyro.sample("b", dist.Normal(-1, 0.5))
        sigma_cafe = numpyro.sample("sigma_cafe", dist.Exponential(1).expand([2]))
        sigma = numpyro.sample("sigma", dist.Exponential(1))
```

```
        Rho = numpyro.sample("Rho", dist.LKJ(2, 2))
        cov = jnp.outer(sigma_cafe, sigma_cafe) * Rho
        a_cafe_b_cafe = numpyro.sample(
            "a_cafe,b_cafe", dist.MultivariateNormal(jnp.stack([a, b]), cov).expand([20])
        )
    a_cafe, b_cafe = a_cafe_b_cafe[:, 0], a_cafe_b_cafe[:, 1]

# TFP version of rendom centered effect-----------------------------------------------------
    alpha = yield Root(tfd.Sample(tfd.Normal(loc=5.0, scale=2.0), sample_shape=1))
    beta = yield Root(tfd.Sample(tfd.Normal(loc=-1.0, scale=0.5), sample_shape=1))

    sigma = yield Root(tfd.Sample(tfd.Exponential(rate=1.0), sample_shape=1))
    sigma_alpha_beta = yield Root(
        tfd.Sample(tfd.Exponential(rate=1.0), sample_shape=2)
    )

    Rho = yield Root(tfd.LKJ(dimension=2, concentration=2.0))
    Mu = tf.concat([alpha, beta], axis=-1)
    scale = tf.linalg.LinearOperatorDiag(sigma_alpha_beta).matmul(tf.squeeze(Rho))

# BI version of rendom centered effect-----------------------------------------------------
    Sigma_individual = bi.exponential('Sigma_individual', [ni], 1 )
    L_individual = bi.lkjcholesky('L_individual', [], ni, 1)
    z_individual = bi.normal('z_individual', [ni,K], 0, 1)
    alpha = random_centered2(Sigma_individual, L_individual, z_individual)
```

By generating these custom functions, we eliminate some data manipulation steps for users, allowing them to more easily adhere to the model's mathematical formula. All custom functions are discussed in their respective model descriptions within the BI documentation.

Finally, The architecture of our software is built around a versatile class object that encapsulates the entire modeling process. The code below demonstrates how user can 1) handle data, 2) define models, 3) specify inference algorithms, 4) run inference, 5) compute posterior distributions, 6) compute model predictions, and 7) plot results from a linear regression.

```
# Setup device & call BI object-------------------------------------------------
m = bi(platform='cpu')

# Import data ------------------------------------------------
m.data('../data/Howell1.csv', sep=';')
m.df = m.df[m.df.age > 18]
m.scale(['weight']) # Sacle continuous data
m.data_to_model(['weight', 'height']) # Cnnvert data to JAX arrays

# Define model -----------------------------------------------
def model(height, weight):
    alpha = dist.normal( 178, 20, name = 'alpha')
    beta = dist.normal(  0, 1, name = 'beta')
    sigma = dist.uniform( 0, 50, name = 'sigma')
    lk("Y", Normal(alpha + beta * weight , sigma), obs = height)

# Run mcmc ---------------------------------------------------
m.run(model)

# Summary ----------------------------------------------------
m.sampler.print_summary(0.89)

# Plot posterior distributions -------------------------------------------------

#TODO
```

This cohesive structure allows users to efficiently manage different stages of their analysis, from data handling to model evaluation without the need to switch from on library to another. BI laverage data handling through pandas library and postrior evaluation througth library Arviz.

1.2.2 Comprehensive Documentation of Models

To further support users in their Bayesian modeling journey, our BI software provides comprehensive documentation for 20 different models specifically designed to address a wide array of research questions. This extensive repertoire not only equips users with essential tools for tackling diverse analytical challenges but also serves as a valuable educational resource. By presenting various modeling options alongside detailed guidance, we empower users to make informed decisions and effectively apply Bayesian methods in their research. The documentation for each model encompasses 1) general principles, 2) underlying assumptions, 3) code snippets, and 4) mathematical details, enabling users to gain a deeper understanding of the modeling process and its nuances. Whether users are interested in hierarchical models, time-series analysis, or cutting-edge network modeling approaches, our library caters to a variety of analytical needs. This accessibility fosters an environment where users can confidently explore and implement Bayesian methods, ultimately enhancing their research capabilities.

Table 1: Table of documented models.

Documented Models
Linear Regression for continuous variable
Multiple continuous Variables.qmd
Interaction between continuous variables
Categorical variable
Binomial model
Beta binomial model
Poisson model
Gamma-Poisson
Multinomial model
Dirichlet model
Zero inflated
Varying intercepts
Varying slopes
Gaussian processes
Measuring error
Missing data
Network model
Sender receiver network model
Block model network
Network Computations
Network Based diffusion approach
Multiplex network model
Multiplex temporal network model
Multilayer network model

For example, the state-of-the-art network models implemented in BI include methods for censoring and exposure control, block modeling, multiplex and multilayer networks, network-based diffusion approaches, and computation of network metrics. Each approach is presented in detail in the documentation, complete with their custom functions. This comprehensive documentation ensures that users can easily navigate the software and identify the models that best suit their research requirements. Furthermore, these diverse approaches can be combined to create a single model, such as a multiplex model with controls for censoring and exposure biases, along with the computation of social network measures (code snippets bellow). This versatility showcases the flexibility and power of the software, allowing users to tailor their analyses to meet specific research needs, which is not possible with other specialized Bayesian network frameworks such as STRAND and BISON. These frameworks feature prebuilt models that either do not allow for direct computation of network measures within the model (BISON requires two separate models) or do not support network measure computation at all (STRAND).

```
# Setup device & call BI object-------------------------------------------------
m = bi(platform='cpu')
```

```
# Import data ------------------------------------------------
m.data('../data/Howell1.csv', sep=';')
m.df = m.df[m.df.age > 18]
m.scale(['weight']) # Sacle continuous data
m.data_to_model(['weight', 'height']) # Cnnvert data to JAX arrays

# Define model -----------------------------------------------
def model(height, weight):
    alpha = dist.normal( 178, 20, name = 'alpha')
    beta = dist.normal(  0, 1, name = 'beta')
    sigma = dist.uniform( 0, 50, name = 'sigma')
    lk("Y", Normal(alpha + beta * weight , sigma), obs = height)

# Run mcmc ---------------------------------------------------
m.run(model)

# Summary ----------------------------------------------------
m.sampler.print_summary(0.89)

# Plot posterior distributions -------------------------------------------------

#TODO
```

1.3 Discussion

The XXX framework is built on top of the popular Python programming language, with a focus on providing a user-friendly interface for model development and interpretation. Our framework is designed to be modular and extensible, allowing users to easily incorporate their own custom models and data types into the framework. We have also developed a set of tutorials and examples to help users get started with the framework and to demonstrate its capabilities.

One of the key features of this software is its comprehensive library of 16 predefined Bayesian models, covering a wide range of common applications and use cases. These models are accompanied by detailed explanations, making it easier for users to understand the underlying assumptions and apply the models to their specific research questions. In addition to these built-in models, the software includes several custom functions tailored for advanced statistical and network modeling. These functions support specialized tasks such as centered random effects, block modeling, and the computation of network measures. ## Aknowledgements ## References