Bayesian Modeling with Stan:
An Introductory Workshop

Jakob Torgander

Department of Statistics, Uppsala University
jakob.torgander@statistik.uu.se

March 19, 2025

Outline

About me/us

- Jakob Torgander, PhD student in Statistics at Uppsala University
- Research in Bayesian Statistics & Probabilistic Programming
- Part of InterBayes network `https://interbayes.github.io`
- Developer of `posteriordb`

Bayesian modeling

- Flexible framework for defining and fitting statistical models
- Incorporating prior information $\rightarrow$ Less data needed
- Framework for updating beliefs/inference as new data is collected
- Intuitive inference. Compare:
  1. $\mathbb{P}(\underline{T}(X) < \theta < \overline{T}(X)) \geq 1 - \alpha$, T is a statistic of data X
  2. $\mathbb{P}(a < \theta < b) \geq 1 - \alpha$

Bayesian modeling

- Flexible framework for defining and fitting statistical models
- Incorporating prior information $\rightarrow$ Less data needed
- Framework for updating beliefs/inference as new data is collected
- Intuitive inference. Compare:
  1. $\mathbb{P}(\underline{T}(X) < \theta < \overline{T}(X)) \geq 1 - \alpha$, T is a statistic of data X
  2. $\mathbb{P}(a < \theta < b) \geq 1 - \alpha$
- Common bottleneck: computation. Topic of this workshop

Posterior distribution

Main component of Bayesian modeling: posterior distribution:

$$p(\theta|x) \propto \prod_{i=1}^{N} \underbrace{f(x_i|\theta)}_{\text{Likelihood}} \cdot \underbrace{p(\theta)}_{\text{Prior}}$$

- $f(x|\theta)$: relationship between parameter θ and data sample x
- $p(\theta)$: prior beliefs about θ (if any)
- Today's task: compute $p(\theta|x)$ using probabilistic programming

Posterior distribution

Why use posterior distribution?

Posterior distribution

Why use posterior distribution?

Posterior contains the necessary information about θ given data

$$\mathbb{E}[h(\theta)] = \int h(\theta)p(\theta|x)d\theta$$

Posterior distribution

Why use posterior distribution?

Posterior contains the necessary information about θ given data

$$\mathbb{E}[h(\theta)] = \int h(\theta) p(\theta|x) d\theta$$

Examples:

$h(\theta)$	$\mathbb{E}[h(\theta)]$
θ	Expected value of θ
$(\theta - \mathbb{E}[\theta])^2$	"Variance of θ"
$\mathbb{1}(\theta > \tau)$	"Probability that θ is larger than τ"

Monte carlo methods

- Problem: posteriors rarely given in closed analytical form.

Monte carlo methods

- Problem: posteriors rarely given in closed analytical form.
- Solution: draw samples $\theta^{(1)}, \dots, \theta^{(N)}$ from the target posterior and use Monte Carlo integration:

$$\mathbb{E}[h(\theta)] \approx \frac{1}{N} \sum_{i=1}^{N} h(\theta^{(i)})$$

Monte carlo methods

- Problem: posteriors rarely given in closed analytical form.
- Solution: draw samples $\theta^{(1)}, \ldots, \theta^{(N)}$ from the target posterior and use Monte Carlo integration:

$$\mathbb{E}[h(\theta)] \approx \frac{1}{N} \sum_{i=1}^{N} h(\theta^{(i)})$$

- As the number of samples grow $\frac{1}{N} \sum_{i=1}^{N} h(\theta^{(i)}) \overset{a.s}{\rightarrow} \mathbb{E}[h(\theta)]$
- For drawing the samples $\theta^{(i)}$, we will use Stan

Stan

- Released in 2012
- Probabilistic programming language for Bayesian inference with MCMC, Variational inference and penalized maximum likelihood estimation with optimization
- Implements MCMC using Hamiltonian Monte Carlo (HMC) and the No-U-Turn Sampler (NUTS)
- Written in C++ using similar syntax. Can be used together with R, Python or Julia. (Lab will be in R)
- Link to documentation & tutorials: `https://mc-stan.org/docs/`

Components of Stan program

Recall, we want to compute

$$p(\theta|x) \propto \prod_{i=1}^{N} \underbrace{f(x_i|\theta)}_{\text{Likelihood}} \cdot \underbrace{p(\theta)}_{\text{Prior}}$$

or equivalently,

$$\log p(\theta|x) \propto \sum_{i=1}^{N} \underbrace{\log f}_{\text{log likelihood}}(x_i|\theta) + \underbrace{\log p}_{\text{log prior}}(\theta)$$

A Stan program specifies the (log) posterior distribution through data, parameter and model blocks

Stan main program blocks

```
data{..}

parameters{..}

model {..}
```

- Declaring input data & arguments
- Specifying model parameters to be fitted
- Defining model (log posterior)

$$\log p(\theta|x) \propto \sum_{i=1}^{N} \underbrace{\log f(x_i|\theta)}_{\text{log likelihood}} + \underbrace{\log p(\theta)}_{\text{log prior}}$$

Stan program blocks

```
functions{..}

data{..}

transformed data{..}

parameters{..}

transformed parameters {..}

model{..}

generated quantities{..}
```

Example: simple linear regression

Consider the standard simple regression model:

$$\mathbb{E}[Y_i] = \alpha + \beta X_i + \epsilon_i, \qquad \epsilon_i \overset{iid}{\sim} \mathcal{N}(0, \sigma^2)$$

A Bayesian model for this using *weakly informative priors* can be defined as follows

$$\begin{aligned} Y_i | X_i, \alpha, \beta &\sim \mathcal{N}(\alpha + \beta X_i, \sigma^2) \\ \alpha &\sim \mathcal{N}(0, 10) \\ \beta &\sim \mathcal{N}(0, 10) \\ \sigma &\sim \mathcal{N}^+(0, 10) \end{aligned}$$

Example: simple linear regression

```
data {
  int<lower=0> N;
  vector[N] x;
  vector[N] y;
}
parameters {
  real alpha;
  real beta;
  real<lower=0> sigma;
}
model {
  y ~ normal(alpha + beta * x, sigma);
  alpha ~ normal(0, 10);
  beta ~ normal(0, 10);
  sigma ~ normal(0, 10);
}
```

Example: simple linear regression

```
data {
  int<lower=0> N;
  vector[N] x;
  vector[N] y;
}
parameters {
  real alpha;
  real beta;
  real<lower=0> sigma;
}
model {
  y ~ normal(alpha + beta * x, sigma);
  alpha ~ normal(0, 10);
  beta ~ normal(0, 10);
  sigma ~ normal(0, 10);
}
```

Comments:

- Variable and types need to be declared
- Each statement ends with ;
- Constraints of variables & parameters enforced using $< \cdots >$-brackets

Stan data types

Most common data types:

- `int`: Integers, eg. discrete data
- `real`: Real numbers, eg. continuous
- `vector`: One dimensional array for storing numbers
- `matrix`: Two dimensional array for storing numbers
- `simplex`: Probability vector (sums to 1)

Stan data types

Most common data types:

- `int`: Integers, eg. discrete data
- `real`: Real numbers, eg. continuous
- `vector`: One dimensional array for storing numbers
- `matrix`: Two dimensional array for storing numbers
- `simplex`: Probability vector (sums to 1)

- Others exist! See manual.
- Array/matrix subsetting same as in standard C/C++

Example: multiple linear regression

```
data {
  int<lower=0> N;
  int<lower=0> K;
  matrix[N, K] x;
  vector[N] y;
}
parameters {
  real alpha;
  vector[K] beta;
  real<lower=0> sigma;
}
model {
  y ~ normal(x * beta + alpha, sigma);
  ...
}
```

Alternative model definition

Previous example uses vectorized notation for defining likelihood and priors (eg. `y` $\sim$ `normal`). A more flexible option is to define the model ”term-wise” using a for-loop

```
model {
 for (i in 1:N){
  target += lpdf_normal(y[i]|alpha + x[i] * beta, sigma);
 }
 target += lpdf_normal(alpha |0, 10);
 target += lpdf_normal(beta |0, 10);
 ...
}
```

Alternative model definition

Previous example uses vectorized notation for defining likelihood and priors (eg. `y ~ normal`). A more flexible option is to define the model "term-wise" using a for-loop

```
model {
 for (i in 1:N){
  target += lpdf_normal(y[i]|alpha + x[i] * beta, sigma);
 }
 target += lpdf_normal(alpha |0, 10);
 target += lpdf_normal(beta |0, 10);
 ...
}
```

Resembles mathematical definition of the log posterior

$$\log p(\theta|x) \propto \sum_{i=1}^{N} \log f(y_i|x_i, \alpha, \beta) + \log p(\alpha) + \log p(\beta) + \log p(\sigma),$$

i.e. `target+=` corresponds adding a log-probability term to the above sum

Generated quantities block

Recall, end goal of computing posterior often is to compute

$$\mathbb{E}[h(\theta)] \approx \frac{1}{N}\sum_{i=1}^{N} h(\theta^{(i)})$$

In Stan, the `generated quantities` block can be used for implementing the computation of $h(\theta^{(i)})$ sample-wise

Generated quantities - example

Example: $h(\beta) = \beta^2$ and $h(\beta) = \mathbb{1}(\beta > 0)$

```
generated quantities{
 real beta_sqr =  beta^2;
 int beta_is_significant = beta > 0;
 }
```

Most common use of `generated quantities` is to generate samples from the predictive distribution $p(\hat{y}|y)$ of new data $\hat{y}$ given current data y as will be shown later.

Model selection

Common likelihood-prior combinations for 1-dim posteriors

Data	Parameter	Likelihood	Prior
Continuous ($y \in \mathbb{R}$)	Mean Variance	Normal	Normal Inv-Gamma
Discrete ($y \in \mathbb{Z}$)	Mean	Poisson Neg.binomial	Normal Normal
Binary ($y \in \{0, 1\}$)	Probability of	Bernoulli	Beta Dirichlet

Model selection

Common likelihood-prior combinations for 1-dim posteriors

Data	Parameter	Likelihood	Prior
Continuous ($y \in \mathbb{R}$)	Mean Variance	Normal	Normal Inv-Gamma
Discrete ($y \in \mathbb{Z}$)	Mean	Poisson Neg.binomial	Normal Normal
Binary ($y \in \{0, 1\}$)	Probability of	Bernoulli	Beta Dirichlet

- Prior can both incorporate previous known knowledge (informative) or be chosen to be weakly informative
- Can also be ignored in the Stan program (vague prior)
- See Stan manual for many more models and likelihood/prior combinations

Motivation

- How do we know that we can trust that the posterior is computed correctly?
- How do we know if our model/prior is the best possible for our data?
- We will here present common model diagnostic tools for answering these questions

Sampler convergence - background

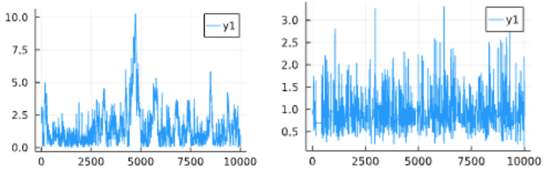
- To be able to ”trust” our samples, we need to assure that our sampler has ”converged” properly.
- Recall, if we are able produce independent samples $\theta^{(i)}$ from the true posterior, then as the number of samples N grows,

$$\frac{1}{N}\sum_{i=1}^{N} h(\theta^{(i)}) \overset{a.s}{\to} \mathbb{E}[h(\theta)]$$

- Formally: Stan produces $\theta^{(i)}$ using Markov Chain Monte Carlo (MCMC) methods, by constructing a Markov chain which has the target posterior as its stationary distribution
- Thus, to trust our samples, we must verify that the underlying Markov chain has converged to its stationary distribution

Traceplots

Idea: verifying convergence by inspecting the samples over time over several chains

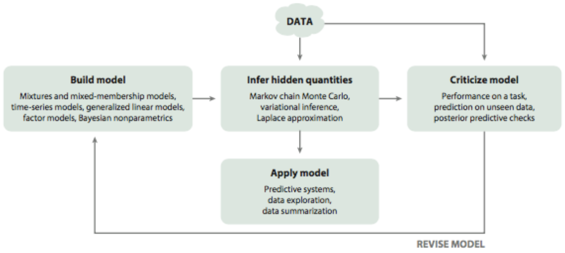


- Left figure: samples clearly correlated, chain not stationary!
- Right: better but not perfect
- Repeat for multiple chains and assure that all chains have found the same range
- Only keep samples after convergence (warm-up phase).

Quantitative diagnostics - $\hat{R}$ and ESS

- $\hat{R}$-statistic: comparing variance within and between different chains. Values "close to" 1 indicates that all chains have converged.
- Effective sample size (ESS): The number of independent samples needed for the same uncertainty as in our estimates. Measure of how independent samples are

Model evaluation - Box's loop


DATA
Build model
Mixtures and mixed-membership models, time-series models, generalized linear models, factor models, Bayesian nonparametrics
Infer hidden quantities
Markov chain Monte Carlo, variational inference, Laplace approximation
Criticize model
Performance on a task, prediction on unseen data, posterior predictive checks
Apply model
Predictive systems, data exploration, data summarization
REVISE MODEL

- Formalized by Blei, David M. "Build, compute, critique, repeat: Data analysis with latent variable models." Annual Review of Statistics and Its Application 1 (2014): 203-232.
- General framework for iterative development of statistical models
- Will now focus on the "Criticise model" block

Posterior predictive distribution

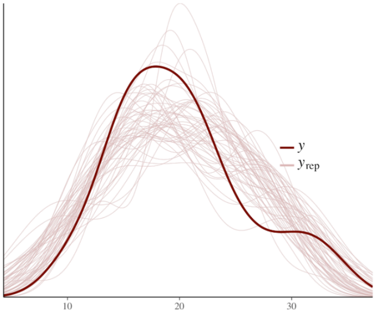
- Idea: Evaluate/criticise model by how well it predicts new data
- Key object: the posterior predictive distibution (PPD)

$$p_{post}(\hat{y}|\theta) = \int p(\hat{y}|\theta)p(\theta|y)dy,$$

  where $\hat{y}$ is a new data point
- Posterior predictive checks
- Leave-one-out cross validation

Posterior predictive check

- Idea: compare samples from the posterior predictive distribution with the actual data



- Useful for identifying obvious mismatches between the model and data
- Can also be done for priors (prior predictive check)

Sampling from the PPD

- In practice, samples from the PPD can be generated by sampling one data point from the likelihood *for each* posterior sample and data point.
- In Stan, this can be done *in parallel* to the posterior sampling using the `generated quantities` block as follows

```
generated quantities {
  array[N] real ypred;
  for (i in 1:N){
  ypred[i] = normal_rng(alpha + x[i] * beta, sigma);
  }
}
```

Frame Title

Why does this work?

$$\begin{aligned}\mathbb{E}[h(Y))] &= \int h(\bar{y})p(\bar{y}|y)dy \\ &= \int\int h(\hat{y})p(\hat{y}|\theta)p(\theta|y)d\theta dy \\ &\approx \frac{1}{N}\frac{1}{M}\sum_{i=1}^{N}\sum_{j=1}^{M} h(\hat{y}^{(i,j)}),\end{aligned}$$

where $\hat{y}^{(i,j)} \sim p(\hat{y}, \theta^{(j)}|y) = p(\hat{y}|\theta^{(j)})p(\theta^{(j)}|y)$

Model comparison

- The PPD p_{post} can also be used for comparing different candidate models
- Useful theoretic metric for this is the expected log predictive density (ELPD)

$$\mathbb{E}[\log(p_{post}(\hat{Y}|\theta))] = \int \log(p_{post}(\hat{y}|\theta))p(\hat{y})d\hat{y},$$

  where $p(\hat{y})$ is the true data generating process
- ELPD measures how well the PPD matches the true distribution on average: if $p(\hat{y})$ is high, $log(p(\hat{y}|\theta))$ should be high and v.v
- However $p(\hat{y})$ is unknown in general and thus ELPD needs to be estimated $\rightarrow$ cross validation

Leave-one-out cross-validation (LOO-CV)

- Leave-one-out cross-validation (LOO-CV) estimates ELPD by

$$\sum_{i}^{n} \log p_{post(-i)}(y_i|\theta),$$

  where $p_{post(-i)|\theta}(y_i)$ is the PPD evaluated at y_i of a posterior computed excluding y_i as a data point.
- Can be done computationally efficient using Pareto smoothed Importance Sampling (PSIS LOO-CV)
- Implemented in the `loo`-package

Computer lab

Now: experiment with Stan on your own computer. Two options:

- A Try Stan on your own research problem/data
- B Workshop demo notebook with corresponding exercises at `https://github.com/JTorgander/interbayes-workshop`