A Short Introduction in Bayesian Modelling Using Stan

Barmpounakis Petros

Athens University of Economics and Business,
Department of Statistics

barmpounakis@aueb.gr

Thursday 11 March, 2021

Overview

Introduction

- Bayesian methodology requires the use of Markov Chain Monte Carlo methods in order to derive the posterior density.
- Famous statistical tools like BUGS (WinBUGS, OpenBUGS, JAGS) and Stan will be compared.
- BUGS implements the Gibbs Sampler and the Metropolis-Hastings algorithms and Stan implements an advanced variant of Hamiltonian Monte Carlo called No-U-Turn Sampler.

Hamiltonian Monte Carlo

1. Hamiltonian Monte Carlo (HMC) is a Markov chain Monte Carlo (MCMC) method that uses the derivatives of the density function being sampled to generate efficient transitions spanning the space of the posterior,, by avoiding the random-walk behaviour that arises in random-walk Gibbs samplers when there is correlation in the posterior. (Betancourt and Girolami, 2013; Neal, 2011).

2. It uses an approximate Hamiltonian dynamics simulation based on numerical integration which is then corrected by performing a Metropolis acceptance step.

3. *Stan* uses the no-U-turn sampler (NUTS), which automatically selects an appropriate number of leapfrog steps in each iteration in order to allow the proposals to traverse the posterior without doing unnecessary work, by avoiding the random-walk behaviour that arises in random-walk Gibbs samplers when there is correlation in the

Momentum Variables

- The goal of sampling is to draw from a density $p(\theta)$ for parameters θ. This is typically a Bayesian posterior $p(\theta|y)$, given data y. HMC introduces auxiliary momentum variables ρ independent of θ and draws from a joint density. $p(\rho, \theta|y) = p(\rho|\theta, y)p(\theta|y)$

$$\rho \sim MultivariateNormal(0, \Sigma)$$

In Stan, this matrix may be set to the identity matrix or estimated from warmup samples and optionally restricted to a diagonal matrix. The inverse Σ^{-1} is known as the mass matrix, and will be a unit, diagonal, or dense if Σ is.

The Hamiltonian

The joint density $p(\rho, \theta|y)$ defines a Hamiltonian

$$H(\rho, \theta|y) = -\log p(\rho, \theta|y)$$

$$= -\log p(\rho|\theta, y) - \log p(\theta|y)$$

$$= T(\rho|\theta, y) + V(\theta|y)$$

,where the term $T(\rho|\theta, y) = -\log p(\rho|\theta, y)$ is called "kinetic energy" and the term $V(\theta|y) = -\log p(\theta|y)$ is called "potential energy". The potential energy is specified by the Stan program through its definition of a log density (Also, called target density). $\log p(\theta|y)$ is the posterior up to a normalizing constant.

Generating transitions

Starting from the current value of the parameters, a transition to a new state is generated in two stages before being subjected to a Metropolis accept step. First, a value for the momentum is drawn independently of the current parameter values. Next, the joint system $p(\rho, \theta | y)$ made up of the current parameter values θ and new momentum ρ is evolved via Hamilton's equations.

$$\frac{d\theta}{dt} = +\frac{\partial H}{\partial \rho} = +\frac{\partial T}{\partial \rho}$$

$$\frac{d\rho}{dt} = -\frac{\partial H}{\partial \theta} = -\frac{\partial V}{\partial \theta}$$

This two-state differential equation is solved in Stan, in agreement with other packages, by using the leapfrog integrator.

Leapfrog Integrator

The leapfrog integrator takes discrete steps of some small time integral ϵ. It begins by drawing a momentum value from the ρ density and the alternate half-step updates of the momentum and full-step updates of the position.

$$\rho \leftarrow \rho - \frac{\epsilon}{2}\frac{\partial V}{\partial \theta}$$
$$\theta \leftarrow \theta + \epsilon \Sigma \rho$$
$$\rho \leftarrow \rho - \frac{\epsilon}{2}\frac{\partial V}{\partial \theta}$$

After L leapfrog steps are applied, a total of Lϵ time is simulated. The resulting state at the end of the simulation (L repetitions of the above three steps) will be denoted $(\rho^*, theta^*)$. At the end, a Metropolis acceptance step is applied, where the probability of keeping the proposal is $min(1, exp(H(\rho, \theta) - H(\rho^*, \theta^*)))$.

No-U-Turn Sampler

- The selection of L leapfrog steps and step size ϵ can prove to be extremely difficult for an uninitiated user.
- Proper calibration requires many trial runs increasing the computational cost, as well as deep knowledge and understanding of the algorithm in order to interpret the results of these trial runs.
- If ϵ is too large, the simulation will be inaccurate, on the other hand if ϵ is to small, the computation will be wasted taking many small steps.
- If L is too small, then successive samples will be too close resulting in random walk behavior and slow mixing. If L is too large, then HMC will generate trajectories that loop back and retrace the steps.
- *Stan* uses the no-U-turn sampler (NUTS), which automatically selects an appropriate number of leapfrog steps in each iteration in order to allow the proposals to traverse the posterior without doing unnecessary work, by avoiding the random-walk behaviour, resulting at a much more user-friendly algorithm.

Seeds: Random effect logistic regression

This example is taken from Table 3 of Crowder (1978), and concerns the proportion of seeds that germinated on each of 21 plates arranged according to a 2 by 2 factorial layout by seed and type of root extract. The model is essentially a random effects logistic, allowing for over-dispersion. If p_i is the probability of germination and r_i and n_i are the number of germinated and the total number of seeds on the i^{th} plate, $i = 1, ..., 21.$, we assume:

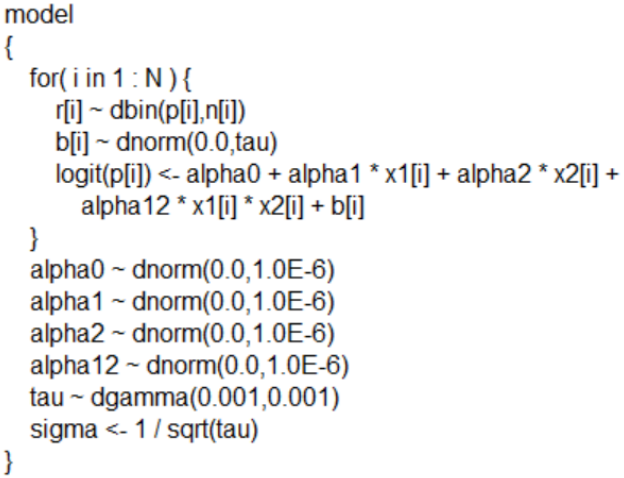
$$r_i \sim Binomial(p_i, n_i)$$

$$logit(p_i) = \alpha_0 + \alpha_1 x_{1i} + \alpha_2 x_{2i} + \alpha_{12} x_{1i} x_{2i} + b_i$$

$$b_i \sim Normal(0, \tau)$$

,where x_{1i}, x_{2i} are the seed type and root extract of the i^{th} plate, and an interaction term $\alpha_{12} x_{1i} x_{2i}$ is included. $\alpha_0, \alpha_1, \alpha_2, \alpha_{12}, \tau$ are given independent "noninformative" priors.

BUGS model



```
model
{
   for( i in 1 : N ) {
      r[i] ~ dbin(p[i],n[i])
      b[i] ~ dnorm(0.0,tau)
      logit(p[i]) <- alpha0 + alpha1 * x1[i] + alpha2 * x2[i] +
         alpha12 * x1[i] * x2[i] + b[i]
   }
   alpha0 ~ dnorm(0.0,1.0E-6)
   alpha1 ~ dnorm(0.0,1.0E-6)
   alpha2 ~ dnorm(0.0,1.0E-6)
   alpha12 ~ dnorm(0.0,1.0E-6)
   tau ~ dgamma(0.001,0.001)
   sigma <- 1 / sqrt(tau)
}
```

Stan model

```
data {
    int<lower=0> I;
    int<lower=0> n[I];
    int<lower=0> N[I];
    vector[I] x1;
    vector[I] x2;
}

transformed data {
    vector[I] x1x2;
    x1x2 <- x1 .* x2;
}
parameters {
    real alpha0;
    real alpha1;
    real alpha12;
    real alpha2;
    real<lower=0> tau;
    vector[I] b;
}
```

```
transformed parameters {
    real<lower=0> sigma;
    sigma  <- 1.0 / sqrt(tau);
}
model {
   alpha0 ~ normal(0.0,1.0E3);
   alpha1 ~ normal(0.0,1.0E3);
   alpha2 ~ normal(0.0,1.0E3);
   alpha12 ~ normal(0.0,1.0E3);
   tau ~ gamma(1.0E-3,1.0E-3);

   b ~ normal(0.0, sigma);
   n ~ binomial_logit(N, alpha0 + alpha1 * x1
   + alpha2 * x2 + alpha12 * x1x2 + b);
}
```

Stochastic Epidemic Model - Construction of Likelihood

- Daily number of deaths attributed to Covid-19 D_t

$$\begin{aligned} D_t &\sim \text{Negative-Binomial}\left(d_t, \phi 1\right) \\ d_t &= ifr_{adj} * \sum_{\tau=0}^{t-1} c_\tau \pi_{t-\tau} \\ \phi 1 &\sim \text{Cauchy}\left(0, 5\right) \\ ifr_{adj} &= ifr_t * ifr_{noise} \\ ifr_{noise} &\sim \text{Normal}\left(1, 0.01\right) \end{aligned} \tag{1}$$

where c_t is the predicted total cases at time t, π_t is the infection to death distribution, and ifr_{adj} is the adjusted infection-to-fatality ratio

$$ifr_t = \begin{cases} ifr_1 & t < t^* \\ ifr_2 & t \geq t^* \end{cases}$$

We consider lower ifr at the second wave, to denote the better means the health systems has during the second wave. (more doctor experience, better medicines).

Stochastic Epidemic Model - Data Augmentation

- Daily predicted total cases of Covid-19 c_t. The predicted generated cases are generated by the following mechanism.

$$\begin{aligned} c_t &= S_t * R_t * \sum_{\tau=0}^{t-1} c_\tau g_{t-\tau} \\ R_t &= R_0 * exp(-\sum_{\kappa=1}^{6} effect_\kappa * I_{k,t}) \\ R_0 &\sim \text{Normal}\,(3,1) \\ effect_\kappa &\sim \text{Normal}\,(0,9) \end{aligned} \tag{2}$$

,where $S_t = 1 - \frac{\sum_{s=1}^{t-1} c_s}{N}$ the shrinkage of R_t due to the reduction of susceptible individuals and $I_{k,t}$ is an indicator function for each stochastic changepoint.

Stochastic Epidemic Model - Stochastic changepoints

- Stochastic Changepoints T_k

$$\begin{aligned} T_1 &\sim \text{Uniform}\left(3, N_{days} - 3\right) \\ T_{i+1} &= T_i + e_i \\ e_i &\sim \text{Uniform}\left(0, 100\right) \end{aligned} \tag{3}$$

We use this formulation on the changepoints to counter the label switching problem.

Stan model

```
data{
  int<lower=0> N0; // number of days to impute
  int<lower=1> Nobs; // number of days observed must be <= N
  int<lower=0> deaths[Nobs]; //daily deaths
  int<lower=0> pop; //population
  int<lower=0> reported_cases[Nobs]; //daily reported cases
  vector<lower=0>[Nobs] s_int_rev1 ; // serial interval reverse order vector for generation distribution
  vector<lower=0>[Nobs] s_int_rev2 ; // serial interval reverse order vector for infection to death distribution
  real<lower=0,upper=1> ifr[2]; // infection-fatality ratio
  int<lower=0> ifr_cp;// changepoint for ifr
  int<lower=0> epi_start;// day to start for deaths
}

parameters{
    real<lower=0> R0;
    vector[3] effect;
    vector<lower=0>[N0] prediction0;
    real<lower=0> ifr_noise[2];
    real<lower=0> phi;
    real<lower=0> tau;
    real<lower=3,upper=Nobs-3> T1;
    real<lower=0,upper=100> e[2]; //extra time for the changepoint number 2, 3
}

transformed parameters{
real<lower=T1,upper=Nobs-3> T2;
real<lower=T2,upper=Nobs-3> T3;
vector<lower=0>[Nobs] Rt ;
vector<lower=0>[Nobs] Rt_adj ;
vector<lower=0>[Nobs] cumul_sum ;
vector<lower=0>[Nobs] prediction;
vector<lower=0>[Nobs] E_deaths;
real<lower=0,upper=1> ifr_adj[2];

T2=T1+e[1];
T3=T2+e[2];
for(i in 1:2){
ifr_adj[i]=ifr_noise[i]*ifr[i];
}
```

```
prediction[1:N0]=prediction0;
cumul_sum[1:N0]=cumulative_sum(prediction[1:N0]);

for( t in 1:Nobs){
Rt[t]=R0*exp(-inv_logit(5* (t-T1))*inv_logit(5* (T2-t))*effect[1]-inv_logit(5* (t-T2))*inv_logit(5* (T3-t))*effect[2]
-inv_logit(5* (t-T3))*effect[3]);
}
Rt_adj[1:N0+1]=Rt[1:N0+1];

for(i in (N0+1):Nobs-1){
real convolution=dot_product(prediction[1:i-1] ,s_int_rev1[Nobs-i+2:Nobs]);
prediction[i]=Rt_adj[i]*convolution;
cumul_sum[i]=cumul_sum[i-1]+prediction[i];
Rt_adj[i+1]=((pop-cumul_sum[i])/pop)*Rt[i+1];
}
prediction[Nobs]=Rt_adj[Nobs]*dot_product(prediction[1:Nobs-1] ,s_int_rev1[2:Nobs]);
cumul_sum[Nobs]=cumul_sum[Nobs-1]+prediction[Nobs];

E_deaths[1]=ifr_adj[1]*prediction[1]*s_int_rev2[Nobs];
for(i in 2:Nobs){
if(i<ifr_cp)
E_deaths[i]=ifr_adj[1]*dot_product(prediction[1:i-1] ,s_int_rev2[Nobs-i+2:Nobs]);
else
E_deaths[i]=ifr_adj[2]*dot_product(prediction[1:i-1] ,s_int_rev2[Nobs-i+2:Nobs]);
}
}

model{
R0 ~ normal(3,1);
effect ~ normal(0,3);
tau ~ exponential(0.03);
prediction0 ~ exponential(1/tau);
ifr_noise ~ normal(1,0.1);
phi ~ cauchy(0,5);
deaths[epi_start:Nobs] ~ neg_binomial_2(E_deaths[epi_start:Nobs], phi);
}
```

Interfaces to run Stan

- You can use Stan through R (Rstan), Python (Pystan) and more.
- In both R and Python, diagnostics and plotting libraries are offered, such as bayesplot and ARVIZ, respectivelly.
- All interfaces run a c++ program. If you've got a complicated model, running the sampler will dominate your processing time, no matter which interface you use.
- In all interfaces you can run the same exact stan model without needing any adjustments in your model code.
- Ultimately, the only differences between Rstan and Pystan are the names of each functions and the manipulation and parsing of data for each programming language.
- For more information about the implementation of HMC and NUTS in Stan, the Stan manual is a well-written and comprehensive instructional book or you can join a very active community at the Stan forum at https://discourse.mc-stan.org/.