Implementação Bayesiana via Stan do modelo de regressão linear múltipla

Prof. Vinícius D. Mayrink - Departamento de Estatística - ICEx - UFMG

Estatística Bayesiana - 1º Semestre de 2025

O primeiro passo é instalar o *software* `R` e o pacote `rstan` em seu computador. Para maiores informações sobre o `Stan` visite a página mc-stan.org.

Limpe a área de trabalho do `R` e carregue o pacore `rstan` com os comandos a seguir.

```
rm(list=ls(all=TRUE))
library(rstan)
# comando para evitar recompilar.
rstan_options(auto_write = TRUE)
# comando para executar diferentes cadeias em paralelo.
options(mc.cores = parallel::detectCores())
# Fixando semente para garantir reproducibilidade.
set.seed(12345)
```

Introdução

A regressão linear múltipla é uma técnica bastante utilizada na estatística. O objetivo principal é avaliar a relação linear entre uma variável resposta Y_i e um grupo de K covariáveis $X_{1i}, X_{2i}, \cdots, X_{Ki}$ observadas para diversos indivíduos $i = 1, \cdots, n$.

A grande maioria dos pacotes computacionais disponíveis em *softwares* estatísticos, realizam o ajuste do modelo de regressão através do procedimento clássico de mínimos quadrados ou máxima verossimilhança. O presente material mostra como é feita a estimação através da abordagem Bayesiana.

O modelo é escrito como segue:

$$Y_i = \beta_0 + \beta_1 X_{1i} + \cdots + \beta_K X_{Ki} + \epsilon_i \quad \text{com} \quad \epsilon_i \sim N(0, \sigma^2)$$

O modelo de regressão sob estudo é composto por $q = K+1$ coeficientes, sendo um deles o intercepto β_0. O valor K representa a quantidade de covariáveis. Veja que os parâmetros de interesse são: $\beta = (\beta_0, \beta_1, \cdots, \beta_K)^\top$ e σ^2. Lembre-se que estes parâmetros são desconhecidos e, portanto, na inferência Bayesiana especificamos uma distribuição *a priori* para descrever nossa incerteza inicial sobre eles. Admita que, dado $\{\beta, \sigma^2\}$, temos independência condicional entre $Y_1, Y_2, \cdots, Y_n$.

Denote $\boldsymbol{Y} = (Y_1, Y_2, \cdots, Y_n)^\top$ e escreva a matriz de covariáveis $(n \times q)$ como segue:

$$\boldsymbol{X} = \begin{bmatrix} 1 & X_{11} & X_{21} & \cdots & X_{K1} \\ 1 & X_{12} & X_{22} & \cdots & X_{K2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & X_{1n} & X_{2n} & \cdots & X_{Kn} \end{bmatrix}.$$

Para obter a função de verossimilhança, perceba que $Y_i \sim N(\boldsymbol{X}_{i\bullet}\beta, \sigma^2)$, sendo $\boldsymbol{X}_{i\bullet}$ o vetor representando a i-ésima linha da matriz $\boldsymbol{X}$. A média da variável aleatória Y_i é uma combinação linear das covariáveis observadas para o indivíduo i. Temos $\boldsymbol{X}_{i\bullet}\beta = \beta_0 + \beta_1 X_{1i} + \cdots + \beta_K X_{Ki}$ sendo usualmente chamado de "preditor linear".

Para o i-ésimo indivíduo, temos a f.d.p.: $f_{Y_i|\beta,\sigma^2,\boldsymbol{X}_{i\bullet}}(y_i) = (2\pi\sigma^2)^{-1/2}\exp\{-\frac{1}{2\sigma^2}(Y_i - \boldsymbol{X}_{i\bullet}\beta)^2\}$.

Por independência condicional escreva: $f_{\boldsymbol{Y}|\beta,\sigma^2,\boldsymbol{X}}(\boldsymbol{y}) = \prod_{i=1}^{n}(2\pi\sigma^2)^{-1/2}\exp\{-\frac{1}{2\sigma^2}(Y_i - \boldsymbol{X}_{i\bullet}\beta)^2\}$.

Gerando dados artificiais

Nesta análise, iremos explorar dados sintéticos que serão gerados com base na estrutura do modelo de regressão sob investigação. Esta estratégia permite avaliar a performance do modelo Bayesiano, pois os valores reais dos parâmetros alvo serão escolhidos para simular (gerar valores de uma distribuição de probabilidade). Durante o ajuste Bayesiano via `Stan`, iremos ignorar o conhecimento destes valores verdadeiros. Na análise dos resultados *a posteriori*, uma comparação "real versus estimado" será feita para julgarmos a qualidade da estimação.

O tamanho amostral e os valores reais dos parâmetros serão os seguintes:

```
n = 200                                 # Tamanho amostral.
beta = c(1.5, 0.5, -0.5, 1.0, -1.0)     # Coeficientes reais.
sigma2 = 2                              # Variância real.
q = length(beta)                        # Número de coeficientes.
real = list(beta, sigma2)               # Lista contendo valores reais.
names(real) = c("beta","sigma2")
```

O próximo passo é gerar as covariáveis. A primeira será binária (ex.: masculino = 1, feminino = 0). As demais serão contínuas e provenientes da distribuição $U(-1, 1)$.

```
x = array(1, c(n, q))                       # Matriz de 1's.
x[,2] = rbinom(n, 1, 0.5)                   # Covariável binária.
for(i in 3:q){ x[,i] = runif(n, -1, 1) }    # Covariáveis contínuas.
```

Uma vez definido os verdadeiros valores dos parâmetros e construída a matriz $\boldsymbol{X}$, seguimos com a geração da variável resposta.

```
y = numeric(n)                          # Vetor de tamanho n.
for(i in 1:n){
 err = rnorm(1, 0, sqrt(sigma2))        # Termo de erro.
 y[i] = x[i,] %*% beta + err            # Variável resposta.
}
```

O código a seguir cria um histograma para avaliar o comportamento da variável resposta.

```r
hist(y, 20, prob = TRUE, xlab = "variável resposta", ylab = "densidade",
     main = "", cex.lab = 1.5, cex.axis = 1.5, col = rgb(0.9,0.9,0.9))
lines(density(y), lwd = 2, col = "red")
```

Após gerar os dados, a próxima etapa é definir as distribuições *a priori* relativas aos parâmetros de interesse. Isso é discutido na próxima seção.

Especificações a priori

A informação inicial sobre $\beta = (\beta_0, \beta_1, \cdots, \beta_k)^\top$ e σ^2 serão expressas através das seguintes distribuições:

$$\beta \sim N_q(m_\beta, S_\beta) \qquad \text{e} \qquad \sigma^2 \sim GI(a_{\sigma^2}, b_{\sigma^2}).$$

Note que uma normal multivariada foi escolhida para o vetor β. Esta opção indica que os coeficientes possuem uma distribuição *a priori* conjunta e, eventualmente, o analista pode definir uma matriz de covariâncias S_β não diagonal. Tal estratégia estabeleceria uma dependência *a priori* entre estes parâmetros. Além de possibilitar a modelagem *a priori* de uma dependência entre os coeficientes, a atribuição de uma distribuição conjunta para o bloco β pode trazer alguns benefícios computacionais em termos de performance do MCMC (amostrar o bloco β pode ser melhor do que amostrar separadamente $\beta_0, \beta_1, \cdots, \beta_K$). Tenha em mente que este ganho pode ser pouco significativo (isto varia de modelo para modelo). Nada impede que o pesquisador escolha especificações *a priori* separadas (normais univariadas) para cada coeficiente. Outra maneira de indicar independência *a priori*, mantendo a amostragem em bloco através da normal multivariada, é assumir $S_\beta = s\, \boldsymbol{I}_q$ (com s escalar). Isto diz ao modelo que $\beta_j \sim N(m_{\beta i}, s)$ independentemente para $j = 0, 1, \cdots, K$. A escolha da normalidade para os coeficiente é apropriada, visto que estes parâmetros podem ser tanto positivos quanto negativos. Além disso a normal é simétrica e proporciona variados formatos através da definição de sua média e variância. No caso da variância σ^2, lembre-se que a sigla GI indica a distribuição Gama Inversa. Esta escolha foi feita tendo em vista a conjugação da Gama Inversa para a variância de um modelo Normal.

Os hiperparâmetros das distribuições *a priori* são:

```r
# Normal Multivariada.
m_beta = rep(0, q)      # Vetor de médias
S_beta = 10 * diag(q)   # Matriz de covariâncias
# Gama Inversa com E(sigma2) = 1 e Var(sigma2) = 10.
a_sigma2 = 2.1          # Parâmetro de escala.
b_sigma2 = 1.1          # Parâmetro de forma.
```

A próxima seção é dedicada a mostrar como os dados simulados e as informações *a priori* serão transmitidas ao `Stan`.

Transmitindo informações para o Stan

O primeiro passo é organizar os resultados amostrais (dados observados) em uma lista a ser lida pelo `Stan`. Esta lista vai incluir: tamanho amostral, dados e hiperparâmetros *a priori*. Um detalhe a ser ressaltado é que dentro da listagem escreve-se, por exemplo, "`n = n`". O `n` à esquerda do sinal

de igualdade refere-se à notação a ser utilizada no código `Stan` (veremos adiante). O `n` à direita do sinal de igualdade refere-se à nomenclatura adotada no ambiente do `R`. Recomenda-se que a notação no `R` e no `Stan` sejam as mesmas, isto evita erros de programação por confusão de notação.

```
data = list(n = n, q = q, y = y, x = x,
            m_beta = m_beta, S_beta = S_beta,
            a_sigma2 = a_sigma2, b_sigma2 = b_sigma2)
```

Outro aspecto que o usuário também deve indicar ao `Stan` são os nomes dos parâmetros que deseja salvar durante a execução do MCMC. A nomenclatura deve ser a mesma empregada no código `Stan`, por exemplo, se você escrever em maiúsculo `Sigma2` aqui e em minúsculo `sigma2` no código `Stan`, uma mensagem de erro será exibida quando for executar o MCMC. Em um modelo multidimensional (muitos parâmetros) podemos não estar interessados em salvar alguns parâmetros, então a lista abaixo será muito útil para escolher os elementos mais relevantes para análise.

```
# Lista requisitando que o vetor beta e o escalar sigma2 sejam salvos.
pars = c("beta", "sigma2")
```

Valores iniciais das cadeias MCMC podem ser informados pelo usuário ao `Stan` para que sejam usados na inicialização do MCMC (isto é opcional). Note que o objeto `R` denominado `init` (ver abaixo) é uma lista contendo 2 sub-listas. Cada sub-lista é relacionada a uma cadeia no cenário em que solicita-se ao `Stan` a construção de 2 ou mais cadeias para cada parâmetros (isto pode ser útil para avaliar convergência).

O `Stan` pode também inicializar o MCMC usando sementes aleatórias, para isso, simplesmente indique `init = "random"`. A semente do gerador de números aleatórios considerado pelo `Stan` pode ser especificada através do argumento `seed`; se `seed` é fixo, as mesmas sementes são usadas em cada execução do MCMC. O padrão (*default*) do programa é gerar aleatoriamente chutes iniciais entre (-2, 2) em um suporte irrestrito (uma esfera) e aplicar uma transformação que define o valor correspondente no suporte original do parâmetro. Usando o argumento adicional `init_r` (isto é opcional) pode-se escolher um valor diferente de 2 para os limites de geração no suporte irrestrito.

Uma última opção de chute inicial do MCMC é a especificação, permitida pelo `Stan`, dada pelo argumento `init = "0"`. Neste caso, todos os parâmetros serão inicializados em 0 dentro do suporte irrestrito (a esfera). Após uma transformação, estes zeros serão mapeados para um valor correspondente no suporte original do espaço paramétrico restrito.

```
# Lista de sementes de inicialização (admita 2 cadeias):
init = list()
init[[1]] = list(beta=rep(0,q),sigma2=1)
init[[2]] = list(beta=runif(q,-1,1),sigma2 = runif(1,0,1))

# Alternativamente, pode-se especificar:
# init = "random"
# init = "0"
```

Conforme já explicado no curso de Estatística Bayesiana, o `Stan` aplica o algorítmo No-U-Turn Sampling (NUTS) visando uma amostragem indireta mais eficiente da distribuição *a posteriori*. O NUTS é uma extensão do *Hamiltonian Monte Carlo* e enquadra-se na classe de métodos MCMC. Precisamos definir o número total de iterações a serem executadas pelo algorítmo e o tamanho do

período de aquecimento das cadeias. Considere os escolhas abaixo:

```
iter = 2000      # Total de iterações (incluindo burn-in).
warmup = 1000    # Número de iterações do burn-in.
chains = 2       # Número de cadeias do MCMC.
# chains = 1 (Stan retorna amostras a posteriori após burn-in da cadeia única).
# chains > 1 (Stan junta as amostras após burn-in de todas as cadeias).
```

Todos os códigos apresentados até aqui devem ser executados no ambiente do `R`. Eles são destinados a cumprir as etapas de: gerar dados sintéticos, especificar os hiperparâmetros das distribuições *a priori* e configurar o MCMC. O conteúdo da próxima caixa de códigos computacionais está escrito na linguagem do `Stan`. Este bloco não será executado diretamente nas linhas de comandos do `R`. O usuário deve salvar este conteúdo em um arquivo que chamaremos de `RegNormal.stan`. Veja que a extensão do arquivo é do tipo "`.stan`". O código abaixo contém dois itens cruciais para definir o modelo Bayesiano: a verossimilhança e as distribuições `a priori`. A linguagem de programação do `Stan` segue a mesma lógica do `C++`. Todos os objetos a serem utilizados nos cálculos devem ser obrigatoriamente declarados. Declarar um objeto significa informar (antes de fazer contas com ele) o tipo de estrutura (escalar, vetor, matriz) e o tipo de valor (inteiro, real, real positivo, etc) que ele receberá. Outro ponto importante é que todos os comandos devem finalizar com o símbolo ";" para sinalizar o seu fim.

```
// Bloco de declaração de dados.
// Declare aqui todos os objetos passados do R para o Stan.
// Estes objetos são aqueles dentro da listagem "data".
data{
  int<lower=1> n;
  int<lower=1> q;
  vector[n] y;
  matrix[n,q] x;
  vector[q] m_beta;
  matrix[q,q] S_beta;
  real<lower=0> a_sigma2;
  real<lower=0> b_sigma2;
}

// Bloco de declaração de parâmetros.
// Declare aqui todos os parâmetros para os quais
// uma distribuição a priori é especificada.
parameters{
  vector[q] beta;
  real<lower=0> sigma2;
}

// Bloco de parâmetros transformados.
// Se necessário, declare aqui novos parâmetros
// construídos como função daqueles
// declarados no bloco anterior.
```

```
transformed parameters{
  vector[n] mu;
  mu = x * beta;
}

// Bloco do modelo.
// Defina aqui a verossimilhança e as distribuições a priori.
model{
  // Verossimilhança
  for(i in 1:n){ y[i] ~ normal(mu[i], sqrt(sigma2)); }

  // Priori 1: Normal Multivariada com
  // vetor de médias e matriz de covariâncias.
  beta ~ multi_normal(m_beta, S_beta);

  // Priori 2: Gama Inversa.
  sigma2 ~ inv_gamma(a_sigma2, b_sigma2);

}
// Deixe vazia a última linha do arquivo ".stan" (isso evita "warnings").
```

Finalmente as informações chave a serem repassadas ao `Stan` estão devidamente organizadas. O comando a seguir deve ser utilizando no `R` para requisitar a execução do MCMC através do `Stan`. Note que o argumento `file` invoca o arquivo `RegNormal.stan` que contém (em código `Stan`) a estrutura principal do modelo Bayesiano (verossimilhança e distribuições *a priori*). A maneira como está escrito o comando abaixo supõe que o arquivo `RegNormal.stan` está salvo dentro da mesma pasta que o usuário definiu como diretório de trabalho do `R`. Digite no `R` o comando `getwd()` para identificar o seu diretório de trabalho.

```
output = stan(file = "RegNormal.stan", data = data,
              iter = iter, warmup = warmup, chains = chains,
              pars = pars, init = init, verbose = FALSE)
```

Explorando os resultados

Os códigos `R` exibidos nesta seção são destinados a desenvolver uma análise exploratória das amostras geradas para formar as cadeias de Markov após *burn-in*. O objeto `output`, salvo na área de tabalho do `R`, contém os resultados do ajuste Bayesiano. Este objeto é da classe `stanfit`, o qual é um formato particular estabelecido pelo `rstan` para guardar informações de saída do NUTS.

Iniciamos com um sumário descritivo, através da função `print`, que será aplicado diretamente ao objeto `output` do tipo `stanfit`. As estatísticas serão calculadas para as amostras coletadas após o período de aquecimento das cadeias. Visto que solicitou-se 2 cadeias para cada parâmetro, o sumário é baseado na junção das duas amostras. A tabela resultante contém as seguintes estimativas *a posteriori*: média (`mean`), desvio padrão (`sd`), percentis (2.5% a 97.5%), tamanho efetivo da amostra (`n_eff`) e estatística $\hat{R}$ (`Rhat`). O `n_eff` é uma medida do nível de autocorrelação presente na cadeia de Markov. Autocorrelação forte indica dependência sequencial entre observações geradas

em iterações vizinhas durante a execução do MCMC. Isto implica que a amostra coletada não se assemelha a uma amostra aleatória, possuindo menos informação (tamanho efetivo menor). O ideal é ter valores próximos ao número total de iterações MCMC. Neste exemplo, adotou-se `iter = 2000` iterações formando 2 cadeias para cada parâmetro, então o tamanho efetivo deve ser comparado a 4000. A estatística $\hat{R}$ (Gelman-Rubin) mede o grau de convergência da cadeia de Markov tomando como base a estabilidade dos resultados dentro e entre cadeias. Valor perto de 1 indicam convergência para a distribuição alvo. Valor maior do que 1.1 sugere ao pesquisador que ele deve fazer uma análise visual da cadeia para verificar o comportamento de estabilização.

```
# Sumário global do objeto stanfit.
print(output, pars = c("beta", "sigma2"))
# Sumário focado em beta2, beta3 e sigma2.
print(output, pars = c(paste0("beta[",c(2,3),"]"), "sigma2"))
```

O próximo comando fornece o gráfico sequencial das cadeias de Markov ao longo das iterações e após *burn-in*. Este tipo de gráfico é geralmente chamado de *traceplot* e serve para fazer a inspeção visual de convergência. As duas cadeias solicitadas ao `Stan` aparecerão sobrepostas e com cores diferentes.

```
traceplot(output, pars = c("beta", "sigma2"))
```

Para realizar uma análise inferencial mais aprofundada, precisaremos extrair do objeto `output` (tipo `stanfit`) a matriz contendo nas colunas a cadeia de Markov gerada para cada parâmetro. Isto é feito através do comando a seguir.

```
# Extração em formato de lista;
# beta e sigma2 separados na lista (um é matriz, o outro é vetor).
samp = extract(output)

# Extração alternativa em formato de matriz;
# beta e sigma2 juntos na mesma matriz.
# samp = as.matrix(output)
```

A próxima caixa de comandos cria uma curva que acompanha o formato dos histogramas obtidos a partir das amostras *a posteriori* para β_0 e σ^2. O valor real destes parâmetros é identificado através de uma linha vertical vermelha.

```
par(mfrow = c(1,2))

{ plot( density(samp$beta[,1]), cex.lab = 1.5, cex.axis = 1.5, lwd = 2,
      main = "Densidade a posteriori de beta0", col = "blue" )
abline( v = real$beta[1], lwd = 2, col = "red" ) }

{ plot( density(samp$sigma2), cex.lab = 1.5, cex.axis = 1.5, lwd = 2,
      main = "Densidade a posteriori de sigma2", col = "blue")
abline( v = real$sigma2, lwd = 2, col = "red" ) }
```

A próxima caixa mostra como os *traceplots* das cadeias podem ser alternativamente construídos a partir das matrizes extraídas do objeto `stanfit`. Destaca-se que o gráfico da cadeia exibido neste resultado é uma junção das duas cadeias solicitadas ao `Stan`.

```r
par(mfrow = c(1,2))

{ plot( samp$beta[,1], type = "l", cex.lab = 1.5, cex.axis = 1.5,
        xlab = "iterações", ylab = "beta0",
        main = "Traceplot de beta0", col = "blue" )
abline( h = real$beta[1], lwd = 2, col = "red" ) }

{ plot( samp$sigma2, type = "l", cex.lab = 1.5, cex.axis = 1.5,
        xlab = "iterações", ylab = "sigma2",
        main = "Traceplot de sigma2", col = "blue")
abline( h = real$sigma2, lwd = 2, col = "red" ) }
```

O código a seguir é destinado a criar uma tabela sumarizadora com os principais resultados de inferência pontual e intervalar *a posteriori*. Iremos incluir os intervalos de credibilidade HPD de 95% obtidos por meio do pacote `coda`.

```r
require(coda)

aux = cbind( samp$beta, samp$sigma2 )
me = apply(aux, 2, mean)     # média
md = apply(aux, 2, median)   # mediana
sd = apply(aux, 2, sd)       # desvio padrão
aux = as.mcmc(aux)
hpd = HPDinterval(aux)
tab = cbind(unlist(real), me, md, sd, hpd[,"lower"], hpd[,"upper"])
rownames(tab) = c( paste0("beta",0:(q-1)),"sigma2")
colnames(tab) = c("true", "mean", "median", "s.d.", "HPD_inf", "HPD_sup")
round(tab,4) # mostrar saída com 4 casas decimais.
```

Exercício

Reveja os slides das aulas e volte aos exemplos, nos quais implementou-se o *Metropolis-Hastings* e o *Gibbs Sampling*, para amostragem indireta *a posteriori*, relativos ao modelo $Y_i \sim N(\mu, 1/\phi)$, com média μ e precisão ϕ desconhecidos. Lembre-se que a amostra $Y_1, \cdots, Y_n$ foi simulada (adote os mesmo valores reais) e admitiu-se independência *a priori* entre μ e ϕ. Além disso, assuma independência condicional de $Y_1, \cdots, Y_n$ dado $\{\mu, \phi\}$. Siga cada passo tratado no presente material, sobre regressão linear múltipla via `Stan`, para implementar o seu código que irá executar o NUTS visando estimar μ e ϕ com uma amostra de tamanho $n = 50$. Mostre gráficos, tabelas e comentários sobre o resultado final.