贝叶斯统计

马尔可夫链蒙特卡洛算法和Stan语言

Markov Chain Monte Carlo and Stan

第三天

- Stan语言
  - 11am – 12:30pm, 20 mins break

- 层次模型
  - 12:50pm – 3pm

其他 MCMC 算法

- 自己写的 MCMC 代码通常依赖于吉布斯采样算法、Metropolis 算法、切片采样（slice sampling）
- 这些模型的代码写起来都非常耗时，并且涉及繁琐的数学运算
- 对模型的小调整都需要很长的时间去重新写代码

概率编程 （Probabilistic Programming Languages）

- 概率编程指用户通过编程语言，指定和估计概率模型 （例如贝叶斯模型）
- 一些较老的软件（BUGS，JAGS），主要使用吉布斯采样算法 和 Metropolis Hastings算法
- 较新的软件大多使用哈密尔顿蒙特卡洛 （Hamiltonian Monte Carlo）算法
  - 此算法的灵感来自物理学。因为其较为复杂，所以我们将在本课程中不做介绍

Stan编程语言

- 使用 C++ 语言
  - 这有助于代码快速运行!
- 它的API适用于多种语言，包括 R、 Stata、 Python 等
- 允许用户从各种各样的概率分布中进行选择，以制定灵活的先验分布和似然函数
- 加速建模的过程

Stanislaw Ulam
Physicist
1909-1984

Stan的一个例子

$$y_i \overset{i.i.d.}{\sim} Normal(\mu, \sigma^2)\ i = 1, \ldots, 1000$$

$$\mu \sim Normal(\mu_0 = 100, \sigma_0^2 = 10\ )$$

$$\sigma^2 \sim Inverse\ Gamma(a = .01, b = .01)$$

```
data {
  int<lower=0> N;
  array[N] real y;
}
parameters {
  real mu;
  real<lower=0> sigmasq;
}
transformed parameters {
    real<lower=0> sigma;
    sigma = sqrt(sigmasq);
}
model {
  y ~ normal(mu, sigma);
  mu ~ normal(100, sqrt(10));
  sigmasq ~ inv_gamma(0.01, 0.01);
}
generated quantities {
  array[N] real sample_dataset;
  for (i in 1:N) {
      sample_dataset[i] = normal_rng(mu, sigma);
  }
}
```

Stan 使用标准差而不是方差作为正态分布的参数

从后验分布中抽取样本，
以进行后验预测检查

R 演示

有效样本量 （Effective Sample Size）

	mean	se_mean	sd	2.5%	5%	50%	95%	97.5%	n_eff	Rhat
mu	104.99	0	0.04	104.91	104.93	104.99	105.05	105.06	3684	1
sigmasq	1.43	0	0.06	1.31	1.33	1.42	1.53	1.56	3209	1

- N_eff：有效样本量（effective sample size）
- 我们生成了 4000 个自相关的样本
- 4000 个自相关样本大致相当于 3684 个独立样本

Rhat （Gelman-Rubin 检验）

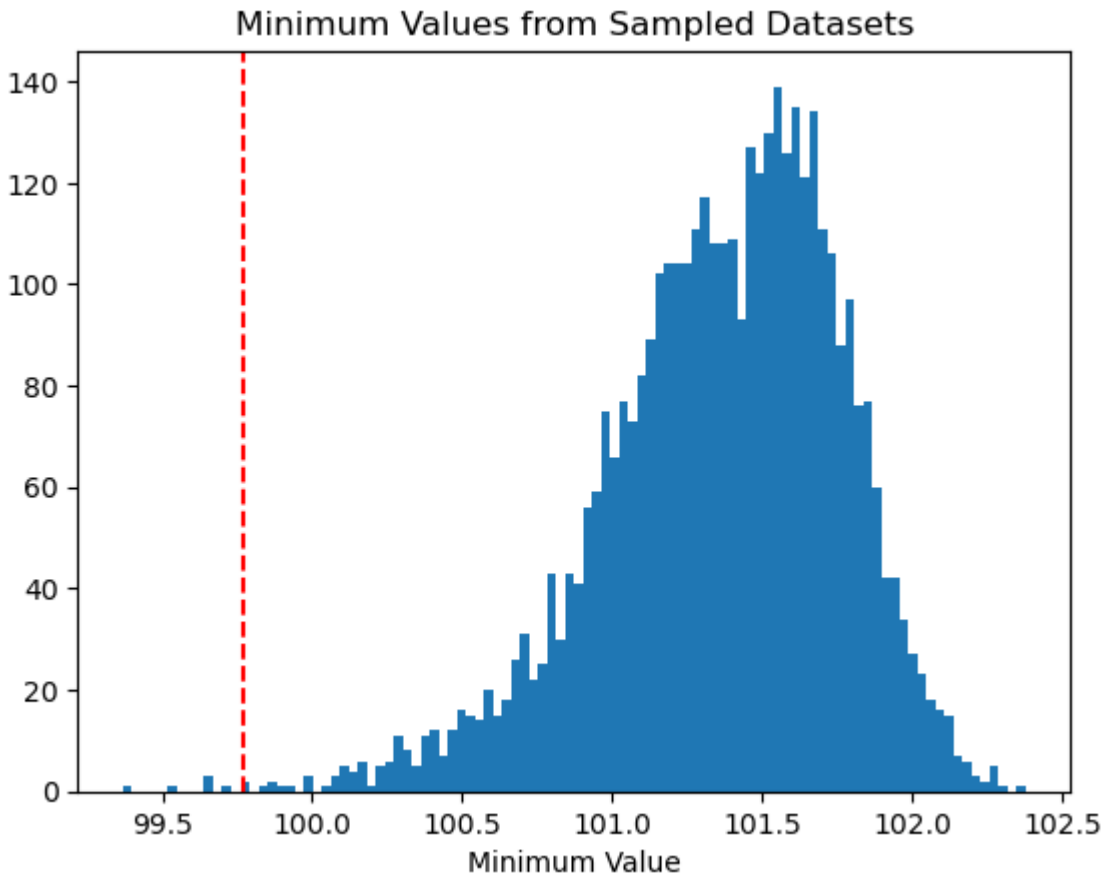
	mean	se_mean	sd	2.5%	5%	50%	95%	97.5%	n_eff	Rhat
mu	104.99	0	0.04	104.91	104.93	104.99	105.05	105.06	3684	1
sigmasq	1.43	0	0.06	1.31	1.33	1.42	1.53	1.56	3209	1

- 运行行多条MCMC链并进行比较
  - 如果链与链之间非常不同，这就是没有收敛的证据

- 当Rhat近于时1代表收敛，且不应大于1.05
- Rhat值越大，说明MCMC链没有收敛
  - 但低值并不能确保收敛！需要进行其他检验，例如迹线图

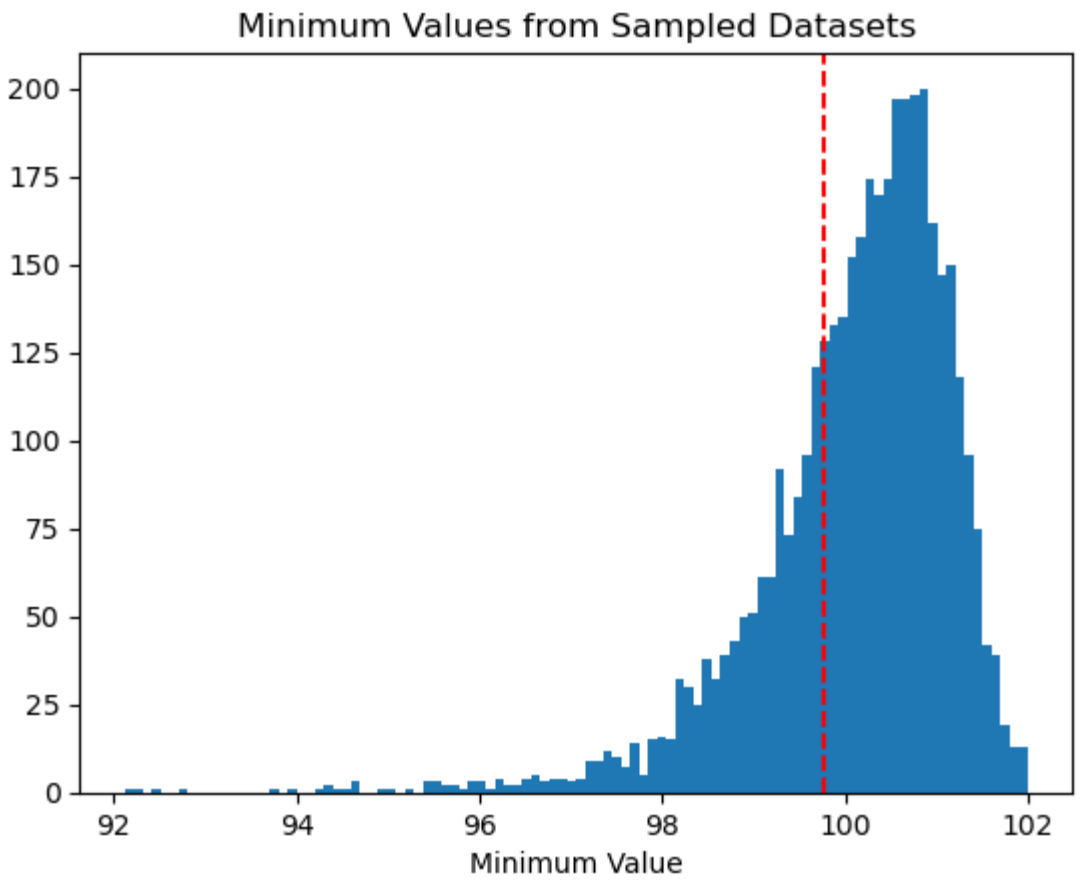
回顾后验预测检查

- 观测到的数据的最小值是 99.8
- 这与我们复制的数据一致吗?
- 我们的 4,000 次复制中只有 7次的最小值在99.8左右



在Stan里可以简便的更改似然函数

```
data {
  int<lower=0> N;
  vector[N] y;
}
parameters {
  real mu;
  real<lower=0> sigmasq;
}
transformed parameters {
  real<lower=0> sigma;
  sigma = sqrt(sigmasq);
}
model {
  y ~ student_t(8, mu, sigma);
  mu ~ normal(100, sqrt(10));
  sigmasq ~ inv_gamma(0.01, 0.01);
}
generated quantities {
  vector[N] sample_dataset;
  for (i in 1:N) {
      sample_dataset[i] = student_t_rng(8, mu, sigma);
  }
}
```



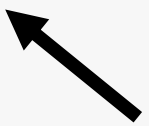
共轭先验分布不再是必须

- 由于 Stan 不使用 吉布斯抽样，因此共轭先验分布不再是必须

- 早些时候，我们提到使用逆伽马先验分布可能存在问题
- 用截断正态分布先验分布（truncated normal distribution）作为先验分布可能更加合理

- 一些建议: https://github.com/stan-dev/stan/wiki/Prior-Choice-Recommendations

```
data {
  int<lower=0> N;
  vector[N] y;
}
parameters {
  real mu;
  real<lower=0> sigma;
}
model {
  y ~ student_t(8, mu, sigma);
  mu ~ normal(100, sqrt(10));
  sigma ~ normal(0, 100);
}
generated quantities {
  vector[N] sample_dataset;
  for (i in 1:N) {
      sample_dataset[i] = student_t_rng(8, mu, sigma);
  }
}
```



例子

- 线性回归
- $\boldsymbol{y} \sim Normal(\alpha + \boldsymbol{x}'\boldsymbol{\beta}, \sigma^2)$
- 如果用户没有指定先验分布，在没有先验的情况下，Stan 将使用默认的非正常先验分布（improper prior）
- https://mc-stan.org/docs/stan-users-guide/linear-regression.html

```
data {
  int<lower=0> N;   // number of data items
  int<lower=0> K;   // number of predictors
  matrix[N, K] x;   // predictor matrix
  vector[N] y;      // outcome vector
}
parameters {
  real alpha;           // intercept
  vector[K] beta;       // coefficients for predictors
  real<lower=0> sigma;  // error scale
}
model {
  y ~ normal(x * beta + alpha, sigma);  // likelihood
}
```

例子

- 逻辑回归
- $\Pr(y=1)=\frac{1}{1+\exp(-(\alpha+\beta x))}$
- https://mc-stan.org/docs/stan-users-guide/logistic-probit-regression.html

```
data {
  int<lower=0> N;
  vector[N] x;
  array[N] int<lower=0, upper=1> y;
}
parameters {
  real alpha;
  real beta;
}
model {
  y ~ bernoulli_logit(alpha + beta * x);
}
```

例子

- AR(1) 时间序列模型
- $y_t \sim Normal(\alpha + \beta y_{t-1}, \sigma^2)$
- https://mc-stan.org/docs/stan-users-guide/autoregressive.html

```
data {
  int<lower=0> N;
  vector[N] y;
}
parameters {
  real alpha;
  real beta;
  real<lower=0> sigma;
}
model {
  for (n in 2:N) {
    y[n] ~ normal(alpha + beta * y[n-1], sigma);
  }
}
```

关于使用Stan的一些小贴士

- 不用再把目光限制在共轭先验分布上

- 缩放你的数据和先验分布，使它们的均值为0，标准差为1
  - 使用“transformed parameters”
  - 无信息先验分布可能会减慢代码运行的速度
  - https://mc-stan.org/docs/stan-users-guide/standardizing-predictors-and-outputs.html

- 查阅Stan的说明书！里面提供了很好的贴士和示例

关于Stan的结语

- Stan使贝叶斯建模变得容易
- Hamiltonian MCMC 算法使它比吉布斯抽样更加高效
- C++ 语言使代码运行速度更快，但也增加了它的复杂性
  - E.g. 很难在 Windows 上安装 PyStan
  - 报错的原因更难理解
- 可能难以处理某些类型的模型 (e.g. 离散参数 discrete parameters)
  - 有变通的办法，但都很复杂
- 在大型数据上运行可能会非常慢
  - 有时重参数化（reparameterization）或缩放（scaling）可以提供帮助