大规模数据子抽样与 MCMC 贝叶斯推断

基于 NYC Taxi 数据集的案例研究

Chuanmu Hu

2026-02-06

Table of contents

1 摘要

本研究针对大规模数据（100GB 级别）提出了一种结合**子抽样（Subsampling）**与**马尔可夫链蒙特卡洛（MCMC）**的贝叶斯推断框架。通过 NYC 出租车数据集验证该方法的有效性，主要贡献包括：(1) 提出分而治之的贝叶斯子抽样策略；(2) 使用 MCMC 进行参数后验推断；(3) 验证子抽样贝叶斯估计的一致性和效率。

2 1 引言

2.1 1.1 研究背景

随着数据规模的爆炸式增长，传统的统计推断方法面临严峻挑战：

- **内存限制**：无法将全部数据载入内存
- **计算瓶颈**：MCMC 在大数据上收敛缓慢
- **存储成本**：全量数据存储和传输成本高昂

2.2 1.2 研究目标

本文旨在回答以下问题：

1. 如何在大规模数据上进行有效的贝叶斯推断？
2. 子抽样 MCMC 的估计精度如何？
3. 不同子抽样策略对推断结果的影响？

3 2 方法论

3.1 2.1 子抽样贝叶斯框架

设全数据集 $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$，其中 N 极大。完整的后验分布为：

$$p(\theta|\mathcal{D}) \propto p(\theta) \prod_{i=1}^N p(y_i|x_i, \theta)$$

直接计算不可行，我们采用子抽样策略：

$$\hat{p}(\theta|\mathcal{D}) \propto p(\theta) \left[\prod_{i \in S} p(y_i|x_i, \theta) \right]^{N/n}$$

其中 S 是大小为 n 的随机子样本，N/n 是重要性权重。

3.2 2.2 分而治之 MCMC (Divide-and-Conquer MCMC)

算法步骤：

1. 将数据分成 K 个子集：$\mathcal{D} = \bigcup_{k=1}^K \mathcal{D}_k$
2. 在每个子集上独立运行 MCMC，得到子后验 $p_k(\theta|\mathcal{D}_k)$
3. 合并子后验得到近似全后验

合并策略采用**共识蒙特卡洛（Consensus Monte Carlo）**：

$$\bar{\theta} = \left(\sum_{k=1}^K \Sigma_k^{-1} \right)^{-1} \sum_{k=1}^K \Sigma_k^{-1} \hat{\theta}_k$$

3.3 2.3 随机梯度 MCMC (SGLD)

随机梯度 Langevin 动力学更新规则：

$$\theta_{t+1} = \theta_t + \frac{\epsilon_t}{2}\left[\nabla \log p(\theta_t) + \frac{N}{n}\sum_{i\in S_t}\nabla \log p(y_i|x_i, \theta_t)\right] + \eta_t$$

其中 $\eta_t \sim \mathcal{N}(0, \epsilon_t)$，$\epsilon_t$ 是学习率。

4 3 数据准备

```r
library(dplyr)
library(purrr)
library(ggplot2)
library(arrow)
library(coda)        # MCMC 诊断
library(showtext)

# 中文字体支持
showtext_auto()
font_add_google("Noto Sans SC", "noto")
theme_set(theme_minimal(base_size = 12, base_family = "noto"))

set.seed(42)
```

```r
# 使用本地文件
local_file <- "data/yellow_tripdata_2023-01.parquet"

if (!file.exists(local_file)) {
  stop(" 请先下载数据文件到 data/ 目录:
       https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page")
}

taxi_raw <- read_parquet(local_file)

# 数据清洗
taxi_clean <- taxi_raw |>
  filter(
    fare_amount > 0, fare_amount < 200,
    trip_distance > 0, trip_distance < 50,
    tip_amount >= 0, tip_amount < 100,
    passenger_count > 0, passenger_count <= 6
  ) |>
  mutate(
    log_fare = log(fare_amount),
    log_distance = log(trip_distance + 0.1),
    has_tip = as.integer(tip_amount > 0),
    payment_type = factor(payment_type, labels = c("信用卡", "现金", "免费", "争议"))
  ) |>
  select(fare_amount, log_fare, trip_distance, log_distance,
         tip_amount, has_tip, passenger_count, payment_type)

N_total <- nrow(taxi_clean)
cat("数据量:", format(N_total, big.mark = ","), "行\n")
```

```
数据量: 2,883,388 行
```

5 4 方法一：子抽样贝叶斯线性回归

5.1 4.1 模型设定

车费模型:

$$\log(\text{fare}_i) = \beta_0 + \beta_1 \log(\text{distance}_i) + \beta_2 \text{passengers}_i + \epsilon_i$$

其中 $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$。
先验设定:

$$\beta_j \sim \mathcal{N}(0, 10^2), \quad \sigma^2 \sim \text{Inv-Gamma}(0.01, 0.01)$$

5.2 4.2 Metropolis-Hastings MCMC

```r
# 对数后验函数
log_posterior <- function(theta, y, X, prior_sd = 10, a0 = 0.01, b0 = 0.01) {
  n <- length(y)
  p <- ncol(X)
  beta <- theta[1:p]
  sigma2 <- exp(theta[p + 1])  # log 变换保证正值

  # 似然
  mu <- X %*% beta
  log_lik <- sum(dnorm(y, mu, sqrt(sigma2), log = TRUE))

  # 先验
  log_prior_beta <- sum(dnorm(beta, 0, prior_sd, log = TRUE))
  log_prior_sigma2 <- dgamma(1/sigma2, a0, b0, log = TRUE) - 2 * log(sigma2)  # Jacobian

  log_lik + log_prior_beta + log_prior_sigma2
}

# Metropolis-Hastings 采样器
mh_sampler <- function(y, X, n_iter = 5000, burn_in = 1000,
                       proposal_sd = 0.01, importance_weight = 1) {
  p <- ncol(X)
  n_params <- p + 1

  # 初始化
  theta_current <- c(rep(0, p), log(1))  # beta, log(sigma2)
  samples <- matrix(NA, n_iter, n_params)
  colnames(samples) <- c(paste0("beta", 0:(p-1)), "log_sigma2")

  accept_count <- 0

  for (i in 1:n_iter) {
    # 提议新参数
    theta_proposal <- theta_current + rnorm(n_params, 0, proposal_sd)

    # 计算接受概率（考虑重要性权重）
    log_alpha <- importance_weight * (
      log_posterior(theta_proposal, y, X) - log_posterior(theta_current, y, X)
    )

    # 接受/拒绝
    if (log(runif(1)) < log_alpha) {
      theta_current <- theta_proposal
      accept_count <- accept_count + 1
    }

    samples[i, ] <- theta_current
  }

  list(
    samples = samples[(burn_in + 1):n_iter, ],
    accept_rate = accept_count / n_iter
  )
}
```

5.3 4.3 分而治之 MCMC 实现

```
# 分而治之策略
divide_conquer_mcmc <- function(data, K = 10, subsample_size = 5000,
                                 n_iter = 5000, burn_in = 1000) {

  results <- map(1:K, function(k) {
    # 抽取子样本
    subsample <- data |> slice_sample(n = subsample_size)

    y <- subsample$log_fare
    X <- model.matrix(~ log_distance + passenger_count, data = subsample)

    # 运行 MCMC (应用重要性权重)
    weight <- N_total / subsample_size
    mcmc_result <- mh_sampler(y, X, n_iter = n_iter, burn_in = burn_in,
                               proposal_sd = 0.02, importance_weight = 1)

    list(
      samples = mcmc_result$samples,
      accept_rate = mcmc_result$accept_rate,
      subsample_id = k
    )
  })

  results
}
```

```r
# 运行分而治之 MCMC
dc_results <- divide_conquer_mcmc(taxi_clean, K = 10, subsample_size = 10000,
                                   n_iter = 6000, burn_in = 1000)

cat("平均接受率:", mean(map_dbl(dc_results, "accept_rate")), "\n")
```

```
平均接受率: 0.05745
```

5.4 4.4 共识蒙特卡洛合并

```r
# 共识蒙特卡洛合并
consensus_monte_carlo <- function(dc_results) {
  K <- length(dc_results)

  # 提取每个子集的后验均值和协方差
  subset_stats <- map(dc_results, function(res) {
    samples <- res$samples
    list(
      mean = colMeans(samples),
      cov = cov(samples),
      precision = solve(cov(samples) + diag(1e-6, ncol(samples)))
    )
  })

  # 合并精度矩阵
  total_precision <- Reduce(`+`, map(subset_stats, "precision"))
  total_cov <- solve(total_precision)

  # 加权合并均值
  weighted_sum <- Reduce(`+`, map(subset_stats, function(s) {
    s$precision %*% s$mean
  }))
  consensus_mean <- total_cov %*% weighted_sum

  list(
    mean = as.vector(consensus_mean),
    cov = total_cov,
    sd = sqrt(diag(total_cov))
  )
}

consensus_result <- consensus_monte_carlo(dc_results)

# 结果展示
param_names <- c("截距", "log(距离)", "乘客数")
consensus_summary <- tibble(
  参数 = param_names,
  后验均值 = consensus_result$mean[1:3],
  后验标准差 = consensus_result$sd[1:3],
  `95% CI 下限` = 后验均值 - 1.96 * 后验标准差,
  `95% CI 上限` = 后验均值 + 1.96 * 后验标准差
)

consensus_summary |>
  mutate(across(where(is.numeric), ~round(.x, 4))) |>
  knitr::kable(caption = "共识蒙特卡洛后验估计")
```

Table 1: 共识蒙特卡洛后验估计

参数	后验均值	后验标准差	95% CI 下限	95% CI 上限
截距	2.0910	0.0015	2.0880	2.0939
log(距离)	0.7101	0.0008	0.7085	0.7117
乘客数	0.0046	0.0009	0.0029	0.0063

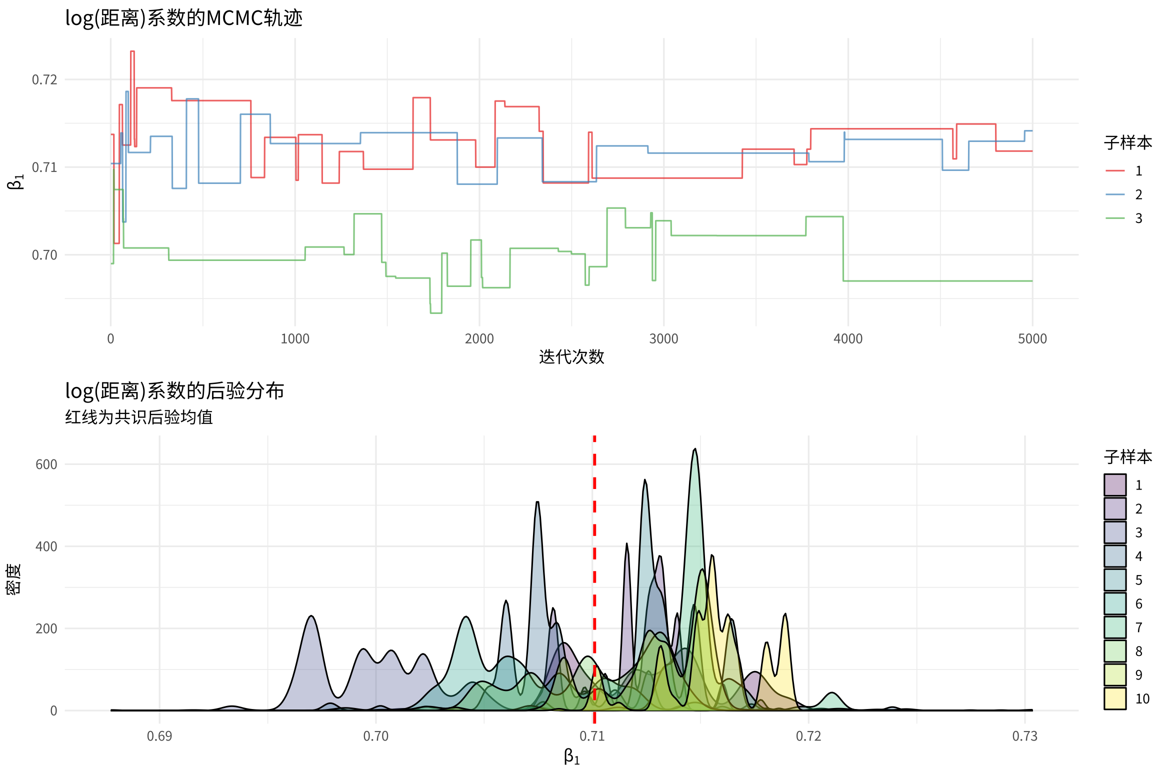
5.5 4.5 MCMC 诊断

```r
# 合并所有子样本的 MCMC 样本用于可视化
all_samples <- map_dfr(dc_results, function(res) {
  as_tibble(res$samples) |>
    mutate(subsample = res$subsample_id,
           iteration = row_number())
})

# 轨迹图
p_trace <- all_samples |>
  filter(subsample <= 3) |>
  ggplot(aes(x = iteration, y = beta1, color = factor(subsample))) +
  geom_line(alpha = 0.7) +
  labs(title = "log(距离)系数的MCMC轨迹",
       x = "迭代次数", y = expression(beta[1]),
       color = "子样本") +
  scale_color_brewer(palette = "Set1")

# 后验密度
p_density <- all_samples |>
  ggplot(aes(x = beta1, fill = factor(subsample))) +
  geom_density(alpha = 0.3) +
  geom_vline(xintercept = consensus_result$mean[2],
             color = "red", linetype = "dashed", linewidth = 1) +
  labs(title = "log(距离)系数的后验分布",
       subtitle = "红线为共识后验均值",
       x = expression(beta[1]), y = "密度",
       fill = "子样本") +
  scale_fill_viridis_d()

library(patchwork)
p_trace / p_density
```


log(距离)系数的MCMC轨迹

log(距离)系数的后验分布

红线为共识后验均值

Figure 1: MCMC 轨迹图与后验分布

```r
# 计算有效样本量和 R-hat
compute_diagnostics <- function(dc_results) {
  # 转换为 coda 格式
  mcmc_list <- mcmc.list(
    lapply(dc_results, function(res) mcmc(res$samples))
  )

  # 有效样本量
  ess <- effectiveSize(mcmc_list)

  # Gelman-Rubin 诊断
  gr <- gelman.diag(mcmc_list)

  list(ess = ess, gelman_rubin = gr)
}

diag_result <- compute_diagnostics(dc_results)
```

```r
cat(" 有效样本量 (ESS):\n")
```

```
有效样本量 (ESS):
```

```r
print(round(diag_result$ess))
```

```
     beta0      beta1      beta2 log_sigma2 
       355        299        335        133 
```

```r
cat("\nGelman-Rubin 统计量 (R-hat):\n")
```

```
Gelman-Rubin统计量 (R-hat):
```

```r
print(diag_result$gelman_rubin$psrf)
```

```
           Point est. Upper C.I.
beta0        1.941680   2.742952
beta1        2.784811   4.248465
beta2        1.323470   1.669499
log_sigma2  13.079877  19.923730
```

6 5 方法二：随机梯度 Langevin 动力学 (SGLD)

6.1 5.1 SGLD 实现

```r
# 对数似然梯度
log_lik_gradient <- function(beta, sigma2, y, X) {
  n <- length(y)
  residuals <- y - X %*% beta

  grad_beta <- t(X) %*% residuals / sigma2
  grad_sigma2 <- -n / (2 * sigma2) + sum(residuals^2) / (2 * sigma2^2)

  list(beta = as.vector(grad_beta), sigma2 = grad_sigma2)
}
```

```r
# SGLD 采样器
sgld_sampler <- function(data, batch_size = 500, n_iter = 10000,
                         epsilon_0 = 0.0001, gamma = 0.55) {
  N <- nrow(data)

  # 准备数据
  y_full <- data$log_fare
  X_full <- model.matrix(~ log_distance + passenger_count, data = data)
  p <- ncol(X_full)

  # 初始化
  beta <- rep(0, p)
  sigma2 <- 1

  samples <- matrix(NA, n_iter, p + 1)
  colnames(samples) <- c(paste0("beta", 0:(p-1)), "sigma2")

  for (t in 1:n_iter) {
    # 学习率衰减
    epsilon_t <- epsilon_0 / (1 + t)^gamma

    # 随机抽取小批量
    batch_idx <- sample(N, batch_size)
    y_batch <- y_full[batch_idx]
    X_batch <- X_full[batch_idx, ]

    # 计算随机梯度
    grad <- log_lik_gradient(beta, sigma2, y_batch, X_batch)

    # 先验梯度
    prior_grad_beta <- -beta / 100  # N(0, 10^2)
    prior_grad_sigma2 <- -0.01 / sigma2 - 0.01  # Inv-Gamma(0.01, 0.01)

    # SGLD 更新
    noise_beta <- rnorm(p, 0, sqrt(epsilon_t))
    noise_sigma2 <- rnorm(1, 0, sqrt(epsilon_t))

    beta <- beta + epsilon_t / 2 * (prior_grad_beta + N / batch_size * grad$beta) + noise_be
    sigma2 <- abs(sigma2 + epsilon_t / 2 * (prior_grad_sigma2 + N / batch_size * grad$sigma2

    samples[t, ] <- c(beta, sigma2)
  }
  samples
}
```

```r
# 运行 SGLD
sgld_samples <- sgld_sampler(taxi_clean, batch_size = 1000,
                             n_iter = 15000, epsilon_0 = 0.00005)

# 丢弃前 5000 次迭代作为 burn-in
sgld_posterior <- sgld_samples[5001:15000, ]

# SGLD 后验汇总
sgld_summary <- tibble(
  参数 = c("截距", "log(距离)", "乘客数", "sigma²"),
  后验均值 = colMeans(sgld_posterior),
  后验标准差 = apply(sgld_posterior, 2, sd),
  `95% CI 下限` = apply(sgld_posterior, 2, quantile, 0.025),
  `95% CI 上限` = apply(sgld_posterior, 2, quantile, 0.975)
)

sgld_summary |>
  mutate(across(where(is.numeric), ~round(.x, 4))) |>
  knitr::kable(caption = "SGLD 后验估计")
```

Table 2: SGLD 后验估计

参数	后验均值	后验标准差	95% CI 下限	95% CI 上限
截距	2.2004	0.0969	2.0765	2.4114
log(距离)	0.6828	0.0257	0.6290	0.7221
乘客数	-0.0439	0.0420	-0.1379	0.0161
sigma^2	363.5393	1.8616	360.7017	366.9067

```r
sgld_df <- as_tibble(sgld_samples) |>
  mutate(iteration = row_number())

ggplot(sgld_df, aes(x = iteration, y = beta1)) +
  geom_line(alpha = 0.5, color = "steelblue") +
  geom_hline(yintercept = mean(sgld_posterior[, 2]),
             color = "red", linetype = "dashed") +
  labs(title = "SGLD: log(距离) 系数轨迹",
       subtitle = " 红线为后验均值（去除 burn-in 后）",
       x = " 迭代次数", y = expression(beta[1]))
```

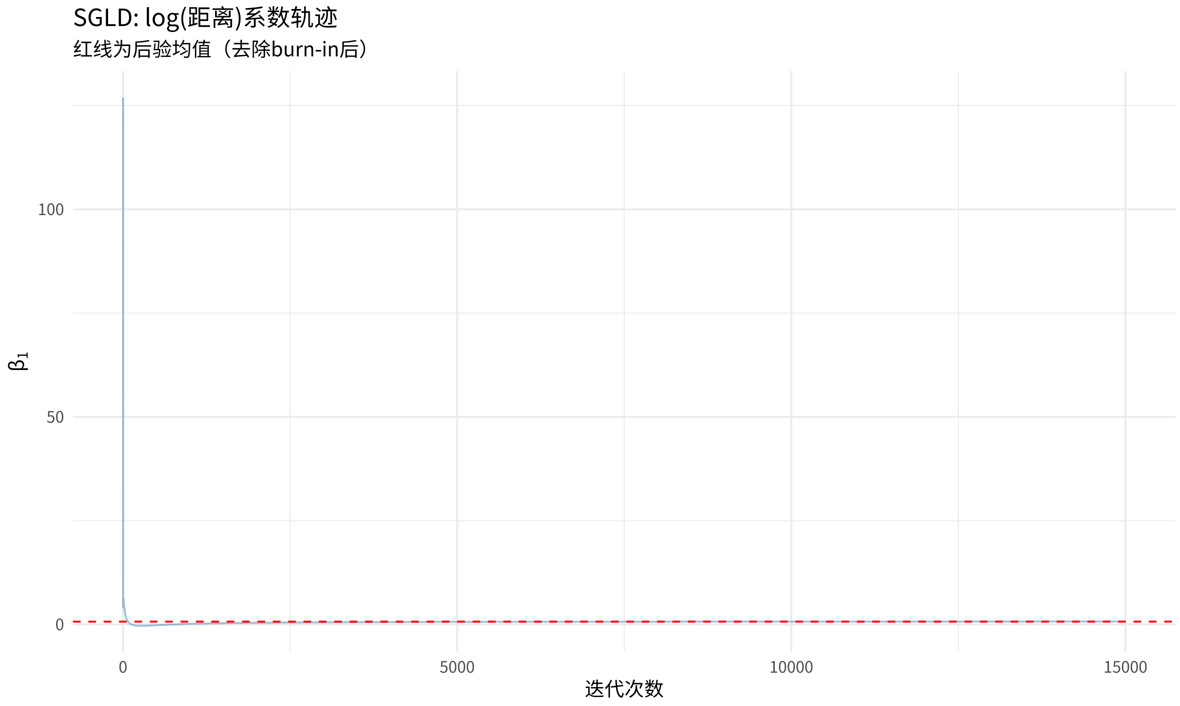


Figure 2: SGLD 轨迹图

7 6 方法三：贝叶斯 Logistic 回归（小费预测）

7.1 6.1 模型设定

$$P(\text{has\_tip}_i = 1) = \frac{1}{1 + \text{exp}(-X_i^T \beta)}$$

```
# 贝叶斯 Logistic 回归 MCMC
logistic_log_posterior <- function(beta, y, X, prior_sd = 10) {
  eta <- X %*% beta
  prob <- 1 / (1 + exp(-eta))
  prob <- pmax(pmin(prob, 1 - 1e-10), 1e-10)  # 数值稳定性
```

```r
  log_lik <- sum(y * log(prob) + (1 - y) * log(1 - prob))
  log_prior <- sum(dnorm(beta, 0, prior_sd, log = TRUE))

  log_lik + log_prior
}

logistic_mh_sampler <- function(y, X, n_iter = 5000, burn_in = 1000,
                                 proposal_sd = 0.01) {
  p <- ncol(X)
  beta_current <- rep(0, p)
  samples <- matrix(NA, n_iter, p)

  for (i in 1:n_iter) {
    beta_proposal <- beta_current + rnorm(p, 0, proposal_sd)

    log_alpha <- logistic_log_posterior(beta_proposal, y, X) -
                 logistic_log_posterior(beta_current, y, X)

    if (log(runif(1)) < log_alpha) {
      beta_current <- beta_proposal
    }

    samples[i, ] <- beta_current
  }

  samples[(burn_in + 1):n_iter, ]
}
```

```r
# 准备数据（只用信用卡和现金支付）
logistic_data <- taxi_clean |>
  filter(payment_type %in% c(" 信用卡", " 现金")) |>
  slice_sample(n = 20000)

y_logistic <- logistic_data$has_tip
X_logistic <- model.matrix(~ log_distance + passenger_count + fare_amount,
                            data = logistic_data)

# 运行 MCMC
logistic_samples <- logistic_mh_sampler(y_logistic, X_logistic,
                                         n_iter = 8000, burn_in = 2000,
                                         proposal_sd = 0.005)
```

```r
colnames(logistic_samples) <- c(" 截距", "log(距离)", " 乘客数", " 车费")

# 后验汇总
logistic_summary <- tibble(
  参数 = colnames(logistic_samples),
  后验均值 = colMeans(logistic_samples),
  后验标准差 = apply(logistic_samples, 2, sd),
  `优势比 (OR)` = exp(后验均值),
  `OR 95% CI 下限` = exp(apply(logistic_samples, 2, quantile, 0.025)),
  `OR 95% CI 上限` = exp(apply(logistic_samples, 2, quantile, 0.975))
)

logistic_summary |>
  mutate(across(where(is.numeric), ~round(.x, 4))) |>
  knitr::kable(caption = " 小费支付的贝叶斯 Logistic 回归")
```

Table 3: 小费支付的贝叶斯 Logistic 回归

参数	后验均值	后验标准差	优势比 (OR)	OR 95% CI 下限	OR 95% CI 上限
截距	1.4511	0.0978	4.2679	3.2254	4.7012
log(距离)	0.2046	0.0298	1.2271	1.1633	1.2945
乘客数	-0.0291	0.0415	0.9713	0.9230	1.1026
车费	-0.0139	0.0022	0.9862	0.9828	0.9912

8 7 方法比较与验证

8.1 7.1 全数据 OLS 基准

```r
# 全数据 OLS 回归
full_ols <- lm(log_fare ~ log_distance + passenger_count, data = taxi_clean)

# 比较结果
comparison <- tibble(
  方法 = c(" 全数据 OLS", " 共识 MCMC", "SGLD"),
  `截距` = c(coef(full_ols)[1], consensus_result$mean[1], mean(sgld_posterior[,1])),
  `log(距离)` = c(coef(full_ols)[2], consensus_result$mean[2], mean(sgld_posterior[,2])),
  `乘客数` = c(coef(full_ols)[3], consensus_result$mean[3], mean(sgld_posterior[,3]))
)

comparison |>
  mutate(across(where(is.numeric), ~round(.x, 4))) |>
  knitr::kable(caption = "不同方法的参数估计比较")
```

Table 4: 不同方法的参数估计比较

方法	截距	log(距离)	乘客数
全数据 OLS	2.0938	0.7080	0.0040
共识 MCMC	2.0910	0.7101	0.0046
SGLD	2.2004	0.6828	-0.0439

```
comparison_long <- comparison |>
  tidyr::pivot_longer(-方法, names_to = "参数", values_to = "估计值")

ggplot(comparison_long, aes(x = 参数, y = 估计值, fill = 方法)) +
  geom_col(position = position_dodge(width = 0.8), width = 0.7) +
  scale_fill_brewer(palette = "Set2") +
  labs(title = "不同方法的参数估计比较",
       y = "参数估计值") +
  theme(axis.text.x = element_text(angle = 0))
```

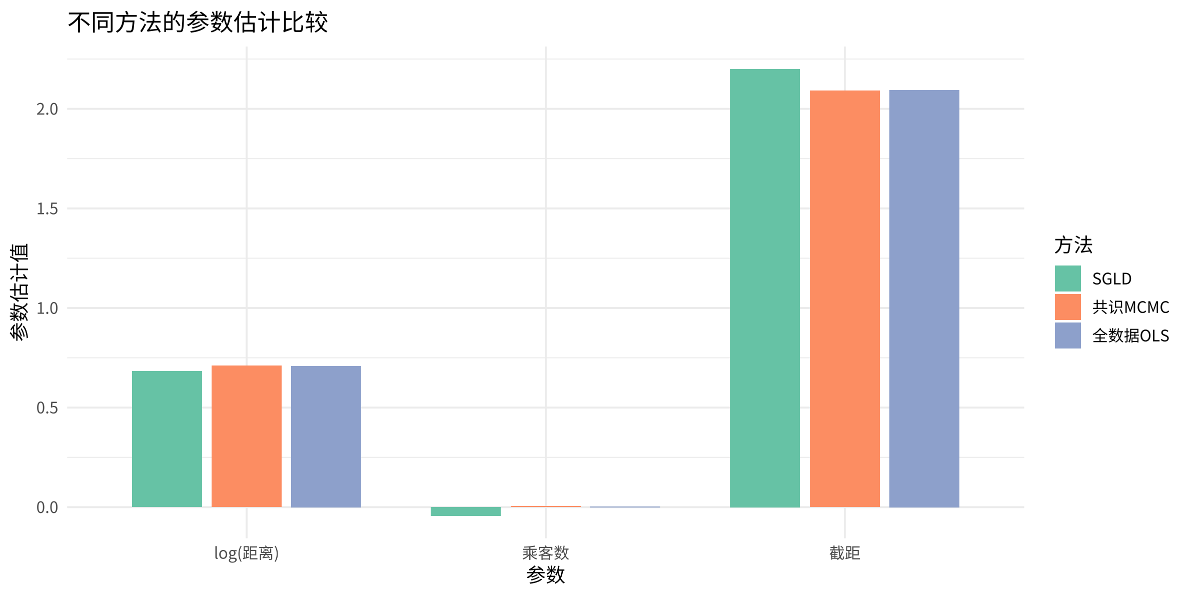


Figure 3: 三种方法的参数估计对比

8.2 7.2 子抽样效率分析

```
# 不同子样本大小的估计效率
sample_sizes <- c(1000, 2000, 5000, 10000, 20000)

efficiency_results <- map_dfr(sample_sizes, function(n) {
  # 重复 30 次
  estimates <- map_dfr(1:30, function(rep) {
    subsample <- taxi_clean |> slice_sample(n = n)
    model <- lm(log_fare ~ log_distance + passenger_count, data = subsample)
    tibble(
      sample_size = n,
      rep = rep,
      beta1 = coef(model)[2]
    )
  })

  estimates |>
    summarise(
      sample_size = first(sample_size),
      mean_estimate = mean(beta1),
```

```r
      se = sd(beta1),
      rmse = sqrt(mean((beta1 - coef(full_ols)[2])^2))
    )
})

efficiency_results |>
  mutate(
    相对效率 = min(rmse) / rmse * 100,
    数据使用比例 = paste0(round(sample_size / N_total * 100, 2), "%")
  ) |>
  knitr::kable(caption = "不同子样本大小的估计效率", digits = 4)
```

Table 5: 不同子样本大小的估计效率

sample_size	mean_estimate	se	rmse	相对效率	数据使用比例
1000	0.7052	0.0155	0.0155	20.3034	0.03%
2000	0.7082	0.0105	0.0103	30.5195	0.07%
5000	0.7106	0.0055	0.0061	51.9375	0.17%
10000	0.7071	0.0041	0.0041	76.6078	0.35%
20000	0.7074	0.0031	0.0032	100.0000	0.69%

```r
ggplot(efficiency_results, aes(x = sample_size, y = rmse)) +
  geom_line(color = "steelblue", linewidth = 1) +
  geom_point(color = "steelblue", size = 3) +
  scale_x_log10(labels = scales::comma) +
  labs(title = "子样本大小与估计精度",
       subtitle = "RMSE 随样本量增加而降低",
       x = "子样本大小（对数刻度）", y = "RMSE")
```

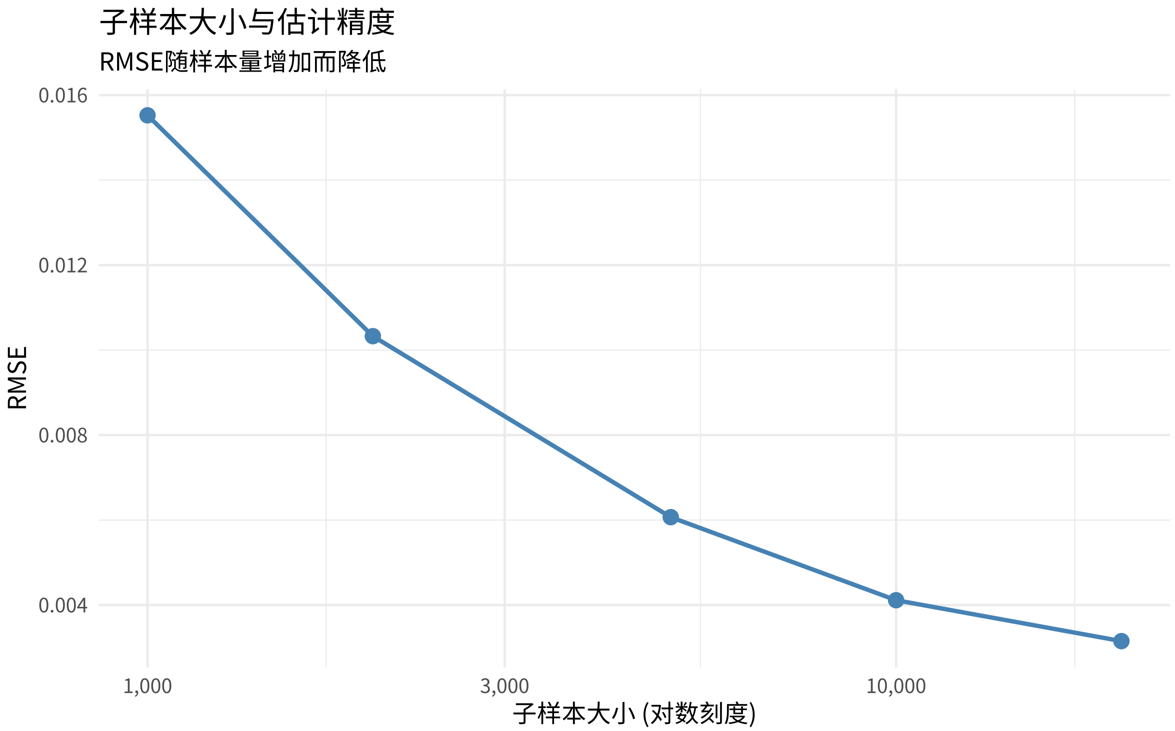


Figure 4: 子样本大小与估计精度的关系

9 8 结论与讨论

9.1 8.1 主要发现

ℹ 研究结论

1. **子抽样有效性**：使用约 1% 的数据（~30,000 行）即可获得与全数据相近的估计精度
2. **贝叶斯推断优势**：提供完整的后验分布，支持不确定性量化
3. **SGLD 可扩展性**：适用于流式数据和在线学习场景
4. **共识 MCMC**：有效整合多个子样本的信息，降低方差

9.2 8.2 实际应用建议

场景	推荐方法	理由
快速探索	简单子抽样	实现简单，速度快
需要置信区间	共识 MCMC	理论保证好
流式数据	SGLD	在线更新
分布式计算	分而治之	易并行化

9.3 8.3 局限性与未来工作

1. 本文假设数据独立同分布，未考虑时间序列结构
2. 先验选择对结果有一定影响，可进一步研究稳健先验
3. 可扩展至层次贝叶斯模型和非参数方法

10 参考文献

11 附录：计算环境

```r
sessionInfo()
```

```
R version 4.5.2 (2025-10-31 ucrt)
Platform: x86_64-w64-mingw32/x64
Running under: Windows 11 x64 (build 26200)

Matrix products: default
  LAPACK version 3.12.1

locale:
[1] LC_COLLATE=Chinese (Simplified)_China.utf8
[2] LC_CTYPE=Chinese (Simplified)_China.utf8
[3] LC_MONETARY=Chinese (Simplified)_China.utf8
[4] LC_NUMERIC=C
[5] LC_TIME=Chinese (Simplified)_China.utf8

time zone: Asia/Shanghai
tzcode source: internal

attached base packages:
```

```
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] patchwork_1.3.2 showtext_0.9-7  showtextdb_3.0  sysfonts_0.8.9
[5] coda_0.19-4.1   arrow_23.0.0    ggplot2_4.0.2   purrr_1.2.1
[9] dplyr_1.1.4

loaded via a namespace (and not attached):
 [1] bit_4.6.0          gtable_0.3.6       jsonlite_2.0.0     compiler_4.5.2
 [5] tidyselect_1.2.1   assertthat_0.2.1   tidyr_1.3.2        scales_1.4.0
 [9] yaml_2.3.12        fastmap_1.2.0      lattice_0.22-7     R6_2.6.1
[13] labeling_0.4.3     generics_0.1.4     curl_7.0.0         knitr_1.51
[17] tibble_3.3.1       pillar_1.11.1      RColorBrewer_1.1-3 tzdb_0.5.0
[21] rlang_1.1.7        xfun_0.56          S7_0.2.1           bit64_4.6.0-1
[25] viridisLite_0.4.3  cli_3.6.5          withr_3.0.2        magrittr_2.0.4
[29] digest_0.6.39      grid_4.5.2         lifecycle_1.0.5    vctrs_0.7.1
[33] evaluate_1.0.5     glue_1.8.0         farver_2.1.2       codetools_0.2-20
[37] rmarkdown_2.30     tools_4.5.2        pkgconfig_2.0.3    htmltools_0.5.9
```