BDA - Assignment X

Anonymous

Contents

a)

For alpha we have $\alpha \sim N(0, 2^2)$ and for beta we have $\beta \sim N(10, 10^2)$. Also, correlation between alpha and beta is $corr(\alpha, \beta) = 0.6$.

The mean vector of the two values can be read straight from the definition of the normal distributions:

$$\mu_{\alpha,\beta} = \begin{bmatrix} \mu_\alpha \\ \mu_\beta \end{bmatrix} = \begin{bmatrix} 0 \\ 10 \end{bmatrix}$$

For covariance, we know that the Pearson correlation coefficient is

$$\rho_{\alpha,\beta} = corr(\alpha, \beta) = \frac{cov(\alpha, \beta)}{\sigma_\alpha \sigma_\beta}$$

We can calculate covariance from the previous formula

$$cov(\alpha, \beta) = cov(\beta, \alpha) = corr(\alpha, \beta)\sigma_\alpha\sigma_\beta = 0.6 * 2 * 10 = 12$$

The covariance matrix of alpha and beta is

$$\Sigma_{\alpha,\beta} = \begin{bmatrix} Var(\alpha) & cov(\alpha, \beta) \\ cov(\beta, \alpha) & Var(\beta) \end{bmatrix} = \begin{bmatrix} 4 & 12 \\ 12 & 100 \end{bmatrix}$$

b

Let's first load the draws for alphas and betas

```
data("bioassay_posterior")

alphas <- bioassay_posterior$alpha
betas <- bioassay_posterior$beta
```

Let's then create a function for calculating Monte Carlo standard mean (MCSE) $E[\theta] = \sqrt{Var(\theta)/S}$.

```
mcse_mean <- function(data) {
  sqrt(var(data) / length(data))
}
```

Then let's use this function to calculate the MCSE for alpha and beta means.

```
msce_mean_alpha <- mcse_mean(alphas)
msce_mean_beta <- mcse_mean(betas)
```

MCSE mean for alpha: 0.01482435
MCSE mean for beta: 0.07560016

We can calculate 5% and 95% quantiles with aaltobda function mcse_quantile. Mcse-quantile returns a dataframe so we need to take the mcse-column to get the value for the quantile.

```
mcse_low_alpha <- mcse_quantile(alphas, 0.05)$mcse
msce_high_alpha <- mcse_quantile(alphas, 0.95)$mcse

mcse_low_beta <- mcse_quantile(betas, 0.05)$mcse
mcse_high_beta <- mcse_quantile(betas, 0.95)$mcse
```

MCSE 5% quantile for alpha: 0.02600412
MCSE 95% quantile for alpha: 0.04206342

MCSE 5% quantile for beta: 0.07043125
MCSE 95% quantile for beta: 0.2412129

Let's round the results based on the MCSE values. Only the 95% quantile will be rounded with zero digits since its error has a non-zero value in the first digit, every other MCSE has non-zero values until the second digit.

```
mean_alpha <- round(mean(alphas), 1)
mean_beta <- round(mean(betas), 1)
quantile_low_alpha <- round(quantile(alphas, 0.05), 1)
quantile_high_alpha <- round(quantile(alphas, 0.95), 1)
quantile_low_beta <- round(quantile(betas, 0.05), 1)
quantile_high_beta <- round(quantile(betas, 0.95), 0)
```

Mean for alpha: 1
Mean for beta: 10.6

5% quantile for alpha: -0.5
95% quantile for alpha: 2.6

5% quantile for beta: 4
95% quantile for beta: 19

Monte Carlo standard error is a measure of accuracy of the results which is proportional to the standard deviation of the data divided by the sample size. So when standard deviation goes up, MCSE also increases. When sample size increases, MCSE goes down.

c)

The importance ratio is defined as follows:

$$w_s = \frac{q(\theta_s)}{g(\theta_s)}$$

where q is the non-normalized target distribution and g is the proposal distribution. In this example, however, q is the posterior distribution $q \propto prior * likelihood$ and the proposal distribution is the prior distribution. Thus, we get

$$w_s = \frac{q(\theta_s)}{g(\theta_s)} = \frac{likelihood * prior}{prior} = likelihood$$

We can calculate the logarithm of the likelihood with aaltobda bioassaylp function.

```
# Load the data for bioassay
data("bioassay")

log_importance_weights <- function(alpha, beta) {
  bioassaylp(alpha, beta, bioassay$x, bioassay$y, bioassay$n)
}
```

Calculating log ratios is better than calculating since this makes the computation more numerically stable. Floating point numbers have limited accuracy depending on the amount of bits in the number (most commonly 64, 32, 16, also 8 is possible). When calculating importance ratios with logarithms, the division turn to substractions which are easy to compute and the computed values are don't get shrinked down or blown out.

d)

Normalized importance ratios can be computed with the following formula:

$$\widetilde{w} = \frac{w(\theta_s)}{\sum_s^S w(\theta_s)}$$

When transferred to log-scale, the division between values also changes to a substraction. We can calculate the normalized importance ratios with this

```
normalized_importance_weights <- function(alpha, beta) {
  log_weights <- log_importance_weights(alpha, beta)
  denominator <- log(sum(exp(log_weights)))
  weights <- log_weights - denominator
  weights <- exp(weights)
  weights
}
```

By having the sum exponentiated and scaled to one, the weights can be used to scale the approximated distribution to a proper density function.

e)

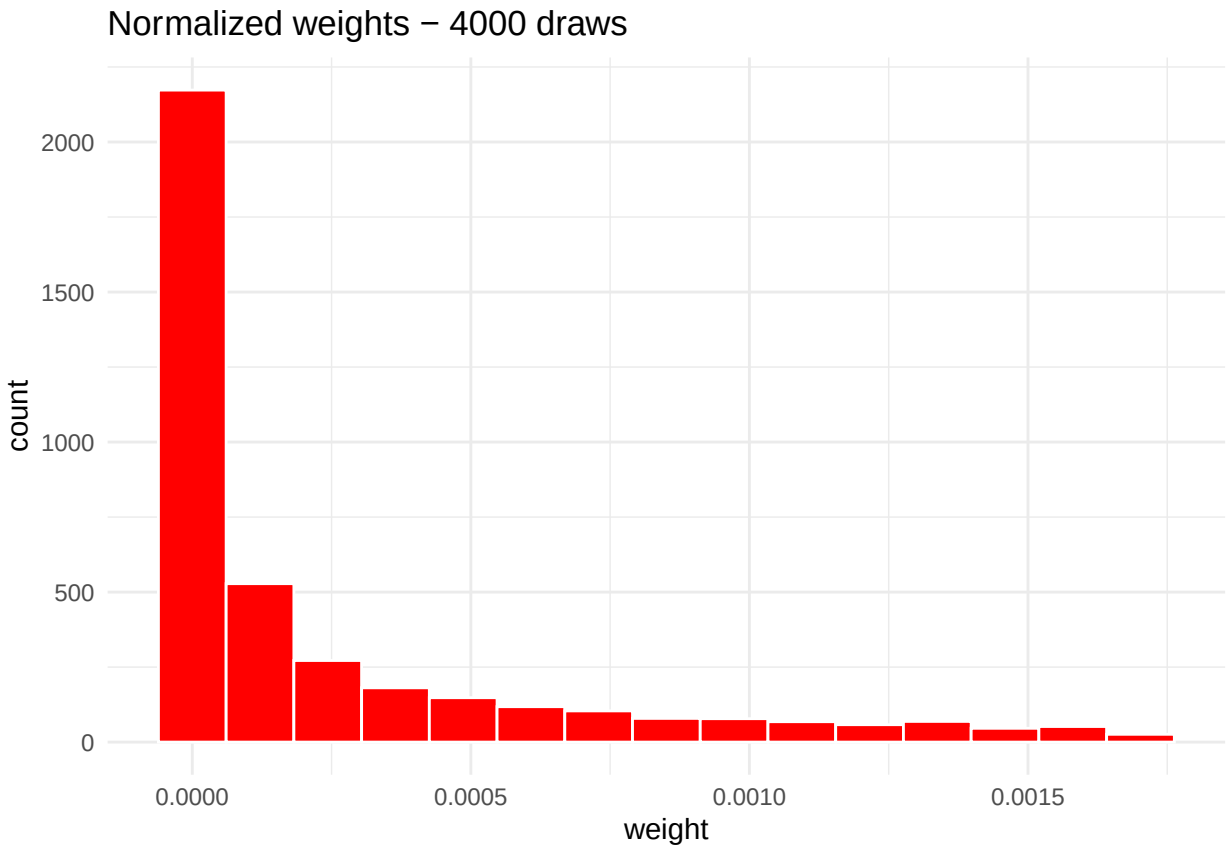
```
# a)-part data
mean_vector <- c(0, 10)
cov_matrix <- matrix(c(4, 12, 12, 100), nrow = 2, ncol = 2)
g <- rmvnorm(4000, mean_vector, cov_matrix)

# Split the matrix into alpha and beta vectors
alpha_vec <- g[, 1]
beta_vec <- g[, 2]

# d)-part funciton
weights <- normalized_importance_weights(alpha_vec, beta_vec)

# Create the distribution plot
```

```
df = data.frame(weights)
ggplot(df, aes(y=weights)) +
      geom_histogram(colour="white", fill="red", bins=15) +
      coord_flip() +
      ggtitle("Normalized weights - 4000 draws") +
      labs(y="weight")
```



f)

Effective sample size can be calculated with the following formula:

$$S_{eff} = \frac{1}{\sum_{s=1}^{S} (\widetilde{w}(\theta^s))^2}$$

```
S_eff <- function(alpha, beta) {
  # Function from part d
  log_weights <-log(normalized_importance_weights(alpha, beta))

  # Log weight squares
  square_log_weights <- log_weights + log_weights

  sum_squares <- sum(exp(square_log_weights))
  1 / sum_squares
}
S_eff(alpha = alpha_vec, beta = beta_vec)
```

```
## [1] 1108.066
```

g)

The effective sample size (ESS) is a rough estimation about how many samples will be enough to reasonably cover the original distribution. If there are a few very high weights, S_eff will be very small since these few weights will make the denominator very large. On the other hand, if the weights are uniformly distributed, then the effective sample size is the original sample size.

We can see from the plot that the general trend is exponentially decaying. This means that the first few weights will have the most influence over the distribution and thus ESS will be again relatively small.

h)

Normalization is inherently integrated into importance sampling since the method takes into account the difference between the proposal and target distributions. The weight of each draw is proportional to the difference between proposal and target distributions at that value, so if the target distribution is higher than the proposal at a value of a draw, that draw has a higher weight. This goes the other way around also: if the proposal distribution has a higher value, then weight is lower.

Functions are need to compute the posterior means for alpha and beta and the squared values of both. The expected values of squares are needed for calculating the importance sampling variances.

Here are the posterior function implementations:

```
posterior_mean <- function(alpha, beta) {
  weights <- exp(log_importance_weights(alpha, beta))
  sum_weights <- sum(weights)

  e_alpha <- sum(alpha * weights) / sum_weights
  e_beta <- sum(beta * weights) / sum_weights

  c(e_alpha, e_beta)
}

posterior_squared_mean <- function(alpha, beta) {
  weights <- exp(log_importance_weights(alpha, beta))
  sum_weights <- sum(weights)

  e_alpha_squared <- sum(alpha * alpha * weights) / sum_weights
  e_beta_squared <- sum(beta * beta * weights) / sum_weights

  c(e_alpha_squared, e_beta_squared)
}
```

Then let's compute the means and MCSE errors for both alpha and beta.

```
means <- posterior_mean(alpha_vec, beta_vec)
squared_means <- posterior_squared_mean(alpha_vec, beta_vec)

alpha_squared_mean <- squared_means[1]
alpha_mean_squared <- means[1] * means[1]
beta_squared_mean <- squared_means[2]
```

```
beta_mean_squared <- means[2] * means[2]

var_alpha <- alpha_squared_mean - alpha_mean_squared
var_beta <- beta_squared_mean - beta_mean_squared

s_eff <- S_eff(alpha_vec, beta_vec)
mcse_alpha <- sqrt(var_alpha / s_eff)
mcse_beta <- sqrt(var_beta / s_eff)
```

MCSE alpha: 0.02627781
MCSE beta: 0.1355264

Because of the MCSE values, the posterior means should be reported with accuracies 1 and 0 decimals for both alpha and beta.

```
means <- posterior_mean(alpha_vec, beta_vec)
squared_means <- posterior_squared_mean(alpha_vec, beta_vec)

alpha_squared_mean <- squared_means[1]
alpha_mean_squared <- means[1] * means[1]

beta_squared_mean <- squared_means[2]
beta_mean_squared <- means[2] * means[2]

var_alpha <- alpha_squared_mean - alpha_mean_squared
var_beta <- beta_squared_mean - beta_mean_squared

s_eff <- S_eff(alpha_vec, beta_vec)
mcse_alpha <- sqrt(var_alpha / s_eff)
mcse_beta <- sqrt(var_beta / s_eff)
```

Mean alpha: 0.9624201
Mean beta: 10.51923

Posterior mean alpha: 1
Posterior mean beta: 11