BDA - Assignment 4

Anonymous

Contents


```
# To install aaltobda, see the General information in the assignment.
remotes::install_github("avehtari/BDA_course_Aalto", subdir = "rpackage", upgrade = "never")
```

```
## Skipping install of 'aaltobda' from a github remote, the SHA1 (38f34d35) has not changed since last i
##   Use `force = TRUE` to force installation
```

```
library(aaltobda)
```

1. Bioassay model

A)

I construct a vector of means with the dimensions 2×1, and a variance-covariance matrix with the dimensions 2×2.

```
mean  <- c(0, 10)
mean  <- matrix(mean, nrow = 2)
sigma <- c(2, .6, .6, 10)
sigma <- matrix(sigma, nrow = 2, ncol = 2)

print(mean)
```

```
##      [,1]
## [1,]    0
## [2,]   10
```

```
print(sigma)
```

```
##      [,1] [,2]
## [1,]  2.0  0.6
## [2,]  0.6 10.0
```

B)

```
data("bioassay_posterior")
# Function to calculate expected value with MCSE
# Also adjusts for leading zeros after the decimal
mcse.estimate <- function(vector) {
  S <- length(vector)

  estimate <- (1/S)*sum(vector)
  variance <- var(vector)
```

```
  mcse <- sqrt(variance/S)

  nu.leadingZeros <- attr(regexpr("(?<=\\.)0+", mcse, perl = TRUE), "match.length")
  estimate <- round(estimate, nu.leadingZeros)

  list <- list("Estimate" =  estimate,
               "MCSE"     = mcse)

  return(list)

}
# Function to calculate quantile value with MCSE
# Also adjusts for leading zeros after the decimal
mcse.estimateQuantile <- function(vector, prob) {

  q    <- quantile(vector, prob = prob)
  mcse <- mcse_quantile(vector, prob = prob)

  nu.leadingZeros <- attr(regexpr("(?<=\\.)0+", mcse, perl = TRUE), "match.length")
  q <- round(q, nu.leadingZeros)

  list <- list("Quantile" = q,
               "MCSE"     = mcse)

  return(list)

}
```

Below I describe the values for α:

```
# Values for alpha
alpha <- bioassay_posterior$alpha
mcseEstimate <- mcse.estimate(alpha)
mcse.estimateQuantile05 <- mcse.estimateQuantile(alpha, prob = 0.05)
mcse.estimateQuantile95 <- mcse.estimateQuantile(alpha, prob = 0.95)

print(mcseEstimate)
```

```
## $Estimate
## [1] 1
##
## $MCSE
## [1] 0.01482435
```

```
print(mcse.estimateQuantile05)
```

```
## $Quantile
##   5%
## -0.5
##
## $MCSE
##         mcse
## 1 0.02600412
```

```
print(mcse.estimateQuantile95)
```

```
## $Quantile
## 95%
## 2.6
##
## $MCSE
##         mcse
## 1 0.04206342
```

Below I describe the values for β:

```
# Values for alpha
beta <- bioassay_posterior$beta
mcseEstimate <- mcse.estimate(beta)
mcse.estimateQuantile05 <- mcse.estimateQuantile(beta, prob = 0.05)
mcse.estimateQuantile95 <- mcse.estimateQuantile(beta, prob = 0.95)

print(mcseEstimate)
```

```
## $Estimate
## [1] 10.6
##
## $MCSE
## [1] 0.07560016
```

```
print(mcse.estimateQuantile05)
```

```
## $Quantile
## 5%
##  4
##
## $MCSE
##         mcse
## 1 0.07043125
```

```
print(mcse.estimateQuantile95)
```

```
## $Quantile
## 95%
##  20
##
## $MCSE
##        mcse
## 1 0.2412129
```

Importance sampling:

C)

Notice here that $w(\theta^s) = \frac{q(\theta^s|y)}{g(\theta)} = q(y|\theta^s)$ when we assumed that the prior distribution $p(\theta)$ equals our proposal distribution $g(\theta)$. Thus, our function is given below. Also, we work with logs because it makes things computaitonally easier!

```
library(aaltobda)
data("bioassay")
bioassay
```

```
##       x n y
## 1 -0.86 5 0
```

```
## 2 -0.30 5 1
## 3 -0.05 5 3
## 4  0.73 5 5
```

```
alpha <- c(1.896, -3.6, 0.374, 0.964, -3.123, -1.581)
beta  <- c(24.76, 20.04, 6.15, 18.65, 8.16, 17.4)

log_importance_weights <- function(alpha, beta) {

  log.like <- bioassaylp(alpha, beta, bioassay$x, bioassay$y, bioassay$n)

  return(log.like)

}

#pointc <- log_importance_weights(alpha, beta)
#round(pointc,2)
```

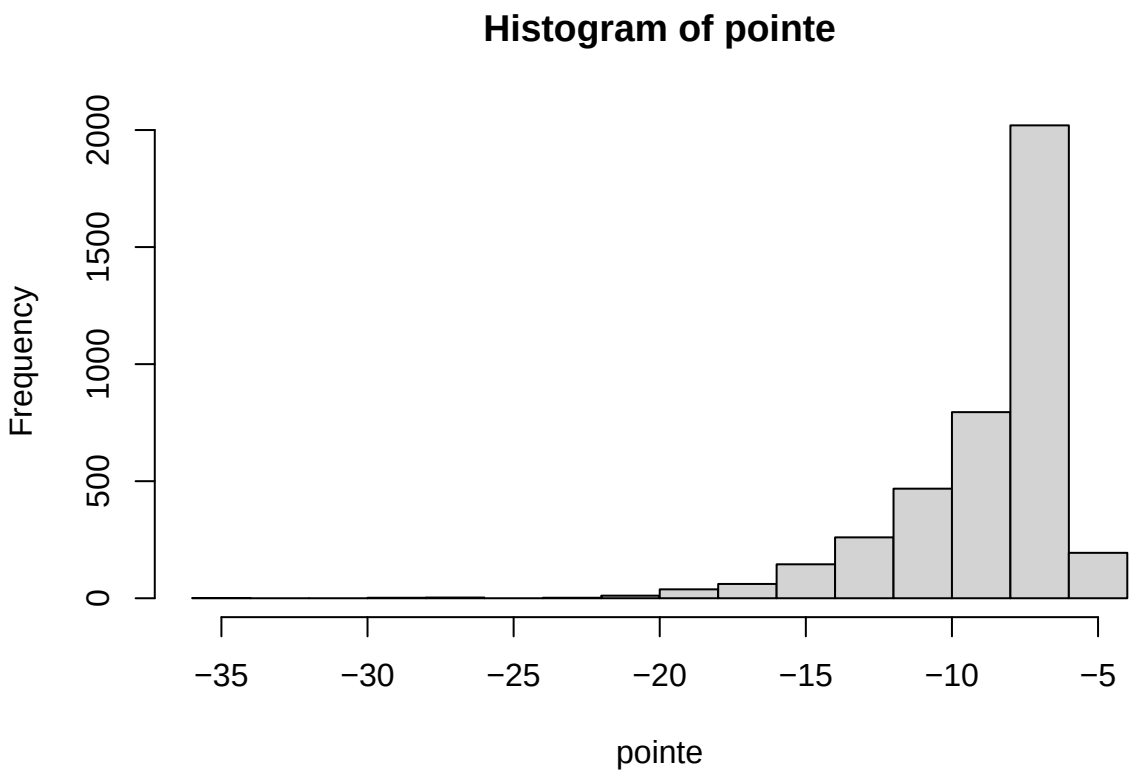
D)

The effect of exponentiation and scaling makes things easier for interpretation.

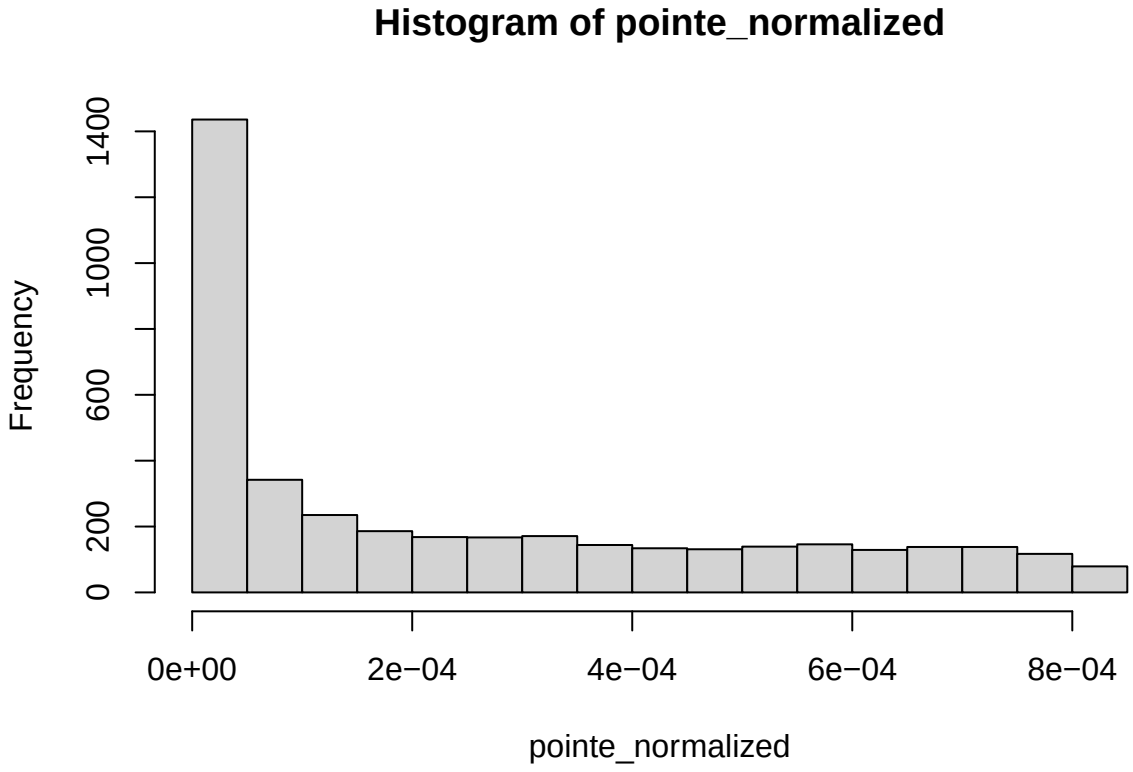
```
normalized_importance_weights <- function(alpha, beta) {

  log.like <- bioassaylp(alpha, beta, bioassay$x, bioassay$y, bioassay$n)
  like <- exp(log.like)

  denominator <- sum(like)

  return(like/denominator)

}

#pointd <- normalized_importance_weights(alpha = alpha, beta = beta)
#round(pointd, 3)
```

E)

```
sims <- 4000
theta <- rmvnorm(sims, mean = mean, sigma = sigma)

alphaE <- theta[,1]
betaE  <- theta[,2]

pointe <- log_importance_weights(alpha = alphaE, beta = betaE)
pointe_normalized <- normalized_importance_weights(alpha = alphaE, beta = betaE)

hist(pointe)
```




```
hist(pointe_normalized)
```



F)

Below I show the sampling effective sample size.

```r
S_eff <- function(alpha, beta) {
  Seff <- 1/sum(normalized_importance_weights(alpha, beta)^2)
  return(Seff)
}

Seff <- S_eff(alpha, beta)
print(Seff)
```

```
## [1] 1.354205
```

G)

To analyze the possibility of missing some extremely large importance weights, we can look at the sampling effective sample size. If small, it indicates that there are only a handful of extremely high weights. If big, then it shows that it has many extremely high weights.

H)

```r
posterior_mean <- function(alpha, beta) {

  theta <- list(alpha, beta)
  lst <- list()
  i <- 0
  for(t in theta) {
    i <- 1 + i

    S <- length(t)
    estimate <- (1/S)*sum(t)
    variance <- var(t)
    mcse <- sqrt(variance/Seff)
    nu.leadingZeros <- attr(regexpr("(?<=\\.)0+", mcse, perl = TRUE), "match.length")
    estimate <- round(estimate, nu.leadingZeros)

    lst[i] <- estimate
    i <- 1 + i
    lst[i] <- mcse[1]
  }
  return(lst)
}

pstmean <- posterior_mean(alpha, beta)
print(pstmean)
```

```
## [[1]]
## [1] 0
##
## [[2]]
## [1] 1.944149
##
## [[3]]
## [1] 20
##
## [[4]]
## [1] 6.201597
```