

Homework 2 Answer Key

Minji Jeong

Problem 1: Working with profile likelihoods

Consider the following dataset: $y = (1, 0, 0, 1, 0, 0, 0, 0)$, which happens to be a series of independent realizations of a Bernoulli distributed random variable. Plot the likelihood function for the Bernoulli parameter π given y . If the experiment were repeated, roughly what fraction of the observations would you expect to be successes ($y = 1$)? Why?

Answer 1a:

In this question, we aim to simulate the likelihood of a Bernoulli distribution and visualize its profile.

For a sequence of n independent Bernoulli trials, each with success probability π , the probability mass function (p.m.f.) of each observation y_i (where $y_i = 1$ for success and $y_i = 0$ for failure) is:

$$f_{\text{Bern}}(y_i | \pi) = \pi^{y_i} (1 - \pi)^{1-y_i} \quad \text{for } y_i = 0, 1.$$

To find the **likelihood** of observing the entire vector $\mathbf{y}' = (y_1, y_2, \dots, y_n)$, we take the product (*joint probability*) of the Bernoulli PMFs for each individual observation. Assuming a constant probability π across all observations, we have:

$$\Pr(\mathbf{y} | \pi) = \prod_{i=1}^n \pi^{y_i} (1 - \pi)^{1-y_i}.$$

The likelihood function $\mathcal{L}(\pi | \mathbf{y})$ for the entire sample $\mathbf{y} = (y_1, y_2, \dots, y_n)$ is the product of the individual p.m.f. values. The **likelihood axiom** states that the likelihood is defined as the product of the p.m.f. and a constant $k(y)$:

$$\begin{aligned} \mathcal{L}(\pi | \mathbf{y}) &= k(y) \Pr(\mathbf{y} | \pi) \\ \mathcal{L}(\pi | \mathbf{y}) &= k(y) \prod_{i=1}^n \pi^{y_i} (1 - \pi)^{1-y_i}. \end{aligned}$$

where $k(y)$ is a **constant** that does not depend on π but affects the scaling of the likelihood function.

Since $k(y)$ is constant with respect to π , it does not influence the maximization of the likelihood with respect to π . Therefore, we can **drop** $k(y)$, making the likelihood only **proportional** to the remaining expression:

$$\mathcal{L}(\pi | \mathbf{y}) \propto \prod_{i=1}^n \pi^{y_i} (1 - \pi)^{1-y_i}$$

This expression is now ready for maximizing with respect to π to obtain the maximum likelihood estimate (MLE). The code below performs this simulation and plotting.

```

y = c(1, 0, 0, 1, 0, 0, 0, 0) # data

L_Bernoulli <- function(data, pi) {
  # data: vector of binary data (0 or 1)
  # pi: success probability parameter for each independent Bernoulli trial

  # The Bernoulli likelihood is the joint probability (product) of individual Bernoulli probabilities
  res <- prod(pi^data * (1 - pi)^(1 - data))

  return(res)
}

# Now create vectors for loop

sims <- 1000

pi <- seq(0, 1, length.out=sims) # probability = [0,1]

lls <- numeric(sims) # to store the likelihoods

for(i in 1:sims) {
  lls[i] <- L_Bernoulli(data = y, pi[i]) # estimate and store L_Ben for every value of pi
}

# Visualize

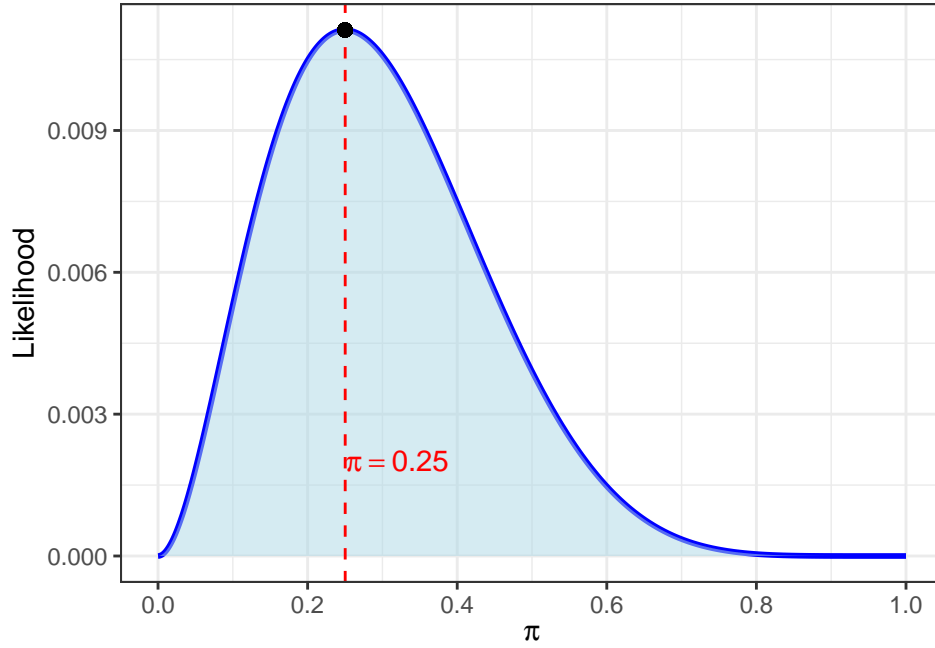
dt <- data.frame(pi,lls)

dt |>
  ggplot(aes(x = pi, y = lls)) +
  geom_line(aes(x = pi,
                y = lls),
            color = "blue", linewidth = 1) +
  # the line below this is optional
  geom_area(fill = "lightblue", alpha = 0.5) +
  geom_vline(linetype = "dashed",
             color = "red",
             xintercept = filter(dt, lls == max(lls))[, "pi"]) +
  geom_point(x = filter(dt, lls == max(lls))[, "pi"], # Selecting pi by maximum ll value
            y = max(dt$lls),
            color = "black", size = 2) +
  annotate(geom="text",
          x = .32,
          y = .002,
          color = "red",
          parse = TRUE, # To allow mathematical expressions
          label = "pi == .25") +

  scale_y_continuous(name = "Likelihood") +
  scale_x_continuous(limits = c(0, 1),
                    breaks = seq(0, 1, 0.2),

```

```
name = expression(pi)) +  
theme_bw()
```



As an alternative approach, we can derive and visualize the log-likelihood function of the Bernoulli distribution below. Recall the likelihood function:

$$\mathcal{L}(\pi \mid \mathbf{y}) = k(y) \prod_{i=1}^n \pi^{y_i} (1 - \pi)^{1-y_i}.$$

To simplify the computation, we take the **natural logarithm** of the likelihood function to obtain the **log-likelihood function**.

$$\log \mathcal{L}(\pi \mid \mathbf{y}) = \log k(y) + \sum_{i=1}^n (y_i \log \pi + (1 - y_i) \log(1 - \pi)).$$

Note that $\log(k(y))$ is constant with respect to π and can be omitted in the computation, as it only scales the likelihood's level without affecting its functional form. However, once we omit the constant, the likelihood becomes proportional to the right hand side of the equation.

Dropping the constant $\log k(y)$ (since it does not affect the maximization), we get:

$$\log \mathcal{L}(\pi \mid \mathbf{y}) \propto \sum_{i=1}^n (y_i \log \pi + (1 - y_i) \log(1 - \pi)).$$

Now we can distribute the summation across terms involving y_i and $1 - y_i$. This allows us to rewrite the log-likelihood as:

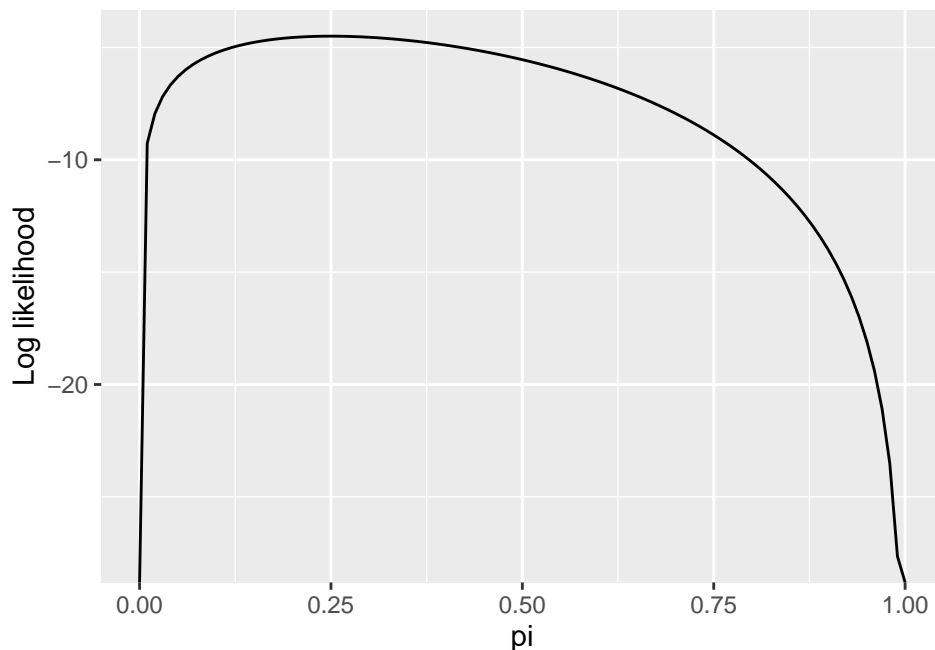
$$\log \mathcal{L}(\pi \mid \mathbf{y}) \propto \left(\sum_{i=1}^n y_i \right) \log \pi + \left(\sum_{i=1}^n (1 - y_i) \right) \log(1 - \pi).$$

The resulting equation provide the computational routine to plot the **log-likelihood profile** of a Bernoulli distribution given some sample of $\mathbf{y}' = (y_1, y_2, \dots, y_n)$.

```
# vector of data on independent Bernoulli realizations
y <- c(1, 0, 0, 1, 0, 0, 0, 0)

# function of profile likelihood
fun1 <- function(pi){
  sum(y)*log(pi) + sum(1-y)*log(1-pi)
}

# visualize the
tibble(pi = 0) |>
  ggplot(mapping = aes(x=pi)) +
  # func argument to plot profile
  stat_function(fun = fun1) +
  # make sure to provide the domain of pi
  xlim(0,1)+
  labs(x = "pi",
       y = "Log likelihood")
```



To identify the maximum point, use `optimize` function.

```
optimize(fun1, interval = c(0,1), maximum = TRUE)

## $maximum
## [1] 0.2500143
##
## $objective
## [1] -4.498681
```

If we were to repeat the experiment, assuming no small sample bias, the expected success rate in our observations ($y = 1$) would be of $1/4$ (the sample mean). As the likelihood maximizes the probability of success at 0.25.

Problem 2: Writing and testing a useful new maximum likelihood estimator

Answer 2a.:

$$\begin{aligned}
 y_i &\sim f_{\text{Poisson}}(\lambda_i) = \frac{\exp(-\lambda_i)\lambda_i^{y_i}}{y_i!} \\
 \lambda_i &= \exp(\mathbf{x}_i\beta) \\
 \mathcal{L}(\lambda \mid y) &= k(y)\Pr(y \mid \lambda) = k(y) \prod_{i=1}^n \frac{\exp(-\lambda_i)\lambda_i^{y_i}}{y_i!} \\
 \log \mathcal{L}(\lambda \mid y) &= \log \prod_{i=1}^n k(y_i) \frac{\exp(-\lambda_i)\lambda_i^{y_i}}{y_i!} \\
 &= \sum_{i=1}^n \log \left(k(y_i) \frac{\exp(-\lambda_i)\lambda_i^{y_i}}{y_i!} \right) \\
 &= \sum_{i=1}^n \left(\log k(y_i) + y_i \log \lambda_i - \log \exp(\lambda_i) - \log y_i! \right) \\
 &\propto \sum_{i=1}^n \left(y_i \log \lambda_i - \lambda_i \right) \\
 \log \mathcal{L}(\beta \mid y) &\propto \sum_{i=1}^n \left(y_i(\mathbf{x}_i\beta) - \exp(\mathbf{x}_i\beta) \right)
 \end{aligned}$$

Answer 2b.:

$$\begin{aligned}
 f_{\text{Poisson}, t}(y \mid \lambda, t) &= \frac{\exp(-\lambda_i t_i)(\lambda_i t_i)^{y_i}}{y_i!} \\
 \mathcal{L}(y \mid \lambda, t) &= \prod_{i=1}^n \frac{\exp(-\lambda_i t_i)(\lambda_i t_i)^{y_i}}{y_i!} \\
 \log \mathcal{L}(\lambda, t \mid y) &= \log \prod_{i=1}^n k(y_i) \frac{\exp(-\lambda_i t_i)(\lambda_i t_i)^{y_i}}{y_i!} \\
 &\propto \sum_{i=1}^n \left(y_i \log(\lambda_i t_i) - \lambda_i t_i \right) \\
 &\propto \sum_{i=1}^n \left(y_i (\log \lambda_i + \log t_i) - \lambda_i t_i \right) \\
 \log \mathcal{L}(\beta \mid y) &\propto \sum_{i=1}^n \left(y_i(\mathbf{x}_i\beta) - t_i \exp(\mathbf{x}_i\beta) \right)
 \end{aligned}$$

Answer 2c.:

```
LHp <- function(param, y, x, t) { # no constant version
                                # so put constant in the xcovariate
                                # when running optim

  x <- as.matrix(x)
  t <- as.matrix(t)
  beta <- param
  xb <- x %*% beta
  -sum(y * (xb) - t * exp(xb))
}

LHp2 <- function(param, y, x, t) { # with constant version
                                   # so don't put constant in the xcovariate
                                   # when running optim

  x <- as.matrix(x)
  os <- rep(1, nrow(x))
  x <- cbind(os, x)
  t <- as.matrix(t)
  beta <- param
  xb <- x %*% beta
  -sum(y * (xb) - t * exp(xb))
}
```

Answer 2d.:

```
obs <- 1000
beta <- c(0, 1, 2)
x0 <- rep(1, obs)
x1 <- runif(obs, 0, 1)
x2 <- runif(obs, 0, 1)
x <- cbind(x0, x1, x2)
t <- sample(c(1,2,3,4,5), size = obs, replace = TRUE)
lambda <- exp(x %*% beta)

y <- rpois(obs, t * lambda)

y %>%
  as_tibble() %>%
  summarise_all(list(mean=mean, sd=sd)) %>%
  pander()
```

mean	sd
16.54	14.53

The mean of y_i is approximately 16.5, and the standard deviation of y_i is approximately 14.7.

Answer 2e.:

```
stval <- c(0, 0, 0)
result <-
  optim(stval,
        LHp2,
        method = "BFGS",
        hessian = TRUE,
        y = y,
        x = x[,-1], # x without constant
        t = t)

pe <- result$par           # point estimate
vc <- solve(result$hessian) # var-cov matrix
se <- sqrt(diag(vc))       # standard errors
(ll <- -result$value)      # likelihood at maximum
```

```
## [1] 14566.01
```

```
r <- tibble(
  param = c("beta0", "beta1", "beta2"),
  Point_estimates = pe,
  Standard_errors = se)

r %>% pander()
```

param	Point_estimates	Standard_errors
beta0	-0.02372	0.02675
beta1	1.03	0.02754
beta2	2.009	0.03022

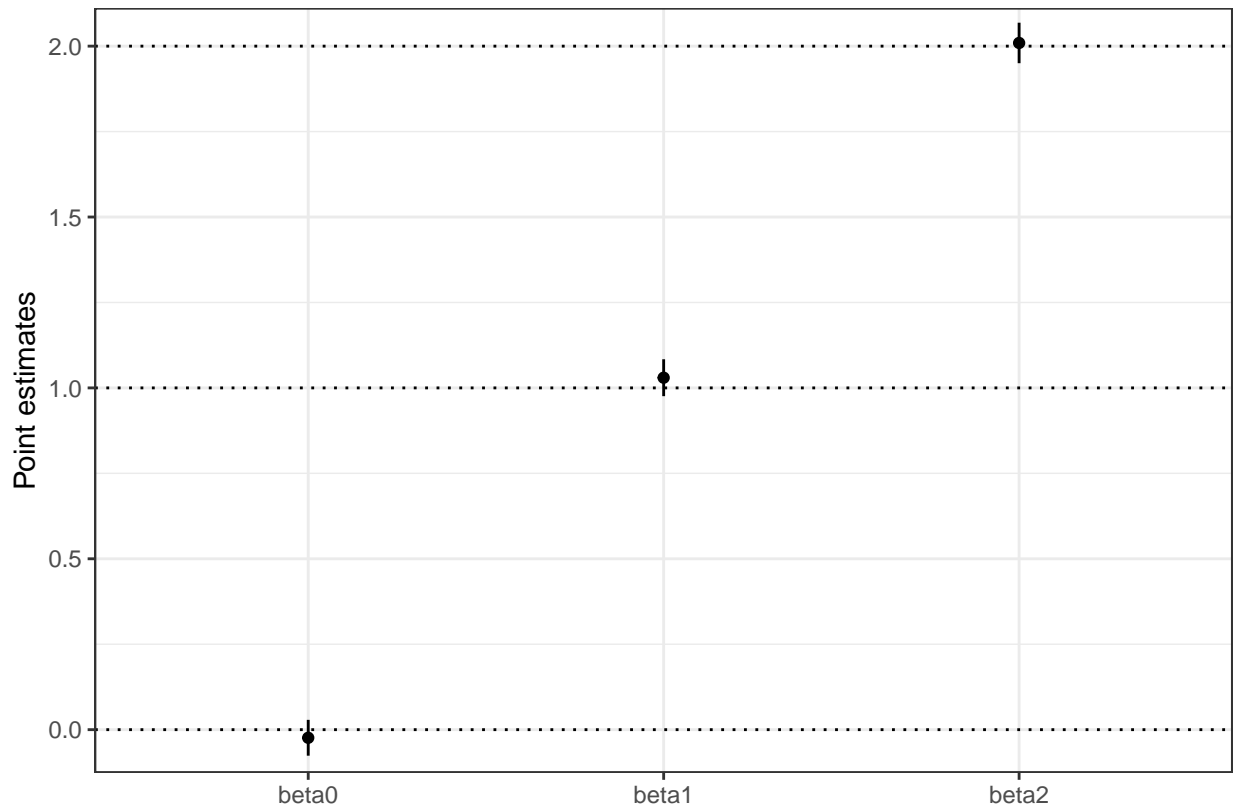
```
# just for your understanding
```

```
dta <-
  tibble(pe = pe,
        se = se,
        trueBeta = c(0,1,2)) %>%
  mutate(lower = pe -1.96*se,
        upper = pe +1.96*se,
        Conf95 = case_when(lower<=trueBeta & upper >= trueBeta ~ 1,
                             TRUE ~0))

dta %>% kable() %>% kable_styling()
```

pe	se	trueBeta	lower	upper	Conf95
-0.023719	0.0267486	0	-0.0761462	0.0287082	1
1.029824	0.0275427	1	0.9758406	1.0838081	1
2.009291	0.0302250	2	1.9500501	2.0685321	1

```
## x2
r %>%
  ggplot(aes(x = param, y = pe))+
  geom_point()+
  geom_linerange(ymin = pe - 1.96*se, ymax = pe + 1.96*se)+
  geom_hline(yintercept = c(0, 1, 2), linetype = "dotted")+
  labs(y = "Point estimates",
       x = "")+
  theme_bw()
```



As the above plot shows, the point estimates are close to true value with small standard error. To improve our confidence, we can increase the number of observations and sample size.

Problem 3: Solving the birthday problem using simulation

A famous probability problem asks “What is the probability that at least two people in a classroom of n people share the same birthday?” Write an R program to solve this problem using simulation. Produce as output a plot of the probabilities of at least one shared birthday for $n = \{2, \dots, 50\}$.

Answer 3:

Method 1: ifelse and loop


```

# Practice -- one class size
sims <- 1000
people <- 15
sameday <- 0
for (k in 1:sims) {
  bday <- sample(1:365, people, replace = TRUE) # assign random birthdays to students
  if (length(unique(bday)) < people){
    # or (length(unique(bday))!=length(bday))
    sameday <- sameday+1
    # Add 1 if at least two of them have the same birthday
  }
}
print(sameday/sims)

```

```
## [1] 0.266
```

```

# Varying class size (2:50)
n = c(2:50) # class size
sims <- 1000
sameday <- rep(NA, sims) # empty vector for storing simulated results
prob <- rep(NA, length(n)) # empty vector for storing probability for each n

for (i in n) { # outer loop for iterating through each class size

  bday <- rep(NA, i) # empty vector for birthdays of students per each class size
                    # changes in every simulation

  for (k in 1:sims){
    bday <- sample(1:365, i, replace=T) # assign random birthdays to students
    sameday[k] <- ifelse(length(unique(bday))!=length(bday), 1, 0)
    # 1 if at least two of them have the same birthday
    # 0 if none shares birthdays
  }

  prob[i-1] <- sum(sameday)/sims # probability for this class size
  # i-1 to store the result of the class size of 2 as the 1st element of this vector
  # (R loop through positions, not values)
  # OR use seq_along(n) in outer loop
}

bday_1 <- data.frame(n=n, prob=prob)

```

Method 2: duplicated in dplyr

```

BdayFunc <- function(n){
  shared <- tibble(sims = 1:1000) %>% # each row represents one simulation
    rowwise() %>% # treat each row as an independent unit of computation
    mutate(bdays = list(sample(c(1:365), size = n, replace = TRUE)),
           twopeople = TRUE %in% duplicated(bdays)) %>% # check if TRUE exists
    ungroup() %>%

```

```

    summarize(prob = mean(twopeople)) %>%
    as.numeric() #change tibble into a numerica value

  shared
}

prob <- sapply(X = 2:50, FUN = BdayFunc)

bday_2<-tibble(n = n, prob = prob)

# use it inside the loop
sims <- 1000
n <- 2:50
prob <- numeric(length(n)) # store results

for (i in n) { # outer loop for iterating through each class size
  sameday <- 0 # starts at 0 every time the outer loop begin
  for (k in 1:sims) { # inner loop for simulation
    bdays <- sample(1:365, i, replace = TRUE) # random birthdays
    if (any(duplicated(bdays))) { # check if any duplicates
      sameday <- sameday + 1 # count shared cases
    }
  }

  prob[i-1] <- sameday / sims # probability for this class size
}

bday_22 <- data.frame(n = n, prob = prob)

```

Method 3: facorial

```

n <- 2:50
prob <- rep(NA,49)
none <- 1 # first product for calculating the probability that none shares birthday
# 365/365

for (i in n) {
  none <- none * ((365 - (i-1)) / 365) # probability that none shares birthday
# none * ith term
  prob[i-1] <- 1-none # probability that at least two have the same birthday
}

bday_3 <- data.frame(n=n, prob=prob)

```

Visualization

```

ggplot(bday_1, aes(x=n, y=prob)) +
  geom_point() +

```

```
labs(x="The size of the classroom",  
     y="Probability that at least two people in a classroom \n share the same birthday",  
     title="Probability that at least two people share the same birthday  
           \n per each size of the classroom") +  
theme_classic()
```

