HW6

Jinhong Du - 12243476

May 23, 2020

Contents

Problem A: Multiple Testing

Consider the following setting in the context of "multiple hypothesis tests". Let $i = 1, \ldots, n$ index individuals and $j = 1, \ldots, m$ index genes (or pixels in an image if you prefer). Assume we have measurements on each individual at each gene for "treatments" $k = 0, 1$. Let Y_{ijk} denote the measurement on individual i, gene j, treatment k, and let D_{ij} denote the difference between the measurements in the two treatments: $D_{ij} := Y_{ij0} - Y_{ij1}$. We will assume that D is sufficient for all our inferences, and so you can forget about Y now and work only with D: I just wanted you to understand where D might come from in principle.

We will assume a model for D:

$$D_{ij} | \beta, \sigma \sim \mathcal{N}(\beta_j, \sigma_j^2) \tag{1}$$

where $\beta = (\beta_1, \ldots, \beta_m)$ is a vector of "treatment effects", where β_j is the effect at gene j, and $\sigma = (\sigma_1, \ldots, \sigma_m)$ is a vector of standard deviation parameters. For each gene j we wish to test the null $H_j : \beta_j = 0$ (that is, that there is no treatment effect). For simplicity you can assume that $\sigma_j = 1$ is known for all j. You can also assume that $n = 10$ and $m = 1000$.

Assume that the true effects β_j are independent, and identically distributed, with

$$\beta_j \sim \pi_0 \delta_0 + (1 - \pi_0) \mathcal{N}(0, \sigma_b^2) \tag{2}$$

where δ_0 denotes a point mass on zero. That is, $\beta_j = 0$ with probability π_0, and $\beta_j \sim \mathcal{N}(0, \sigma_b^2)$ with probability $1 - \pi_0$.

i) Write an `R` function to simulate data D under this model, for userspecified π_0 and σ_b. The function should take π_0 and σ_b as input, and return a list, with elements D (a matrix) and β (a vector).

[1]:
```
generate_data <- function(n, m, pi_0, sigma_b){
    beta <- rnorm(m, 0, sigma_b)
    beta[rbinom(m, size = 1, prob=pi_0)==1] <- 0
    D <- mapply(function(x,y){rnorm(x,y,n=n)},x=beta,y=1)
    return(list(D=D,beta=beta))
}
```

ii) Write an `R` function to compute a p value p_j for each column of the data matrix D, testing $H_0 : \beta_j = 0$. This function should take as input the data matrix D and output a vector of p values. You can use any reasonable two-sided test, but state which test you use. Apply your `R` function to data simulated under a) $\pi_0 = 1$, b) $\pi_0 = 0.5$, $\sigma_b = 3$; c) $\pi_0 = 0$, $\sigma_b = 3$. Provide histograms of the p values in each case and comment on their distributions.

Since $D_{1j}, \dots D_{n,j}|\beta \sim \mathcal{N}(\beta_j, 1)$ with known variance, we can use z test for testing $H_0 : \beta_j = 0$ versus $H_1 : \beta_j \neq 0$. The statistic is

$$Z_j = \frac{\frac{1}{n}\sum_{i=1}^{n} D_{ij} - 0}{\sqrt{\frac{1}{n}}} = \sqrt{\frac{1}{n}} \sum_{i=1}^{n} D_{ij} \overset{H_0}{\sim} \mathcal{N}(0, 1),$$

and the p value is given by $p_j = 2\Phi(-|Z_j|)$ where Φ is the cumulative distribution function of the standard normal random varaible.

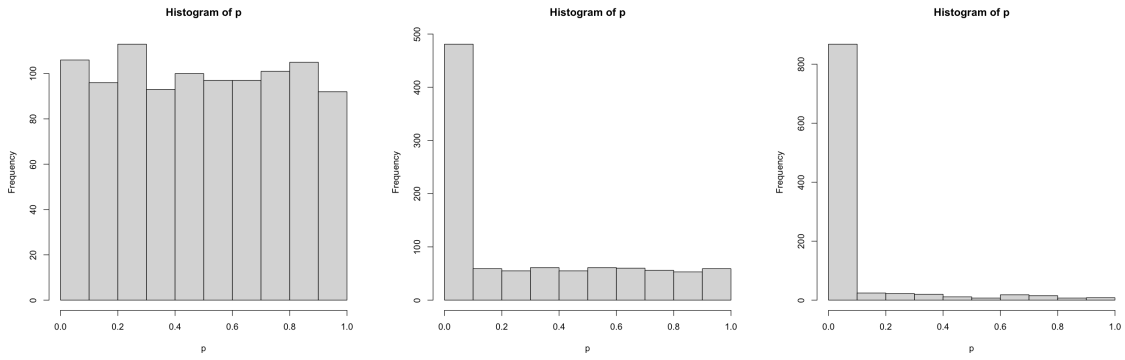
[2]:
```
compute_p <- function(D){
    n <- dim(D)[1]
    p <- 2 * pnorm(-abs(apply(D, 2, sum))/sqrt(n))
    return(p)
}

set.seed(0)
n <- 10
m <- 1000

pi_0 <- 1
sigma_b <- 1
data <- generate_data(n, m, pi_0, sigma_b)
p <- compute_p(data$D)
hist(p)

pi_0 <- 0.5
sigma_b <- 3
data <- generate_data(n, m, pi_0, sigma_b)
p <- compute_p(data$D)
hist(p)

pi_0 <- 0
sigma_b <- 3
data <- generate_data(n, m, pi_0, sigma_b)
p <- compute_p(data$D)
hist(p)
```



When $\pi_0 = 1$, the values of p_j are uniformly distributed on $(0, 1)$. In the second case, almost half of p_j is near zero and the rest of p_j is uniformly distributed on $(0, 1)$. In the third case, most values of p_j are near zero.

iii) Write an R function to apply the Benjamini–Hochberg rule to control FDR at a user-specified level α. This function should input a vector of p values, and a level α, and output a vector of binary (0/1) indicators, $\gamma = (\gamma_1, \ldots, \gamma_m)$ say, where $\gamma_j = 1$ indicates that the rule would reject $H_j : \beta_j = 0$.

[3]:
```
BH <- function(p, alpha){
    m <- length(p)
    gamma <- rep(0,m)
    sort_p <- sort(p, index.return=TRUE)
    if(sort_p$x[1]<=alpha/m){
        gamma[sort_p$ix[c(1:max(which(sort_p$x <= c(1:m) * alpha / m)))]] <- 1
    }
    return(gamma)
}
```

iv) Write an R function to compute the empirical False Discovery Rate (i.e. the number of false discoveries divided by the number of discoveries) for any given value for the vector β of true values of β, and the vector γ of reject decisions. That is, the function should return V/R in the notation of the notes. Remember to deal correctly with the special case of no discoveries, $R = 0$.

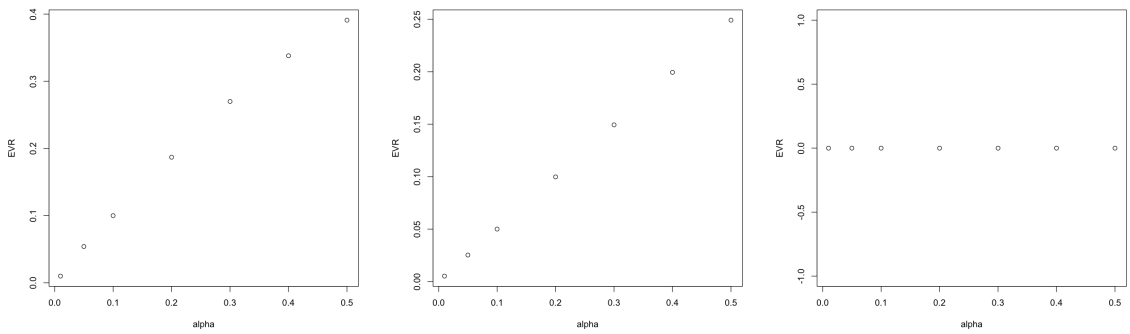
[4]:
```
FDR <- function(beta, gamma){
    R <- sum(gamma==1)
    if(R==0){
        return(0)
    }
    else{
        V <- sum((beta==0)&(gamma==1))
        return(V/R)
    }
}
```

v) Perform a simulation study to estimate the actual FDR ($\mathbb{E}(V/R)$) achieved by the BH rule in the three cases a), b) and c) above. In each case perform the test procedure for different levels α, and plot the estimated $\mathbb{E}(V/R)$ as a function of α (say for $\alpha = (0.05, 0.1, \ldots, 0.5)$). Comment on the results. [NOTE: to estimate the actual FDR you have to estimate $\mathbb{E}(V/R)$ where the expectation is over datasets D. To do this you will want to do a simulation study where you simulate a large number of datasets D, not just one dataset!]

[5]:
```
set.seed(0)
n_simulation <- 1000
n <- 10
m <- 1000
alpha <- c(0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5)

run_simu_FDR <- function(n_simulation, n, m, alpha, pi_0, sigma_b){
    res <- matrix(0,length(alpha),n_simulation)
    for(i in 1:n_simulation){
        data <- generate_data(n, m, pi_0, sigma_b)
        p <- compute_p(data$D)
        gamma <- lapply(alpha, BH, p=p)
        res[,i] <- unlist(lapply(gamma, FDR, beta=data$beta))
    }
    EVR <- apply(res, 1, mean)
    plot(alpha, EVR)
}


run_simu_FDR(n_simulation, n, m, alpha, 1, 1)
run_simu_FDR(n_simulation, n, m, alpha, 0.5, 3)
run_simu_FDR(n_simulation, n, m, alpha, 0, 3)
```



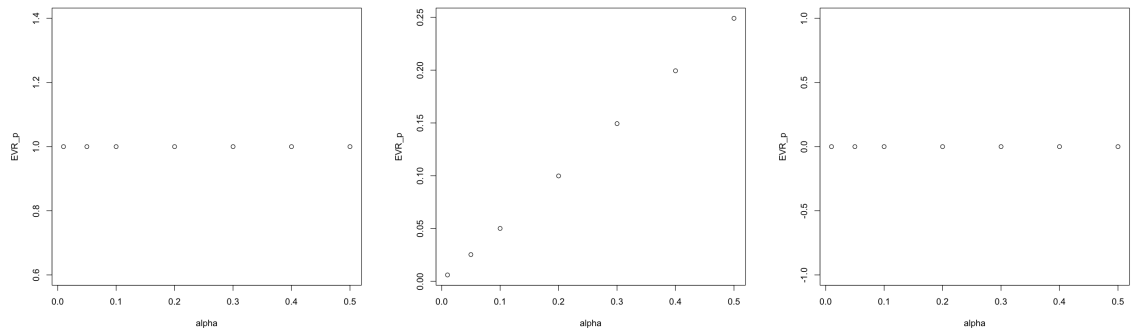
As we can see, FDR is linear with respect to α. The slope of the first plot is largest while the one of the last plot is smallest, which is 0.

vi) Repeat the simulation study, but this time estimate the pFDR instead of the FDR, and plot this as a function of α.

[6]:
```
set.seed(0)
n_simulation <- 1000
n <- 10
m <- 1000
alpha <- c(0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5)


run_simu_pFDR <- function(n_simulation, n, m, alpha, pi_0, sigma_b){
    res <- matrix(0,length(alpha),n_simulation)
    for(i in 1:n_simulation){
        data <- generate_data(n, m, pi_0, sigma_b)
        p <- compute_p(data$D)
        gamma <- lapply(alpha, BH, p=p)
        res[,i] <- unlist(lapply(gamma, FDR, beta=data$beta))
    }
    EVR_p <- apply(res, 1, sum)/apply(res>0, 1, sum)
    EVR_p[which(!is.finite(EVR_p))] <- 0
    plot(alpha, EVR_p)
}


run_simu_pFDR(n_simulation, n, m, alpha, 1, 1)
run_simu_pFDR(n_simulation, n, m, alpha, 0.5, 3)
run_simu_pFDR(n_simulation, n, m, alpha, 0, 3)
```



The difference between FDR and $pFDR$ is that when H_0 always holds, $pFDR$ is always one as in the first plot. While in other case (second and third plots), they are similar.

Problem B: HMM

In this question you will extend the HMM in https://stephens999.github.io/fiveMinuteStats/hmm.html to treat the means of the two states as unknown and to be estimated. (Note that the true values of the means in the simulation are 1 and 2).

1. Derive and implement the EM algorithm for estimating the means.

Let $Z_t \in \{0, 1\}$ be two states and the emission distributions in state k being normal with unknown mean μ_k and known standard error σ. The transition matrix for the Markov chain is symmetric, with probability 0.9 of staying in the same state, and 0.1 of switching at each step. The forward probabilities are

$$\alpha_{tk} = p(X_1, \ldots, X_t, Z_t = k),$$

so $\alpha_{1k} = \pi_k p(X_1|Z_1 = k)$ where $\pi_k = \mathbb{P}(Z_1 = k)$. Further,

$$\begin{aligned}\alpha_{t+1,k} &= \sum_j p(X_1, \ldots, X_t, Z_t = j, Z_{t+1} = k) \\ &= \sum_j \alpha_{tj} P_{jk} p(X_{t+1}|Z_{t+1} = k) \\ &= (\alpha_{t\cdot} P)_k p(X_{t+1}|Z_{t+1} = k).\end{aligned}$$

So the likelihood is given by

$$p(X_1, \ldots, X_T) = \sum_k \alpha_{Tk}.$$

The backward probabilities are

$$\beta_{tk} = p(X_{t+1}, \ldots, X_T|Z_t = k)$$

with boundary condition $\beta_{Tk} = 1$. So

$$\beta_{tk} = \sum_j p(X_{t+1}, \ldots, X_T|Z_t = k) = \sum_j \beta_{t+1,j} P_{kj} p(X_{t+1}|Z_{t+1} = j).$$

By the definitions of α and β we have:

$$p(X_1, \ldots, X_T, Z_t = k) = \alpha_{tk}\beta_{tk}.$$

Next, we take into consideration $\boldsymbol{\mu}$. Since

$$p(\boldsymbol{X}|\boldsymbol{Z}, \boldsymbol{\mu}) = \prod_{t=1}^{T} p(X_t|Z_t, \boldsymbol{\mu}), \qquad p(\boldsymbol{Z}|\boldsymbol{\mu}) = \pi_{Z_1} \prod_{t=2}^{T} P_{Z_{t-1},Z_t},$$

the complete-data log-likelihood is given by

$$\begin{aligned}\log p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\mu}) &= \log p(\boldsymbol{X}|\boldsymbol{Z}, \boldsymbol{\mu}) + \log p(\boldsymbol{Z}|\boldsymbol{\mu}) \\ &= \log \left[\pi_{Z_1} \prod_{t=2}^{T} P_{Z_{t-1},Z_t} \prod_{t=1}^{T} \prod_{k=0}^{1} \left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(X_t - \mu_k)^2} \right)^{\mathbb{1}_{\{Z_t = k\}}} \right] \\ &= \log(\pi_{Z_1}) + \sum_{t=2}^{T} \log(P_{Z_{t-1},Z_t}) + \sum_{t=1}^{T} \sum_{k=0}^{1} \left[-\frac{1}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2}(X_t - \mu_k)^2 \right] \mathbb{1}_{\{Z_t = k\}}.\end{aligned}$$

The Q function is given by

$$\begin{aligned} Q(\boldsymbol{\mu}, \boldsymbol{\mu}') &= \mathbb{E}_{\boldsymbol{Z}}[\log p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\mu})|\boldsymbol{\mu}'] \\ &= \sum_{k=0}^{1} \log(\pi_k) p(\boldsymbol{X}, Z_1 = k|\boldsymbol{\mu}') + \sum_{i=0}^{1}\sum_{j=0}^{1}\sum_{t=1}^{T} \log(P_{ij}) p(\boldsymbol{X}, Z_{t-1} = i, Z_t = j|\boldsymbol{\mu}') \\ &\quad + \sum_{k=0}^{1}\sum_{t=1}^{T} \left[-\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}(X_t - \mu_k)^2 \right] p(\boldsymbol{X}, Z_t = k|\boldsymbol{\mu}') \end{aligned}$$

Taking derivative with respect to $\boldsymbol{\mu}$,

$$\frac{\partial Q(\boldsymbol{\mu}, \boldsymbol{\mu}')}{\partial \mu_k} = \sum_{t=1}^{T} \frac{1}{\sigma^2}(X_t - \mu_k) p(\boldsymbol{X}, Z_t = k|\boldsymbol{\mu}'), \qquad k = 0, 1$$

and setting them to zero yields

$$\mu_k = \frac{\sum_{t=1}^{T} X_t p(\boldsymbol{X}, Z_t = k|\boldsymbol{\mu}')}{\sum_{t=1}^{T} p(\boldsymbol{X}, Z_t = k|\boldsymbol{\mu}')} = \frac{\sum_{t=1}^{T} X_t \alpha_{tk}(\boldsymbol{\mu}')\beta_{tk}(\boldsymbol{\mu}')}{\sum_{t=1}^{T} \alpha_{tk}(\boldsymbol{\mu}')\beta_{tk}(\boldsymbol{\mu}')}.$$

[7]:
```
emit = function(mu,x){
    return(dnorm(x,mean=mu,sd=sd))
}

forw <- function(X, mu, K, P, pi){
    T <- length(X)
    alpha = matrix(nrow = T,ncol=K)
    for(k in 1:K){
      alpha[1,k] = pi[k] * emit(mu[k],X[1])
    }

    for(t in 1:(T-1)){
      m = alpha[t,] %*% P
      for(k in 1:K){
        alpha[t+1,k] = m[k]*emit(mu[k],X[t+1])
      }
    }
    return(alpha)
}

back <- function(X, mu, K, P){
    T <- length(X)
    beta = matrix(1, nrow = T,ncol=K)
    for(t in (T-1):1){
      for(k in 1:K){
        beta[t,k] = sum(beta[t+1,]*P[k,]*emit(mu,X[t+1]))
      }
```

```
    }
    return(beta)
}

EM <- function(X, mu, K, P, pi, epsilon=1e-5){
    T <- length(X)
    eps <- 1
    l <- c()
    while(eps>epsilon){
        # E step
        alpha <- forw(X, mu, K, P, pi)
        beta <- back(X, mu, K, P)

        # M step
        mu <- apply(X * alpha * beta, 2, sum)/apply(alpha * beta, 2, sum)

        l <- c(l, log(sum(alpha[T,])))
        if(length(l)>1){
            eps <- l[length(l)] - l[length(l)-1]
        }

    }
    return(list(mu=mu,l=l))
}
```

2. Check your implementation by running it on the example and seeing that the log-likelihood is increasing. [Hint: note that the forwards algorithm gives you the likelihood.]

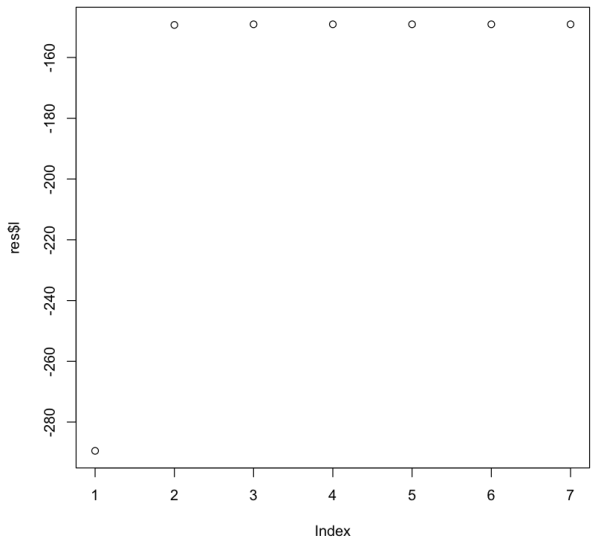
[8]:
```
set.seed(1)
T = 200
K = 2
sd= 0.4
P = cbind(c(0.9,0.1),c(0.1,0.9))

# Simulate the latent (Hidden) Markov states
Z = rep(0,T)
Z[1] = 1
for(t in 1:(T-1)){
  Z[t+1] = sample(K, size=1, prob=P[Z[t],])
}

# Simulate the emitted/observed values
X= rnorm(T,mean=Z,sd=sd)
#Assumed prior distribution on Z_1
pi = c(0.5,0.5)

res <- EM(X,c(0.5,2.5), K, P, pi)
plot(res$l)
cat(res$mu)
```

```
0.9865201 2.022183
```

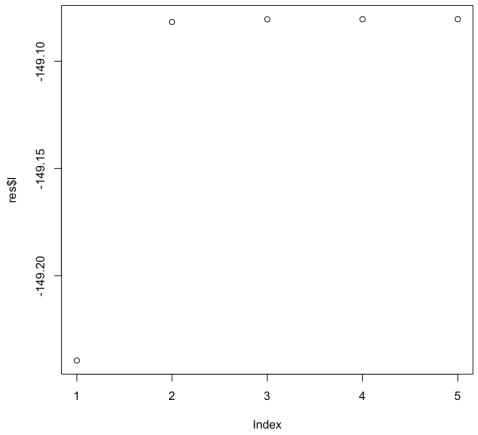


As we can see, the log-likelihood is increasing.

3. Try running the EM algorithm multiple times from different starting points. Does it get stuck in local optima of the log-likelihood?

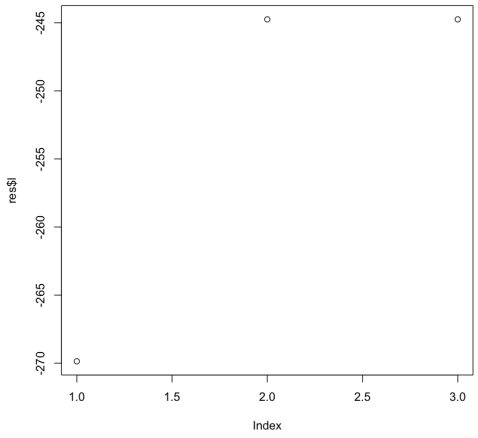
[9]:
```
res <- EM(X,c(2.0,1.0), K, P, pi)
plot(res$l)
cat(res$mu)
```

```
2.022178 0.9865181
```



[10]:
```
res <- EM(X,c(1.5,1.5), K, P, pi)
plot(res$l)
cat(res$mu)
```

```
1.299578 1.299578
```



The algorithm gets stuck in local optima when $\mu_{init} = (2, 1)$ and $\mu_{init} = (1.5, 1.5)$.