

# Assignment 6

Yelie Yuan

9/28/2019

1. Follow the lecture notes to verify the validity of the provided E- and M-steps. That is, derive the updating rules in the given algorithm based on the construction of an EM algorithm.

Density of  $y_i$  is

$$f(y_i | \mathbf{x}_i, \Psi) = \sum_{j=1}^m \pi_j \phi(y_i; \mathbf{x}_i^\top \beta_j, \sigma^2), \quad i = 1, \dots, n,$$

So density of  $y_i - \mathbf{x}_i^\top \beta_j$  is

$$f(y_i - \mathbf{x}_i^\top \beta_j | \Psi) = \sum_{j=1}^m \pi_j \phi(y_i - \mathbf{x}_i^\top \beta_j; 0, \sigma^2), \quad i = 1, \dots, n,$$

From the notes, we have

$$\begin{aligned} Q(\theta | \theta_t) &= \sum_z p(z | \mathbf{x}, \theta_t) \ln p(\mathbf{x}, z | \theta) \\ &= \sum_{i=1}^n \sum_{k=1}^K p(z_i = k | \mathbf{x}_i, \theta_t) \ln p(z_i = k, \mathbf{x}_i | \theta) \end{aligned}$$

In this exercise,

$$Q(\Psi | \Psi^{(k)}) = \sum_{i=1}^n \sum_{j=1}^m p(z_{ij} | y_i, \mathbf{x}_i, \Psi^{(k)}) \log p(z_{ij}, y_i, \mathbf{x}_i | \Psi)$$

By Bayes rule,

$$\begin{aligned} p(z_{ij} | y_i, \mathbf{x}_i, \Psi^{(k)}) &= \frac{p(z_{ij}, y_i, \mathbf{x}_i | \Psi^{(k)})}{p(y_i, \mathbf{x}_i | \Psi^{(k)})} \\ &= \frac{p(z_{ij}, y_i, \mathbf{x}_i | \Psi^{(k)})}{\sum_{j=1}^m p(z_{ij}, y_i, \mathbf{x}_i | \Psi^{(k)})} \\ &= \frac{\pi_j^{(k)} \phi(y_i - \mathbf{x}_i^\top \beta_j^{(k)}; 0, \sigma^{2^{(k)}})}{\sum_{j=1}^m \pi_j^{(k)} \phi(y_i - \mathbf{x}_i^\top \beta_j^{(k)}; 0, \sigma^{2^{(k)}})} \\ &= p_{ij}^{(k+1)} \\ p(z_{ij}, y_i, \mathbf{x}_i | \Psi) &= \pi_j \phi(y_i - \mathbf{x}_i^\top \beta_j; 0, \sigma^2) \end{aligned}$$

So we have

$$\begin{aligned}
Q(\Psi \mid \Psi^{(k)}) &= \sum_{i=1}^n \sum_{j=1}^m p_{ij}^{(k+1)} \{\log \pi_j + \log \phi(y_i - \mathbf{x}_i^\top \beta_j; 0, \sigma^2)\} \\
&= \sum_{i=1}^n \sum_{j=1}^m p_{ij}^{(k+1)} \log \pi_j + \sum_{i=1}^n \sum_{j=1}^m p_{ij}^{(k+1)} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(y_i - \mathbf{x}_i^\top \beta_j)^2}{2\sigma^2}\right\} \\
&= \sum_{i=1}^n \sum_{j=1}^m p_{ij}^{(k+1)} \log \pi_j - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^m p_{ij}^{(k+1)} \log 2\pi\sigma^2 - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^m p_{ij}^{(k+1)} \frac{(y_i - \mathbf{x}_i^\top \beta_j)^2}{\sigma^2} \\
&= I_1 - \frac{1}{2} I_2 - \frac{1}{2} I_3
\end{aligned}$$

The M-Step maximizes  $Q(\Psi \mid \Psi^{(k)})$ , for  $I_1$ , we have

$$\begin{cases} \pi_1 + \pi_2 + \dots + \pi_m = 1 \\ L(\pi_1, \dots, \pi_m) = \sum_{i=1}^n \sum_{j=1}^m p_{ij}^{(k+1)} \log \pi_j - \lambda \{\sum_{j=1}^m \pi_j - 1\} \end{cases}$$

with  $\lambda$  a Lagrange multiplier. Then

$$\pi_j = \frac{\sum_{i=1}^n p_{ij}^{(k+1)}}{\sum_{i=1}^n \sum_{j=1}^m p_{ij}^{(k+1)}} = \frac{1}{n} \sum_{i=1}^n p_{ij}^{(k+1)}$$

Only  $I_3$  contains  $\beta_j$ , to minimize it, each  $I_{3j}$  must be minimized. For each  $\beta_j$ , take derivative of  $I_{3j}$  and let it equal to 0. We have

$$\sum_{i=1}^n p_{ij}^{(k+1)} (-2\mathbf{x}_i y_i + 2\mathbf{x}_i \mathbf{x}_i^\top \beta_j) = -2 \sum_{i=1}^n p_{ij}^{(k+1)} \mathbf{x}_i y_i + 2 \sum_{i=1}^n p_{ij}^{(k+1)} \mathbf{x}_i \mathbf{x}_i^\top \beta_j = 0$$

Solve this equation, we have

$$\beta_j = \left( \sum_{i=1}^n \mathbf{x}_i \mathbf{x}_i^\top p_{ij}^{(k+1)} \right)^{-1} \left( \sum_{i=1}^n \mathbf{x}_i p_{ij}^{(k+1)} y_i \right)$$

$I_2$  and  $I_3$  contain  $\sigma^2$ , take the derivative of  $I_2 + I_3$  with respect to  $\sigma^2$  and let it equals 0, we have

$$\frac{1}{\sigma^2} \sum_{i=1}^n \sum_{j=1}^m p_{ij}^{(k+1)} - \frac{1}{\sigma^4} \sum_{i=1}^n \sum_{j=1}^m p_{ij}^{(k+1)} (y_i - \mathbf{x}_i^\top \beta_j)^2 = 0$$

Then we have

$$\sigma^2 = \frac{\sum_{i=1}^n \sum_{j=1}^m p_{ij}^{(k+1)} (y_i - \mathbf{x}_i^\top \beta_j)^2}{\sum_{i=1}^n \sum_{j=1}^m p_{ij}^{(k+1)}} = \frac{\sum_{i=1}^n \sum_{j=1}^m p_{ij}^{(k+1)} (y_i - \mathbf{x}_i^\top \beta_j)^2}{n}$$

To sum up, at the  $k$ th iteration, the updating rules are as follows (notice that we need to update  $\beta_j^{(k+1)}$  before we use it to update  $\sigma^{2(k+1)}$ )

$$\begin{cases} \pi_j^{(k+1)} = \frac{1}{n} \sum_{i=1}^n p_{ij}^{(k+1)} \\ \beta_j^{(k+1)} = \left( \sum_{i=1}^n \mathbf{x}_i \mathbf{x}_i^\top p_{ij}^{(k+1)} \right)^{-1} \left( \sum_{i=1}^n \mathbf{x}_i p_{ij}^{(k+1)} y_i \right) \\ \sigma^{2(k+1)} = \frac{\sum_{i=1}^n \sum_{j=1}^m p_{ij}^{(k+1)} (y_i - \mathbf{x}_i^\top \beta_j^{(k+1)})^2}{n} \end{cases}$$

2. Implement this algorithm in R with a function `regmix_em`. The inputs of the functions are `y` for the response vector, `xmat` for the design matrix, `pi.init` for initial values of  $\pi_j$ 's (a vector of  $K \times 1$  vector), `beta.init` for initial values of  $\beta_j$ 's (a matrix of  $p \times K$  where  $p$  is `ncol(xmat)` and  $K$  is the number of components in the mixture), `sigma.init` for initial values of  $\sigma$ , and a control list for controlling max iteration number and convergence tolerance. The output of this function is the EM estimate of all the parameters.

In this part, we need to translate the prior maximization rules to codes. It will be more readable if we separate the rules as several parts. Firstly, write the following functions to update the parameters. `p_ij_fun` will be used to update  $p_{ij}$ , `beta_fun` will be used to update  $\beta$ , `sigma_fun` will be used to update  $\sigma$ . As for  $\pi_j$ , it's relatively easy to compute once we have  $p_{ij}$ , we will calculate it in the function `regmix_em`. And we need to get the length of  $\pi_j$ , number of components we have, for future use.

```
p_ij_fun <- function(y, xmat, pi, beta, sigma) {
  m <- length(pi)
  p_ij <- matrix(0, n, m)
  for (j in 1:m) {
    p_ij[, j] <- pi[j] * sapply(y - xmat %*% beta[, j], dnorm, mean = 0, sd = sigma)
  }
  return(p_ij / apply(p_ij, 1, sum))
}

beta_fun <- function(y, xmat, p_ij, beta) {
  for (j in 1:m) {
    temp1 <- 0
    temp2 <- 0
    p <- NROW(beta)
    for (i in 1:n) {
      temp1 <- temp1 + matrix(xmat[i, ], p, 1) %*% xmat[i, ] * p_ij[i, j]
      temp2 <- temp2 + matrix(xmat[i, ], p, 1) * p_ij[i, j] * y[i]
    }
    beta[, j] <- solve(temp1, temp2)
  }
  return(beta)
}

sigma_fun <- function(y, xmat, p_ij, beta) {
  temp3 <- 0
  for (j in 1:m) {
    temp3 <- temp3 + sum(p_ij[, j] * (y - xmat %*% beta[, j])^2)
  }
  return((temp3 / n)^0.5)
}
```

Then, `regmix_em` can be written as following. Set initial values of each parameter, store the results of  $\beta_t$  and  $\pi_t$  in  $k$ th iteration and use them to update the parameters in the next iteration. Once the code reached the max step or  $\beta$  and  $\pi$  stopped updating, the code will stop and return the parameters. There are four components in the result. The first one is our iteration steps, the second part is  $\beta$ , the third part is  $\pi$ , the last one is  $\sigma$ .

```
regmix_em <- function(y, xmat, pi.init, beta.init,
                      sigma.init, control) {
  tolerance <- 1
  steps <- 0
  xmat <- as.matrix(xmat)
  y <- as.matrix(y)
```

```

pi <- pi.init
beta <- beta.init
sigma <- sigma.init
while (steps < control$maxit & tolerance > control$tol) {
  beta_t <- beta
  pi_t <- pi

  p_ij <- p_ij_fun(y, xmat, pi_t, beta_t, sigma)
  pi <- apply(p_ij, 2, sum) / n
  beta <- beta_fun(y, xmat, p_ij, beta_t)
  sigma <- sigma_fun(y, xmat, p_ij, beta)

  steps <- steps + 1
  tolerance <- sum(abs(beta_t - beta)) + sum(abs(pi_t - pi))
}
return(list(Step = steps, Beta = beta, Pi = pi, Sigma = sigma))
}

```

3. Here is a function to generate data from the mixture regression model.

```

regmix_sim <- function(n, pi, beta, sigma) {
  K <- ncol(beta)
  p <- NROW(beta)
  xmat <- matrix(rnorm(n * p), n, p) # normal covaraites
  error <- matrix(rnorm(n * K, sd = sigma), n, K)
  ymat <- xmat %*% beta + error # n by K matrix
  ind <- t(rmultinom(n, size = 1, prob = pi))
  y <- rowSums(ymat * ind)
  data.frame(y, xmat)
}

```

Generate data with the following and estimate the parameters.

```

n <- 400
pi <- c(.3, .4, .3)
bet <- matrix(c( 1, 1, 1,
                -1, -1, -1), 2, 3)
sig <- 1
set.seed(1205)
dat <- regmix_sim(n, pi, bet, sig)
## regmix_em(y = dat[,1], xmat = dat[,-1],
##           pi.init = pi / pi / length(pi),
##           beta.init = bet * 0,
##           sigma.init = sig / sig,
##           control = list(maxit = 500, tol = 1e-5))

```

Use the following code and initial values to estimate the parameters.

```

regmix_em(y = dat[,1], xmat = dat[,-1],
          pi.init = pi / pi / length(pi),
          beta.init = matrix(c(1, 2, 3, -2, -1, -4), 2, 3),
          sigma.init = sig / sig,
          control = list(maxit = 500, tol = 1e-5))

```

```

## $Step
## [1] 53

```

```
##
## $Beta
##      [,1]      [,2]      [,3]
## [1,] 0.8796608 0.9911851 -0.9136978
## [2,] 0.9341963 -1.2424571 -1.1990372
##
## $Pi
## [1] 0.3858219 0.2687873 0.3453908
##
## $Sigma
## [1] 1.023598
```

We can set the initial values of  $\beta$  randomly, but the order of  $\beta_j$  in our results might be different. The original  $\beta$  we used to generate  $y$  is  $\beta_1 = [1; 1], \beta_2 = [1; -1], \beta_3 = [-1; -1]$ . But our estimated  $\beta_j$  might have a different order according to the original order, for example our estimated result can be  $\beta_1 \approx [-1; -1], \beta_2 \approx [1; -1], \beta_3 \approx [1; 1]$ . But it doesn't mean our estimated results are wrong. If we check  $p_{ij}$ , the possibilities of every observation from each component, we can find that the values are basically same. The different order is just different sequence of group. In general, the same group still has the same "members".