Expectation-Maximization Algorithm

Tomas Meade

Part (a)

The parameters we are trying to estimate are given by $\theta = (\beta_0, \beta_1, \sigma^2)$. Assuming Y_i $N(\beta_0 + \beta_1 x_i, \sigma^2)$, then we are maximizing the expectation of the following log-likelihood function,

$$logL(\theta) = \sum_{i=1}^{n} -\frac{1}{2} log(2\pi\sigma^2) - \frac{(y_i - b_0 - b_1 x_i)^2}{2\sigma^2}.$$

So for the algorithm, we will maximize the expectation of the log-likelihood above given the current value of θ, which we can call θ_t and observed data.

For the expectation at each interation, $x_i, \sigma^2, \beta_0, \beta_1$ are all fixed. The only term that involves randomness is y_i.

Thus, we can split the log-likelihood into two sums,

$$logL(\theta) = \sum_{i=1}^{n-n_c} -\frac{1}{2} log(2\pi\sigma^2) - \frac{(y_i - b_0 - b_1 x_i)^2}{2\sigma^2} + \sum_{j=n-n_c}^{n} -\frac{1}{2} log(2\pi\sigma^2) - \frac{(y_j - b_0 - b_1 x_j)^2}{2\sigma^2}$$

where the y_i in the first sum are fixed and the y_j in the second sum on the right are random. The first sum is constant, so taking the expectation gives,

$$logL(\theta) = \sum_{i=1}^{n-n_c} -\frac{1}{2} log(2\pi\sigma^2) - \frac{(y_i - b_0 - b_1 x_i)^2}{2\sigma^2} + \sum_{j=n-n_c}^{n} -\frac{1}{2} log(2\pi\sigma^2) - \frac{E[y_j^2] - 2E[y_j](b_0 - b_1 x_j) + (b_0 + b_1 x_i)^2}{2\sigma^2}.$$

Given a set threshold value τ we know that $E[y_j] = \mu_j + \sigma_j \rho(\tau*)$ and $V[y_j] = \sigma_j^2(1 + \tau * \rho(\tau*) - \rho(\tau*)^2)$ where $\rho(\tau*) = \frac{\phi(\tau*)}{1-\Phi(\tau*)}$ and $\tau* = (\tau - \mu_j)/\sigma_j$. Using these we can then also solve for the second moment since $E[y_j^2] = V[y_j] + E[y_j]^2$.

Now we have an equation that we can analytically maximize to find optimal *theta*. For ease of notation, let $\sum_o = \sum_{i=1}^{n-n_c}$ or in other words that we are summing over the observed/uncensored data and let $\sum_u = \sum_{i=n-n_c+1}^{n}$ or that we are summing over the unobserved/censored data. Also, let $Y_i^t = E[y_j]$ and let $Y_i^{t^2} = E[y_j^2]$

Taking the derivative with respect to each parameter and setting equal to zero gives,

$$\frac{\partial l(\theta)}{\partial \beta_0} = 1/\sigma^2 \sum_o (y_i - \beta_0 - \beta_1 x_i) + 1/\sigma^2 \sum_u (Y_i^t - \beta_0 - \beta_1 x_i) = 0$$

$$\hat{\beta_0} = 1/n(\sum_o y_i + \sum_u Y_i^t) - \hat{b_1}\bar{x}$$

$$\frac{\partial l(\theta)}{\partial \beta_1} = 1/\sigma^2 \sum_o (y_i - \beta_0 - \beta_1 x_i)(-x_i) + 1/\sigma^2 \sum_u (Y_i^t - \beta_0 - \beta_1 x_i)(-x_i) = 0$$

$$\hat{\beta}_1 = \frac{(\sum_o x_i y_i + \sum_u x_i Y_i^t) - \bar{x}(\sum_o y_i + \sum_u Y_i^t)}{\sum_{i=1}^n x_i^2 - n\bar{x}^2}$$

$$\frac{\partial l(\theta)}{\partial \sigma^2} = -n/2\sigma^2 + 1/2\sigma^4 \sum_o (y_i - \beta_0 - \beta_1 x_i)^2 + 1/2\sigma^4 \sum_u (Y_i^t - \beta_0 - \beta_1 x_i)^2 = 0$$

$$\hat{\sigma}^2 = 1/n(\sum_o (y_i - \beta_0 - \beta_1 x_i)^2 + \sum_u (Y_i^{t^2} - 2(\beta_0 - \beta_1 x_i)Y_i^t + (\beta_0 + \beta_1 x_i)^2))$$

The algorithm will compute the optimal values of θ using the above estimators and some initial values and then repeat the process using the optimal values found in the previous iteration. Using the above equations, β_1 needs to be solved for first and then plugged in to find β_0 and then both of these plugged in to find σ^2.

Part (b)

Reasonable starting values are any values that are on the scale of the observations and where the value of the variance is not something that would not make sense in the context of the problem like zero or negative variance. One way to pick starting values would be to use β_0 and β_1 from a simple linear regression model on the observed or uncensored data. And then use the variance of the observed y_i as the initial variance. The code below does this.

```
# Set up data

## problem 3

set.seed(1)
n <- 100
beta0 <- 1
beta1 <- 2
sigma2 <- 6

x <- runif(n)
yComplete <- rnorm(n, beta0 + beta1*x, sqrt(sigma2))

sum(yComplete > 4)/length(yComplete)
```

```
## [1] 0.2
```

```
## Parameters above were chosen such that signal in data is moderately strong.
## The estimate divided by std error is approximately 3.
mod <- lm(yComplete ~ x)
summary(mod)$coef
```

```
##              Estimate Std. Error  t value    Pr(>|t|)
## (Intercept) 0.5607442  0.5041346 1.112290 0.268734381
## x           2.7650812  0.8657927 3.193699 0.001889262
```

```
# Set Up data with 20% exceedances
t <- quantile(yComplete, 0.8)

y <- yComplete[yComplete <= t]

x_with_y <- x[yComplete <= t]
```

```r
x_no_y <- x[yComplete > t]
```

```r
# Get starting values

# Fit linear model to find initial b0 and b1
mod <- lm(y ~ x_with_y)
summary(mod)$coef
```

```
##              Estimate Std. Error   t value    Pr(>|t|)
## (Intercept) 0.1440021  0.3838236 0.3751778 0.708546520
## x_with_y    1.9287315  0.6887203 2.8004569 0.006430566
```

```r
b0 <- summary(mod)$coef[1]
b1 <- summary(mod)$coef[2]

# Get initial variance
sigma2 <- var(y)

# Show starting values
theta_0 <- c(b0, b1, sigma2)

print(theta_0)
```

```
## [1] 0.1440021 1.9287315 2.8345110
```

Based on the code above the approximate initial value for β_0 would be 0.144, the initial value for β_1 would be 1.929 and the initial value for $sigma^2$ would be 2.835

Part (c)

To implement the algorithm I made a helper function (update_theta()) that does one update to θ and an overall function (EM()) that estimates the parameters by using update_theta repeatedly. In my EM function, I added a variable that allows the user to specify the maximum amount of iterations the algorithm will do. For my stopping criteria, I check to see if the sum of the absolute value of the difference between the value of each of the parameters at the current iteration and their value in the previous iteration is less than 1×10^{-9}.

```r
# Function to update theta once
update_theta <- function(x, y, x_no_y, x_with_y, b01, b11, sig21, t){

  # Get t*
  t1 <- (t-(b01+b11*x_no_y))/sqrt(sig21)

  # Get p*(t*)
  p1 <- dnorm(t1)/(1-pnorm(t1))

  # Get first moment
  Yi <- b01 + b11*x_no_y + sqrt(sig21)*p1

  # Get second moment
  Yi2 <- sig21*(1 + t1*p1 - p1^2) + Yi^2

  # Estimate b1
  b12 <- ((sum(x_with_y*y) + sum(x_no_y*Yi)) - mean(x)*(sum(y) + sum(Yi)))/(sum(x^2) - length(x)*mean(x

  # Estimate b0
  b02 <- ((sum(y) + sum(Yi))/length(x)) - b12*mean(x)
```

```
  #Estimate sigma^2
  sig22 <- (sum((y - b02 - b12*x_with_y)^2) + sum(Yi2 - 2*(b02 + b12*x_no_y)*Yi + (b02 + b12*x_no_y)^2)
  return(c(b02, b12, sig22))


}


# Overall EM function
EM <- function(x, y, x_no_y, x_with_y, b01, b11, sig21, t, iter){

  # Store initial values
  theta_0 <- c(b01, b11, sig21)

  # Do one update
  theta_t <- update_theta(x, y, x_no_y, x_with_y, b01, b11, sig21, t)

  # Iteratively update until the sum of the absolute
  for (i in 1:iter){

    prev <- theta_t

    theta_t <- update_theta(x, y, x_no_y, x_with_y, theta_t[1], theta_t[2], theta_t[3], t)

    if(sum(abs(theta_t - prev)) < 1*10^-9){

      return(theta_t)
    }
  }
  return(theta_t)
}
```

```
b0 <- theta_0[1]
b1 <- theta_0[2]
sig2 <- theta_0[3]

# Test one update
update_theta(x, y, x_no_y, x_with_y, b0, b1, sig2, t)
```

```
## [1] 0.4650357 2.6168434 3.9550767
```

```
# Estimate parameters
EM(x, y, x_no_y, x_with_y, b0, b1, sig2, t, iter = 1000)
```

```
## [1] 0.4566128 2.8241081 4.6188762
```

Using the starting values specified in part (b) and 20% exceedances, the EM function estimates the parameters to be $\hat{\theta} = (\hat{\beta}_0 = 0.4566128, \hat{\beta}_1 = 2.8241081, \hat{\sigma}^2 = 4.6188760)$

```
# Set Up data with 80% exceedances
t <- quantile(yComplete, 0.2)

y <- yComplete[yComplete <= t]
```

```r
x_with_y <- x[yComplete <= t]

x_no_y <- x[yComplete > t]

# Estimate parameters

EM(x, y, x_no_y, x_with_y, b0, b1, sig2, t, iter = 1000)
```

```
## [1] 0.3126394 2.8792202 3.8419643
```

Again using the starting values specified in part (b), but this time with 80% exceedances, the EM function estimates the parameters to be $\hat{\theta} = (\hat{\beta}_0 = 0.3126384, \hat{\beta}_1 = 2.8792170, \hat{\sigma}^2 = 3.8419568)$. These values are relatively similar to 20% exceedances.

Part (d)

I wrote a function below that directly maximizes the log-likelihood of the observed data. To do this I separated the log-likelihood into two sums again, the first being the sum over the uncensored x's and y's and the second being a sum of the censored data. For the first sum, I just used dnorm with mean given by $\beta_0 + \beta_1 x_i$ where the x_i are the x's with uncensored y's and variance equal to σ^2. For the second sum, I used pnorm with mean given by $\beta_0 + \beta_1 x_j$ where the x_j are the x's with censored y's and then variance equal to σ^2. I then used optim to find the estimated parameters and then used the Hessian to find the standard errors.

The estimated parameters using this method and 20% exceedances were $\hat{\theta} = (\hat{\beta}_0 = 0.4577333, \hat{\beta}_1 = 2.8221096, \hat{\sigma}^2 = 4.6194727)$. This is almost exactly what the estimated parameters were using the EM algorithm in part (c) which should be the case. The standard error of $\beta_0, \beta_1, \sigma^2$ were 0.4772766, 0.8308774 and 0.7670780, respectively.

The estimated parameters using this method and 80% exceedances were $\hat{\theta} = (\hat{\beta}_0 = 0.4577333, \hat{\beta}_1 = 2.8221096, \hat{\sigma}^2 = 4.6194727)$. This is also almost exactly what the estimated parameters were using the EM algorithm in part (c). The standard error of $\beta_0, \beta_1, \sigma^2$ for this test case were 0.572204 1.135917, 1.430705, respectively.

```r
# log-likelihood function
lik <- function(theta) {
  b0 <- theta[1]
  b1 <- theta[2]
  sig2 <- theta[3]

  obs <- sum(dnorm(y, mean = b0 + b1*x_with_y, sd = sqrt(sig2), log = TRUE))

  unobs <- sum(pnorm(t, mean = b0 + b1*x_no_y, sd = sqrt(sig2), log.p = TRUE, lower.tail = FALSE))

  return(-(obs + unobs))
}

# Reset test case for 20% exceedances
t <- quantile(yComplete, 0.8)

y <- yComplete[yComplete <= t]

x_with_y <- x[yComplete <= t]

x_no_y <- x[yComplete > t]

# Optimize using optim function
```

```r
model <- optim(c(b0, b1, sig2), lik, method = "BFGS", hessian = TRUE)

# Output model and estimates
model
```

```
## $par
## [1] 0.4577333 2.8221096 4.6194727
##
## $value
## [1] 195.0124
##
## $counts
## function gradient
##       14        8
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##            [,1]       [,2]       [,3]
## [1,] 20.6759383 10.5585450 -0.7456167
## [2,] 10.5585450  6.8453925 -0.4726718
## [3,] -0.7456167 -0.4726718  1.7321995
```

```r
# Output standard error of the estimates
sqrt(diag(solve(model$hessian)))
```

```
## [1] 0.4772766 0.8308774 0.7670780
```

```r
# Reset test case for 80% exceedances
t <- quantile(yComplete, 0.2)

y <- yComplete[yComplete <= t]

x_with_y <- x[yComplete <= t]

x_no_y <- x[yComplete > t]

# Optimize using optim function
model <- optim(c(b0, b1, sig2), lik, method = "BFGS", hessian = TRUE)

# Output model and estimates
model
```

```
## $par
## [1] 0.312670 2.879223 3.842109
##
## $value
## [1] 69.01469
##
## $counts
## function gradient
##       14        7
```

```
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##           [,1]      [,2]      [,3]
## [1,] 13.155747  5.779679 -3.451085
## [2,]  5.779679  3.436892 -1.779674
## [3,] -3.451085 -1.779674  1.471200
```

```
# Output standard error of the estimates
sqrt(diag(solve(model$hessian)))
```

```
## [1] 0.572204 1.135917 1.430705
```