EM Algorithm

Tzu Feng Leung

20 April, 2022

EM Algorithm for missing data.

Goal:

Consider a censored regression problem. We assume a simple linear regression model, $Y_i \sim N(\beta_0 + \beta_1 x_i, \sigma^2)$. Suppose we have an iid sample, but that for any observation with $Y > \tau$, all we are told is that Y exceeded the threshold and not its actual value. In a given sample, n_c of the n observations will (in a stochastic fashion) be censored, depending on how many exceed the fixed τ. A real world example (but with censoring in the left tail) is in measuring pollutants, for which values below a threshold are reported as below the limit of detection. Another real world example is US tax revenue data where the incomes of wealthy taxpayers may be reported as simply exceeding, say, 1 million dollars.

Propose reasonable starting values for the three parameters as functions of the observations.

A reasonable starting values for $\theta_0 = (\beta_{0,0}, \beta_{1,0}, \sigma_0^2)$ would be the parameters we get from running regression on the uncensored data.

Write an R function, with auxiliary functions as needed, to estimate the parameters. Make use of the initialization from part (b). You may use lm() for updating β. You'll need to include criteria for deciding when to stop the optimization.

Test your function using data simulated based on the code in ps8.R with

- (a) a modest proportion of exceedances expected, say 20%, and
- (b) a high proportion, say 80%.

```r
## data given
set.seed(1)
n <- 100
beta0 <- 1
beta1 <- 2
## variance
sigma2 <- 6
```

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

## Parameters above were chosen such that signal in data is moderately strong.
## The estimate divided by std error is approximately 3.
## 2.74/.86 = 3.18
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
```

```
## For problem 3c, you'll need to simulate the censoring process.

## I will use quantile to split the data to find tau
tau20 <- quantile(yComplete, .80) # this tau will gives us 20% censored data
tau80 <- quantile(yComplete, .20) # this tau will gives us 80% censored data

##
df = data.frame(x = x, y = yComplete)
```

```
get_regression_results <- function(df){
  ## extract b0, b1, and sigma^2 from linear model Y~X
  fitted_lm = lm(df$y~df$x)
  beta_0 = fitted_lm$coefficients[1]
  beta_1 = fitted_lm$coefficients[2]
  sigma = summary(fitted_lm)$sigma
  return(c(beta_0, beta_1, sigma^2))
}

EM_Algorithm = function(df, tau = 4, max_iterations = 100, tol = 1e-6){

  iteration = 1

  censored_data <- df[ df$y > tau, ]
  uncensored_data <- df[ !(df$y > tau), ]

  n= nrow(df)

  ## use the uncensored data to generate the starting parameters
  initial_para = get_regression_results(uncensored_data)

  ## this var is for determining how close the t and t+1 parameters
  old_minus_new_para = c(1,1,1)
  while ( (iteration <= max_iterations) && !all(abs(old_minus_new_para) < tol) ){

    ## keep going if iteration < max iteration
    ##  and if the abs difference of the are greater than tau

    beta0_t = initial_para[1]
    beta1_t = initial_para[2]
    sigma2_t = initial_para[3]

    #### This is the E step
    ####
    mu_t = beta0_t + beta1_t * censored_data$x
    tau_star = (tau- mu_t) / sqrt(sigma2_t)
    rho_star = dnorm(tau_star) / (1-pnorm(tau_star))
    E_Z = mu_t + sqrt(sigma2_t) * rho_star
    V_Z =sigma2_t * (1+ tau_star * rho_star - (rho_star)^2)
    E2_Z = V_Z + (E_Z )^2


    #### This is the M step
    ####
    temp1 = sum(uncensored_data$x * uncensored_data$y ) +
              sum(censored_data$x * E_Z) -
              mean(df$x) * ( sum(E_Z) +  sum(uncensored_data$y))
    temp2 = sum(df$x^2) - n*(mean(df$x))^2
    beta1_new = temp1 / temp2
    ##
    beta0_new = 1/n * ((sum(E_Z) + sum(uncensored_data$y))) - beta1_new * mean(df$x)
    ##
    temp3 <- sum((uncensored_data$y - (beta0_new + beta1_new*uncensored_data$x))^2)
    temp4 <- sum(E2_Z - 2*( beta0_new + beta1_new * censored_data$x)*E_Z +
                   (beta0_new + beta1_new * censored_data$x)^2)
    sigma2_new = 1/n * (temp3 + temp4)

    ## save the results
    new_para = c(beta0_new, beta1_new, sigma2_new)

    old_minus_new_para = new_para - initial_para
    # update parameters
    initial_para = new_para

    ## next iteration
    iteration = iteration+1

  }
  ##
  print(paste('The EM Algorithm ends at iteration: ', iteration))
  return(initial_para)
}
EM_Algorithm(df, tau = tau20)
```

```
## [1] "The EM Algorithm ends at iteration:  15"

## [1] 0.4566128 2.8241081 4.6188760
```

```
EM_Algorithm(df, tau = tau80)
```

```
## [1] "The EM Algorithm ends at iteration:  101"

## [1] 0.3125087 2.8787937 3.8409602
```

We know that the true answer should be (1,2,6). And "tau20" gave us a better approximation because the data set contains less censored data. It makes sense that "tau80" will give us a poor approximate because most of the data are cencorsed;

3d

A different approach to this problem just directly maximizes the log-likelihood of the observed data, which for the censored observations just involves the likelihood terms, $P(Y_i > \tau)$.

Estimate the parameters (and standard errors) for your test cases using optim() with the BFGS option in R. You will want to consider reparameterization, and possibly use of the parscale argument. Compare how many iterations EM and BFGS take.

Note that parts (c) and (d) together provide a nice test of your EM derivation and code, since you should get the same results from the two optimization approaches.

Here are the codes:

```r
NegLogLik = function(para, df){
  ##
  beta0=para[1]
  beta1=para[2]
  sigma2=para[3]
  ## split the log lik into 2 parts: uncensored and censored
  mu = beta0 + beta1 * df$x
  Prob_Uncen = sum(dnorm(df$y[df$y <= tau], mu[df$y <= tau],
                sqrt(sigma2), log = TRUE)) # uncensored data
  ##
  Prob_Cen = sum(pnorm(tau, mu[df$y > tau], sqrt(sigma2),
                lower.tail = FALSE, log = TRUE) )# censored data
  return(-(Prob_Uncen + Prob_Cen))
}
# use tau20 to test optimization results
tau = tau20
censored_data <- df[ df$y > tau, ]
uncensored_data <- df[ !(df$y > tau), ]

starting_para <- get_regression_results(uncensored_data)
tau = 4
##
result <-optim(starting_para,
      fn = NegLogLik, df = df,
      method = "BFGS",
      hessian = TRUE)
result
```

```
## $par
## (Intercept)        df$x
##   0.4563302   2.8218980   4.6100165
##
## $value
## [1] 194.9244
##
## $counts
## function gradient
##       30       11
##
## $convergence
## [1] 0
##
## $message
## NULL
##
## $hessian
##              (Intercept)       df$x
## (Intercept)  20.7173985 10.5795182 -0.7477369
## df$x         10.5795182  6.8588952 -0.4746591
##              -0.7477369 -0.4746591  1.7400948
```

```
result$par
```

```
## (Intercept)        df$x
##   0.4563302   2.8218980   4.6100165
```

```
##
negHessian <- result$hessian
var_para <- diag(solve(negHessian))
se_para <- sqrt(var_para)
se_para
```

```
## (Intercept)        df$x
##   0.4767934   0.8300703   0.7653479
```

The optimal parameters are: $\beta_0, \beta_1, \sigma^2 = (0.4563302, 2.8218980, 4.6100165)$

- their se are $(0.4767934, 0.8300703, 0.7653479)$, respectively.

I found the se by calculating: $se = \Big[-\mathbb{E}(\text{Hessian}) \Big]^{-1}$.

Note that the second approach took 30 iterations while the EM algorithm only took 15 iterations. This means that the EM algorithm is more efficient. Both optimization methods gives the same results, this confirms that the EM algorithm is working properly.