Expectation Maximization Algorithms for Generalized Linear Models

Andrew Ma

January 19, 2022

Github: http://github.com/EM-aAlgorithm

1 Introduction

Why do we care about Expectation-Maximization Algorithms? In a sentence, they are very useful for fitting models onto data containing latent or truncated observations. For example, take a look at the following setup; note that there exist latent variables.

To this end, we will break this analysis down into four sections, each complete with examples.

First, let's define the random variables and parameters of the log likelihood function $L(\theta|X)$ that we want to maximize with our algorithm:

- $Y_{complete} = (X_i, Z_i)$
- X_i (Uncensored/Known Data)
- Z_i (Censored Data

i) E step: To maximize the quantity $Q(\theta, \theta^t)$, we first construct the log density based on our starting values

ii) M step: The M step consists of maximizing $Q(\theta, \theta^t)$ from the E-step **with respect to** θ, and updating this value to be our next theta in the E-step.

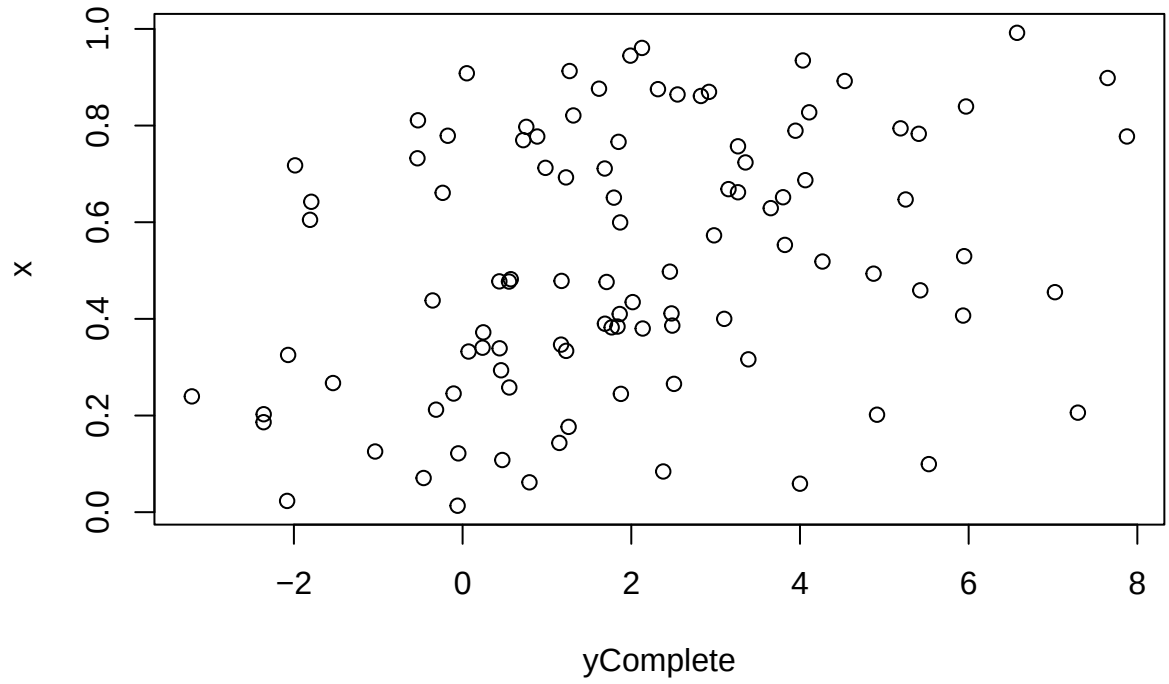
iii) Take the θ^t we get from the M step, and plug that in as θ for the E-step. While we are guaranteed to arrive at a local minimum, we should give deep consideration for our starting values. Repeat until convergence.

Reasonable starting values may be found by fitting a biased regression model on the truncated data since that is the best(and only) information we are given. I would say that testing with multiple starting values would also be smart because MLE's and log likelihoods can have multiple local minima that our algorithm may get "stuck" on.

```
## initialize testing data
set.seed(1)
n <- 100
beta0 <- 1; beta1 <- 2; 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.
mod <- lm(yComplete ~ x)
invisible(summary(mod))
```

```
plot(yComplete,x)
```




```
## helper functions
## ------------------------ Censor Data Function   -------------
## Input: x vector, complete original uncensored y data,
##threshold we want to censor (between 0 & 1)
## Returns a list of uncensored y values, censored y values,
##a combination of the former 2, indices for known & censored data,
## the tau value calculated from threshold given

censor_data <- function(x, yUncensored, thresh = 0.80){
  complete <- yUncensored
  tau <- as.numeric(quantile(complete, probs = thresh))
  known <- complete < tau
  censor <- complete > tau
  complete[complete > tau] <- tau
  yKnown <- complete[known]
  yCensored <- complete[censor]
  complete <- list(yKnown, yCensored, c(yKnown,yCensored),known, censor,tau)
  names(complete) <- c('yKnown' , 'yCensored', 'yTotal', 'indexKnown', 'indexCensored', 'tau')
```

```
  return(complete)
}

## ---------------------------- grab Starting values ---------------------------
## Input: x vector, complete original uncensored y data,
##threshold we want to censor (between 0 & 1)
## Fits a lm model on censored y values against x and uses those as starting points
getStart <- function(x, yUncensored, thresh = 0.80){

  begin <- rep(0,3)
  y <- censor_data(x, yUncensored , thresh = thresh)$yTotal
  values <- lm(y ~ x)

  ## set starting values
  begin[1] <- values$coefficients[1]
  begin[2] <- values$coefficients[2]
  begin[3] <- (summary(values)$sigma)**2

  return(begin)
}
```

```
## EM function
EM <- function(x, yUncensored, thresh = 0.8, max_iter = 1e5){

  #create censored data with threshold
  total <- censor_data(x, yComplete, thresh = thresh)

 #generate starting estimates from fitting censored regression
  begin <- getStart(x, total$yTotal, thresh = thresh)

  ## initialize parameters
  b0 <- begin[1]; b1 <- begin[2]; var1 <- begin[3]

  ## solve for expected and variance of the truncated normal
  ## that we assume our censored data comes from

  updateParam <- function(b0, b1, var1){
    mu_censored <-  b0 + (b1 * x[total$indexCensored])
    tau_star <- (total$tau - mu_censored)/sqrt(var1)
    rho <- dnorm(tau_star, mean = mu_censored, sd = sqrt(var1)) /
      (1-pnorm(tau_star, mean = mu_censored, sd = sqrt(var1)))

    EZ1 <- mu_censored + sqrt(var1)*rho
    VARZ1 <- var1 * (1 + (tau_star*rho) - rho^2)
    EZ21 <- VARZ1 + EZ1 # E(X^2) = VAR(X) + (EX)^2

    temp <- list(EZ1,EZ21); names(temp) <- c('E', 'E2')
    return(temp)
  }

  censor <- total$indexCensored
  known <- total$indexKnown
```

```
  Z <- updateParam(b0,b1,var1)
  count <- 0

  #loop through our values and get updated valies
  while(count <= max_iter) {

    b1_new <- (sum(x[known] * total$yKnown) + sum(x[censor]*Z$E) -
                 mean(x) * (sum(total$yKnown) + sum(Z$E)))/
      (sum(x*x) - length(x) * mean(x)^2)

    b0_new <- (sum(total$yKnown) + sum(Z$E))/length(x) - b1_new*mean(x)

    var1_new <- (sum((total$yKnown - b0_new - b1_new * x[known])^2) +
                 sum(Z$E2 - 2*(b0 + b1 * x[censor])*Z$E +
                       (b0_new + b1_new * x[censor])^2)) / length(x)

    #set updated values and recalculate parameters of our truncated normal
    b0 <- b0_new; b1 <- b1_new; var1 <- var1_new
    Z <- updateParam(b0,b1,var1)
    count <- count + 1

  }

  result <- list(b0,b1,var1,count)
  names(result) <- c('beta0', 'beta1','variance','count')

  return(result)
}
```

```
#20% of top values are censored
test1 <- EM(x,yComplete,thresh = 0.8)
cat("Estimates when censoring top 20% of data: \n", "beta0 = ", test1$beta0,
    "\n beta1 = ", test1$beta1,
    "\n sigma^2 =", test1$variance,
    "\n Iteration Count =", test1$count)
```

```
## Estimates when censoring top 20% of data:
##  beta0 =  0.327419
##  beta1 =  1.987912
##  sigma^2 = 2.296002
##  Iteration Count = 100001
```

```
#80% of top values are censored
test2 <- EM(x,yComplete,thresh = 0.2)
cat("Estimates when censoring top 80% of data: \n", "beta0 = ", test2$beta0,
    "\n beta1 = ", test2$beta1,
    "\n sigma^2 = ", test2$variance,
    "\n Iteration Count =", test1$count)
```

```
## Estimates when censoring top 80% of data:
##  beta0 =  -0.1659665
##  beta1 =  1.581843
```

```
##  sigma^2 =  2.114968
##  Iteration Count = 100001
```

Let Y be the complete data and Y_i be the variables in the uncensored subset of Y. Then the MLE of Y is given by $log([\prod_{i-1}^{n} \mathbb{P}(Y = Y_i)][\prod_{i-1}^{n} \mathbb{P}(Y > \tau)]) = log([\prod_{i-1}^{n} \mathbb{P}(Y = Y_i)]) + log([\prod_{i-1}^{n} \mathbb{P}(Y > \tau)]) = \sum_{i=1}^{n} log(\mathbb{P}(Y = Y_i)) + \sum_{i=1}^{n} log(\mathbb{P}(Y > \tau))$.

```
param <- getStart(x, yComplete)

logmax <- function(begin = eval(parse(text = 'param <- getStart(x, yComplete)')),
                   x, yUncensored, thresh = 0.8){

  ## ----- sanity check ----------
  validate_that(length(begin) > 0)
  validate_that(length(x) == length(yUncensored))
  validate_that(as.numeric(thresh) > 0 && as.numeric(thresh) < 1)

  #grab all components we need with our helper function
  total <- censor_data(x, yComplete, thresh = thresh)

  # initialize parameters
  b0 <- begin[1]; b1 <- begin[2] ; sig2 <- sqrt(begin[3])

  #mean for known and censored data
  mu_known <- b0 + b1 * x[total$indexKnown]
  mu_censored <-  b0 + b1 * x[total$indexCensored]

  #complete log-likelihood
  logscore <- sum(sum(dnorm(total$yKnown, mean = mu_known, sd = sig2, log = TRUE)),
                  sum(pnorm(total$tau, mean = mu_censored, sd = sig2,
                            log = TRUE,lower.tail = FALSE)))

  ## finding the max of a positive is the same as finding the minimum of its negative
  ## optim() defaults to minimization of a function

  return(-logscore)
}
```

```
# #optim starting at recommended default values
# testDefault<- optim(par =  eval(parse(text = 'param <- getStart(x, yComplete)')),
#                     x = x, yUncensored = yComplete,
#              fn = logmax, method = 'BFGS', hessian = TRUE)
#
#
# #optim starting very close to the true solution
# testOptim <- optim(par = c(5,2,6), x = x, yUncensored = yComplete,
#                    fn = logmax, method = 'BFGS', hessian = TRUE)
#
# #optim estimated values, iteration count, and estimated errors
# cat("When using optim() to maximize our log likelihood and
#     using our porposed starting values from part(b),\n our estimates (beta0,beta1,variance) =",
#     testDefault$par, "with iteration count", as.numeric(testDefault$counts[1]),
#     "and \n standard errors",
#     diag(solve(testDefault$hessian)) ,"respectively.\n")
```

```
#
# cat("When choosing (1,1,1) as our staring value,
#     our estimates (beta0,beta1,variance) =",
#     testOptim$par, "with iteration count \n",
#     as.numeric(testOptim$counts[1]),
#       "and standard errors",diag(solve(testDefault$hessian)) ,"respectively.")
```

$\rightarrow$ If we compare optim() with the BFGS method against our EM algorithm, we notice that the optim() solutions are very close to the true values of $\theta = (\beta_0, \beta_1, \sigma^2) = (1, 2, 6)$ with reasonable iteration counts, while our EM algorithm is further off and has over 10,000 iterations, suggesting that we are not converging to a solution. There seems to be some further issues with my EM algorithm that I haven't fixed yet, but I believe the error arises from my formulas of the first derivatives.

Next, let's look at

References Video Source)