Expectation Maximization Algorithms for Generalized Linear Models

Andrew Ma

January 20, 2022

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

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, tax data may contain incomes that are bracketed above a certain level - '$500,000 & over' may be the label of high income tax-payers. In voter data, we may have censored values for . Surprisingly, even policing data is censored as inspections [4].

To this end, we will break this analysis down into four sections, each complete with examples. We will be constructing an EM algorithm for linear regression on right-censored data. For the purposes of this demonstration, we will be utilizing R - this method can be implemented and tested in Python for identical results.

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; I will provide a concise proof of the EM Algorithm below.

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

Proof. $\rightarrow$ Let

So how do we translate this mathematical text into functional code? Let's take the following steps:

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.

One issue with EM Algorithms choosing reasonable starting values. I propose that 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 mini-ma that our algorithm may get "stuck" on.

2 Design

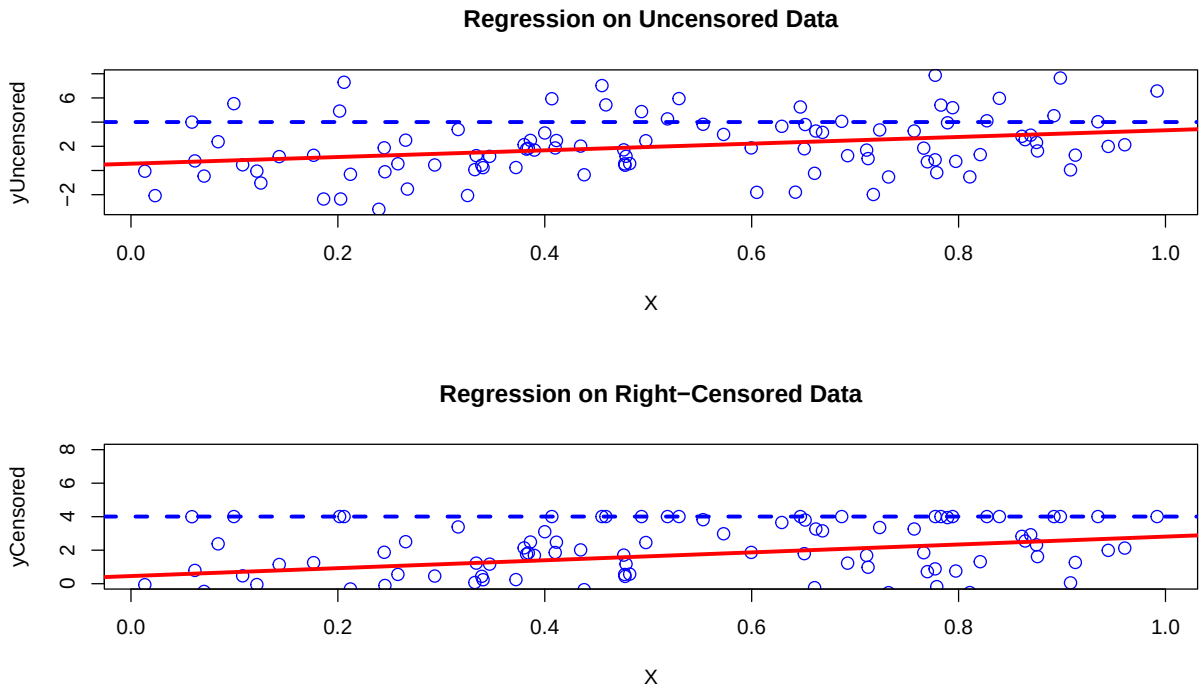
For the purposes of demonstration, let's construct randomized data for a si. A plot of this generated data is shown below:

```
## 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)
coef(mod)
```

```
## (Intercept)           x
##   0.5607442   2.7650812
```

```
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
  yTotal <- complete
  yKnown <- complete[known]
  yCensored <- complete[censor]
  complete <- list(yKnown, yCensored, yTotal ,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)
}
```

```r
par(mfrow = c(2,1))

graphC <- censor_data(x, yComplete)

#Uncensored data plot
plot(x, yComplete, cex = 1.3, col = "blue", main = "Regression on Uncensored Data", xlab = "X", ylab = "
abline(lm(yComplete ~ x), col = 'red', lwd = 3)
abline(h=graphC$tau, col="blue", lwd=3, lty=2)

## plot of censored data
plot(x, graphC$yTotal, ylim = c(0,8), cex = 1.3, col = "blue", main = "Regression on Right-Censored Data
abline(lm(graphC$yTotal ~ x), col = 'red', lwd = 3)
abline(h=graphC$tau, col="blue", lwd=3, lty=2)
```




```r
cat("Coefficients for Linear Regression on Right-Censored Data at a 20% level are: \n")
```

```
## Coefficients for Linear Regression on Right-Censored Data at a 20% level are:
```

```r
coef(lm(graphC$yTotal ~ x))
```

```
## (Intercept)           x
##   0.4563778   2.3539225
```

3 Implementation

Compare this to the uncensored data above. Note that fitting a base regression model would not be a good idea here:

```r
## EM function
EM <- function(x, yUncensored, thresh = 0.8, max_iter = 1e5, error = 1e-5){

  #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 fro
  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 <- 1
  #loop through our values and get updated values
  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)
    #termination conditions
    if(abs(b0_new - b0) < error && abs(b1_new - b1) < error && abs(var1_new - var1) < error){
      break
    }
    #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)
}
```

4 Results

```
test0 <- EM(x,yComplete, thresh = 0.95, error = 1e-5)
cat("Estimates when censoring top 5% of data: \n", "beta0 = ", test0$beta0,
    "\n beta1 = ", test0$beta1,
    "\n sigma^2 =", test0$variance,
    "\n Iteration Count =", test0$count)
```

```
## Estimates when censoring top 5% of data:
##  beta0 =  0.5372085
##  beta1 =  2.386995
##  sigma^2 = 3.977809
##  Iteration Count = 6
```

```
#20% of top values are censored
test1 <- EM(x,yComplete, thresh = 0.8, error = 1e-5)
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 =  0.3274201
##  beta1 =  1.987911
##  sigma^2 = 2.295996
##  Iteration Count = 12
```

```
#40% of top values are censored
test2 <- EM(x,yComplete,thresh = 0.6, error = 1e-5)
cat("Estimates when censoring top 40% of data: \n", "beta0 = ", test2$beta0,
    "\n beta1 = ", test2$beta1,
    "\n sigma^2 =", test2$variance,
    "\n Iteration Count =", test2$count)
```

```
## Estimates when censoring top 40% of data:
##  beta0 =  0.1766261
##  beta1 =  1.607193
##  sigma^2 = 1.491806
##  Iteration Count = 18
```

```
#60% of top values are censored
test3 <- EM(x,yComplete,thresh = 0.4, error = 1e-5)
cat("Estimates when censoring top 60% of data: \n", "beta0 = ", test3$beta0,
```

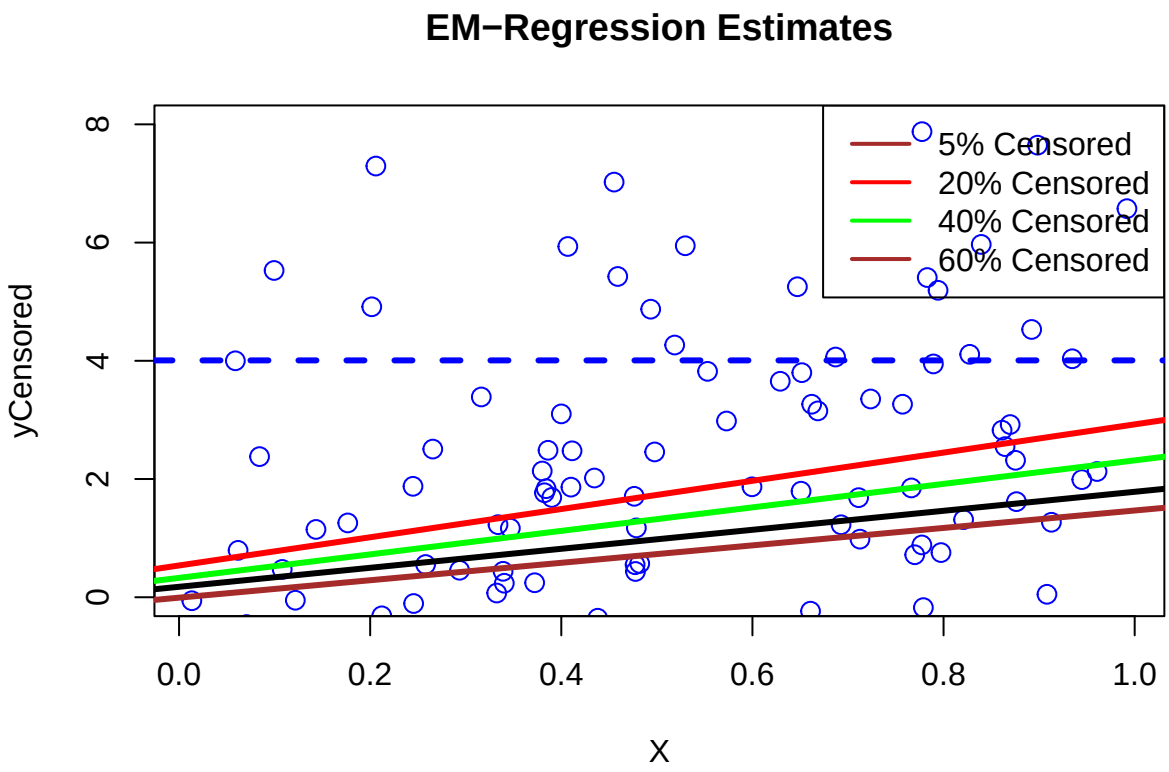
```
    "\n beta1 = ", test3$beta1,
    "\n sigma^2 =", test3$variance,
    "\n Iteration Count =", test3$count)
```

```
## Estimates when censoring top 60% of data:
##  beta0 =  -0.007311921
##  beta1 =  1.474401
##  sigma^2 = 1.588118
##  Iteration Count = 15
```

```
#80% of top values are censored
test4 <- EM(x,yComplete,thresh = 0.2, error = 1e-5)
cat("Estimates when censoring top 80% of data: \n", "beta0 = ", test4$beta0,
    "\n beta1 = ", test4$beta1,
    "\n sigma^2 = ", test4$variance,
    "\n Iteration Count =", test4$count)
```

```
## Estimates when censoring top 80% of data:
##  beta0 =  -0.1659699
##  beta1 =  1.581858
##  sigma^2 =  2.114954
##  Iteration Count = 33
```

```
## plot of EM Algorithm results
plot(x, yComplete, ylim = c(0,8), cex = 1.3, col = "blue", main = "EM-Regression Estimates", xlab = "X"

abline(a=test0$beta0, b=test0$beta1, col = 'red', lwd = 3)
abline(a=test1$beta0, b=test1$beta1, col = 'green', lwd = 3)
abline(a=test2$beta0, b=test2$beta1, col = 'black', lwd = 3)
abline(a=test3$beta0, b=test3$beta1, col = 'brown', lwd = 3)
abline(h=graphC$tau, col="blue", lwd=3, lty=2)

legend(x = "topright",          # Position
       legend = c("5% Censored", "20% Censored", "40% Censored", "60% Censored"),  # Legend texts
       lty = c(1),           # Line types
       col = c('brown','red','green','brown'),  # Line colors
       lwd = 2)                 # Line width
```



5 Comparisons

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(2,2,6), 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
```

```
## When using optim() to maximize our log likelihood and using our porposed starting values from part(b)
##  our estimates (beta0,beta1,variance) = 0.4566455 2.824068 4.618924 with iteration count 16 and
##  standard errors 0.2277647 0.6903112 0.5881268 respectively.
##
```

```
cat("When choosing (2,2,6) as our staring value, our estimates (beta0,beta1,variance) =", testOptim$par
```

```
## When choosing (2,2,6) as our staring value, our estimates (beta0,beta1,variance) = 0.4566271 2.82411
```

→ 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

```
microbenchmark('EM' = EM(x,yComplete,thresh = 0.8),
               'log-lik optim' = optim(par =  eval(parse(text = 'param <- getStart(x, yComplete)')),
                     x = x, yUncensored = yComplete,
                    fn = logmax, method = 'BFGS', hessian = TRUE)
               )
```

```
## Unit: milliseconds
##           expr       min        lq      mean    median        uq       max
##             EM  2.367222  3.003738  3.905831  3.451075  4.017391  11.02546
```

```
##  log-lik optim 38.289030 43.864610 58.466636 49.211936 56.496561 212.28080
##  neval
##    100
##    100
```

From the benchmarking comparison above, we can see a clear increase in computational speed for our EM-Algorithm. It seems that easy problems like . It would be fair to assume that EM-Algorithms provide a great benefit over log-likelihood optimization for models with low numbers of covariates or a small number of observations.

6 Discussion

Vardi's Expectation-Maximization (EM) algorithm is frequently used for computing the nonparametric maximum likelihood estimator of length-biased right-censored data, which does not admit a closed-form representation. The EM algorithm may converge slowly, particularly for heavily censored data. We studied two algorithms for accelerating the convergence of the EM algorithm, based on iterative convex minimization and Aitken's delta squared process

References 1. Video Source 2. Paper on EM-optimization & efficiency 3. Martin Haugh - EM Algorithm 4. Harvard Business Review