Expectation Maximization Algorithms for Generalized Linear Models

Andrew Ma

March 29, 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. Often, we wish to find the Maximum Likelihood Estimator for data we suspect follows a Normal Distribution, but contains latent variables whose true value is hidden. The advantages of an Expectation-Maximization algorithms is the fact that we are almost sure to converge to a solution; furthermore, EM-Algorithms are built off fundamental concepts of probability & statistics, allowing us to explore the precise inner workings of our algorithm. In fact, **every iteration in the EM algorithm results in a guaranteed increase in likelihood.** However, there are also disadvantages to this method; it has slow convergence, converges to local optima only, and requires both forward & backward probabilities while numerical optimization requires only forward probability.

So when should we attempt to implement an EM Algorithm? 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 surrounding total number of votes, or even automobile data on MPG may be labeled as '30 Mpg. & over', '40 Mpg & over', etc. Surprisingly, even local government data is censored as inspections on low-income housing structures and restaurants in Los-Angeles were "capped" on official reports[3].

To this end, 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.

A brief proof of the EM algorithm is shown here, but this article provides rigorous detail and step-by-step derivations for those who are less familiar with this concept.

Next, 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, lets sample from a normal distribution with a known mean and variance - this will allow us to induce moderately strong signals into our data to influence the results. For this demonstration, we are seeking to understand the effects of missing/truncated data with respect to OLS estimates under a simple linear model; note that this result can be generalized to linear models consisting of alternative link functions. The code utilized for data generation is shown below, alongside a plot and the true parameters under a simple linear regression model with full & complete data.

```
## 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)
cat("True Parameters \n ")
```

```
## True Parameters
##
```

```
coef(mod)
```

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

```
k = length(mod$coefficients)-1 #Subtract one to ignore intercept
SSE = sum(mod$residuals**2)
```

```
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)
}
```

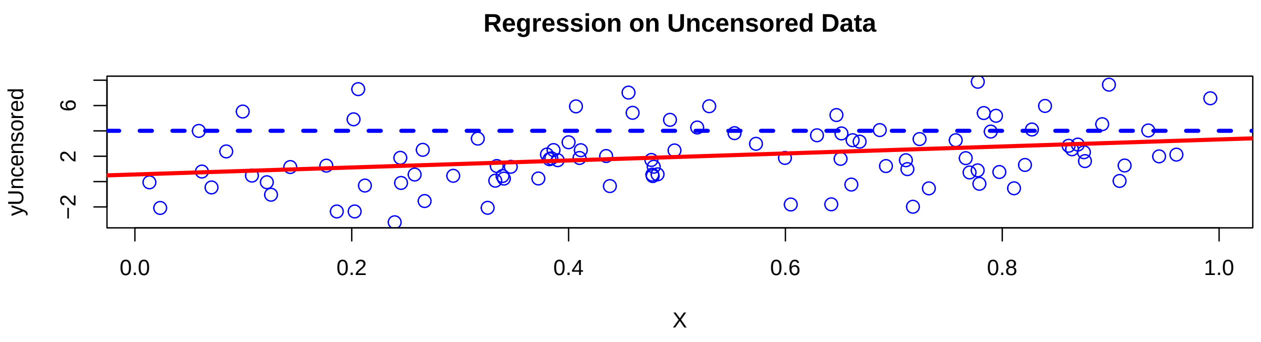
To give you a better understanding of this truncated or censored data concept, we can construct two plots - one fitting a linear regression model with no censored data against a linear model where the top 20% of values are truncated; ie, they are set to the 80% quantile value, represented by the dotted blue line.

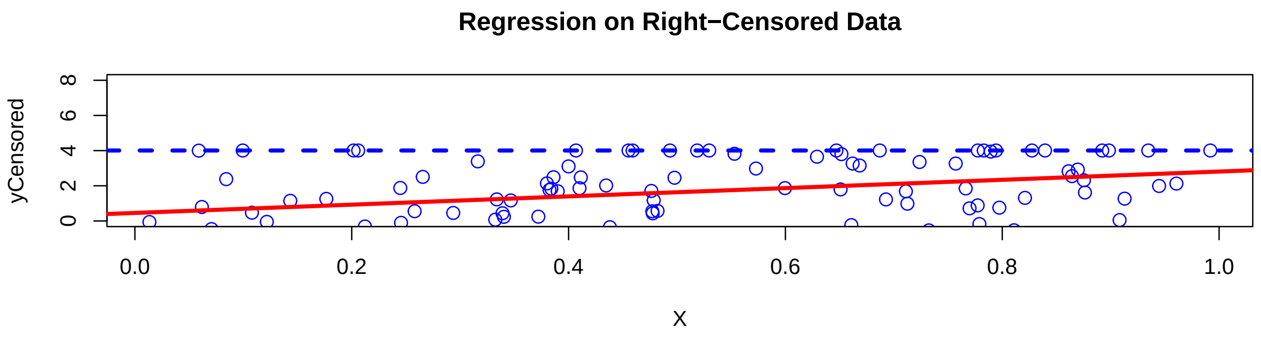
```
par(mfrow = c(2,1))

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)
```





```
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:
```

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

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

3 Implementation

We can see that our estimates are actually not too far off from the true values if we set the top 20% of data to a constant - I propose that these estimates based off the censored data will be valid starting values of our algorithm. In practice, starting values should be chosen after incorporating both prior knowledge & context of the model or data. Finally, we will set termination conditions to include a max iteration count of 10,000 alongside an 'error' of 0.00001, ie, our algorithm will stop after a certain number of loops or we reach estimates where $\hat{\beta}_0^{(t+1)} - \hat{\beta}_0^{(t)} < 0.0001$ & $\hat{\beta}_1^{(t+1)} - \hat{\beta}_1^{(t)} < 0.0001$.

```
## 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

To test our algorithm, we will utilize varying levels of censorship levels: 5%, 20%, 40%, 60%, 80%. An error level of 1e-5 is chosen as incremental increases to the order of 1e-5 would not provide substancial increases in the accuracy of our estimates.

```
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
```

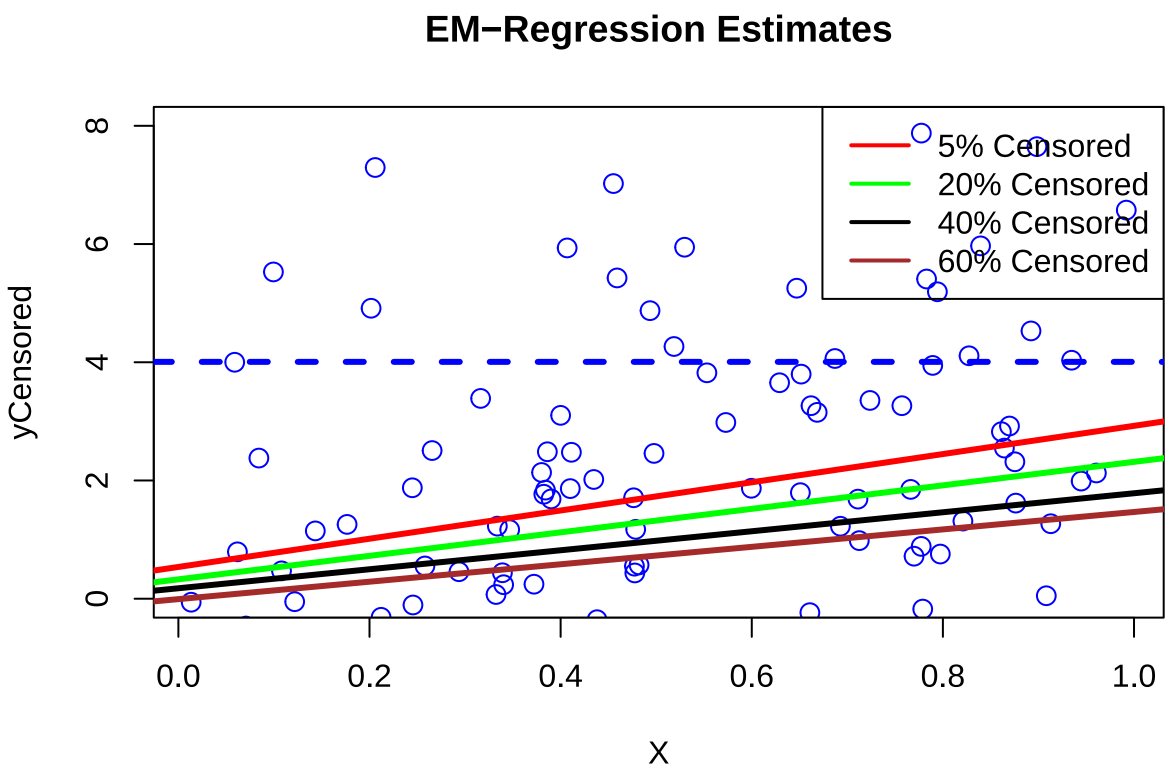
```
##table for wordpress blog
tab <- matrix(c(test0$beta0, test0$beta1, test0$variance,
                test1$beta0, test1$beta1, test1$variance,
                test2$beta0, test2$beta1, test2$variance,
                test3$beta0, test3$beta1, test3$variance,
                test4$beta0, test4$beta1, test4$variance,
                coef(lm(graphC$yTotal ~ x))[1], coef(lm(graphC$yTotal ~ x))[1], SSE), ncol=6, byrow=FAL
colnames(tab) <- c('5% Censor','20% Censor','40$ Censor','60% Censor','80% Censor', 'True Parameters')
rownames(tab) <- c('beta0','beta1','sigma^2')
tab <- as.table(tab)
tab
```

```
##             5% Censor    20% Censor    40$ Censor    60% Censor    80% Censor
## beta0     0.537208476   0.327420138   0.176626149  -0.007311921  -0.165969905
## beta1     2.386994549   1.987911068   1.607193310   1.474401208   1.581857627
## sigma^2   3.977808526   2.295995718   1.491805543   1.588117858   2.114953712
##         True Parameters
## beta0       0.456377825
## beta1       0.456377825
## sigma^2   520.729100608
```

```
## 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 = 1,           # Line types
       col = c('red','green','black','brown'),  # Line colors
       lwd = 2)                     # Line width
```



→ From the plot above, we can confirm our intuition that higher levels of censorship result in larger errors for our parameters. Surprisingly, note that our error does not seem to increase linearly with percentage of missing data - when we censor above 50% of the data points, we halve the sample size; thus, the consequences of further missing data is not as large as when compared to a fully observable dataset.

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 $log([\prod_{i-1}^{n}\mathbb{P}(Y = Y_i)][\prod_{i-1}^{n}\mathbb{P}(Y > \tau)]) = log([\prod_{i-1}^{n}\mathbb{P}(Y = Y_i)]) + log([\prod_{i-1}^{n}\mathbb{P}(Y > \tau)]) = \sum_{i=1}^{n} log(\mathbb{P}(Y = Y_i)) + \sum_{i=1}^{n} log(\mathbb{P}(Y > \tau))$. We can code this into R and utilize the built-in `optim()` function to arrive at optimal coefficient values.

```
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. Similarly, our EM algorithm provides quick convergence to an acceptable solution when the level is censoring is below 20% - these results confirm the functionality of our EM Algorithm.

```
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 neval
##             EM  1.53923  2.170971  2.848201  2.506603  3.373345  7.722169   100
##  log-lik optim 19.81583 31.740714 38.078424 36.251404 44.541785 69.457961   100
```

From the benchmarking comparison above, we can see a clear increase in computational speed for our EM-Algorithm. It seems that simple models such as the example above with only a singular covariate do not induce computational strain. 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. However, optimization methods such as BFGS or Quasi-Newton methods are sure to provide better convergence rates due to the large number of calculations done by the EM function.

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. Two algorithms for accelerating the convergence of the EM algorithm are based on iterative convex minimization and Aitken's delta squared process - more information on this can be found in this paper

References 1. Chan, Kwun Chuen Gary. "Acceleration of Expectation-Maximization Algorithm for Length-Biased Right-Censored Data." Lifetime Data Analysis, U.S. National Library of Medicine, Jan. 2017, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5716484/. 2. Haugh, Matin. IEOR E4570: Machine Learning for or … - Columbia University. 2015, http://www.columbia.edu/~mh2078/MachineLearningORFE/EM_Algorithm.pdf. 3. Beane, Matt. "Learning to Work with Intelligent Machines." Harvard Business Review, 2018, pp. 120–131.