Problem Set 3: EM Algorithm

Alexandra Norris

Due Friday April 1, 2022 by midnight

Please upload answers to all questions (including computational questions and any related graphics and R code *with comments*, in .rmd and knitted pdf forms to Canvas. For any problems that require calculations, please show your work. Finally, when doing simulations or draws from a random distribution in `R`, please set your seed immediately using `set.seed(2018)`.

EM for clustering

The EM algorithm can be seen an unsupervised clustering method based on mixture models. It follows an iterative approach, sub-optimal, which tries to find the parameters of the probability distribution that has the maximum likelihood of its attributes in the presence of missing/latent data.

The algorithm's input are the data set X, the total number of clusters/models K, the accepted error to converge ϵ and the maximum number of iterations.

For each iteration, first it is executed what's called the Expectation Step (E-step), that estimates the probability of each point belonging to each model, followed by the Maximization step (M-step), that re-estimates the parameter vector of the probability distribution of each model. The algorithm finishes when the distribution parameters converges or reach the maximum number of iterations. Convergence is assured since the algorithm increases the likelihood at each iteration until it reaches the (eventually local) maximum.

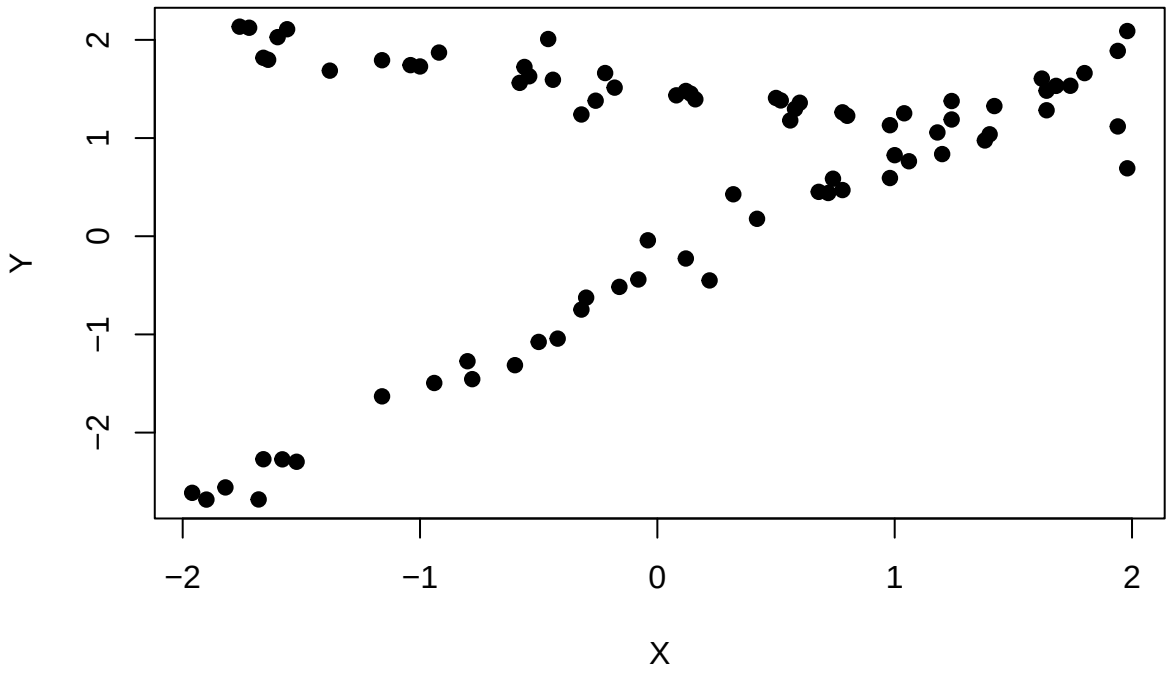
We generate a dataset such that observations are drawn from one of two linear processes C_1 and C_2:

```
set.seed(2018)

slope1 <- -.3; intercept1 <- 1.5    # generating data from C_1
xs1 <- sample(seq(-2,2,len=201), 40)
ys1 <- intercept1 + slope1*xs1 + rnorm(length(xs1),0,.15) # add some noise

slope2 <- 1.2; intercept2 <- -.4    # generating data from C_2
xs2 <- sample(seq(-2,2,len=201), 40)
ys2 <- intercept2 + slope2*xs2 + rnorm(length(xs1),0,.15)

mydata <- rbind( cbind(xs1,ys1), cbind(xs2,ys2) )
plot(mydata, pch=19, xlab="X", ylab="Y")
```



You need to do two things,

1) Find the parameters (slope and intercept) of the two models
2) Assignment of each datapoint x_i to the correct DGP

Knowing one makes the other easy to find. If we know the parameters, we can apply MLE to find which model is more probable to have generated x_i, ie compute $p(x_i|C_j)$. If we know the classifications we could just perform two linear regressions to find their respective parameters.

Recall the basic structure of the EM algorithm:

1) Init step: Assign random values to the model's parameters.
2) E step: Assign points to the model that fits each one best (these assignments are continuous, not binary)
3) M step: Update the parameters using the points assigned in the previous step
4) Iterate until parameter values converge (e.g., Frobenius norm of old parameter vector - new parameters vector < tolerance) or it reaches the maximum number of iterations.

Question 1: Init step

Create a function, `init_params` to initialize your parameters.

```
set.seed(2018)
# step 1 of EM algorithm - assign random values to model's parameters

# set slope and intercept variables to 0

i1 <- 0
s1 <- 0
i2 <- 0
s2 <- 0

init_params <- function() {
  # use runif to generate random values from a normal distribution for the slope and
```

```
  # intercepts - multiply values by 2 because data exists between -2 and 2
  i1 <- 2*runif(1)
  s1 <- 2*runif(1)
  i2 <- 2*runif(1)
  s2 <- 2*runif(1)
  c(i1,s1,i2,s2)
}
```

Question 2: E-step

In this question, we will write a function for E-step. Read the following carefully and write a function called `e.step`.

Assume the parameters are given for each model at this step. For each point x_i, we can compute two weights w_{i1} and w_{i2} (defined below) for the soft assignments of model 1 and model 2. Let $j \in \{1, 2\}$ index model, α_j denote the intercept and β_j denote the slope. The residuals r_{ij} of the two models for each x_i is:

$$r_{ij} = y_i - \alpha_j - \beta_j x_i$$

The weights w_{ij} is defined as:

$$w_{ij} = \frac{p(r_{ij}|x_i \in C_j)}{p(r_{i1}|x_i \in C_1) + p(r_{i2}|x_i \in C_2)}$$

Assume that normal distributions are assigned to these probabilities:

$$r_{ij}|x_i \in C_j \sim \mathcal{N}(0, \sigma^2)$$

which means

$$p(r_{ij}|x_i \in C_j) = \frac{1}{\sqrt{2\pi}\sigma} \exp\{-\frac{r_{ij}^2}{2\sigma^2}\}$$

Plugging in distributions into w_{ij} formula yields:

$$w_{ij} = \frac{\exp\{-\frac{r_{ij}^2}{2\sigma^2}\}}{\exp\{-\frac{r_{i1}^2}{2\sigma^2}\} + \exp\{-\frac{r_{i2}^2}{2\sigma^2}\}}$$

Note that σ is a free parameter and corresponds to the expected noise in the sample. Assume it is equal to 0.5 for our purposes here.

Write a function `e.step` that takes data and parameters as inputs and returns w_{i1} and w_{i2} values.

```
e.step <- function(data, parameters, sigma = 0.5){

  # create data frames with NA values for wi1 and wi2
  wi1 <- rep(NA, nrow(data))
  wi2 <- rep(NA, nrow(data))

  # use equations provided above to calculate residuals for each of the models
  # residual = y value - intercept - slope*x value
  for(i in 1:nrow(data)) {

    resid1 <- data[i,2] - parameters[1] - parameters[2]*data[i,1]
    resid2 <- data[i,2] - parameters[3] - parameters[4]*data[i,1]

    # calculate exp values for each model - e to the power of the result of the equation
    # equation provided above: value = exp((-residual^2)/ 2 sigma^2)
```

```
    exp1 <- exp(-resid1^2/(2*sigma^2))
    exp2 <- exp(-resid2^2/(2*sigma^2))

    # compute the weights for all datapoints
    # equation provided above: exp value of model / (exp1 + exp2)
    wi1[i] <- exp1 / (exp1+exp2)
    wi2[i] <- exp2 / (exp1+exp2)

  }

  # put different weignts together
  return(cbind(wi1, wi2))
}
```

Question 3: M-step

Next, we will write a function `wls` for M-step.

Assume the weights are given for each data point x_i at this step. To estimate the parameters of each process use weighted least squares. Recall that the OLS estimator of the parameters θ is given by:

$$\hat{\theta} = (X^\top X)^{-1} X^\top Y$$

To estimate θ_j for the j-th model, we add weights w_j to the estimator. Specifically, make a diagonal matrix W_j with these values and compute:

$$\hat{\theta}_j = (X^\top W_j X)^{-1} X^\top W_j Y$$

Write a function `wls` that takes X, Y and W as inputs and solves for the weighted least squares $\hat{\theta}_j$. Write a function `m.step` that returns the $\hat{\theta}_j$ for both models and has two arguments: the data, and the w_{i1} and w_{i2} output from the E-step.

```
# create the wls function
# use above equation: theta hat = INVERTED(transposed matrix X * weight *matrix X)
# * transposed matrix X * weiht * matrix Y

# solve() inverts a matrix in R, have to use %*% because this is matrix multiplication

wls <- function(w,x,y) {

  theta_j <- solve(t(x) %*% w %*% x) %*% t(x) %*% w %*% y

  return(theta_j)
}

# create the m.step function
# e_w is the weights found in the e step

m.step <- function(data, e_w) {

  # using wls function so need to define x and y matrices - x is x values and y is y values
  X <- cbind(rep(1, nrow(data)), data[,1])
  Y <- as.matrix(data[,2], ncol=1)

  # calculate theta hat values for both models. Weight matrices are diagonal so use "diag()"
```

```
  theta_hat1 <- wls(diag(e_w[,1]),X,Y)
  theta_hat2 <- wls(diag(e_w[,2]),X,Y)

  c(theta_hat1, theta_hat2)
}
```

Question 4: EM together

Put together the E and M steps into an EM algorithm wrapped in a function called `em.2lines`. The function takes three arguments: your data, the tolerance level (set to `1e-2` as a default), and the maximum number of iterations (set to `1e3` for now as a default). Return the estimated parameters, the final weights for each data point x_i, and the predicted class label for each data point (e.g., if $w_{i1} > w_{i2}$ then the class label of i-th data point is 1 and 0 otherwise).

Plot the results and color the datapoints according to their predicted class; draw appropriately colored linear regressions through the two sets of data using your EM estimated slopes and intercepts.

```
em2lines <- function(data, tol = 1e-2, max.iter = 1e3){

  # set the initial conditions of the function - iterations and slopes/intercepts
  iter <- 0

  i1 <- 0
  s1 <- 0
  i2 <- 0
  s2 <- 0

  # load in initial parameters
  params <- init_params()

  # run through the function until either the tolerance or maximum iterations is reached

  repeat{
    weights <- e.step(mydata, params)
    thetas <- m.step(mydata, weights)

    # define conditions for stopping
    # convergence: when difference between random initial parameters and estimated theta values becomes
    if(norm(as.matrix(params - thetas), type = "F") < tol)
       break

    # have iterations increase as iterations increase - stop when greater than max iterations
    iter <- iter+1

    if(iter > max.iter)
      break
  }

  # print out the thetas/parameters, weights, and classes of the weights
  # weight classes: 1 if w1>w2, 0 otherwise

  list(parameters = thetas,
```

```
       weights = weights,
       # have to use apply so that it happens for each row rather than just once
       class = apply(weights, 1, function(z) if (z[1]>z[2]) 1 else 0))

}


# run function
results <- em2lines(mydata)
results
```

```
## $parameters
## [1] 0.59660006 0.95820770 0.66387939 0.02926501
##
## $weights
##                wi1        wi2
##  [1,] 0.015061681 0.98493832
##  [2,] 0.244284822 0.75571518
##  [3,] 0.877839908 0.12216009
##  [4,] 0.794393488 0.20560651
##  [5,] 0.906382117 0.09361788
##  [6,] 0.033776344 0.96622366
##  [7,] 0.832903250 0.16709675
##  [8,] 0.847773482 0.15222652
##  [9,] 0.921900783 0.07809922
## [10,] 0.817703379 0.18229662
## [11,] 0.165303194 0.83469681
## [12,] 0.917635499 0.08236450
## [13,] 0.361977299 0.63802270
## [14,] 0.870752945 0.12924706
## [15,] 0.920484426 0.07951557
## [16,] 0.868202113 0.13179789
## [17,] 0.027752327 0.97224767
## [18,] 0.039042612 0.96095739
## [19,] 0.591302517 0.40869748
## [20,] 0.802443262 0.19755674
## [21,] 0.912655903 0.08734410
## [22,] 0.011186407 0.98881359
## [23,] 0.536926168 0.46307383
## [24,] 0.685379594 0.31462041
## [25,] 0.359689825 0.64031018
## [26,] 0.764269257 0.23573074
## [27,] 0.494415624 0.50558438
## [28,] 0.152284696 0.84771530
## [29,] 0.893651410 0.10634859
## [30,] 0.001481913 0.99851809
## [31,] 0.037792409 0.96220759
## [32,] 0.904678328 0.09532167
## [33,] 0.677402777 0.32259722
## [34,] 0.842310069 0.15768993
## [35,] 0.912170262 0.08782974
## [36,] 0.186243174 0.81375683
## [37,] 0.855273580 0.14472642
## [38,] 0.902520950 0.09747905
```

```
## [39,] 0.624671397 0.37532860
## [40,] 0.044415743 0.95558426
## [41,] 0.921779447 0.07822055
## [42,] 0.610727274 0.38927273
## [43,] 0.967039911 0.03296009
## [44,] 0.168656796 0.83134320
## [45,] 0.466644190 0.53335581
## [46,] 0.183876128 0.81612387
## [47,] 0.501898433 0.49810157
## [48,] 0.789061465 0.21093854
## [49,] 0.890089470 0.10991053
## [50,] 0.125712893 0.87428711
## [51,] 0.583247914 0.41675209
## [52,] 0.243223381 0.75677662
## [53,] 0.949328710 0.05067129
## [54,] 0.534622683 0.46537732
## [55,] 0.065785077 0.93421492
## [56,] 0.176519794 0.82348021
## [57,] 0.361048425 0.63895157
## [58,] 0.153139921 0.84686008
## [59,] 0.436200365 0.56379963
## [60,] 0.937226991 0.06277301
## [61,] 0.179162631 0.82083737
## [62,] 0.372344527 0.62765547
## [63,] 0.194682914 0.80531709
## [64,] 0.165691354 0.83430865
## [65,] 0.487073929 0.51292607
## [66,] 0.150665795 0.84933421
## [67,] 0.868380674 0.13161933
## [68,] 0.228488648 0.77151135
## [69,] 0.213818423 0.78618158
## [70,] 0.326135581 0.67386442
## [71,] 0.851400625 0.14859938
## [72,] 0.294753558 0.70524644
## [73,] 0.966415311 0.03358469
## [74,] 0.267601489 0.73239851
## [75,] 0.156632334 0.84336767
## [76,] 0.191527045 0.80847296
## [77,] 0.181822390 0.81817761
## [78,] 0.478265227 0.52173477
## [79,] 0.215888008 0.78411199
## [80,] 0.685564207 0.31443579
##
## $class
##  [1] 0 0 1 1 1 0 1 1 1 1 0 1 0 1 1 1 0 0 1 1 1 0 1 1 0 1 0 0 1 0 0 1 1 1 1 0 1 1
## [39] 1 0 1 1 1 0 0 0 1 1 1 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0
## [77] 0 0 0 1
```

```
library(ggplot2)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

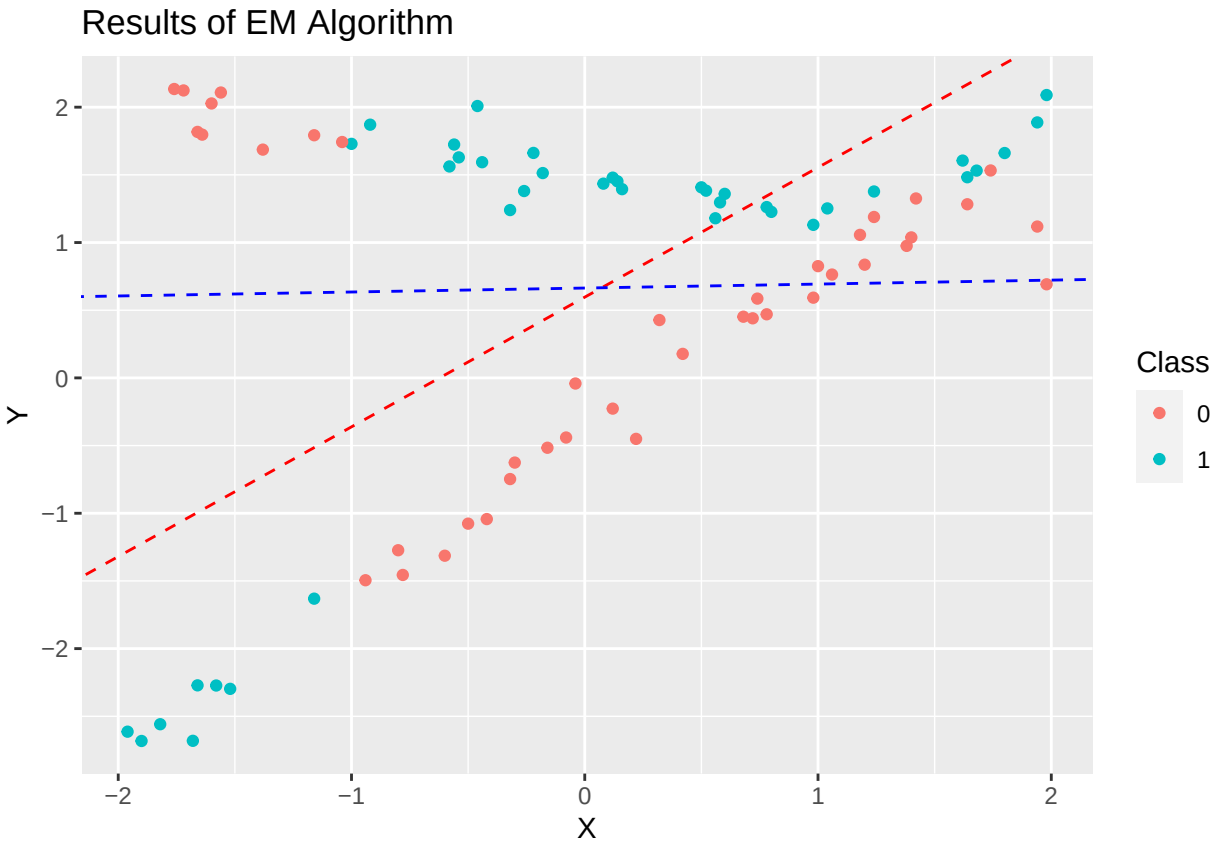
```
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
```

```
## v readr   2.0.2     v forcats 0.5.1
## v purrr   0.3.4
```

```
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
# add class data to dataframe so I can use ggplot
new_data <- data.frame(cbind(mydata, results$weights, results$class)) %>%
  mutate(class = as.character(V5))

# plot the results
new_data %>%
  ggplot(aes(x = xs1, y = ys1, color = class)) +
  geom_point() +
  # add lines based off of parameters
  geom_abline(intercept=results$parameters[1], slope=results$parameters[2], lty = 2, col = "red") +
  geom_abline(intercept=results$parameters[3], slope=results$parameters[4], lty = 2, col = "blue") +
  labs(x = "X", y = "Y", color = "Class", title = "Results of EM Algorithm")
```



Interestingly, it seems that the algorithm I created has difficulty classifying values on the far ends of the data. Both values close to -2 and 2 seem to be switched. This switching of results causes the lines drawn (using the parameters) to be less accurate. The line for class 1 should be negative but instead it is slightly positive.