ST340 Lab 4: Expectation Maximization

2020–21

1: Expectation Maximization—mixture of Gaussians

```
library(mvtnorm)
```

(a) Define parameters for $K = 3$ multivariate normal distributions.

```
K <- 3
mus.actual <- matrix(0,3,2)
mus.actual[1,] <- c(0,0)
mus.actual[2,] <- c(4,4)
mus.actual[3,] <- c(-4,4)
```

(b) Generate the covariance matrices randomly.

```
Sigmas.actual <- list()
for (k in 1:K) {
  mtx <- matrix(1,2,2)
  mtx[1,2] <- runif(1)*2-1
  mtx[2,1] <- mtx[1,2]
  Sigmas.actual[[k]] <- mtx*exp(rnorm(1))
}
```

(c) Generate some random mixture weights.

```
ws <- runif(K)
ws <- ws/sum(ws)
```

(d) Generate 1000 data points in $\mathbb{R}^2$. Hint: look at `?rmvnorm`.

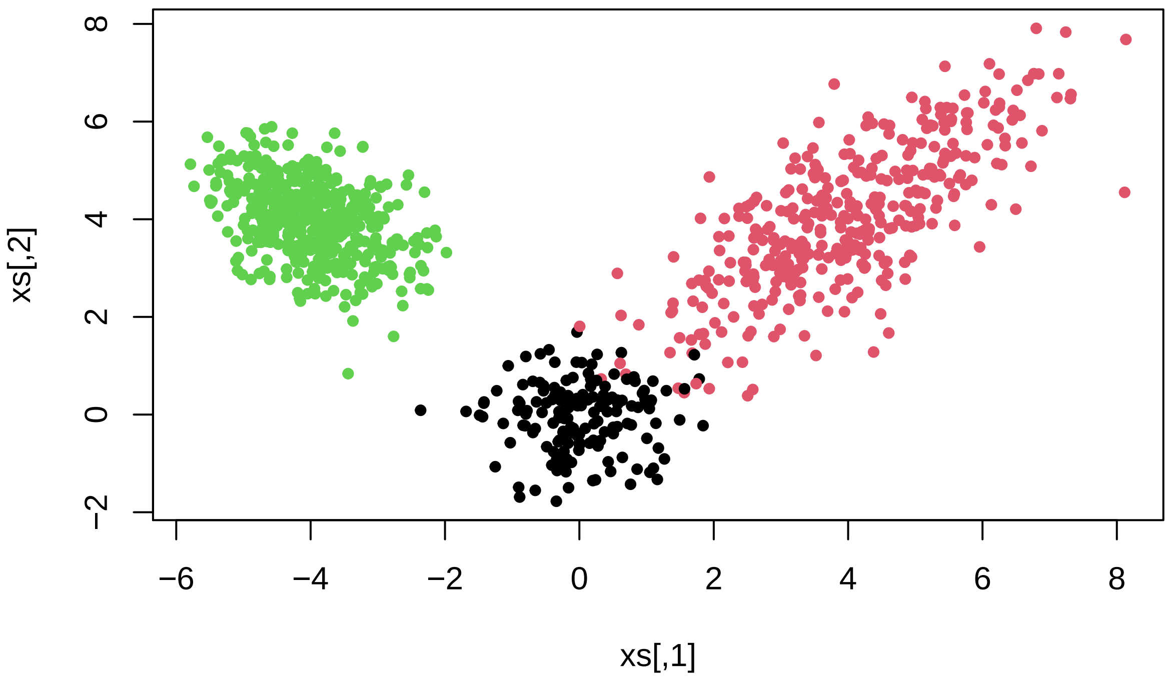
```
n <- 1000
p <- 2
xs <- matrix(0,n,p)
cols <- rep(0,n)
for (i in 1:n) {
  # sample from the mixture by sampling a mixture component k...
  k = sample(1:3,1, prob = ws)

  # ...and then sampling from that mixture component
  xs[i,] = rmvnorm(1, mean = mus.actual[k,], sigma = Sigmas.actual[[k]])


  cols[i] <- k
}
```

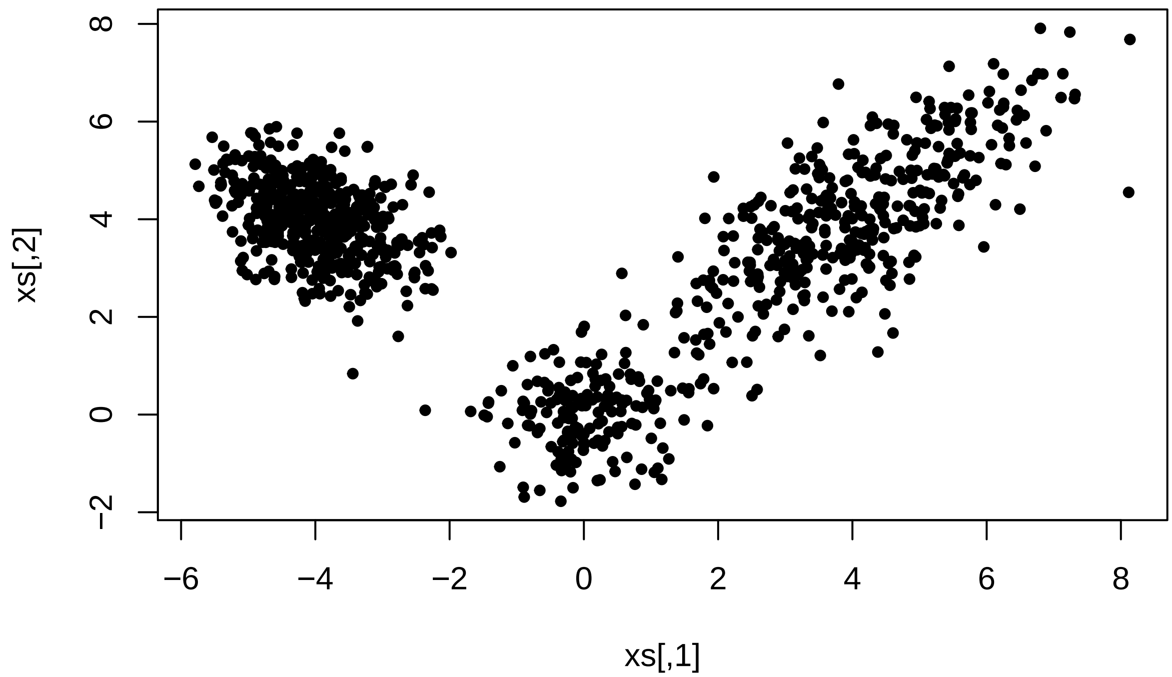
(e) Plot the data points coloured by cluster.

```
plot(xs,col=cols,pch=20)     #plotting generated data points from random MVN samples and colour coded ba
```



(f) Plot the data points without the colours.

```
plot(xs,pch=20)
```



(g) Run the EM algorithm on your generated data. You can try seeing what happens if $K = 2$ or $K = 4$ as well. . .

```
source("data/em_mixture_gaussians.R")
print(system.time(out <- em_mix_gaussian(xs,K=3)))
```

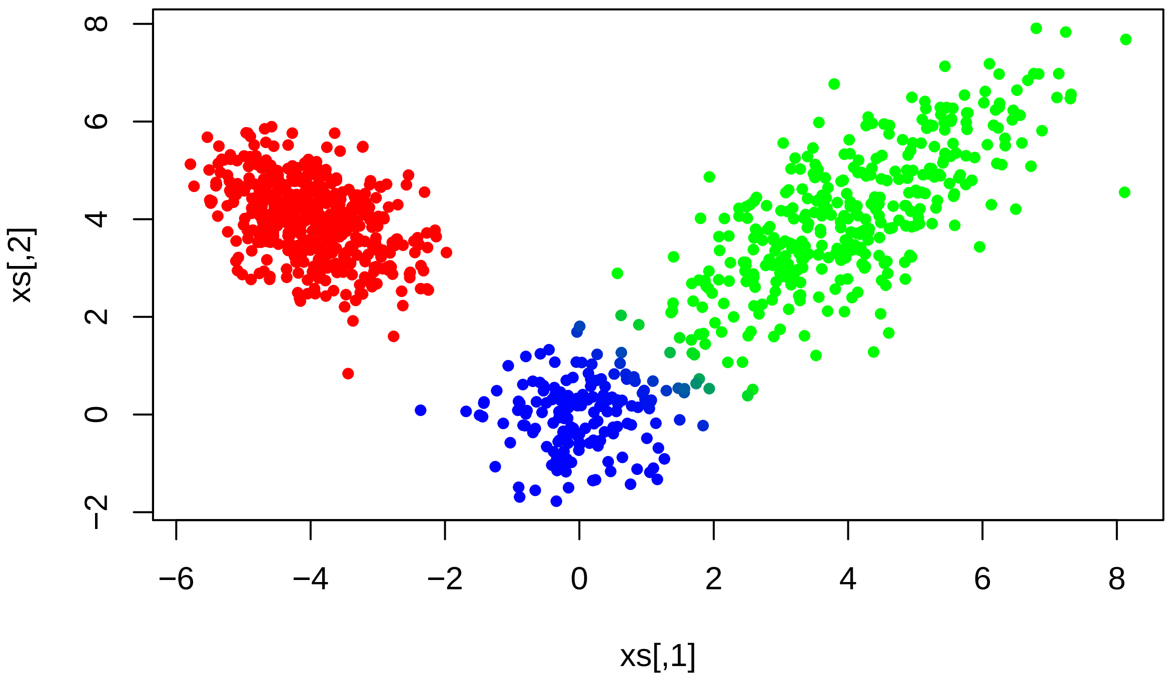
```
## [1] "iteration : log-likelihood"
## [1] "1 :  -4903.41663019164"
## [1] "2 :  -4381.23809750707"
## [1] "3 :  -4034.78528831957"
## [1] "4 :  -3774.78580435992"
## [1] "5 :  -3690.94611726114"
## [1] "6 :  -3668.68988526176"
## [1] "7 :  -3651.44027706681"
## [1] "8 :  -3629.93636120883"
## [1] "9 :  -3596.34030845992"
## [1] "10 :  -3564.10569337843"
## [1] "11 :  -3558.57407274484"
## [1] "12 :  -3558.27957127078"
## [1] "13 :  -3558.198664301"
## [1] "14 :  -3558.16623871572"
## [1] "15 :  -3558.15302057402"
## [1] "16 :  -3558.14761623762"
## [1] "17 :  -3558.14540496592"
## [1] "18 :  -3558.14449996429"
## [1] "19 :  -3558.14412953486"
## [1] "20 :  -3558.14397790315"
```

```
## [1] "21 :  -3558.14391583155"
## [1] "22 :  -3558.14389042135"
## [1] "23 :  -3558.14388001902"
## [1] "24 :  -3558.14387576048"
##    user  system elapsed
##   16.72    0.00   16.73
```

```
my.colors <- rep(0,n)
for (i in 1:n) {
  my.colors[i] <- rgb(out$gammas[i,1],out$gammas[i,2],out$gammas[i,3])
}

# recall that gamma_i_k is the probability that xi is associated with cluster k

plot(xs,col=my.colors,pch=20)
```



2: Expectation Maximization—mixture of Bernoullis

(a) Create a file called `em_mixture_bernoullis.R` which contains a function called `em_mix_bernoulli` that is the analogue of `em_mix_gaussian`. You could use `em_mixture_gaussians.R` as a template.

Hint: do not initialize any of the cluster `mus` to be either 0 or 1. (Do you know why?)

Hint: if, by numerical error, any of the parameters μ_{kj} are greater than 1, the algorithm will most likely fail. To avoid this, you can, after the M step, perform the update

```r
mus[which(mus > 1,arr.ind=TRUE)] <- 1 - 1e-15
```

where `mus` is a $K \times p$ matrix.

(b) Test your code.

```r
n <- 500; p <- 50                           # n data points where each data point is a vector of length
K.actual <- 2                               # actual number of clusters

mix <- runif(K.actual)                      # the mixture probabilities w_k. The probabilities for sele
mix <- mix / sum(mix)                       # normalising the probs

mus.actual <- matrix(runif(K.actual*p),K.actual,p)     # matrix 2 rows 50 columns. This is selecting our
zs.actual <- rep(0,n)                       # actual cluster assignment for each of the 500 data points

xs <- matrix(0,n,p)                         # no we are simulating a dataset
for (i in 1:n) {
  cl <- sample(K.actual,size=1,prob=mix)    # important to note that: If x has length 1, is numeric and
                                            # selects cluster 1 or 2

  zs.actual[i] <- cl                        # zs.actual takes the cluster value

  xs[i,] <- (runif(p) < mus.actual[cl,])    # this is used to simulate our n data points. A large value

}
```

(c) Calculate the mixture parameters.

```r
source("data/em_mixture_bernoullis.R")
source('my bernoulli EM function.R')
print(system.time(out <- em_mix_bernoulli(xs,K.actual)))
```

```
## [1] "iteration : log-likelihood"
## [1] "1 :  -18336.1837684482"
## [1] "2 :  -13849.2089806227"
## [1] "3 :  -12936.4045822234"
## [1] "4 :  -12894.1192353394"
## [1] "5 :  -12894.0769279279"
## [1] "6 :  -12894.0769268927"
##    user  system elapsed
##    0.16    0.01    0.17
```

(d) Check if the learned parameters are close to the truth.

```r
v1 <- sum(abs(out$mus-mus.actual))
v2 <- sum(abs(out$mus[2:1,]-mus.actual))
vm <- min(v1,v2)/p/2
print(vm)
```

```
## [1] 0.02049125
```

```r
if (vm > .3) print("probably not working") else print("might be working")
```

```
## [1] "might be working"
```

```r
# Checking if my algorithm gives the same results

out = my_em_mix_bernoulli(xs,K = K.actual)
```

```
## [1] "1 log-likelihood =  -13619.5077418673"
## [1] "2 log-likelihood =  -12908.3600165341"
## [1] "3 log-likelihood =  -12894.0786220622"
## [1] "4 log-likelihood =  -12894.0769269252"
## [1] "5 log-likelihood =  -12894.0769268927"
```

```
v1 <- sum(abs(out$Mu-mus.actual))
v2 <- sum(abs(out$Mu[2:1,]-mus.actual))
vm <- min(v1,v2)/p/2
print(vm)
```

```
## [1] 0.02049125
```

```
if (vm > .3) print("probably not working") else print("might be working")
```

```
## [1] "might be working"
```