Solutions for ST340 Lab 4

2019–20

1: Expectation Maximization—mixture of Gaussians

```
if (!require(mvtnorm)) {
  install.packages("mvtnorm")
  library(mvtnorm)
}
```

```
## Loading required package: mvtnorm
```

```
## Warning: package 'mvtnorm' was built under R version 3.5.2
```

(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`.

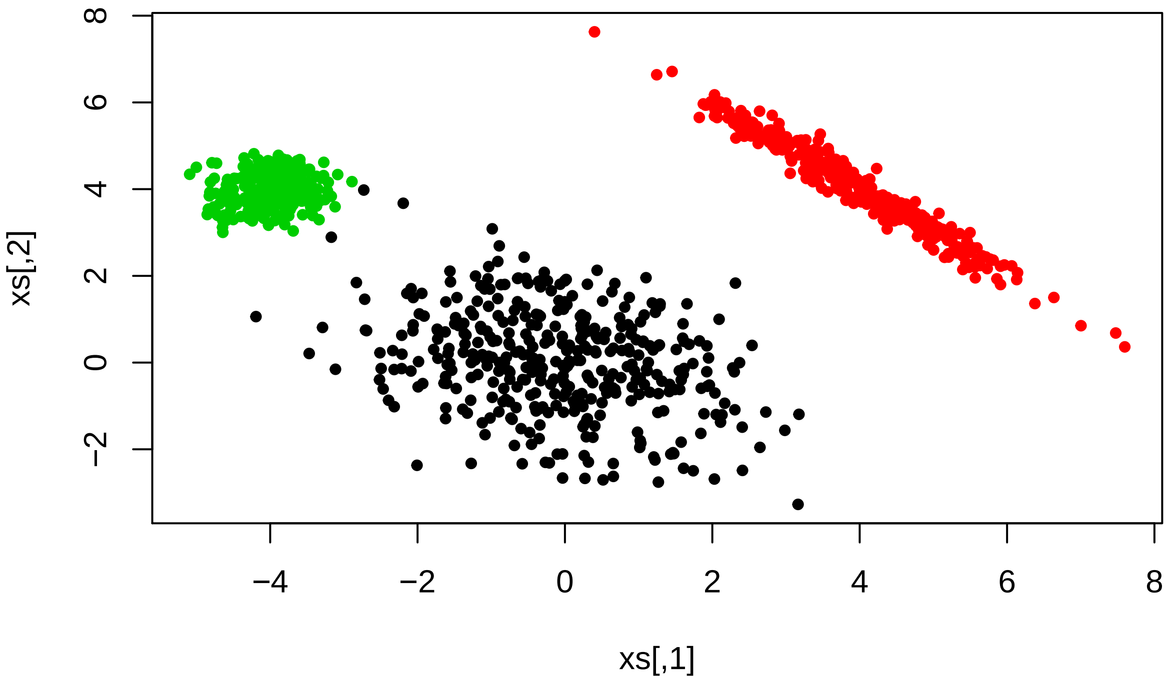
```
n <- 1000
p <- 2
xs <- matrix(0,n,p)
cols <- rep(0,n)
for (i in 1:n) {
  # sample from the mixture by sampling a mixture component k...
  k <- sample(K,1,prob=ws)

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

  cols[i] <- k
}
```

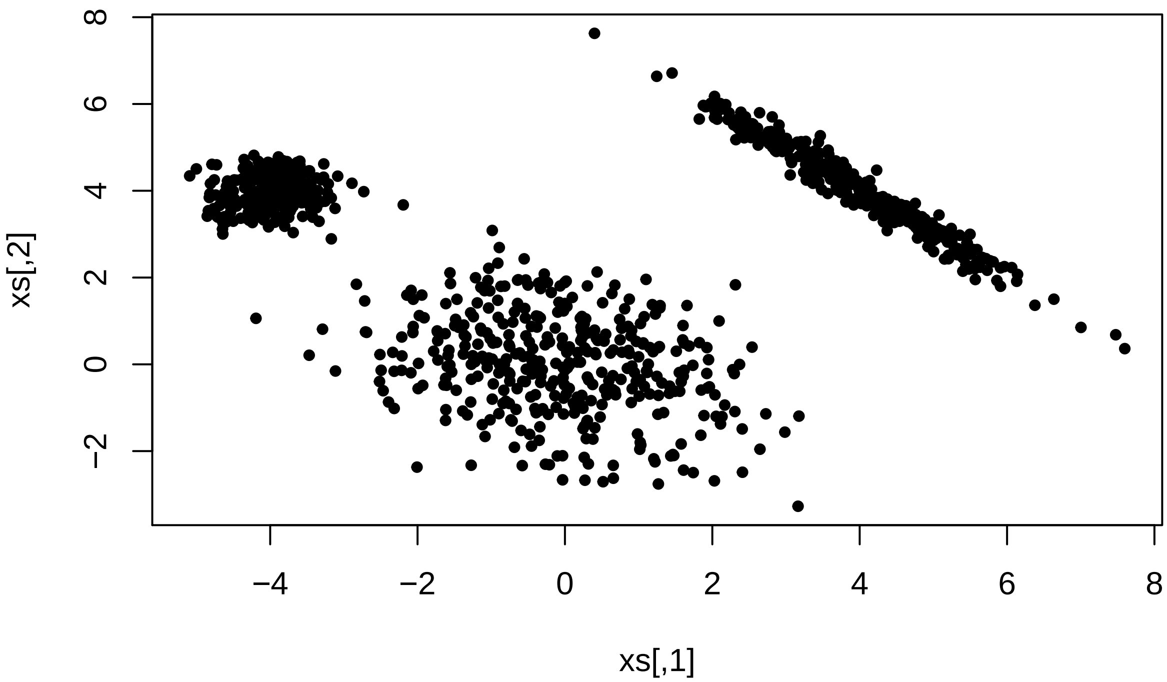
(e) Plot the data points coloured by cluster.

```r
plot(xs,col=cols,pch=20)
```



(f) Plot the data points without the colours.

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



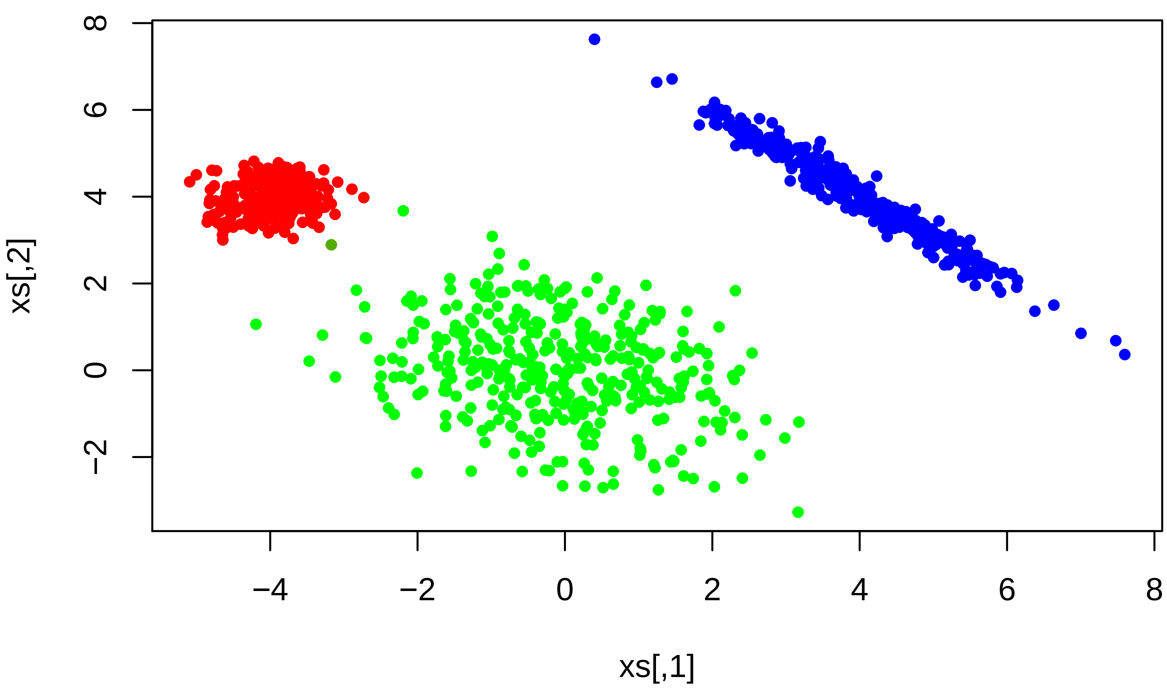
(g) Run the EM algorithm on your generated data. You can try seeing what happens if $K = 2$ or $K = 4$ as well…

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

```
## [1] "iteration : log-likelihood"
## [1] "1 :  -5050.21273524314"
## [1] "2 :  -4596.99170291037"
## [1] "3 :  -4473.96214878028"
## [1] "4 :  -4411.66808338001"
## [1] "5 :  -4340.72058899531"
## [1] "6 :  -4199.31121311593"
## [1] "7 :  -3728.80581034353"
## [1] "8 :  -3590.98657799273"
## [1] "9 :  -3555.09551849044"
## [1] "10 :  -3533.16403263067"
## [1] "11 :  -3515.31843303265"
## [1] "12 :  -3497.89320461474"
## [1] "13 :  -3477.6766173766"
## [1] "14 :  -3449.88601831471"
## [1] "15 :  -3405.67402282529"
## [1] "16 :  -3330.09839577783"
## [1] "17 :  -3187.53878564801"
## [1] "18 :  -3081.20375815544"
## [1] "19 :  -3079.87201031866"
## [1] "20 :  -3079.84878788525"
## [1] "21 :  -3079.84835094201"
## [1] "22 :  -3079.8483428046"
##    user  system elapsed
##   9.662   0.031   9.714
```

```r
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])
}

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

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

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

(b) Test your code.

```
n <- 500; p <- 50
K.actual <- 2
mix <- runif(K.actual); mix <- mix / sum(mix)
mus.actual <- matrix(runif(K.actual*p),K.actual,p)
zs.actual <- rep(0,n)

xs <- matrix(0,n,p)
for (i in 1:n) {
  cl <- sample(K.actual,size=1,prob=mix)
  zs.actual[i] <- cl
  xs[i,] <- (runif(p) < mus.actual[cl,])
}
```

(c) Calculate the mixture parameters.

```
source("em_mixture_bernoullis.R")
print(system.time(out <- em_mix_bernoulli(xs,K.actual)))
```

```
## [1] "iteration : log-likelihood"
## [1] "1 :  -18364.6115992419"
## [1] "2 :  -13464.3556491071"
## [1] "3 :  -13356.4783877141"
## [1] "4 :  -13354.8312877939"
## [1] "5 :  -13354.831142274"
## [1] "6 :  -13354.8311422595"
##    user  system elapsed
##   0.114   0.011   0.126
```

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

```
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] 0.03233699
```

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

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