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)))

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

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

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