CS CM226, PS3, Prob. 2

Niko Darci-Maher, UID #504924547

11/16/2021

Implement EM

Goal: cluster individuals into populations with unknown allele frequency based on genotype

Functions for running the algorithm

```
library(ggplot2)

# helper for r and Q function; computes the l function for every individual and cluster
l_all <- function(alpha, f, x) {
  # vectorized version of the l function (derived in problem 1)
  res = t(log(alpha) + t(x %*% log(f) + (1 - x) %*% log(1 - f)))
  return(res)
}

# r function; the soft assignment for a given individual and cluster
# also computes Q, the expectation of the log-likelihood
compute_r_Q <- function(alpha, f, x, n, K) {
  l = l_all(alpha, f, x)
  # lstar adjusts for underflow of the likelihood
  lstar = apply(l, 1, max)
  # use exp(log likelihood) to avoid calculating a product
  top = exp(l - lstar)
  bottom = rowSums(top)
  bottom = matrix(rep(bottom, each = K), nrow = n, byrow = T)
  r = top/bottom
  # Q function is the product of r and the l function,
  # summed over every individual and cluster
  Q = sum(r * l)
  return(list(r = r, Q = Q))
}

# updatealpha function; update weights from old to new
updatealpha <- function(r, n) {
  # vectorized version of function derived in prob. 1
  return(colSums(r) / n)
}

# updatef function; update allele freq from old to new
updatef <- function(x, r) {
  # vectorized version of function derived in prob. 1
```

```
  tops = t(r) %*% x
  bottoms = colSums(r)
  return (t(tops/bottoms))
}

# runEM function; run the EM algorithm to estimate
# population membership based on genotype
runEM <- function(x, K) {
  n = nrow(x)
  m = ncol(x)
  # initialize first guess of parameters
  alpha0 = rexp(K)
  alpha0 = alpha0 / sum(alpha0)
  f0 = matrix(runif(m*K), nrow = m)
  # iterate until stopping condition is met
  niter = 100
  oldalpha = alpha0
  oldf = f0
  Qvec = rep(NA, niter)
  for (iter in 1:niter) {
    # E-STEP: compute soft assignments using the current guesses of alpha and f
    rq = compute_r_Q(oldalpha, oldf, x, n, K)
    thisr = rq$r
    # remember likelihood at this iteration
    # Q is *equal* to log-likelihood here because it is a tight lower bound
    thisQ = rq$Q
    Qvec[iter] = thisQ
    # check if stopping condition has been met
    if (iter > 1) {
      if(abs(thisQ - Qvec[iter-1]) < 1e-8) { break }}
    # else, continue iterating
    # M-STEP: use the just-computed r_ik to update alpha and f
    newalpha = updatealpha(thisr, n)
    newf = updatef(x, thisr)
    oldalpha = newalpha
    oldf = newf
    # jerry rig thing, nudge allele freqs off of 0 and 1
    smallnumber = 0.001
    oldf[which(oldf == 0)] <- smallnumber
    oldf[which(oldf == 1)] <- 1 - smallnumber
  }
  return(list(alphahat = oldalpha, fhat = oldf, z_post = thisr,
              Qvec = Qvec, maxQ = max(c(na.omit(Qvec)))))
}
```

Import dataset 1

```
# genotypes
readGeno <- function(filename) {
  g = readLines(filename)
  g = strsplit(g, split = '')
  g = data.frame(g)
  colnames(g) = seq(1, ncol(g))
```

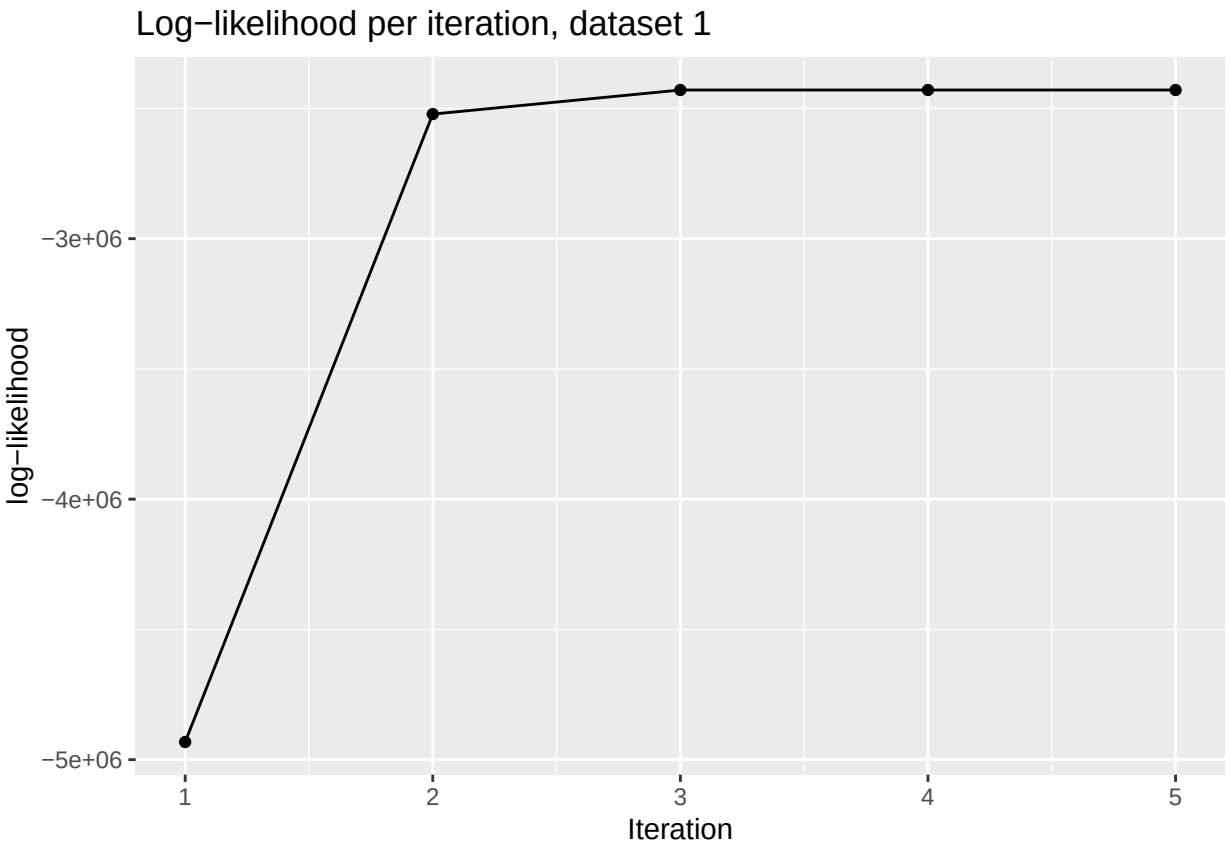
```
  g = apply(g, 2, as.numeric)
  return(g)
}
geno1 = readGeno("data/Q2/mixture1.geno")
# true alphas
alpha1 = read.table("data/Q2/mixture1.alpha")$V1
# true allele freqs
f1 = as.matrix(read.table("data/Q2/mixture1.freq"))
# true cluster assignments
clust1 = as.matrix(read.table("data/Q2/mixture1.ganc"))

# define global constants
# alpha is a K-length vector of cluster weights
# r is an nxK matrix of soft assignments for each cluster
# f is an mxK matrix of allele frequencies in each population
# x is an nxm matrix of genotype data (haploid, 0 or 1)
```

Run EM on dataset 1

```
NRESTARTS = 3
EMresult1 = sapply(1:NRESTARTS, function(x) runEM(geno1, K = 2))
EMresult1_max = EMresult1[,which.max(sapply(1:NRESTARTS, function(x) EMresult1[,x]$maxQ))]
```

Part (a)

```
# plot expected likelihood over iterations
ggplot(mapping = aes(x = 1:length(na.omit(EMresult1_max$Qvec)),
                             y = na.omit(EMresult1_max$Qvec))) +
  geom_line() + geom_point() +
  xlab("Iteration") + ylab("log-likelihood") +
  ggtitle("Log-likelihood per iteration, dataset 1")
```



Part (b)

```
# MLE of pi for dataset 1
print(EMresult1_max$alphahat)
```

```
## [1] 0.606 0.394
```

Part (c)

```
# compute the population guesses based on posterior probabilities
getAccuracy <- function(em, trueclust) {
  modelassign = apply(em$z_post, 1, which.max)
  trueassign = head(apply(trueclust, 1, which.max), length(modelassign))
  acc = sum(modelassign == trueassign) / length(trueassign)
  # this is valid because the population labels in the solution are arbitrary:
  if(acc < 0.5) {acc = 1 - acc}
  return(acc)
}

accuracy1 = getAccuracy(EMresult1_max, clust1)
print(paste0("Accuracy of classifications on dataset 1: ", round(accuracy1*100, 2), "%"))
```

```
## [1] "Accuracy of classifications on dataset 1: 100%"
```

Part (d)

```
dtab = data.frame(matrix(rep(NA, NRESTARTS*3), nrow = NRESTARTS))
colnames(dtab) = c("initialization", "log_likelihood", "accuracy")
for (i in 1:NRESTARTS) {
  dtab[i,] = c(i, EMresult1[,i]$maxQ, getAccuracy(EMresult1[,i], clust1))
}
dtab$accuracy = paste(round(dtab$accuracy*100, 2), "%")
dtab
```

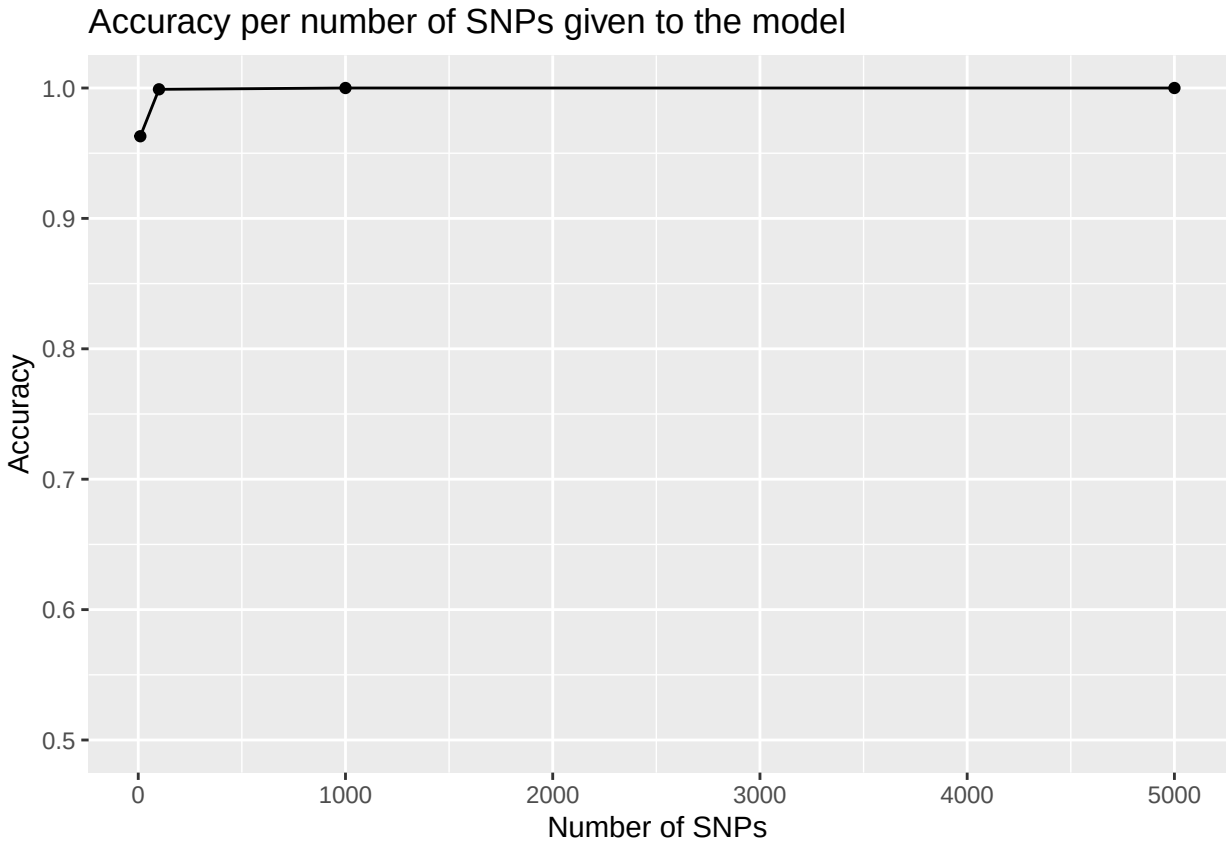
```
##   initialization log_likelihood accuracy
## 1              1       -2429181    100 %
## 2              2       -2429182    100 %
## 3              3       -2429180    100 %
```

From the table, we can see that the results are very similar across 3 different random starts.

```
# now that we've seen that the random restarts give similar results,
# define a function to run random restarts and take the max likelihood
runEM_restart <- function(x, K, nrestarts) {
  EMresult = sapply(1:nrestarts, function(foo) runEM(x, K))
  EMresult_max = EMresult[,which.max(sapply(1:nrestarts, function(x) EMresult[,x]$maxQ))]
  return(EMresult_max)
}
```

Part (e)

```
nsnpvec = c(10, 100, 1000, 5000)
EMresult1_sub = sapply(nsnpvec, function(nsnp) runEM_restart(geno1[,1:nsnp], K = 2, NRESTARTS))
acc_sub = rep(NA, length(nsnpvec))
for (i in 1:length(acc_sub)) {
  acc_sub[i] = getAccuracy(EMresult1_sub[,i], clust1)
}
ggplot(mapping = aes(x = nsnpvec, y = acc_sub)) +
  geom_line() + geom_point() +
  ylim(0.5, 1) +
  xlab("Number of SNPs") + ylab("Accuracy") +
  ggtitle("Accuracy per number of SNPs given to the model")
```



Part (f)

Import dataset 2

```
geno2 = readGeno("data/Q2/mixture2.geno")
```

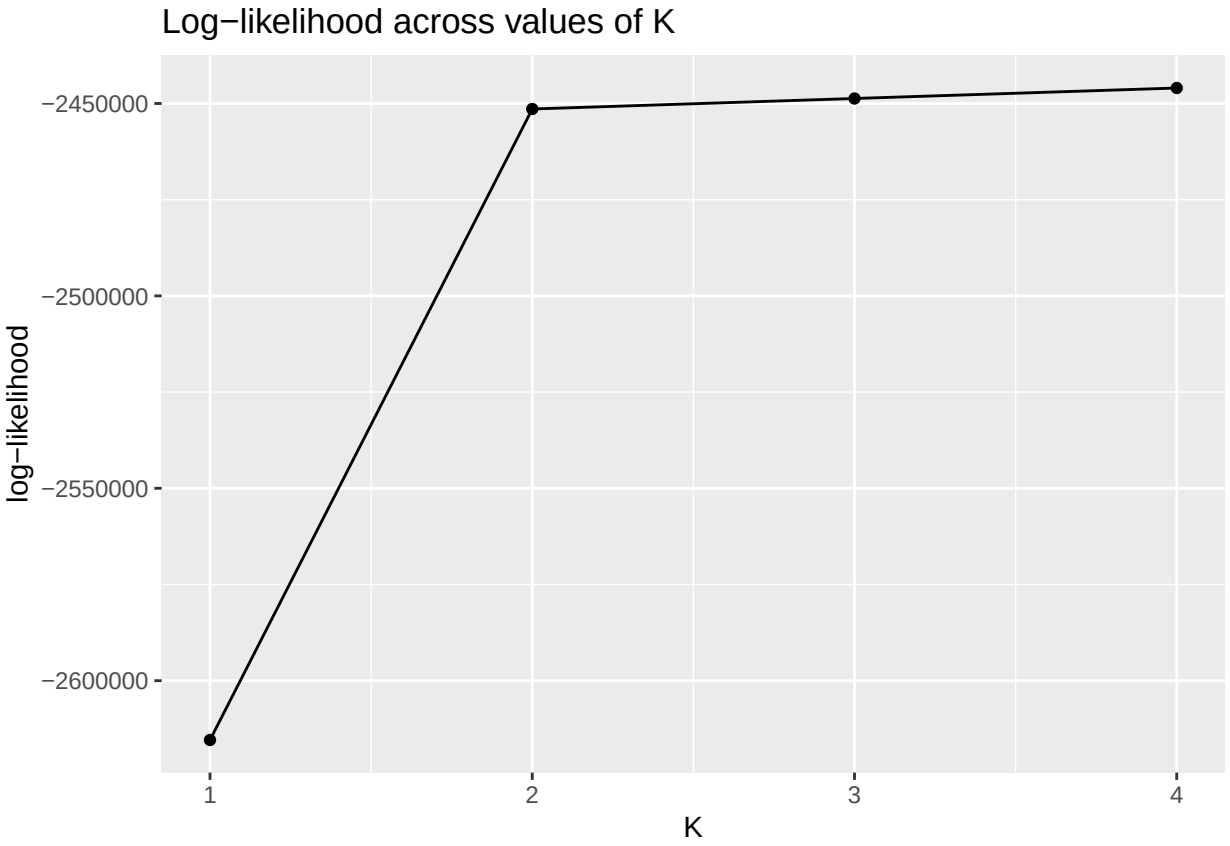
Run EM on dataset 2

```
EMresult2 = runEM_restart(geno2, K = 2, NRESTARTS)
# MLE of pi for dataset 2:
print(EMresult2$alphahat)
```

```
## [1] 0.224 0.776
```

Part (g)

```
kvec = 1:4
EMresult2_varyK = sapply(kvec, function(K_vary) runEM_restart(geno2, K_vary, NRESTARTS))
ll_varyK = sapply(1:length(kvec), function(i) EMresult2_varyK[,i]$maxQ)
ggplot(mapping = aes(x = kvec, y = ll_varyK)) +
  geom_line() + geom_point() +
  xlab("K") + ylab("log-likelihood") +
  ggtitle("Log-likelihood across values of K")
```



Based on this plot, I would choose K = 2 as the number of clusters, because it seems like increasing K above 2 only improves the likelihood a tiny bit, and it's computationally easier to cluster the data into 2 clusters than more than 2 clusters.