STAT 542 / CS 598: Homework 5

Fall 2019, by Regan Chan (ttchan2)

Due: Monday, Oct 28 by 11:59 PM Pacific Time

Contents

Question 1 [50 Points] K-Means Clustering

Let's consider coding a K-means algorithm. Perform the following:

- Load the `zip.train` (handwritten digit recognition) data from the ElemStatLearn package, and the goal is to identify clusters of digits based on only the pixels.
- [15 Points] Write your own code of k-means that iterates between two steps, and stop when the cluster membership does not change.
  - updating the cluster means given the cluster membership
  - updating the cluster membership based on cluster means

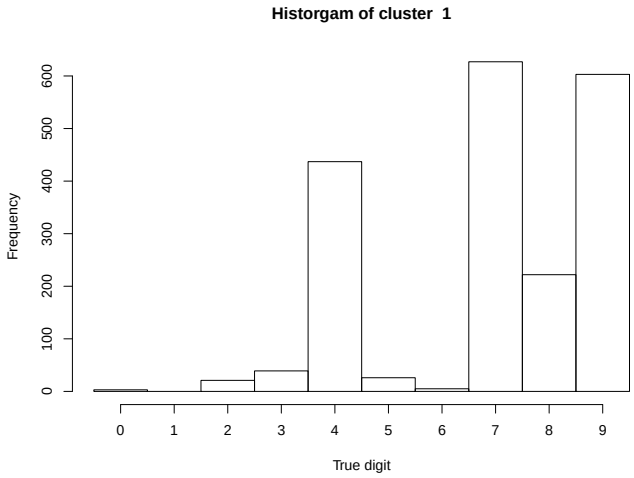
```
library(parallel)
mc.cores <- detectCores()
myKmeans <- function(data, K) {
  n <<- nrow(data)
  old_cluster_num <<- rep(0, n)
  new_cluster_num <<- sample(1:K, n, replace=TRUE)
  iterations <<- 0
  while(any(new_cluster_num != old_cluster_num)) {
    old_cluster_num <<- new_cluster_num
    centers <<- mcmapply(function(j){     # Column vectors
      colMeans(data[old_cluster_num==j, ])
    }, 1:K, mc.cores=mc.cores)
    new_cluster_num <<- mcmapply(function(i){ # Should be apply(data, 1, func(row))
      row <- data[i, ]
      which.min(colSums((centers-row)^2))
    }, 1:n, mc.cores=mc.cores)
    iterations <<- iterations+1
  }
  new_cluster_num
}
```

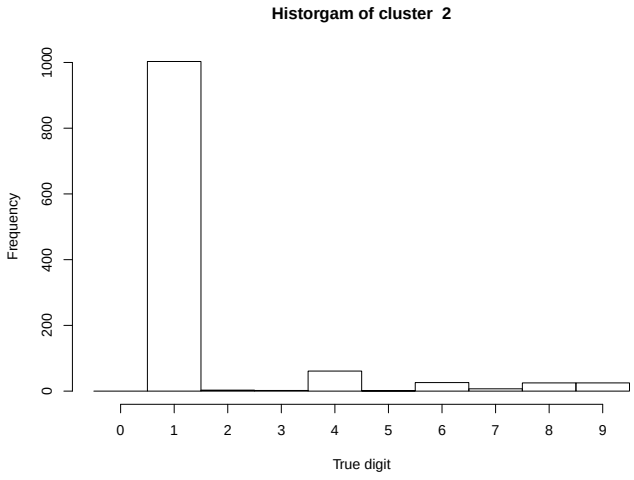
- [10 Points] Perform your algorithm with one random initialization with $k = 5$
  - For this question, compare your cluster membership to the true digits. What are the most prevalent digits in each of your clusters?

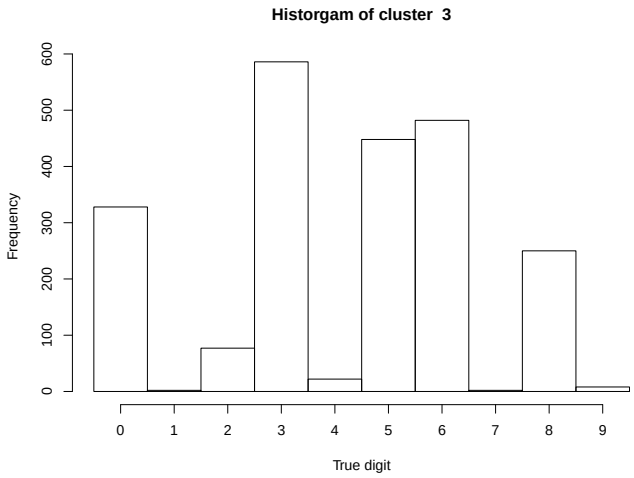
Ans: 1) Cluster 1 contains mostly 7's, 9's and 4's. 2) Cluster 2 contains almost exclusively 1's. 3) Cluster 3 contains 3's, 6's and 5's. 4) Cluster 4 contains almost all 2's. 5) Cluster 5 is mostly 0's.

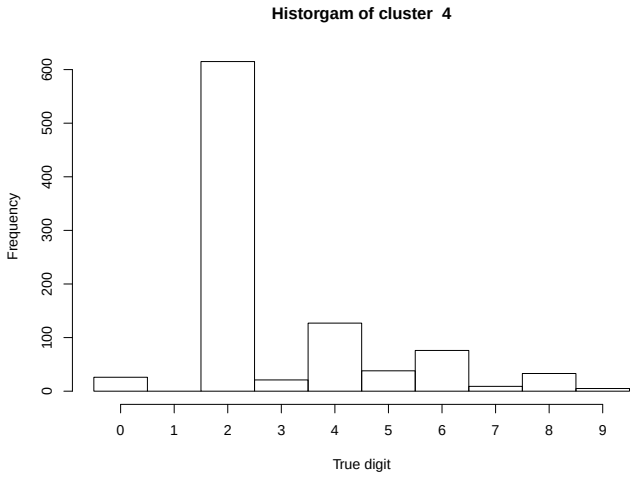
The dataset contains a lot more of 0's and 1's so they are more prevalent and got their own (almost) exclusive clusters. They are also simpler in shape, more distinctive from other digits in their corresponding images, hence they got recognized/clustered better. 2 is also a relatively easy to recognizable shape so it also got its own cluster

```r
library(ElemStatLearn)
data(zip.train)
set.seed(0)
K <- 5
k_result <- myKmeans(zip.train[, 2:257], K)
k_histograms <- function(k_result, K) {
  par(c(1, K))
  for(i in 1:K) {
    hist(zip.train[k_result==i, 1], main=paste("Historgam of cluster ", i), breaks=-.5:9.5, xaxt="n", x
    axis(side=1, at=0:9, labels=0:9)
  }
}
k_histograms(k_result, K)
```









- [10 Points] Perform your algorithm with 10 independent initiations with $k = 5$ and record the best
  - For this question, plot your clustering results on a two-dimensional plot, where the two axis are the first two principle components of your data

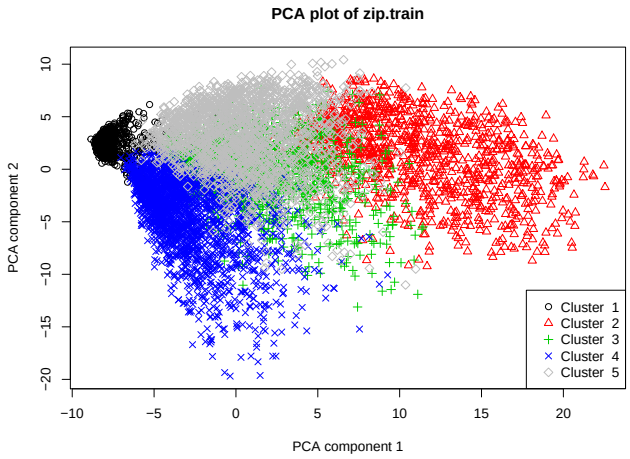
Ans: With the built-in k-means, we got a similar result: we have clusters dedicated to 0, 1 and 2. Then 3, 5, 6 are in a cluster, 4, 7 and 9 are in the last cluster.

```
best_ss <- Inf
for(initiation_num in 1:10) {
```

```
  mydata <- zip.train[, 2:257]
  k_result <- myKmeans(mydata, K)
  rows.centers <- t(centers[, k_result])
  ss <- sum((mydata - rows.centers)^2)

  is_better_ss <- (ss < best_ss)
  print(paste(ifelse(is_better_ss, "*", ""), "Last SS: ", best_ss, ", New SS: ", ss, sep=""))
  if(is_better_ss) {
    best_ss <- ss
    best_k_result <- k_result
  }
}
```

```
## [1] "*Last SS: Inf, New SS: 640171.986769608"
## [1] "Last SS: 640171.986769608, New SS: 642921.336039735"
## [1] "Last SS: 640171.986769608, New SS: 642921.14050212"
## [1] "Last SS: 640171.986769608, New SS: 643131.774811025"
## [1] "*Last SS: 640171.986769608, New SS: 639699.024513558"
## [1] "Last SS: 639699.024513558, New SS: 640163.762790428"
## [1] "Last SS: 639699.024513558, New SS: 642920.951600597"
## [1] "Last SS: 639699.024513558, New SS: 639700.95749664"
## [1] "Last SS: 639699.024513558, New SS: 642921.336039735"
## [1] "Last SS: 639699.024513558, New SS: 639699.024513558"
```

```
pca <- prcomp(mydata, scale=T)
color_codes <- c(1:4, 8)
pcaPlot <- function(k_result, K) {
  plot(pca$x[,1], pca$x[,2], type="n", main="PCA plot of zip.train", xlab="PCA component 1", ylab="PCA
  for(k in 1:K) {
    cluster.points <- pca$x[k_result==k, ]
    points(cluster.points[,1], cluster.points[,2], pch=k, col=color_codes[k])
  }
  legend("bottomright", legend=paste("Cluster ", 1:5), pch=1:K, fill=F, border=0, color_codes, color_co
}
pcaPlot(k_result, K)
```

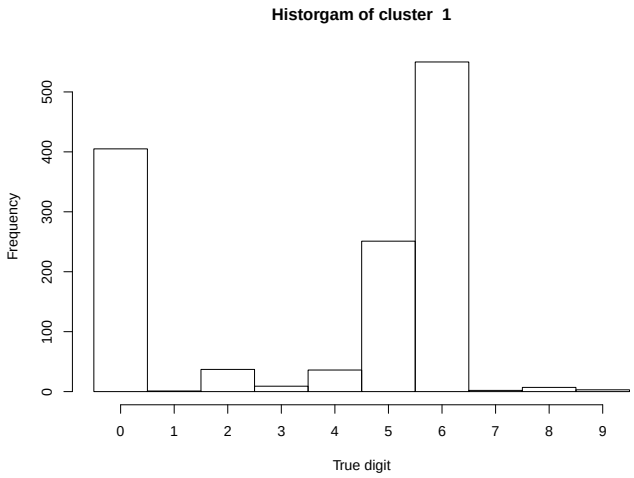


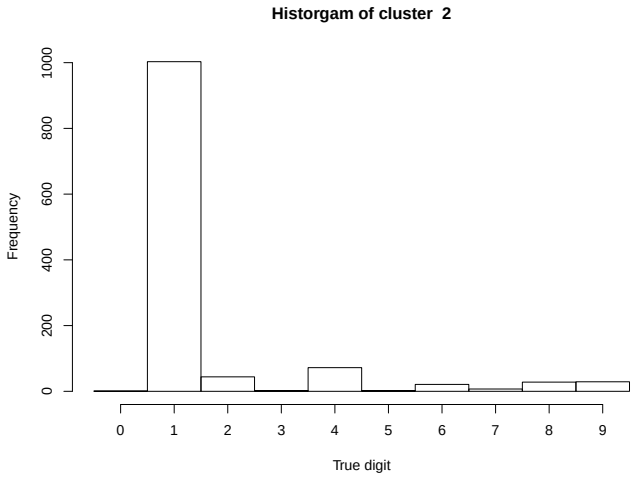
Ans: Each point has coordinates by the first 2 PCA components, the cluster they belong to are uniquely colored

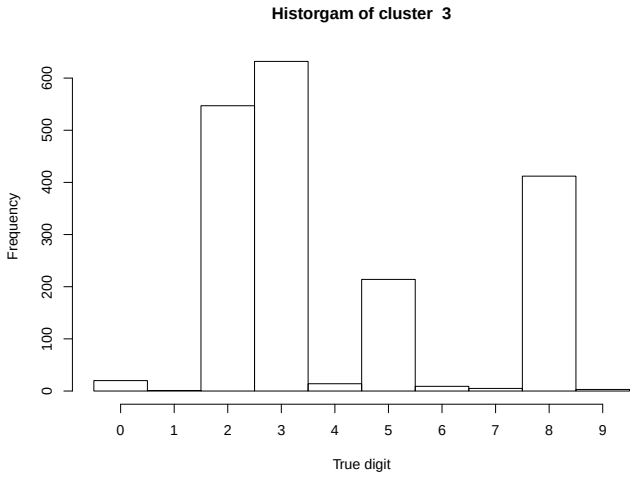
- [15 Points] Compare the clustering results from the above two questions with the built-in `kmeans()`

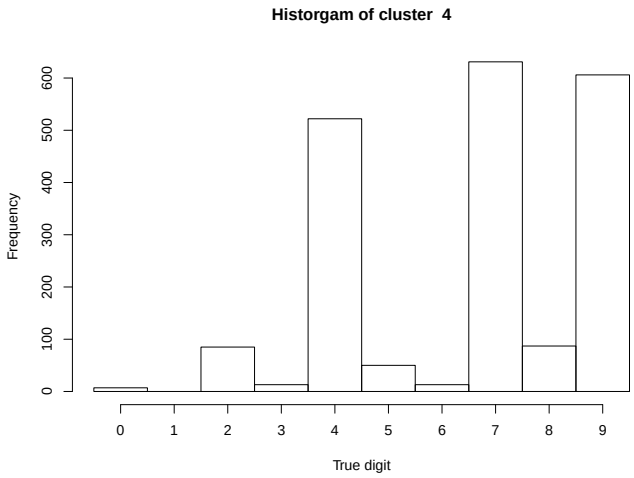
function in R. Use tables/figures to demonstrate your results and comment on your findings.

```
lib_result <- kmeans(mydata, centers=K)
k_histograms(lib_result$cluster, K)
```



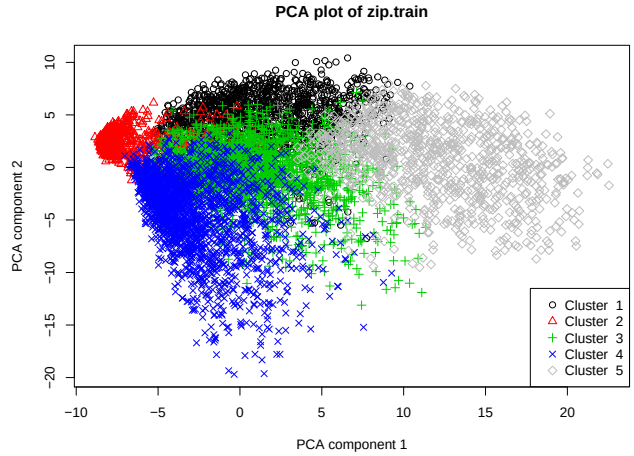






Ans: The colourings are swapped around, but the cluster shapes are similar to our kmeans implementation.

```
pcaPlot(lib_result$cluster, K)
```



Question 2 [50 Points] Two-dimensional Gaussian Mixture Model

We consider an example of the EM algorithm, which fits a Gaussian mixture model to the Old Faithful eruption data. For a demonstration of this problem, see the figure provided on Wikipedia. As the end result, we will obtain the distribution parameters of the two underlying distributions. We consider the problem as

follows. For this question, you are allowed to use packages that calculate the densities of normal distributions.

- We use both variables `eruptions` and `waiting`. We assume that the underlying distributions given the unobserved latent variables are both two-dimensional normal: $N(\mu_1, \Sigma_1)$ and $N(\mu_2, \Sigma_2)$, respectively, while μ_1, Σ_1, μ_2, and Σ_2 are unknow parameters that we need to solve.
- We assume that the unobserved latent variables (that indicate the membership) follow i.i.d. Bernoulli distribution, with parameter p.
- Based on the logic of an EM algorithm, we will first initiate some values of the parameters in the normal distribution. I provided a choice of them, and the normal density plots based on the initial values.
- Your goal is to write the EM algorithm that progressively updates the parameters and the latent variable distribution parameter. Eventually, we will reach a stable model fitting result that approximate the two underlying distributions, as demonstrated on the Wikipedia page. Choose a reasonable stopping criterion. To demonstrate your results, you should provide at least the following information.
  - The distribution parameters p, μ_1, Σ_1, μ_2, and Σ_2
  - A histogram of the underlying probabilities of the latent variables
  - Plot the normal densities at the 2nd, 3rd, 4th and the final iteration of your algorithm

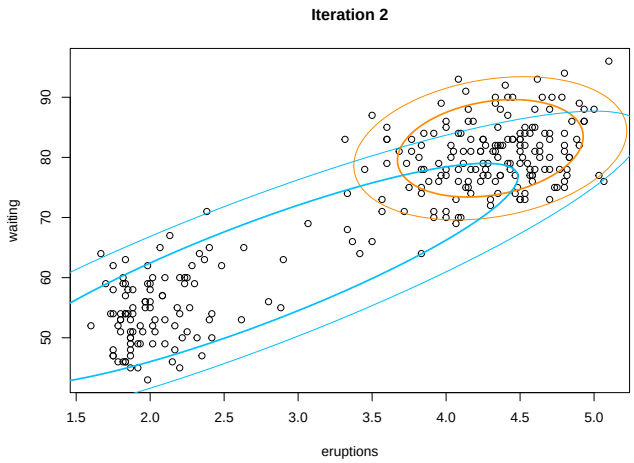
Ans: The final results are: P = 0.35 mu_1 = 4.29, 80 Sigma1 = 0.1677268 0.9134951 0.9134951 35.7592252 mu_2 = 2.04, 54.5 Simga2 = 0.07099924 0.4565369 0.45653691 33.8842131

```
library(mixtools)
library(mnormt)
library(Matrix)
# load the data
faithful = read.table("..//data//faithful.txt")

# the parameters
mu1 = c(3, 80)
mu2 = c(3.5, 60)
Sigma1 = matrix(c(0.1, 0, 0, 10), 2, 2)
Sigma2 = matrix(c(0.1, 0, 0, 50), 2, 2)

par(c(1,5))
```

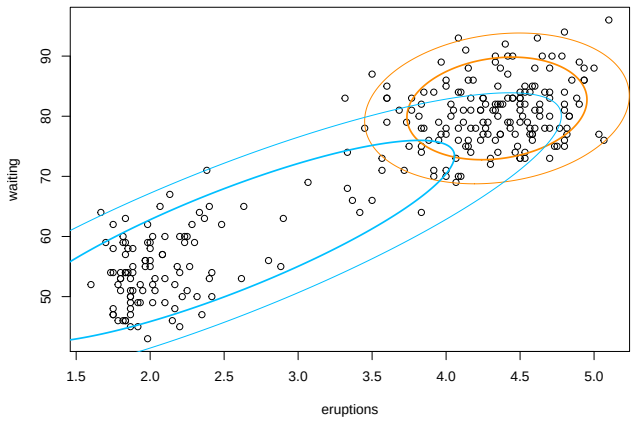
```
## NULL
```

```
addellipse <- function(mu, Sigma, ...)
{
  ellipse(mu, Sigma, alpha = .05, lwd = 1, ...)
  ellipse(mu, Sigma, alpha = .25, lwd = 2, ...)
}

plotEm <- function(result, iteration) {
  plot(faithful, main=paste("Iteration", iteration))

  addellipse(result$mu1, result$Sigma1, col = "darkorange")
  addellipse(result$mu2, result$Sigma2, col = "deepskyblue")
}

em <- function(x, mu1, mu2, Sigma1, Sigma2) {
  PI <- 0.5
  last_ez <- rep(Inf, nrow(faithful))
  for( iterations in 1:100 ) {
    # E step
    d1 <- PI * dmnorm(x, mean=mu1, varcov=Sigma1)
```

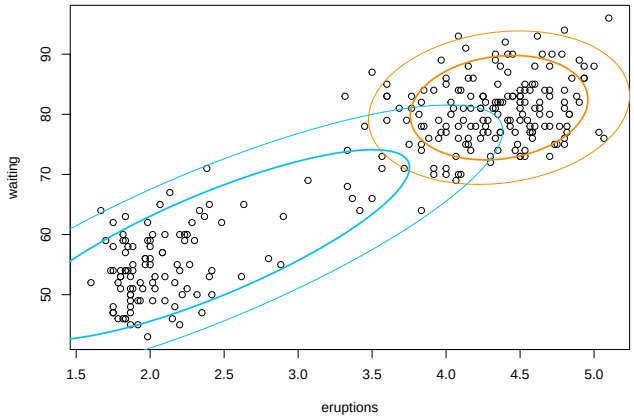
```
    d2 <- (1-PI) * dmnorm(x, mean=mu2, varcov=Sigma2)
    ez <- d2 / (d1+d2)
    if(sum((ez-last_ez)^2) < 0.001) {
      break
    }
    last_ez <- ez

    # M step
    PI <- mean(ez)
    mu1 <- colSums((1-ez)*x) / sum(1-ez)
    mu2 <- colSums(ez*x) / sum(ez)

    temp1 <- (t(x)-mu1)
    Sigma1 <- (temp1 %*% ((1-ez)*t(temp1))) / sum(1-ez)
    Sigma1 <- as.matrix(forceSymmetric(Sigma1))
    temp2 <- (t(x)-mu2)
    Sigma2 <- (temp2 %*% (ez*t(temp2))) / sum(ez)
    Sigma2 <- as.matrix(forceSymmetric(Sigma2))

    result <- list(mu1=mu1, mu2=mu2, Sigma1=Sigma1, Sigma2=Sigma2, PI=PI, ez=ez)
    if(1 < iterations && iterations < 5){
      plotEm(result, iterations)
    }
  }
  plotEm(result, iterations)
  plot(density(ez), main="Z=1 probability distribution")
  result
}
result <- em(faithful, mu1, mu2, Sigma1, Sigma2)
```



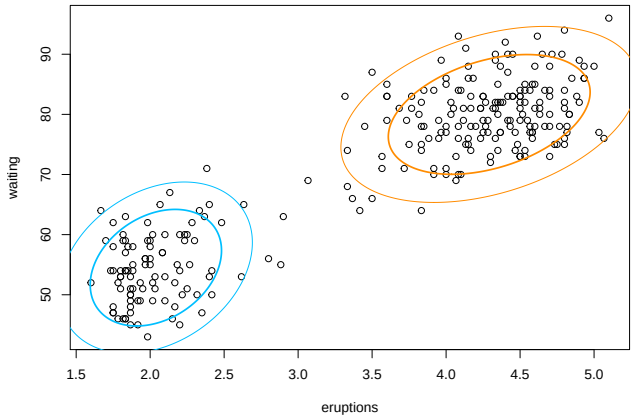
Iteration 3

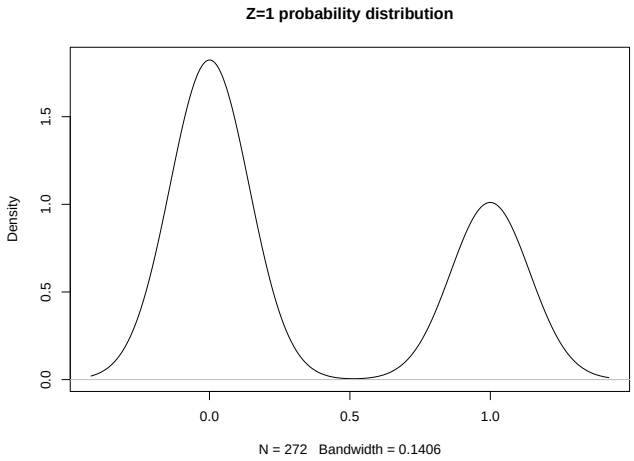


Iteration 4



Iteration 14

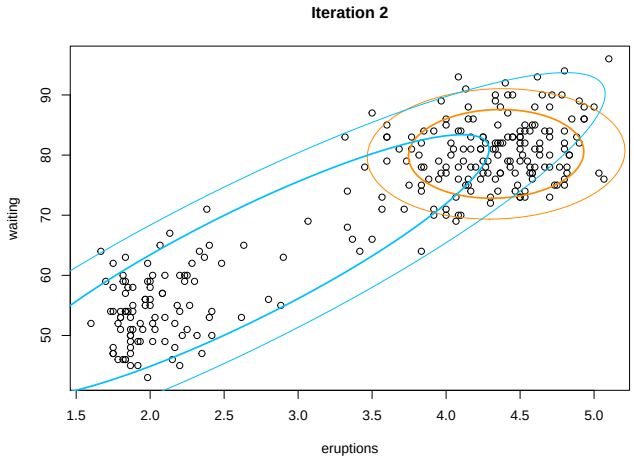


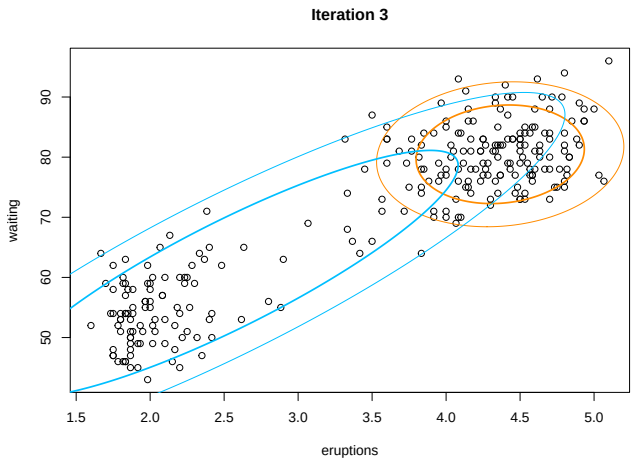


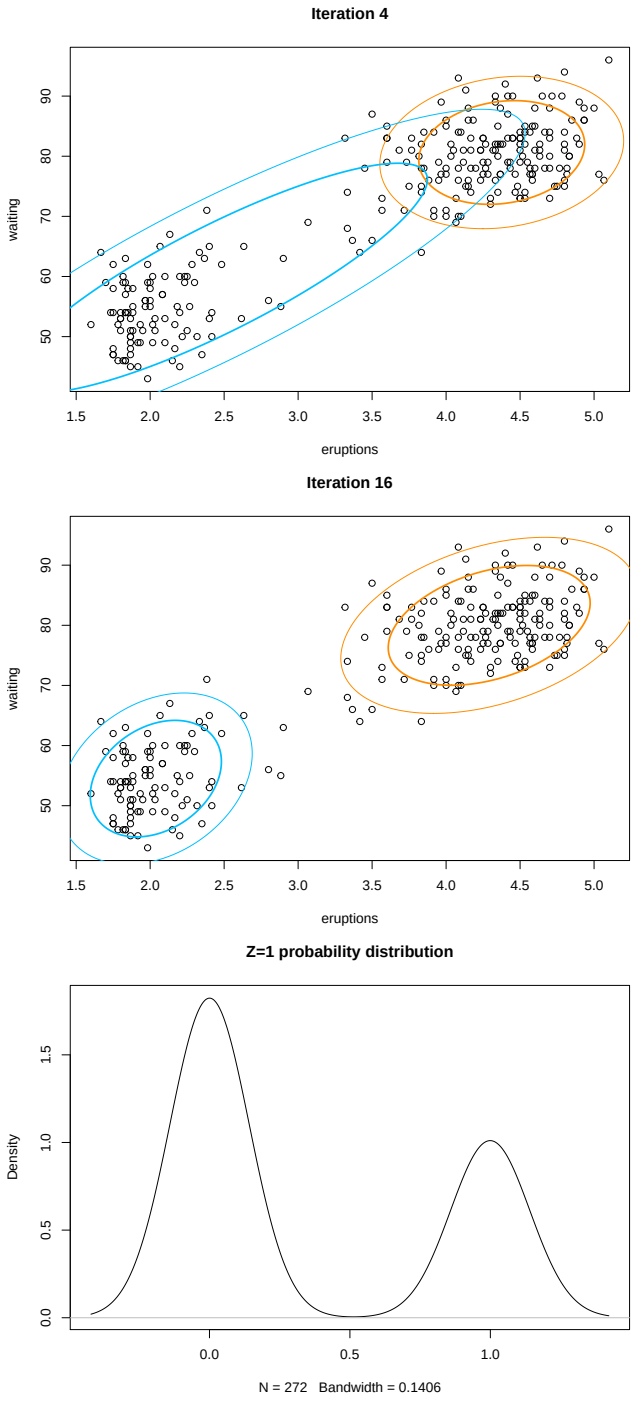
- Now, experiment a very different initial value of the parameters and rerun the algorithm. Comment on the efficiency and convergence speed of this algorithm.

Ans: Even though I set the inital centers for both clusters to be the same, they still managed to pull themselves apart after a few more iterations

```
result <- em(faithful, mu1, mu1, Sigma1, Sigma2)
```



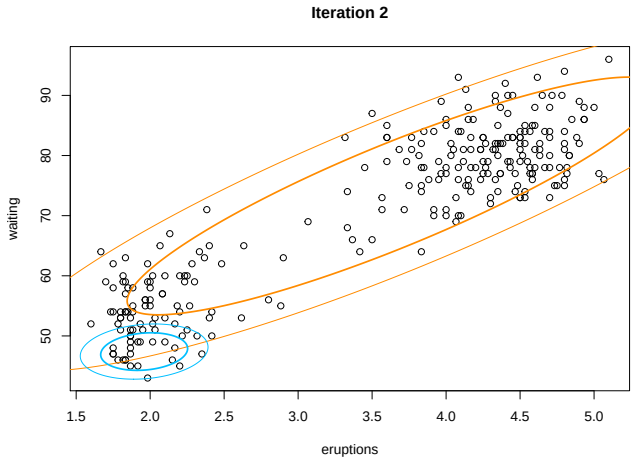


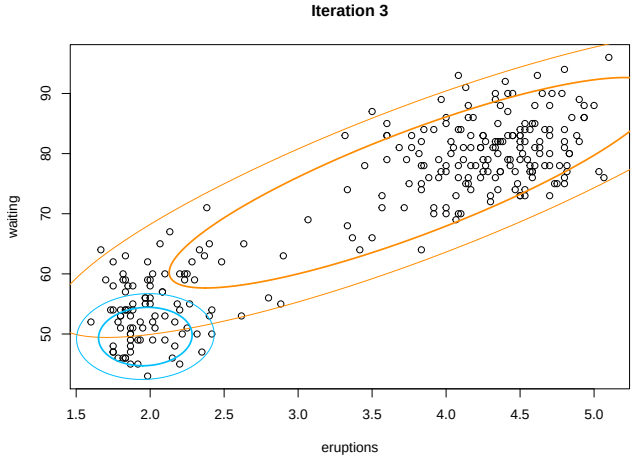


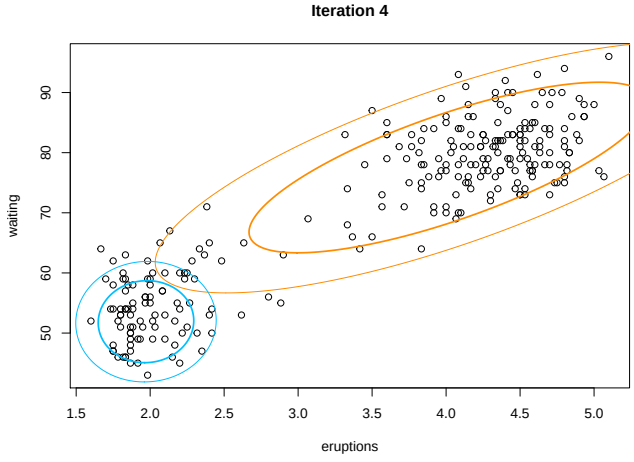
Next, I assumed I have no idea what the distribution of the data is, and just blindly set the centers very far away

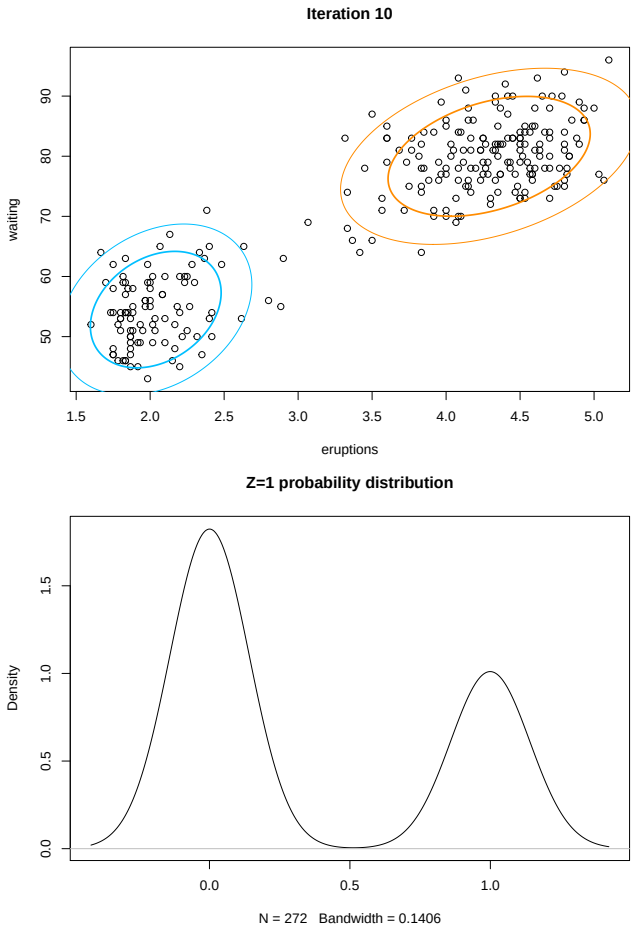
When I set the inital mu values to be very far out, I have to also set a higher sigma value so that the normal density function won't return 0's on all the values. But it doesn't impact the speed of convergence, in fact it converged faster in this case

```
result <- em(faithful, c(-100, 100), c(100,-100), matrix(c(200, 100, 100, 200), 2, 2), matrix(c(200, 100
```








Iteration 10
waiting
eruptions
50
60
70
80
90
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Z=1 probability distribution
Density
0.0
0.5
1.0
1.5
N = 272 Bandwidth = 0.1406