CS 598: Homework 5

Pushpit Saxena (netid: pushpit2)

Contents

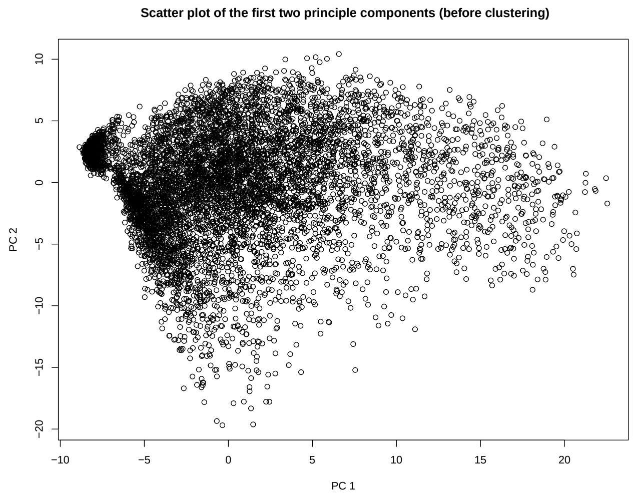
Question 1 K-Means Clustering

- Load the `zip.train`

```
library(ElemStatLearn)
data(zip.train)
X <- as.matrix(zip.train[, -c(1)])
Y <- as.matrix(zip.train[, 1])
k=5
C <- array(numeric(), dim=c(k,256))
set.seed(13)
for (i in 1:k){
  C[i,] <-runif(ncol(X), min = apply(X, 2, min), max = apply(X, 2, max))
}
```

- Generating Principle components of the data (to be used in the plots latter):

```
pr1 <- prcomp(X, scale = T)
plot(pr1$x[,1], pr1$x[,2], xlab="PC 1",
     ylab="PC 2",
     main=" Scatter plot of the first two principle components (before clustering)")
```



- Method to plot the clustering results

```
plot.kmeans <- function(pr1, clusters, plot_with_ellipse_or_centroid=1, title="MyKmeans") {
  df = data.frame("PC1" = pr1$x[,1], "PC2" = pr1$x[,2], "Cluster" = as.factor(clusters))
  centroids <- aggregate(df[, 1:2], list("cluster"=df$Cluster), mean)
  library(ggplot2)
  ggplot(df, aes(df$PC1,df$PC2, color=df$Cluster, shape=df$Cluster)) +
    geom_point() + stat_ellipse(type="norm", linetype=2) +
    stat_ellipse(type="t") +
    labs(x="PC 1", y="PC 2", title = paste(title, " Clustering results")) +
    scale_color_discrete(name="Clusters") + scale_shape_discrete(name="Clusters")
}
```

Write your own k-means:

```
library(Rfast)
clusterAssignment <- function(X,c) {
  distMat <- dista(X, c)
  clusters <- as.matrix(apply(distMat, 1, which.min))
  return(list("clusters" = clusters, "distMat" = distMat))
}

mykmeans <- function(X, C, K=5) {
  centroids <- C
  clustersAndDist <- clusterAssignment(X, centroids)
  iter <- 1
  while (TRUE) {
    oldCluster <- clustersAndDist
    newCentroids <-
      t(sapply(1:K, function(c) colMeans(X[which(clustersAndDist$clusters == c), ])))
    clustersAndDist <- clusterAssignment(X, newCentroids)
    if (identical(oldCluster$clusters, clustersAndDist$clusters)) {
      centroids <-
        t(sapply(1:K, function(c) colMeans(X[which(clustersAndDist$clusters == c), ])))
      # cat("Kmeans converged at iteration: ", iter, "\n")
      break
    }
    iter <- iter + 1
  }
  dataForWithinness <-
    data.frame(cluster=clustersAndDist$clusters,
               distance = apply(clustersAndDist$distMat, 1, min) ^ 2)
  diff.d <- aggregate(dataForWithinness[, 2],
                      list("cluster" = dataForWithinness$cluster), sum)
  mytotWithinss <- sum(diff.d$x)
  return (list("clusters" = clustersAndDist, "centroids" = centroids,
               "tot.withiness" = mytotWithinss, "covergence_iter" = iter))
}

getMembershipCount <- function(X, clusters) {
  clusterCountData <- array(0, dim=c(k, 10))
  for(i in 1:nrow(Y)) {
    cluster_id <- clusters[i]
    clusterCountData[cluster_id, Y[i] + 1] <- clusterCountData[cluster_id, Y[i] + 1] + 1
  }

  colnames(clusterCountData) <- c("Digit 0", "Digit 1", "Digit 2",
                                  "Digit 3", "Digit 4", "Digit 5",
                                  "Digit 6", "Digit 7", "Digit 8", "Digit 9")
  rownames(clusterCountData) <- c("Cluster 1", "Cluster 2",
                                  "Cluster 3", "Cluster 4", "Cluster 5")
  clusterCountdf <- data.frame(clusterCountData)
  clusterCountdf
  return(clusterCountdf)
}
```

Perform your algorithm with one random initialization with $k = 5$

```
mykmean.fit.single <- mykmeans(X, C)
cat("Total withinness for 1 random intialization clustering is ", mykmean.fit.single$tot.withiness, "\n
```

```
## Total withinness for 1 random intialization clustering is  640165.2
```

- Compare your cluster membership to the true digits

```
clusterCountdf <- getMembershipCount(X, mykmean.fit.single$clusters$clusters)
clusterCountdf
```

	Digit.0	Digit.1	Digit.2	Digit.3	Digit.4	Digit.5	Digit.6	Digit.7	Digit.8	Digit.9
Cluster 1	7	0	85	13	524	51	13	631	87	605
Cluster 2	1	1003	44	2	72	2	21	7	28	30
Cluster 3	761	0	18	2	8	38	71	0	8	3
Cluster 4	22	1	547	633	14	224	9	5	413	3
Cluster 5	403	1	37	8	34	241	550	2	6	3

- Most prevalent digit in each cluster:

```
max_val_Df <- data.frame(Cluster = rownames(clusterCountdf),
                         "MostPrevalentDigit" =
                           colnames(clusterCountdf)[apply(clusterCountdf, 1, which.max)])
max_val_Df
```

Cluster	MostPrevalentDigit
Cluster 1	Digit.7
Cluster 2	Digit.1
Cluster 3	Digit.0
Cluster 4	Digit.3
Cluster 5	Digit.6

Comparing single initialization clustering with "kmeans" from library functions (even though it is asked later I am putting it here for single run for better clarity and consistency, comparison with 10 runs will come later):

```
kmeans.fit.single <- kmeans(X, C)
```

The most frequent digits given by **built-in kmeans** and mykmeans are matching(See earlier and below)

```
kmclusterCountdf <- getMembershipCount(X, kmeans.fit.single$cluster)
max_val_Df <- data.frame(Cluster = rownames(kmclusterCountdf),
                         "MostPrevalentDigit" =
                           colnames(kmclusterCountdf)[apply(kmclusterCountdf, 1, which.max)])
max_val_Df
```

Cluster	MostPrevalentDigit
Cluster 1	Digit.7
Cluster 2	Digit.1
Cluster 3	Digit.0
Cluster 4	Digit.3
Cluster 5	Digit.6

- Withiness diff:

```
abs(mykmean.fit.single$tot.withiness - kmeans.fit.single$tot.withinss)
```

```
## [1] 2.528433
```

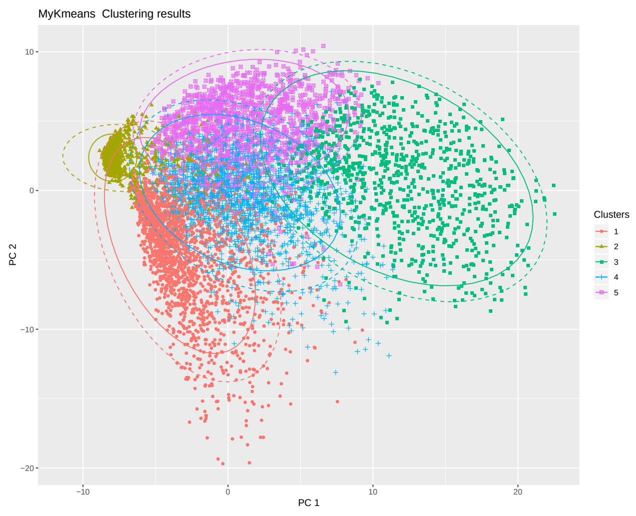
- Clustering results:

```
z <- (mykmean.fit.single$clusters$clusters == kmeans.fit.single$cluster)
cat("Ratio of records for which the cluster assignment is same
    between one random run of my kmeans vs kmeans
    library function (with same initial centroids) is
    ", length(z[z==TRUE])/nrow(Y), "\n")
```

```
## Ratio of records for which the cluster assignment is same
##       between one random run of my kmeans vs kmeans
##       library function (with same initial centroids) is
##        0.9971197
```

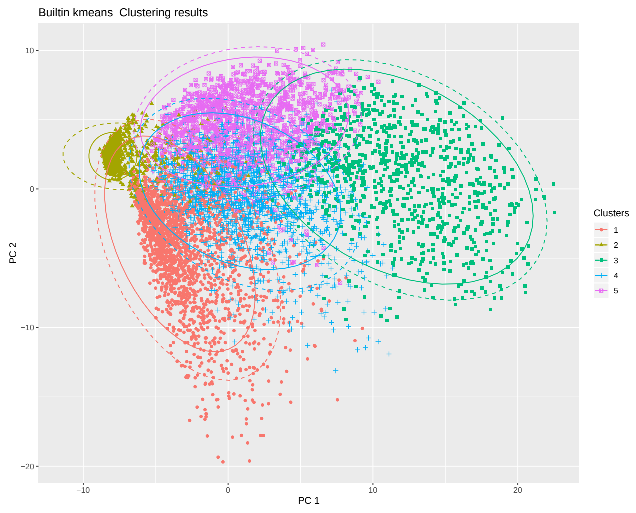
- Plot for 1 random run for myKmeans:

```
plot.kmeans(pr1, mykmean.fit.single$clusters$clusters)
```



- Plot for builtin kmeans with same centroids:

```
plot.kmeans(pr1, kmeans.fit.single$cluster, title = "Builtin kmeans")
```



- Difference between the mykmeans centroids and kmeans centroids is also very small:

```
mean((mykmean.fit.single$centroids - kmeans.fit.single$centers)^2)
```

```
## [1] 7.403067e-06
```

So, in conclusion mykmeans is performing equivalently good as the built in k-means for 1 random initialization, with 10 different initialization I was able to get better results(see below).

Perform your algorithm with 10 independent initiations with k=5 and record the best

- Code to run mykmeans for 10 different initiations

```
set.seed(1513)
C <- array(numeric(), dim=c(k,256))
bestMyKmeansFit <- NULL
```

```
best_C <- NULL
for (iter in 1:10) {
  for (i in 1:k){
    C[i,] <-runif(ncol(X), min = apply(X, 2, min), max = apply(X, 2, max))
  }
  mykmeans.fit <- mykmeans(X, C)
  if (iter == 1 || mykmeans.fit$tot.withiness < bestMyKmeansFit$tot.withiness) {
    bestMyKmeansFit <- mykmeans.fit
    best_C <- C
  }
}
```

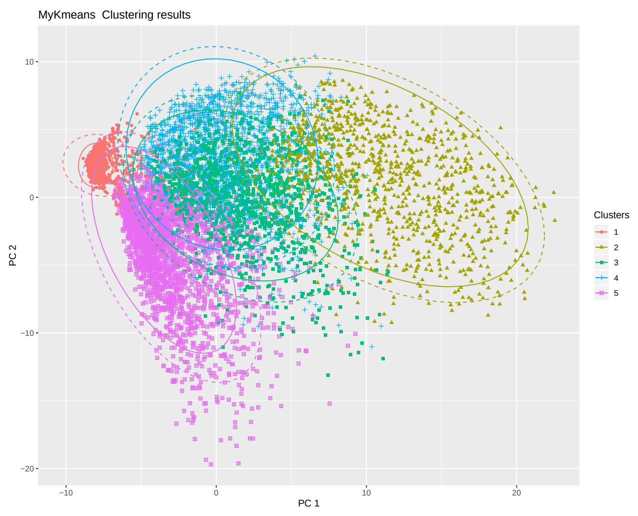
- The total withiness for the best clustering:

```
bestMyKmeansFit$tot.withiness
```

```
## [1] 639700.8
```

- Plot your clustering results on a two-dimensional plot, where the two axis are the first two principle components of your data

```
plot.kmeans(pr1, bestMyKmeansFit$clusters$clusters)
```



Compare the clustering results from the above two questions with the built-in kmeans() function in R

```
kmeans_fit <- kmeans(X, best_C)
kmeans_best_fit <- kmeans(X, 5)
```

- Total withiness diff between mykmeans and builtin kmeans:

```
abs(bestMyKmeansFit$tot.withiness - kmeans_fit$tot.withinss)
```

```
## [1] 1.940864
```

- Clustering results:

```
  z <- (bestMyKmeansFit$clusters$clusters == kmeans_fit$cluster)
  cat("Ratio of records for which the cluster assignment
      is same between one random run of my kmeans vs kmeans
      library function (with same initial centroids) is
      ", length(z[z==TRUE])/nrow(Y), "\n")
```

```
## Ratio of records for which the cluster assignment
##       is same between one random run of my kmeans vs kmeans
##       library function (with same initial centroids) is
##        0.9984913
```

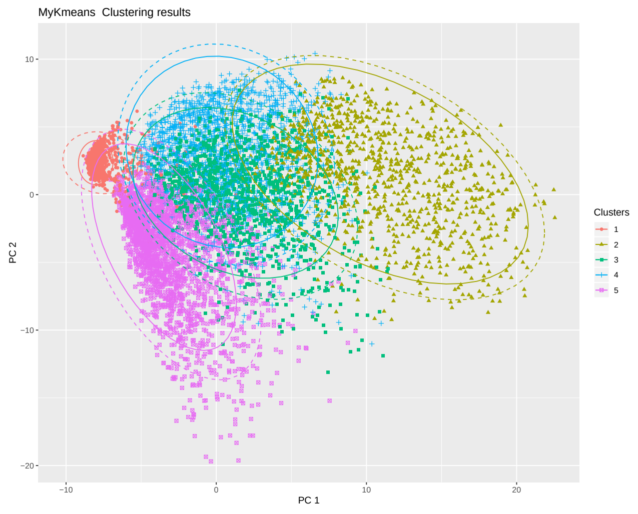
- Difference between the mykmeans centroids and kmeans centroids is very small:

```
mean((bestMyKmeansFit$centroids - kmeans_fit$centers)^2)
```

```
## [1] 2.016035e-06
```

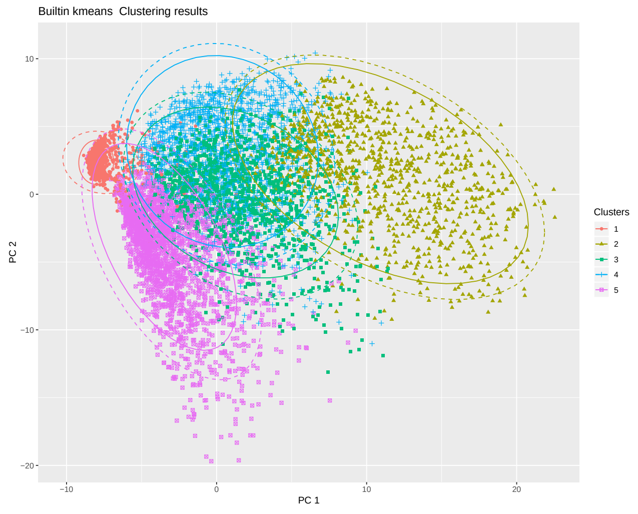
- Plot for mykmeans (best selected based on lowest totalwithiness as choosen above)

```
plot.kmeans(pr1, bestMyKmeansFit$clusters$clusters)
```



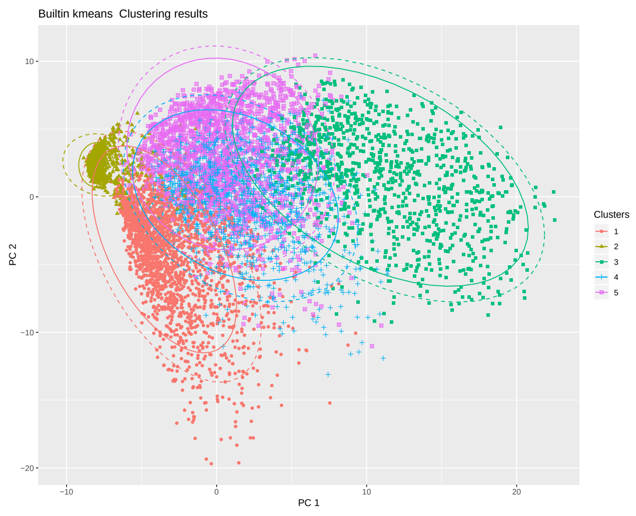
- Plot for builtin kmeans (with centroid provided):

```
plot.kmeans(pr1, kmeans_fit$cluster, title = "Builtin kmeans")
```



- Plot for builtin kmeans (with no centroid provided):

```
plot.kmeans(pr1, kmeans_best_fit$cluster, title = "Builtin kmeans")
```



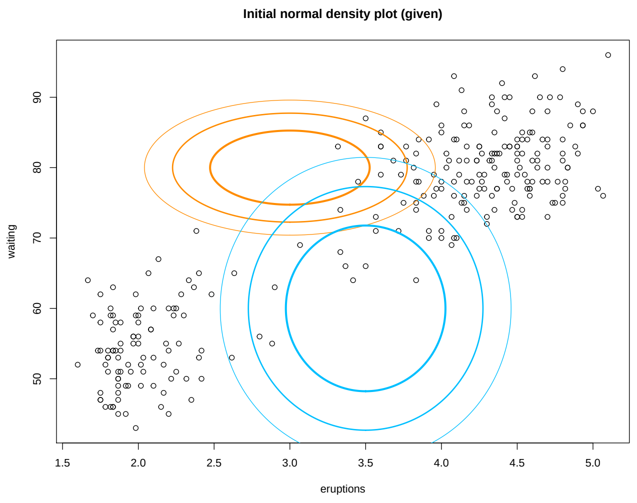
- **Comparision summary**: I have used a couple of stats here *tot.withiness* and *iterationToConvergence* to compare my implemenation of kmeans vs the builtin kmeans. It can be clearly seen that with my implementation of kmeans I am able to achieve similar level of total withiness for the dataset as the kmeans function but my implementation generally takes more iteration to converge. On reading a litte bit about this I think the convergence condition that I am using to check and the one used in kmeans are different as well the initialization are also different. If I got more time, I would have experimented with different convergence conditions and see if I can reduce the number of iterations it takes for my kmeans implemenation to converge

Model	tot.withiness	iterationToConvergence
SingleInit-MyKmeans	640165.2	41
BuiltInKMeansWithSameInitCentroids	640162.7	5
10InitBestMyKmeans	639700.8	26
BuiltInKMeansWithBestInitCentroids	639698.9	6
BestBuiltinKmeans	639698.9	5

Question 2 Two-dimensional Gaussian Mixture Model

- Loaded the data and initialize the mu1, mu2, Sigma1, Sigma2

```
# load the data
  faithful = as.matrix(read.table("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)
  library(mixtools)
  plot(faithful, main="Initial normal density plot (given)")

  addellipse <- function(mu, Sigma, ...)
  {
    ellipse(mu, Sigma, alpha = .05, lwd = 2, ...)
    ellipse(mu, Sigma, alpha = .25, lwd = 3, ...)
    ellipse(mu, Sigma, alpha = .01, lwd = 1, ...)
  }

  addellipse(mu1, Sigma1, col = "darkorange")
  addellipse(mu2, Sigma2, col = "deepskyblue")
```



EM Algorithm:

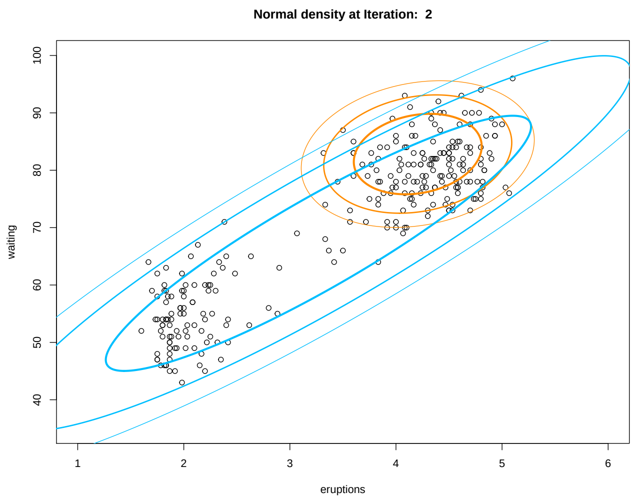
- I have used several convergence criteria:
  - Difference between component means becomes lower than the set threshold between two runs (use convergence_criteria="mean")
  - Difference between the expected value of log-likelihood becomes smaller than the set threshold (default method, can be switched to other methods)
  - Difference between the logsum of responsibilities(gamma) values become lower than the set threshold (use convergence_criteria="gamma")

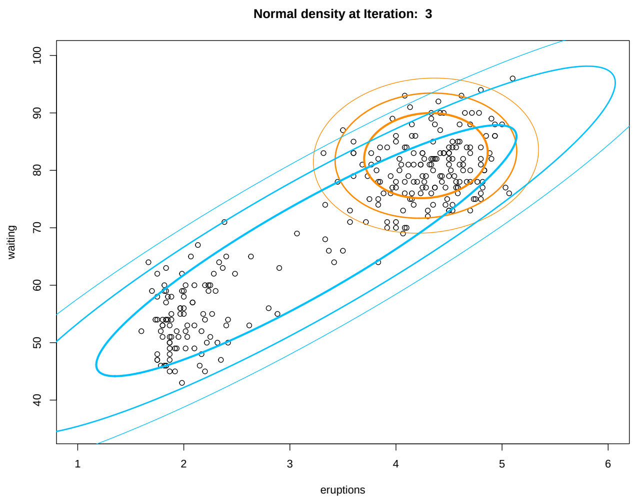
```
library(lattice)
gmm.fromscratch <- function(X, k, mu1, mu2, Sigma1, Sigma2,
                            convergence_criteria="llh", tolerance=1e-6, plot=TRUE)
  {
```

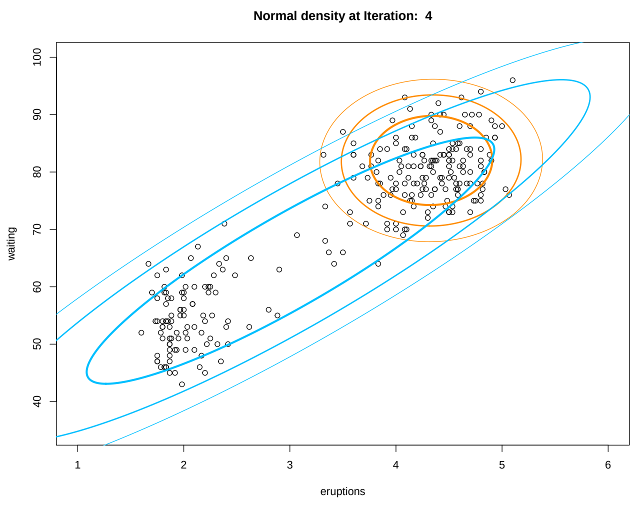
```
  Delta <- 1; iter <- 1;
  mu <- rbind(mu1, mu2)
  mu_mem <- mu
  cov <- list(Sigma1, Sigma2)
  llh <- 0; pi <- 0.5; best_gamma <- NULL; log_gamma <- 0
  best_pi <- 0
  while(TRUE) {
    # E-step
    z <- cbind(dmvnorm(X, mu[1, ], sigma = cov[[1]]),
               dmvnorm(X, mu = mu[2, ], sigma = cov[[2]]))

    r <- cbind((pi * z[, 1])/rowSums(t((t(z) * pi))),
               (1-pi * z[, 2])/rowSums(t((t(z) * (1-pi)))))

    new_llh <- sum(log(((1-pi) * z[,1]) + (pi * z[,2])))

    gamma <- (pi * z[,2])/(((1-pi) * z[,1]) + (pi *z[,2]))

    # M-step
    pi <- mean(gamma)
    # Update our Means
    mu <- rbind(colSums((1-gamma) * X)/sum(1-gamma), colSums((gamma * X)/sum(gamma)))
    # Update Sigma 1
    cov[[1]] <- Reduce("+", sapply(1:nrow(X), function(i) {
      ddd <- (X[i,] - mu[1,]) %*% t(X[i, ] - mu[1, ])
      mul <- (1 - gamma[i]) * ddd
    }, simplify = FALSE))/sum(1-gamma)
    # Update Sigma 2
    cov[[2]] <- Reduce("+", sapply(1:nrow(X), function(i) {
      ddd <- (X[i,] - mu[2,]) %*% t(X[i, ] - mu[2, ])
      mul <- gamma[i] * ddd
    }, simplify = FALSE))/sum(gamma)

    ifelse (convergence_criteria == "llh", {Delta <- abs(llh - new_llh); llh <- new_llh},
            ifelse(convergence_criteria == "mean", Delta <- sum((mu - mu_mem) ^ 2),
                   {Delta <- abs(log(sum(gamma)) - log_gamma); log_gamma <- log(sum(gamma))}))

    if (Delta < 1e-6) {
      best_gamma <- gamma
      best_pi <- pi
      break
    }

    if (plot && (iter == 2 || iter == 3 || iter == 4)) {
      plot(faithful, xlim = c(1, 6), ylim = c(35, 100), main=paste("Normal density at Iteration: ", ite
      addellipse(mu[1,], cov[[1]], col = "darkorange")
      addellipse(mu[2,], cov[[2]], col = "deepskyblue")
    }
    mu_mem <- mu; iter <- iter+1
  }

  return(list(means=mu, cov=cov, gamma=best_gamma, pi=best_pi, Z=z, r=r, final_iteration=iter))
}
```

Applying my EM algorithm implemented above on old-faitful data

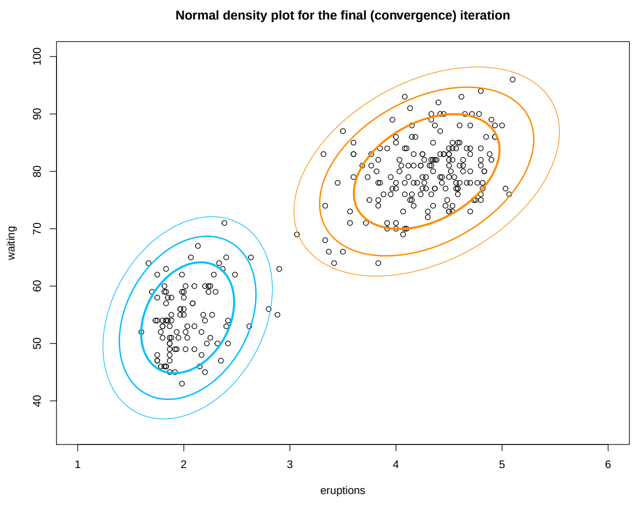
```r
set.seed(13)
X <- faithful
mu1 = c(3, 80)
mu2 = c(3.5, 60)
Sigma1 = matrix(c(0.1, 0, 0, 10), 2, 2)
Sigma2 = matrix(c(0.1, 0, 0, 50), 2, 2)

gmm <- gmm.fromscratch(X, 2, mu1, mu2, Sigma1, Sigma2, plot=T)
```







```r
plot(faithful, xlim = c(1, 6), ylim = c(35, 100),
     main="Normal density plot for the final (convergence) iteration")
addellipse(gmm$means[1,], gmm$cov[[1]], col = "darkorange")
addellipse(gmm$means[2,], gmm$cov[[2]], col = "deepskyblue")
```



```r
# Running gmm again without plotting the intermediate graphs this will be used to compare
# the efficiency with different experiments below
start.time = Sys.time()
gmm.noPlot <- gmm.fromscratch(X, 2, mu1, mu2, Sigma1, Sigma2, plot=F)
gmm.noPlot.tt <- Sys.time() - start.time

gmmCompDf <- data.frame("Model" = c("GMM with given values"),
                  "Iteration to convergence" = c(gmm.noPlot$final_iteration),
                  "Total RunTime" = c(paste(gmm.noPlot.tt)), stringsAsFactors = FALSE)
```

The Final distribution parameters p, μ_1, Σ_1, μ_2, and Σ_2

- μ_1, μ_2

eruptions	waiting

```r
gmm$means
```

eruptions	waiting
4.289662	79.96812
2.036389	54.47852

- Σ_1, Σ_2

```r
gmm$cov
```

```
## [[1]]
##      eruptions    waiting
## [1,] 0.1699681  0.9406045
## [2,] 0.9406045 36.0461574
##
## [[2]]
##       eruptions    waiting
## [1,] 0.06916794  0.4351704
## [2,] 0.43517040 33.6973010
```

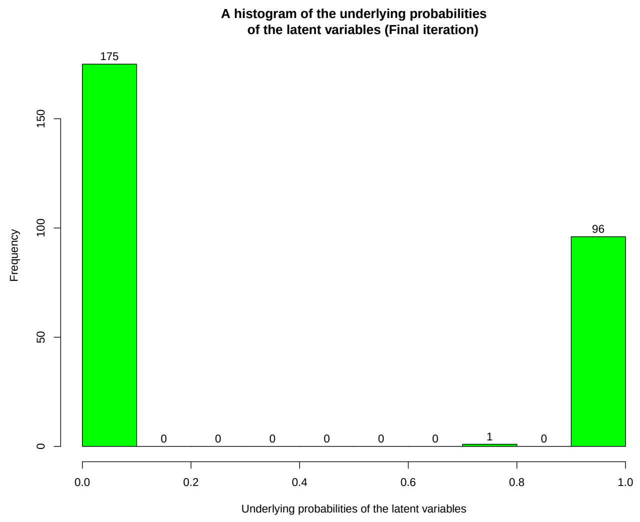
- p

```r
gmm$pi
```

```
## [1] 0.355873
```

A histogram of the underlying probabilities of the latent variables

```r
hist(gmm$gamma, xlab="Underlying probabilities of the latent variables",
main = "A histogram of the underlying probabilities
     of the latent variables (Final iteration)", col="green", labels=TRUE)
```



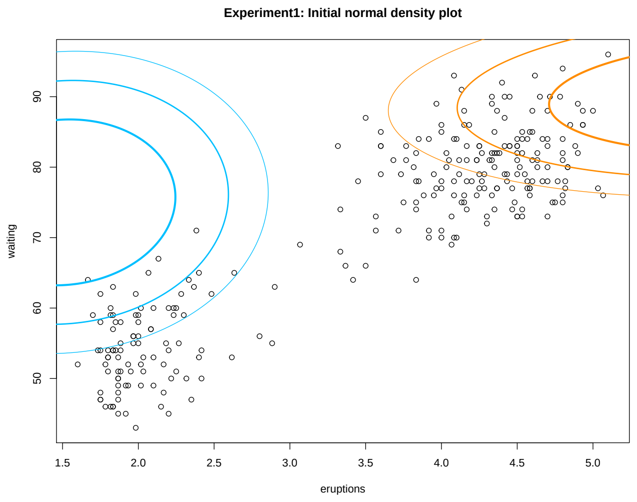
Plot the normal densities at the 2nd, 3rd, 4th and the final iteration of your algorithm

- Plot are already shown before for all these iterations

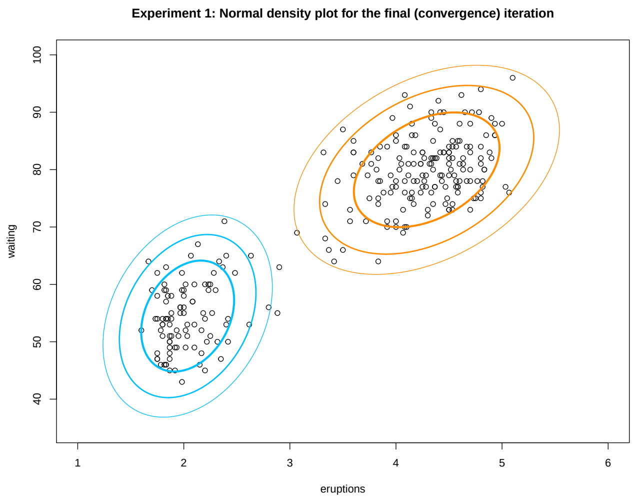
Now, experiment a very different initial value of the parameters and rerun the algorithm. Comment on the efficiency and convergence speed of this algorithm.

- Experiment 1: Manually selecting some different values for mean(μ_1, μ_2) and sigma(Σ_1, Σ_1)

```r
# the parameters
set.seed(13)
mu1 = c(6, 90)
mu2 = c(1.5, 75)
Sigma1 = matrix(c(0.6, 0.5, 0.5, 22), 2, 2)
Sigma2 = matrix(c(0.2, 0.2, 0.2, 50), 2, 2)
plot(faithful, main="Experiment1: Initial normal density plot")
addellipse(mu1, Sigma1, col = "darkorange")
addellipse(mu2, Sigma2, col = "deepskyblue")
```



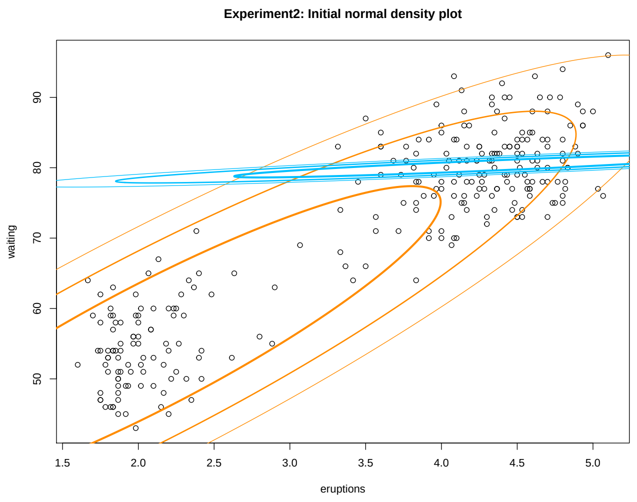
```r
start.time <- Sys.time()
gmm.exp1 <- gmm.fromscratch(X, 2, mu1, mu2, Sigma1, Sigma2, plot = FALSE)
gmm.exp1.tt <- Sys.time() - start.time
plot(faithful, xlim = c(1, 6), ylim = c(35, 100),
     main="Experiment 1: Normal density plot for the final (convergence) iteration")
addellipse(gmm.exp1$means[1,], gmm.exp1$cov[[1]], col = "darkorange")
addellipse(gmm.exp1$means[2,], gmm.exp1$cov[[2]], col = "deepskyblue")
```



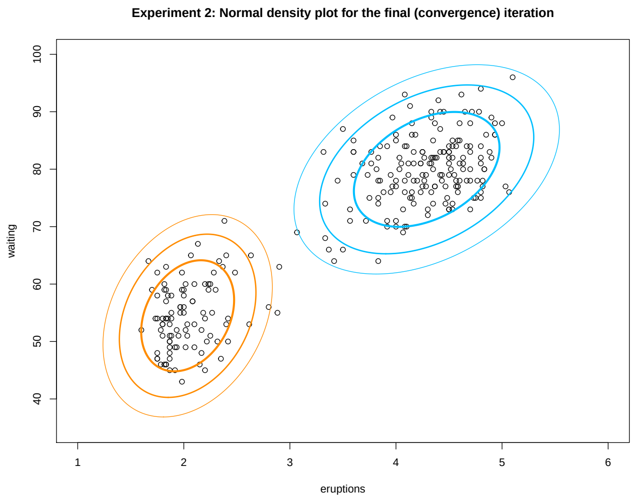
```
  gmmCompDf <- rbind(gmmCompDf,
                     c("Experiment 1",
                     gmm.exp1$final_iteration, paste(gmm.exp1.tt)))
```

- Experiment 2: Selecting means as **kmeans cluster centroids** and **cov** and **cor** as two Sigma matrix

```
set.seed(13)
km <- kmeans(X, 2)
mu1 <- km$centers[1,]
mu2 <- km$centers[2,]
Sigma1 = as.matrix(cov(X))
Sigma2 = as.matrix(cor(X))
plot(faithful, main="Experiment2: Initial normal density plot")
addellipse(mu1, Sigma1, col = "darkorange")
addellipse(mu2, Sigma2, col = "deepskyblue")
```



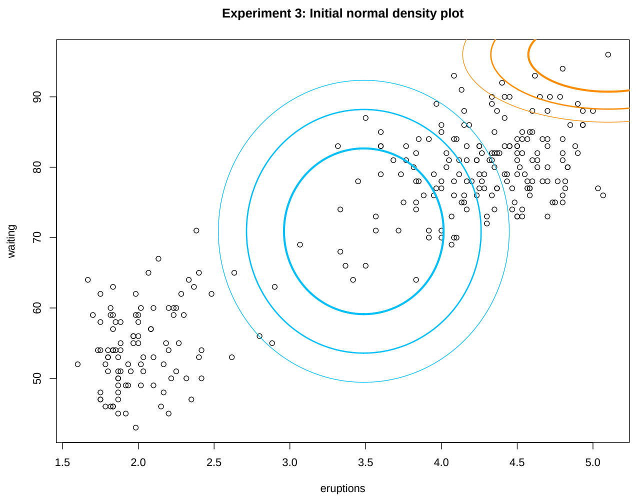
```
start.time <- Sys.time()
gmm.exp2 <- gmm.fromscratch(X, 2, mu1, mu2, Sigma1, Sigma2, plot = FALSE)
gmm.exp2.tt <- Sys.time() - start.time
plot(faithful, xlim = c(1, 6), ylim = c(35, 100),
     main="Experiment 2: Normal density plot for the final (convergence) iteration")
addellipse(gmm.exp2$means[1,], gmm.exp2$cov[[1]], col = "darkorange")
addellipse(gmm.exp2$means[2,], gmm.exp2$cov[[2]], col = "deepskyblue")
```



```
gmmCompDf <- rbind(gmmCompDf,
                   c("Experiment 2",
                   gmm.exp2$final_iteration, paste(gmm.exp2.tt)))
```

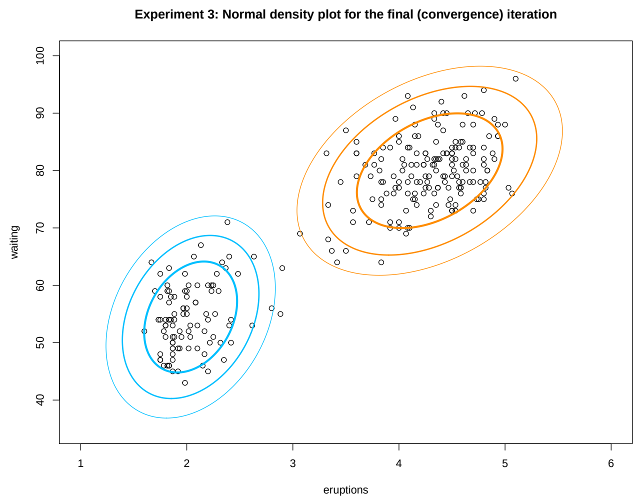
- Experiment 3: Selecting μ_1 as colMax(faithful) and μ_2 as colMeans(faithful), Σ_1, Σ_2 are the same as given one

```
  # the parameters
  set.seed(13)
  mu1 = c(apply(X, 2, max))
  mu2 = c(apply(X, 2, mean))
  Sigma1 = matrix(c(0.1, 0, 0, 10), 2, 2)
  Sigma2 = matrix(c(0.1, 0, 0, 50), 2, 2)
  plot(faithful, main="Experiment 3: Initial normal density plot")
  addellipse(mu1, Sigma1, col = "darkorange")
  addellipse(mu2, Sigma2, col = "deepskyblue")
```



```
  start.time <- Sys.time()
  gmm.exp3 <- gmm.fromscratch(X, 2, mu1, mu2, Sigma1, Sigma2, plot = FALSE)
  gmm.exp3.tt <- Sys.time() - start.time
  plot(faithful, xlim = c(1, 6), ylim = c(35, 100),
       main="Experiment 3: Normal density plot for the final (convergence) iteration")
  addellipse(gmm.exp3$means[1,], gmm.exp3$cov[[1]], col = "darkorange")
```

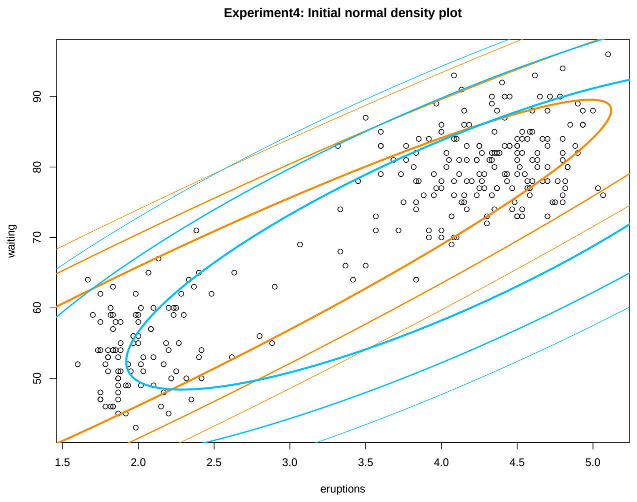
```r
addellipse(gmm.exp3$means[2,], gmm.exp3$cov[[2]], col = "deepskyblue")
```



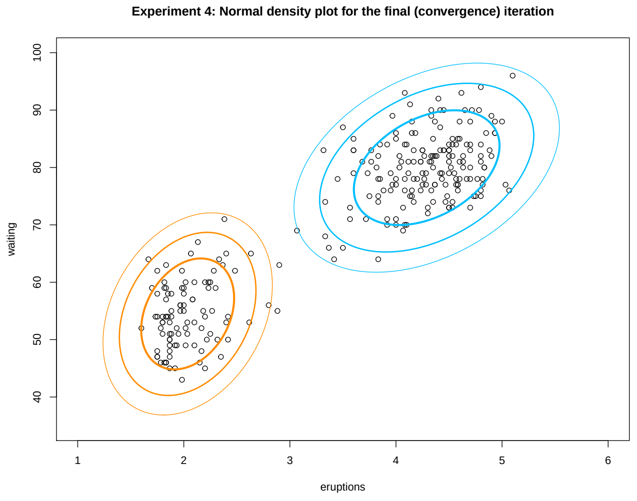
```r
gmmCompDf <- rbind(gmmCompDf,
                   c("Experiment 3",
                   gmm.exp3$final_iteration, paste(gmm.exp3.tt)))
```

- Experiment 4: Selecting values for intial μ and Σ as per the instructions in the book **Elements of Statistical Learning (Section 8.5)**

```r
set.seed(13)
dddd <- sample(seq_len(nrow(X)),size = 2)
mu <- X[dddd,]
# the parameters
mu1 = c(mu[1,])
mu2 = c(mu[2,])

Sigma1 <- as.matrix(Reduce("+", sapply(1:nrow(X), function(i) {
      ddd <- (X[i,] - mu[1,]) %*% t(X[i, ] - mu[1, ])
    }, simplify = FALSE))/nrow(X))

Sigma2 = as.matrix(Reduce("+", sapply(1:nrow(X), function(i) {
      ddd <- (X[i,] - mu[2,]) %*% t(X[i, ] - mu[2, ])
    }, simplify = FALSE))/nrow(X))

plot(faithful, main="Experiment4: Initial normal density plot")
addellipse(mu1, Sigma1, col = "darkorange")
addellipse(mu2, Sigma2, col = "deepskyblue")
```



```r
start.time <- Sys.time()
gmm.exp4 <- gmm.fromscratch(X, 2, mu1, mu2, Sigma1, Sigma2, plot = FALSE)
gmm.exp4.tt <- Sys.time() - start.time
plot(faithful, xlim = c(1, 6), ylim = c(35, 100),
     main="Experiment 4: Normal density plot for the final (convergence) iteration")
addellipse(gmm.exp4$means[1,], gmm.exp4$cov[[1]], col = "darkorange")
addellipse(gmm.exp4$means[2,], gmm.exp4$cov[[2]], col = "deepskyblue")
```



```r
gmmCompDf <- rbind(gmmCompDf,
                   c("Experiment 4",
                   gmm.exp4$final_iteration, paste(gmm.exp4.tt)))
```

- **Comparison:**

Following is the summary of different experiments (four), that I have done above and the iterations each of them took to converge and total run time also. I have found the my experiment 1 which is randomly (manually) selecting intial values converges the fatest. Although all of the experiments converges pretty quickly and performed really well on the dataset. Ideally we should define a grid of these initial values and peform multiple experiments to come up with the optimal initial values for the data. I liked the method given in ESL book (Experiment 4) as it also performed reasonably well and provide a methodical way of generating initial values

Model	Iteration.to.convergence	Total.RunTime
GMM with given values	20	0.130139112472534
Experiment 1	10	0.0632288455963135
Experiment 2	27	0.164202928543091
Experiment 3	25	0.151464939117432
Experiment 4	18	0.155349016189575