STA 6106 Project 4

Tyler Grimes

April 14, 2016

Mixture of Normals

Consider the density function

$$f(x; \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{j=1}^{k} \pi_j f_i(x; \mu_j, \Sigma_j)$$

where $\mu_i \in R$ is the mean of distribution i, $\Sigma_i \in R^+$ is the variance of distribution i, and $\pi_i \in (0, 1)$ is the probability that an observation comes from the i^{th} distribution, for $i = 1, 2, ...k$ with $\sum_{i=1}^{k} \pi_i = 1$. We assume that each $f_i(x|\mu_i, \Sigma_i) \sim N(\mu_i, \Sigma_i)$, so that this is a mixture of k univariate normal distributions. Note, Σ_i here is a scalar; however, if we instead used multivariate normal distributions, the Σ_i's would be the covariance matricies (and μ_i a vector of means).

We could also include the latent variable Z, which indicates the distribution that an observation came from. That is, $Z_{i,j} = 1$ if observation x_i comes from f_j, and otherwise is equal to zero. Note this also implies the condition $\sum_{j=1}^{k} Z_j = 1$. If we incorporate this variable, the density becomes

$$f(x; \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{Z}) = \sum_{j=1}^{k} \pi_j f_j(x; \mu_j, \Sigma_j) I_{(Z_j=1)}$$

where $I_{(Z_j=1)}$ is an indicator function, and

$$P(Z_j = 1) = \pi_j$$

The goal is to estimate the proportions π_j given some data set. We will use the EM algorithm to fit the parameters of this model. A derivation of the EM algorithm is not given here, but an outline of the algorithm is provided. A secondary goal is to determine how well this model predicts the unknown latent variable. After we fit the model, we can choose a value for $Z_{i,j}$, $i = 1, ..., n$, $j = 1, ..., k$. These can then be compared to the known values of $Z_{i,j}$ in the simulation. The proportion of observations that were correctly classified will be our measure of how well EM performs for this task.

EM Algorithm

Given a data set of n observations, we assume the density of the population is a mixture of univariate normals $f(x; \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{Z})$. The EM algorithm gives estimates for the parameters $\boldsymbol{\pi}$, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$.

1. Initialize the parameters $\pi_i^{(0)}$, $\mu_i^{(0)}$, and $\Sigma_i^{(0)}$. Set $r = 1$.

2. (E step): Set

$$z_{i,j}^{(r)} = \frac{\pi_j^{(r-1)} N(x_i; \mu_i^{(r-1)}, \Sigma_i(r-1))}{\sum_{j=1}^{k} \pi_j^{(r-1)} N(x_i; \mu_j^{(r-1)}, \Sigma_j^{(r-1)})}$$

   and $n_j = \sum_{i=1}^{n} z_{i,j}^{(r)}$.
3. (M step) Estimate the parameters by

$$\mu_j^{(r)} = \frac{1}{n_j} \sum_{i=1}^{n} z_{i,j}^{(r)} x_i$$
$$\Sigma_j^{(r)} = \frac{1}{n_j} \sum_{i=1}^{n} z_{i,j}^{(r)} (x_i - \mu_i^r)^2$$
$$\pi_j^{(r)} = \frac{n_j}{n}$$

4. Evaluate the log-likelihood

$$log\_lik = \sum_{i=1}^{n} ln \left(\sum_{j=1}^{k} \pi_j^{(r)} N(x_i; \mu_j^{(r)}, \Sigma_j(r)) \right)$$

5. Check for convergence of the parameters or the log-likelihood.
6. Repeat 2 to 5 until convergence.

We will also classify the i^{th} observation using $\hat{Z}_i$, where $\hat{Z}_{i,j} = 1$ for $j = argmax_j(z_{i,j}))$ and is 0 otherwise. Here we're using the values of $z_{i,j}$ from the last iteration of the EM algorithm. If there is a tie, we will default to the smallest j. The percentage of correct classifications C_p is computed by $C_p = \frac{1}{n} \sum_{i=1}^{n} I(Z_i = \hat{Z}_i)$.

In this project, the default initialization of the parameters will be $\pi_i^{(0)} = \frac{1}{k}$ and $\Sigma_i^{(0)} = 1$ for $i = 1, ...k$. For $\mu_i^{(0)}$, when $k = 2$, we will use $\mu_1^{(0)} = min(x)$ and $\mu_2^{(0)} = max(x)$.

Simulation

We consider five different scenarios and determine how the EM algorithm performs in each. In each case $m = 1000$ samples of size $n = 100$ are drawn from a fixed distribution. The EM estimates of the parameters are computed, along with C_p, the percentage of correct classifications. The distribution of these estimates will be shown by a histogram, and a visual inspection will be used to judge the performance. In addition, we will calculuate a 95% confidence interval for the expected value of π_j using $\bar{\hat{\pi}}_j \pm 1.96SE(\hat{\pi}_j)$, and a 95% confidence interval for the proportion of correct classifications using $\bar{C}_p \pm 1.96SE(C_p)$.

Case 1: different means with small variances.

The first distribution we consider is

$$f(x; \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = 0.4N(0, 1) + 0.6N(3, 1)$$

This represents a basic case with nothing extreme going on, just a mixture of two normals with noticably different means and relatively small variances. The following two graphs show one sample from this distribution; one histogram has the true classifications colored, while the histogram other is colored based on the classification given from the EM.

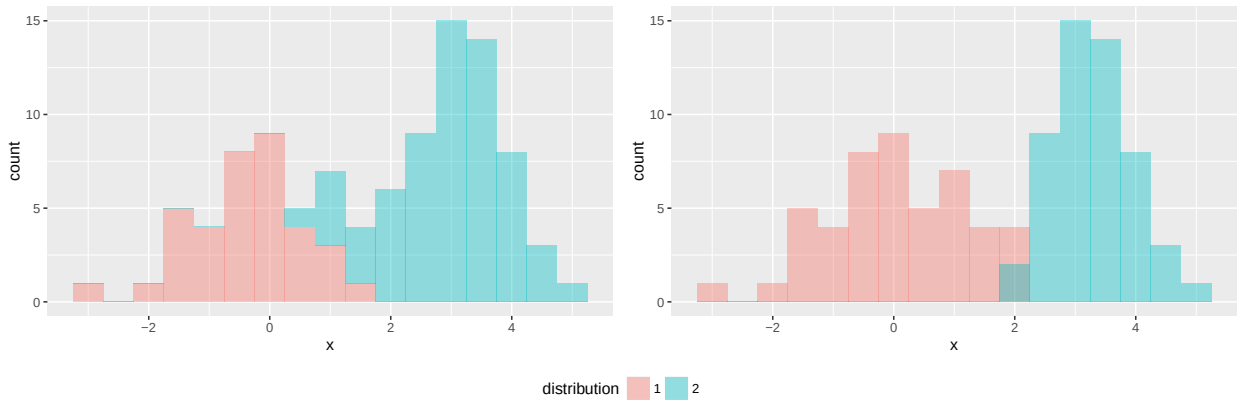

Figure 1: One sample of size n = 100 from the distribution f(x) = 0.4N(0, 1) + 0.6N(3, 1). The histogram on the left is colored according to the true classification on the observations, and the histogram on the right shows how the EM algorithm classifies the observations.

After simulating the EM algorithm $m = 1000$ times, we obtain the following distribution of estimates.

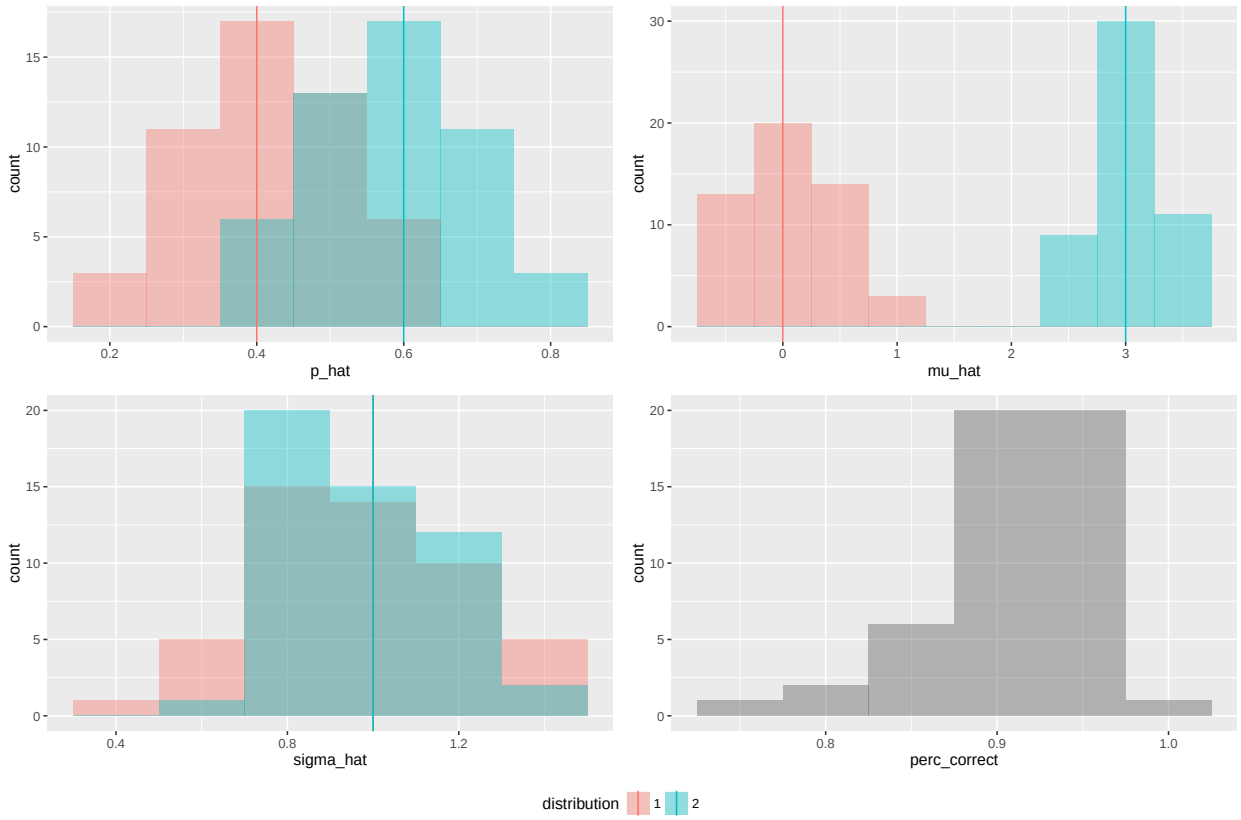

Figure 2: The distribution of EM parameter estimates and percentage of correct classifications from m = 1000 simulations and sample size n = 50 for the population in case 1: f(x) = 0.4N(0, 1) + 0.6N(3, 1). The vertical lines in the first three histograms mark the true parameter values.

The 95% confidence interval for $E(\hat{\pi}_1)$ is (0.2088, 0.6141), for $E(\hat{\pi}_2)$ is (0.3859, 0.7912), and for the average percentage of correct classifications is (0.816, 1.000).

Case 2: different means with large variances.

Consider a similar scenario as case 1, except now the normals have much larger variances.

$$f(x; \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = 0.4N(0, 16) + 0.6N(3, 16)$$

In this case, since the variances are large relative to the difference in means, there will be signifiance overlap in the distributions. Again, we first show one sample from this distribution. It's easy to imagine that the EM algorithm will have a difficult time finding these distinct distributions.

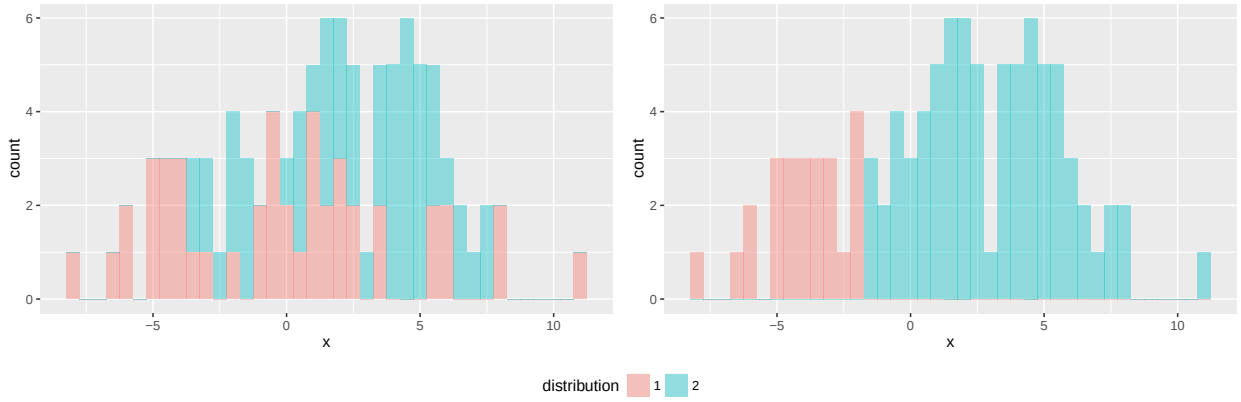

Figure 3: One sample of size n = 100 from the distribution f(x) = 0.4N(0, 16) + 0.6N(3, 16). The histogram on the left is colored according to the true classification on the observations, and the histogram on the right shows how the EM algorithm classifies the observations.

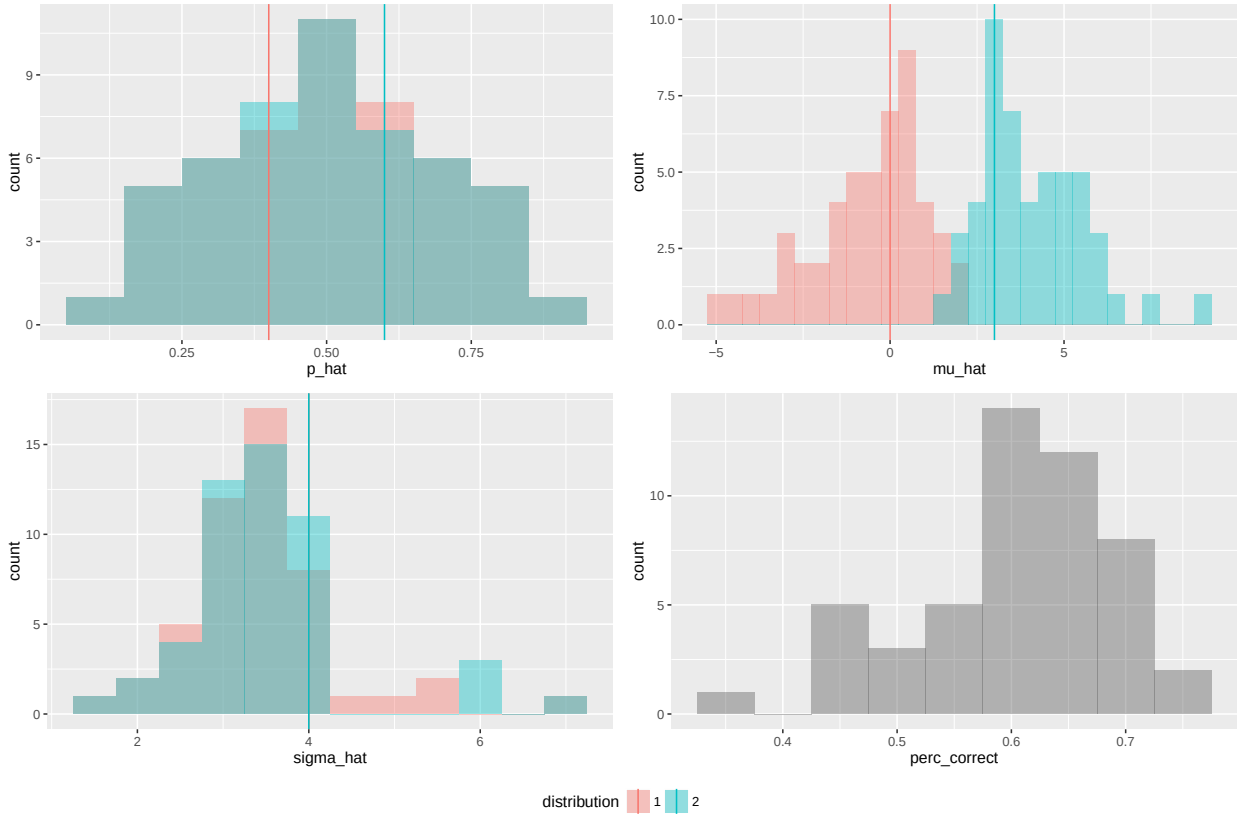

Figure 4: The distribution of EM parameter estimates and percentage of correct classifications from m = 1000 simulations and sample size n = 100 for the population in case 2: f(x) = 0.4N(0, 16) + 0.6N(3, 16). The vertical lines mark the true parameter values.

The 95% confidence interval for $E(\hat{\pi}_1)$ is (0.1293, 0.8722), for $E(\hat{\pi}_2)$ is (0.1278, 0.8707), and for the average percentage of correct classifications is (0.4378, 0.7642). The EM correctly classifies the observations about 50% of the time, on average. The algorithm seems to converge to two normals that have the appropriate means, but will chose one of the distributions to have a much larger variance than the other. Because of this, it tends to over-estimate one proportion and under-estimate the other.

Case 3: same means with different variances.

The distribution we consider is

$$f(x; \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = 0.4N(0, 1) + 0.6N(0, 9)$$

This is a sort of extreme example of case 2, with one distribution now completely dominating the other. The two distributions have the same mean, with one having a smaller variance than the other.

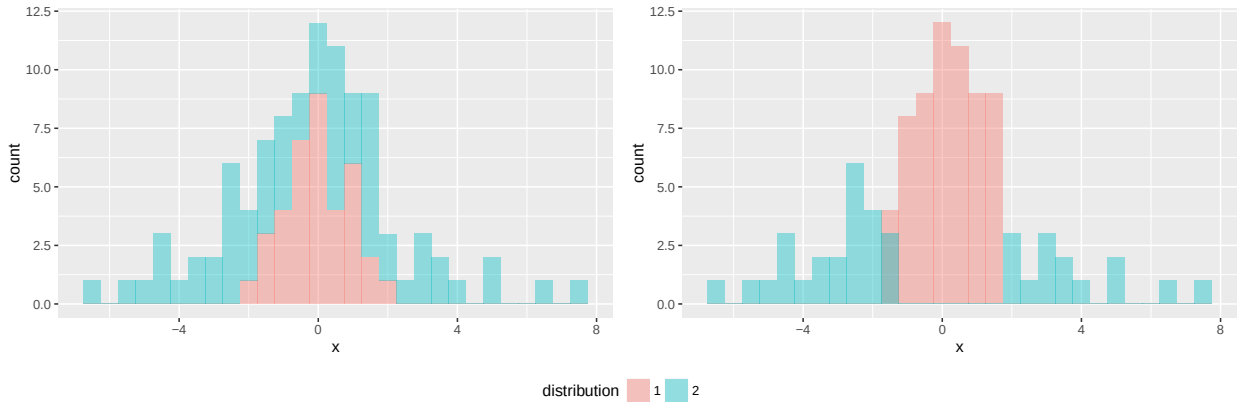

Figure 5: One sample of size n = 100 from the distribution f(x) = 0.4N(0, 1) + 0.6N(0, 9). The histogram on the left is colored according to the true classification on the observations, and the histogram on the right shows how the EM algorithm classifies the observations.

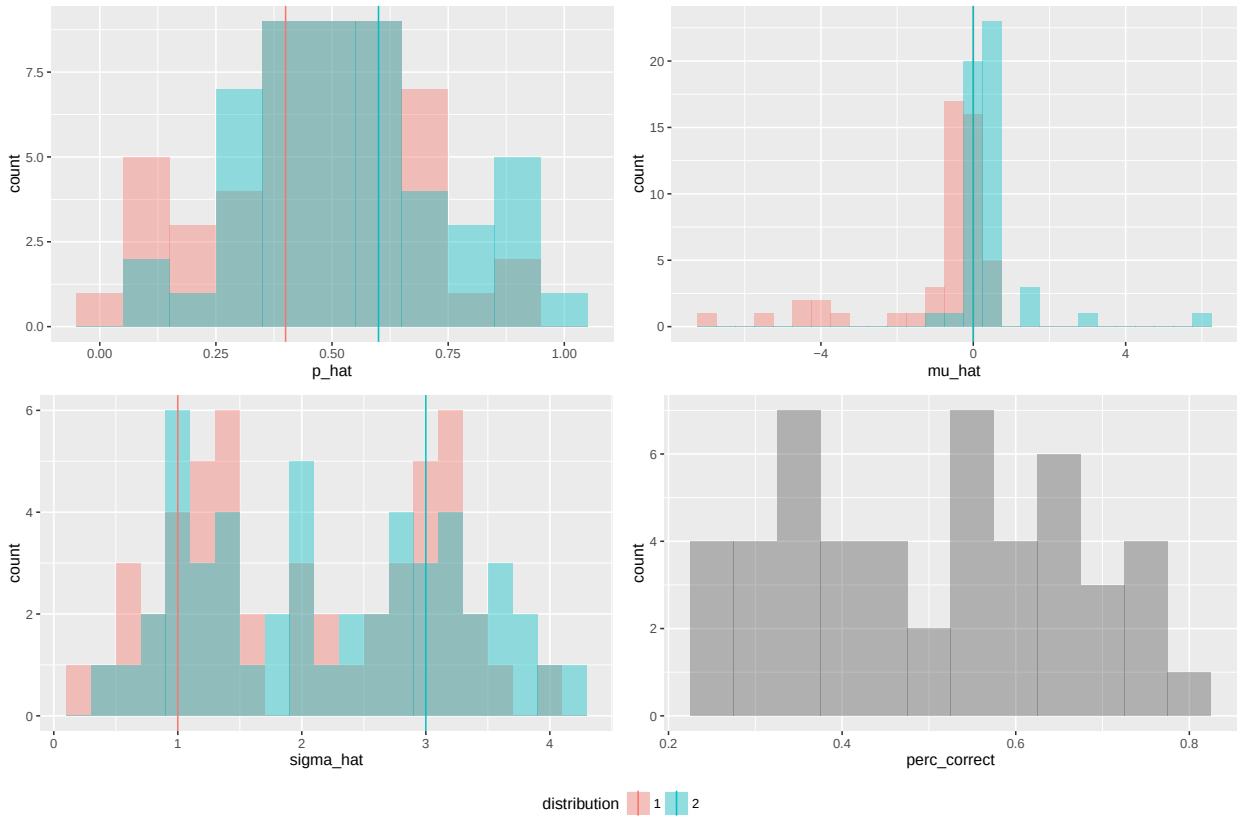

Figure 6: The distribution of EM parameter estimates and percentage of correct classifications from m = 1000 simulations and sample size n = 100 for the population in case 3: f(x) = 0.4N(0, 1) + 0.6N(0, 9). The vertical lines mark the true parameter values.

The 95% confidence interval for $E(\hat{\pi}_1)$ is (0.04128, 0.89524), for $E(\hat{\pi}_2)$ is (0.1048, 0.9587), and for the average percentage of correct classifications is (0.1851, 0.8153).

Surprisingly, the EM algorithm works quite well. Since we already know there are two normals to be found, it correctly centers both at the same mean, and gives one distribution a larger variance than the other.

Case 4: small sub-population

The distribution we consider is

$$f(x; \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = 0.1N(0, 1) + 0.9N(3, 1)$$

In some distrubtions, we might have a relatively small sub-population. In this example, we look at how the EM performs when the smaller population is only 10% of the mixture. We set the means and variances to moderate values.

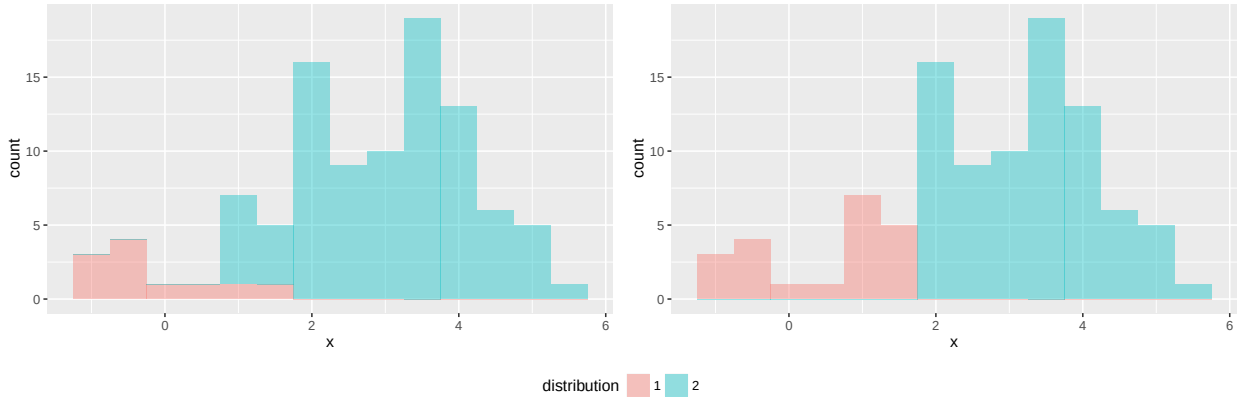

Figure 7: One sample of size n = 100 from the distribution f(x) = 0.1N(0, 1) + 0.9N(3, 1). The histogram on the left is colored according to the true classification on the observations, and the histogram on the right shows how the EM algorithm classifies the observations.

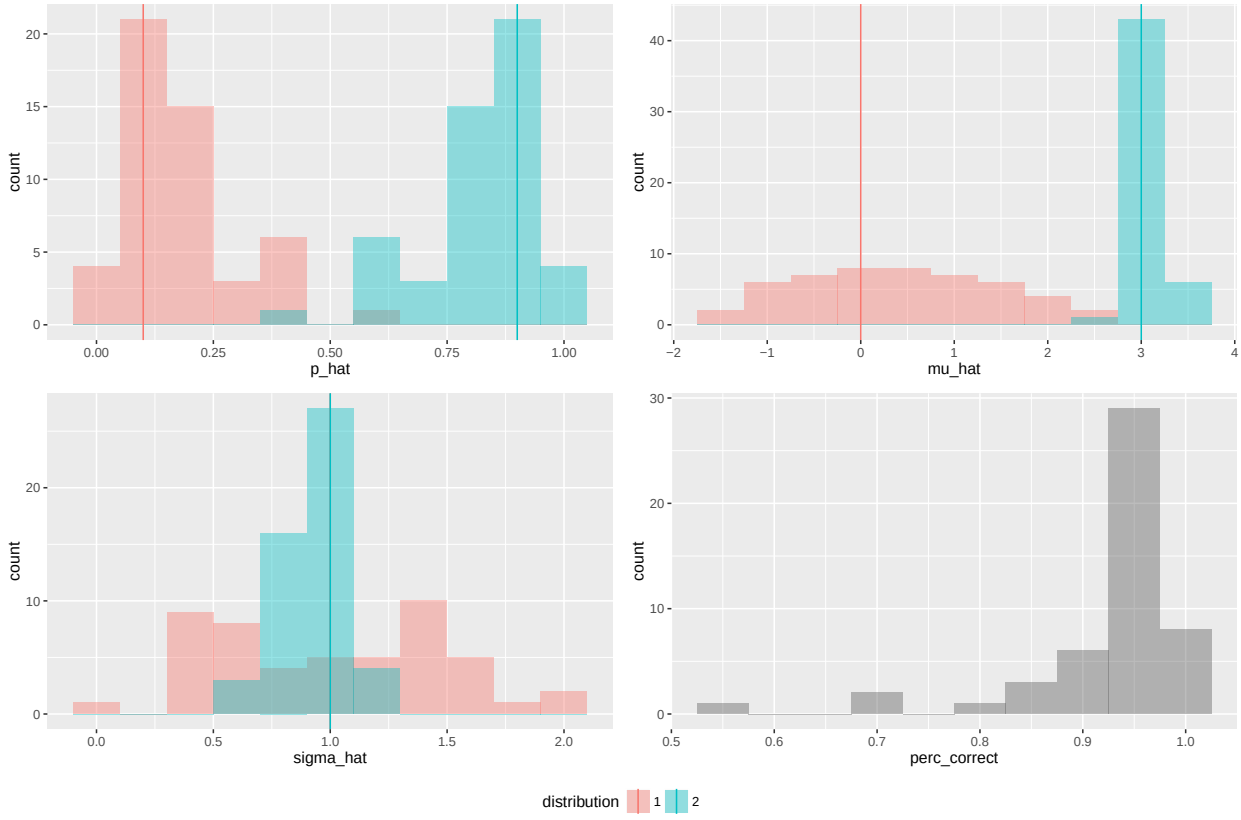

Figure 8: The distribution of EM parameter estimates and percentage of correct classifications from m = 1000 simulations and sample size n = 100 for the population in case 4: f(x) = 0.1N(0, 1) + 0.9N(3, 1). The vertical lines mark the true parameter values.

The 95% confidence interval for $E(\hat{\pi}_1)$ is (0.0000, 0.4322), for $E(\hat{\pi}_2)$ is (0.5678, 1.0000), and for the average percentage of correct classifications is (0.7735, 1.0000).

The EM algorithm handles this scenario well; it tends to get the classificatins correct, but will often overestimate the proportion of the smaller sub-population.

Case 5: k = 4 sub-populations

Finally, we will consider a mixture of $n = 4$ univariate normals

$$f(x; \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = 0.25N(0, 1) + 0.25N(3, 1) + 0.25N(6, 1) + 0.25N(9, 1)$$

and here we will draw samples of size $n = 100$. This scenario is most similar to case 1, whereby there are no extreme paramater values. Instead we just have four equiprobable normals, each with noticably different means, and relatively small variances.

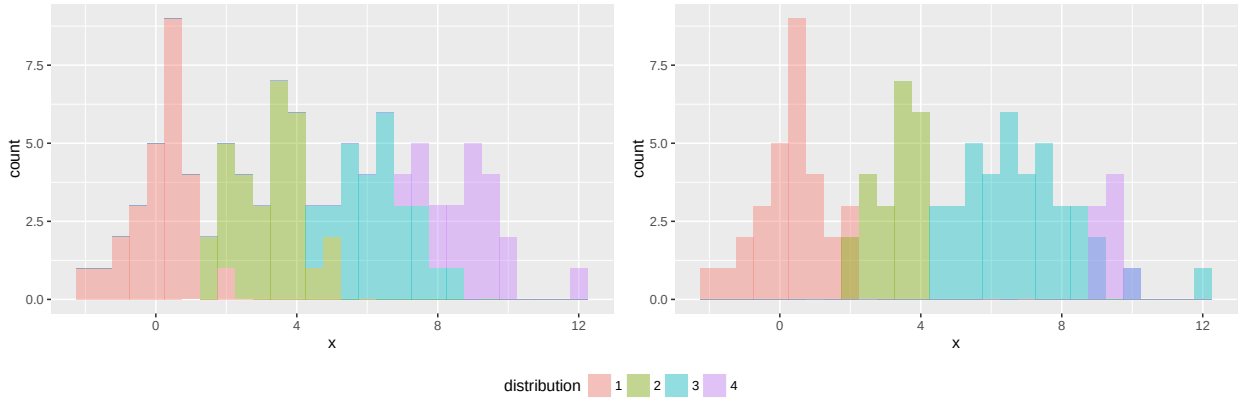

Figure 9: One sample of size n = 100 from the distribution f(x) = 0.25N(0, 1) + 0.25N(3, 1) + 0.25N(6, 1) + 0.25N(9, 1). The histogram on the left is colored according to the true classification on the observations, and the histogram on the right shows how the EM algorithm classifies the observations.

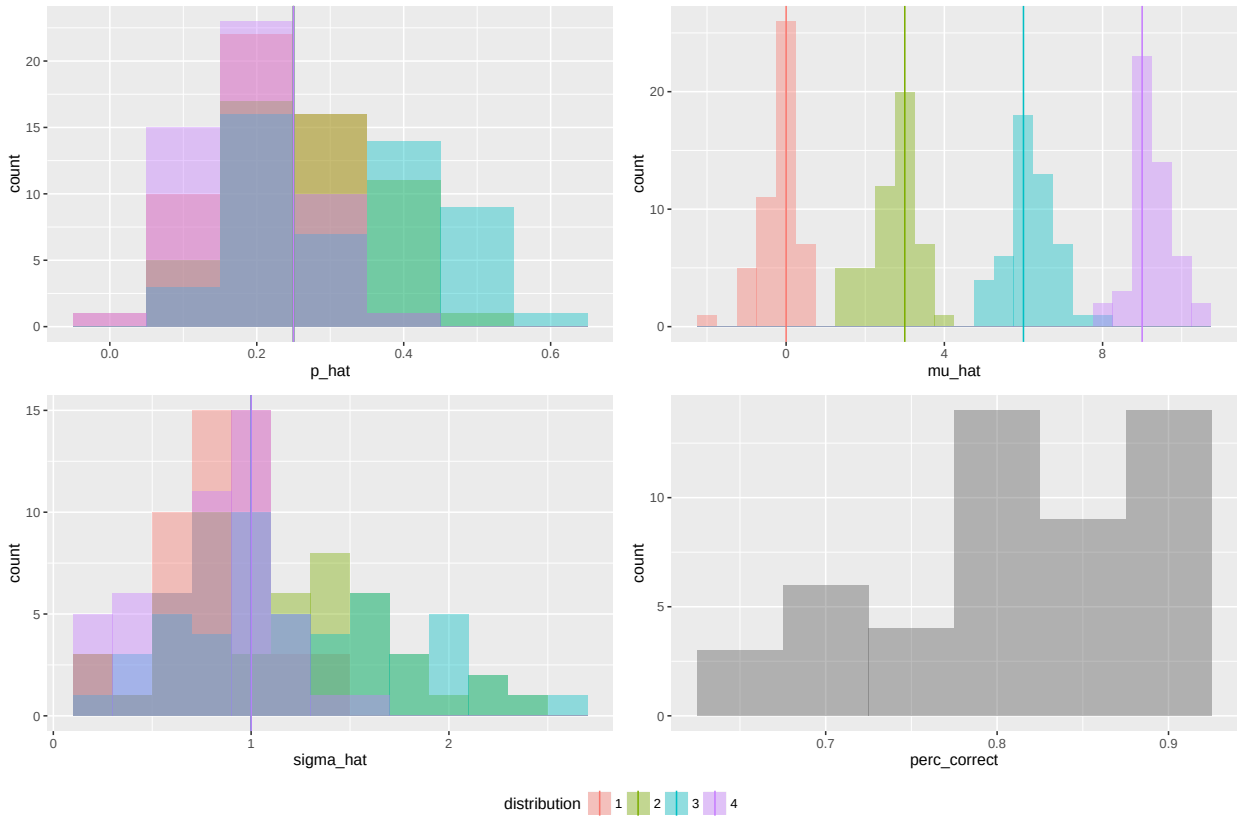

Figure 10: The distribution of EM parameter estimates and percentage of correct classifications from m = 1000 simulations and sample size n = 100 for the population in case 5: f(x) = 0.25N(0, 1) + 0.25N(3, 1) + 0.25N(6, 1) + 0.25N(9, 1). The vertical lines mark the true parameter values.

The 95% confidence interval for $E(\hat{\pi}_1)$ is (0.08125, 0.34774), for $E(\hat{\pi}_2)$ is (0.09565, 0.45380), $E(\hat{\pi}_3)$ is (0.08657, 0.56405), $E(\hat{\pi}_4)$ is (0.03864, 0.33230), and for the average percentage of correct classifications is (0.6580, 0.9676).

This example demonstrates that, if we know how many distributions to look for, EM does a good job finding them.

Discussion

There are many different ways to assess the performance of the EM algorithm. We can vary the sample sizes, the amount of difference in the means, the magnitude of the variation in each distribution, the size of each proportion, and even the the number of mixtures to use. With so many possible combinations, it is difficult to give an exhaustive demonstration that the EM algorithm works well for all mixture distributions.

If we had a particular data set at hand, we should set up a simulation around the characteristics of the data to get a better idea of how EM will perform. For example, if we notice that there are three modes in the data, and perhaps two are close together while the third is further away, we could create scenarios to simulate that replicate that particular situation.

However, from the five cases considered here, it appears that the EM algorithm will work quite well for a variety of different cases, so long as we know how many distributions to look for.

Code

```
library("ggplot2")
library("gridExtra")

#Simulation parameters.
k <- 2      #Number of mixtures to apply in EM algorithm
n <- 50     #Sample size
MAX_ITER <- 100     #Maximum number of iterations for EM.
EPSILON <- 10^(-10)

#Graph parameters.
binwidth <- c(1/10, 1/2, 1/5)

#Define parameters for mixture model.
mu <- c(0, 3)
sigma <- c(1, 1)
p <- c(0.4, 0.6)

#rmix returns n observations from the mixture.
rmix <- function(n = 1) {
  data <- data.frame(x = vector("numeric", n),
                     dist = vector("numeric", n))
  for(i in 1:n) {
    u <- runif(1)
    for(j in 1:length(p)) {
      if(u <= p[j]) {
        data$x[i] <- rnorm(1, mu[j], sigma[j])
        data$dist[i] <- j
        break
      } else {
        u <- u - p[j]
      }
    }
  }
  return(data)
}

mean_mix <- function() {
  sum(p*mu)
}
```

```
var_mix <- function() {
  sum(p*(mu^2 + sigma^2)) - mean_mix()^2
}

EM_mix <- function(x, k = 2, mu_hat_0 = NULL,
                   sigma_hat_0 = NULL, p_hat_0 = NULL) {
  #Store estimates from each iteration.
  mu_hat <- matrix(0, nrow = MAX_ITER + 1, ncol = k)
  sigma_hat <- matrix(0, nrow = MAX_ITER + 1, ncol = k)
  p_hat <- matrix(0, nrow = MAX_ITER + 1, ncol = k)
  log_likelihood <- matrix(0, nrow = MAX_ITER + 1, ncol = 1)

  iter <- 1
  #Use first "iteration"" to initialize paramters.
  if(is.null(mu_hat_0)) {
    mu_hat[iter, ] <- quantile(x, seq(0, 1, by = 1/(k - 1)))
  } else {
    mu_hat[iter, ] <- mu_hat_0
  }
  if(is.null(sigma_hat_0)) {
    sigma_hat[iter, ] <- rep(1, k)
  } else {
    sigma_hat[iter, ] <- sigma_hat_0
  }
  if(is.null(p_hat_0)) {
    p_hat[iter, ] <- rep(1/k, k)
  } else {
    p_hat[iter, ] <- p_hat_0
  }
  log_likelihood[iter, ] <- -Inf
  z <- NULL

  iter <- 2
  #In each iteration, do the following.
  EM_mix_iterate <- function(data) {
    #E-step
    z <<- t(sapply(1:n, function(i) {
      den <- sum(sapply(1:k, function(j) {
        p_hat[iter - 1, j]*dnorm(x[i], mu_hat[iter - 1, j],
                                 sigma_hat[iter - 1, j])
      }))
      sapply(1:k, function(j) {
        p_hat[iter - 1, j] * dnorm(x[i], mu_hat[iter - 1, j],
                                   sigma_hat[iter - 1, j])
      }) / den
    }))
```

```
    #M-step
    N <- apply(z, 2, sum)
    mu_hat[iter, ] <<- (t(z) %*% x)/N
    sigma_hat[iter, ] <<- sqrt(sapply(1:k, function(j) {
      z[, j] %*% (x - mu_hat[iter, j])^2 / N[j]
    }))
    p_hat[iter, ] <<- N/n

    #Check for convergence.
    log_likelihood[iter, ] <<- sum(sapply(1:n, function(i) {
      log(sum(sapply(1:k, function(j) {
        p_hat[iter, j]*dnorm(x[i], mu_hat[iter, j], sigma_hat[iter, j] )
      })))
    }))

    iter <<- iter + 1
  }

  plots <- list(ggplot(data.frame(x = x), aes(x)) +
                  geom_histogram(binwidth = binwidth[2],
                                 alpha = 0.4, position = "identity"))

  #Start the iterative process.
  while(iter <= MAX_ITER) {
    EM_mix_iterate()
    plots <- c(plots,
               list(ggplot(data.frame(x = x,
                                      z = apply(z, 1, function(x) {
                                        which(x == max(x))}
                                      )),
                           aes(x, fill = factor(z))) +
                      geom_histogram(binwidth = binwidth[2],
                                     alpha = 0.4, position = "identity")))
    if(abs(log_likelihood[iter - 1] - log_likelihood[iter - 2]) < EPSILON) {
      break
    }
  }

  return(list(mu_hat = mu_hat[1:(iter - 1), ],
              sigma_hat = sigma_hat[1:(iter - 1), ],
              p_hat = p_hat[1:(iter - 1), ],
              log_lik = log_likelihood[1:(iter - 1)],
              x = x, z = z, iterations = iter - 1, plots = plots))
}


#Simulation to obtain distribution of EM estimates.
```

```
m <- 10     #Number of simulations
estimates <- list(p_hat = matrix(0, nrow = m, ncol = k),
                  mu_hat = matrix(0, nrow = m, ncol = k),
                  sigma_hat = matrix(0, nrow = m, ncol = k))
perc_correct <- vector("numeric", m)

#Show the results for one draw from the population.
obs <- rmix(n)
results <- EM_mix(obs$x, k)

grid.arrange(results$plots[[1]], results$plots[[2]], results$plots[[3]],
             results$plots[[4]], results$plots[[5]], results$plots[[6]],
             ncol = 3, nrow = 2)

grid.arrange(ggplot(data.frame(x = obs$x, dist = obs$dist), aes(x, fill = factor(dist))) +
               geom_histogram(binwidth = binwidth[2], alpha = 0.4),
             results$plots[[results$iterations]],
             ncol = 2)

p
results$p_hat[results$iterations, ]
mu
results$mu_hat[results$iterations, ]
sigma
results$sigma_hat[results$iterations, ]

#Perform the simulation.
for(i in 1:m) {
  obs <- rmix(n)
  results <- EM_mix(obs$x, k)
  classification <- apply(results$z, 1, function(x) which(x == max(x)))
  estimates$p_hat[i, ] <- results$p_hat[results$iterations, ]
  estimates$mu_hat[i, ] <- results$mu_hat[results$iterations, ]
  estimates$sigma_hat[i, ] <- results$sigma_hat[results$iterations, ]
  perc_correct[i] <- mean(obs$dist == classification)
}

#Plot the distributions of the estimators.
plots <- list()
plots[[1]] <- ggplot(data.frame(p_hat = c(estimates$p_hat[, 1:k]),
                                dist = rep(1:k, each = m)),
                     aes(p_hat, fill = factor(dist))) +
  geom_histogram(binwidth = binwidth[1], alpha = 0.4, position = "identity") +
  geom_vline(data = data.frame(x = p, dist = 1:k),
             aes(xintercept = x, color = factor(dist)))
plots[[2]] <- ggplot(data.frame(mu_hat = c(estimates$mu_hat[, 1:k]),
                                dist = rep(1:k, each = m)),
```

```
       aes(mu_hat, fill = factor(dist))) +
  geom_histogram(binwidth = binwidth[2], alpha = 0.4, position = "identity") +
  geom_vline(data = data.frame(x = mu, dist = 1:k),
             aes(xintercept = x, color = factor(dist)))
plots[[3]] <- ggplot(data.frame(sigma_hat = c(estimates$sigma_hat[, 1:k]),
                                dist = rep(1:k, each = m)),
       aes(sigma_hat, fill = factor(dist))) +
  geom_histogram(binwidth = binwidth[3], alpha = 0.4, position = "identity") +
  geom_vline(data = data.frame(x = sigma, dist = 1:k),
             aes(xintercept = x, color = factor(dist)))
plots[[4]] <- ggplot(data.frame(perc_correct = perc_correct),
                     aes(perc_correct)) +
  geom_histogram(binwidth = 1/20, alpha = 0.4)

layout <- rbind(c(1, 2, 3, 4), rep(5, 4))
g <- ggplotGrob(plots[[1]] + theme(legend.position="bottom"))$grobs
legend <- g[[which(sapply(g, function(x) x$name) == "guide-box")]]
lheight <- sum(legend$height)
gl <- lapply(plots, function(x) x + theme(legend.position="none"))

grid.arrange(grobs = c(gl, list(legend)), layout_matrix = layout,
             heights = grid::unit.c(rep((unit(1, "npc") - lheight)*(1/(nrow(layout)-1)),nrow(la
```