Laboration report in Machine Learning

Computer lab 2 block 1

732A99

Simge Cinar
Duc Tran
William Wiik

Division of Statistics and Machine Learning
Department of Computer Science
Linköping University

04 december 2023

Contents

1 Assignment 1. ENSEMBLE METHODS **1**

2 Assignment 2. MIXTURE MODELS **5**

2.1 EM-algorithm with 3 clusters 7

2.2 EM-algorithm with 2 clusters 11

2.3 EM-algorithm with 4 clusters 13

3 Statement of Contribution **15**

4 Appendix **15**

1 Assignment 1. ENSEMBLE METHODS

Your task is to learn some random forests using the function randomForest from the R package randomForest. The training data is produced by running the following R code:

```
x1 <- runif(100)
x2 <- runif(100)
trdata <- cbind(x1,x2)
y <- as.numeric(x1<x2)
trlabels <- as.factor(y)
```

The task is therefore classifying Y from X_1 and X_2, where Y is binary and X_1 and X_2 contin- uous. You should learn a random forest with 1, 10 and 100 trees, which you can do by setting the argument ntree to the appropriate value. Use nodesize = 25 and keep.forest = TRUE. The latter saves the random forest learned. You need it because you should also com- pute the misclassification error in the following test dataset (use the function predict for this purpose):

```
set.seed(1234)
x1 <- runif(1000)
x2 <- runif(1000)
tedata <- cbind(x1,x2)
y <- as.numeric(x1<x2)
telabels <- as.factor(y)
#plot(x1 ,x2 , col = (y+1))
```

- Repeat the procedure above for 1000 training datasets of size 100 and report the mean and variance of the misclassification errors. In other words, create 1000 training datasets of size 100, learn a random forest from each dataset, and compute the mis-classification error in the same test dataset of size 1000. Report results for when the random forest has 1, 10 and 100 trees.

```
# Function to create 1000 training datasets
sim_random_forest <- function(tedata, telabels, assignment, nodesize = 25) {

  misclass_ntree_1 <- c()
  misclass_ntree_10 <- c()
  misclass_ntree_100 <- c()

  for(i in 1:1000) {

      x1 <- runif(100)
      x2 <- runif(100)

      # Assign y depending on assignment
      if(assignment == 1) {

        trdata <- cbind(x1, x2)
        y <- as.numeric(x1 < x2)
        trlabels <- as.factor(y)
```

```
    } else if(assignment == 2) {

      trdata <- cbind(x1, x2)
      y <- as.numeric(x1 < 0.5)
      trlabels <- as.factor(y)

    } else if(assignment == 3) {

      trdata <- cbind(x1, x2)
      y <- as.numeric((x1 < 0.5 & x2 < 0.5) | (x1 > 0.5 & x2 > 0.5))
      trlabels <- as.factor(y)

    }

    # Train data to data.frame
    train <- data.frame(y = trlabels, x1, x2)

    # ntree = 1 ----------------------------------------------------------------
    r1 <-  randomForest(y ~ ., data=train, ntree=1,  nodesize=nodesize, keep.forest=TRUE)
    pred_r1 <-  predict(r1, tedata)
    confusion_r1 <- table(telabels, pred_r1)
    misclass_ntree_1[i] <-  (confusion_r1[1,2] + confusion_r1[2,1]) / sum(confusion_r1)


    # ntree = 10 ----------------------------------------------------------------
    r2 <-  randomForest(y ~ ., data=train, ntree=10,  nodesize=nodesize, keep.forest=TRUE)
    pred_r2 <-  predict(r2, tedata)
    confusion_r2 <- table(telabels, pred_r2)
    misclass_ntree_10[i] <-  (confusion_r2[1,2] + confusion_r2[2,1]) / sum(confusion_r2)


    # ntree = 100 ----------------------------------------------------------------
    r3 <-  randomForest(y ~ ., data=train, ntree=100,  nodesize=nodesize, keep.forest=TRUE)
    pred_r3 <-  predict(r3, tedata)
    confusion_r3 <- table(telabels, pred_r3)
    misclass_ntree_100[i] <-  (confusion_r3[1,2] + confusion_r3[2,1]) / sum(confusion_r3)

  }

  return(list(misclass_ntree_1, misclass_ntree_10, misclass_ntree_100))

}


set.seed(1234)
x1 <- runif(1000)
x2 <- runif(1000)
tedata <- cbind(x1, x2)
```

```
y <- as.numeric(x1 < x2)
telabels <- as.factor(y)

result1 <- sim_random_forest(tedata, telabels, assignment = 1)

r1 <- data.frame(mean = c(mean(result1[[1]]), mean(result1[[2]]), mean(result1[[3]])),
                 variance = c(var(result1[[1]]), var(result1[[2]]), var(result1[[3]])))

rownames(r1) <- c("ntree = 1", "ntree = 10", "ntree = 100")

kable(r1, digits = 4, caption = "Mean and variance for first condition")
```

Table 1: Mean and variance for first condition

	mean	variance
ntree = 1	0.2099	0.0034
ntree = 10	0.1350	0.0010
ntree = 100	0.1112	0.0009

- Repeat the exercise above but this time use the condition $(x1 < 0.5)$ instead of $(x1 < x2)$ when producing the training and test datasets.

```
set.seed(1234)
x1 <- runif(1000)
x2 <- runif(1000)
tedata <- cbind(x1, x2)
y <- as.numeric(x1 < 0.5)
telabels <- as.factor(y)
#plot(x1, x2, col = (y+1))

result2 <- sim_random_forest(tedata, telabels, assignment = 2)


r2 <- data.frame(mean = c(mean(result2[[1]]), mean(result2[[2]]), mean(result2[[3]])),
                 variance = c(var(result2[[1]]), var(result2[[2]]), var(result2[[3]])))

rownames(r2) <- c("ntree = 1", "ntree = 10", "ntree = 100")

kable(r2, digits = 4, caption = "Mean and variance for second condition")
```

Table 2: Mean and variance for second condition

	mean	variance
ntree = 1	0.0930	0.0173
ntree = 10	0.0138	0.0005

	mean	variance
ntree = 100	0.0064	0.0001

- Repeat the exercise above but this time use the condition $((x1 < 0.5 \ \& \ x2 < 0.5)|(x1 > 0.5 \ \& \ x2 > 0.5))$ instead of $(x1 < x2)$ when producing the training and test datasets. Unlike above, use nodesize = 12 for this exercise.

```
set.seed(1234)
x1 <- runif(1000)
x2 <- runif(1000)
tedata <- cbind(x1, x2)
y <- as.numeric((x1 < 0.5 & x2 < 0.5) | (x1 > 0.5 & x2 > 0.5))
telabels <- as.factor(y)
#plot(x1, x2, col = (y+1))

result3 <- sim_random_forest(tedata, telabels, assignment = 3, nodesize = 12)

r3 <- data.frame(mean = c(mean(result3[[1]]), mean(result3[[2]]), mean(result3[[3]])),
                 variance = c(var(result3[[1]]), var(result3[[2]]), var(result3[[3]])))

rownames(r3) <- c("ntree = 1", "ntree = 10", "ntree = 100")

kable(r3, digits = 4, caption = "Mean and variance for third condition")
```

Table 3: Mean and variance for third condition

	mean	variance
ntree = 1	0.2512	0.0133
ntree = 10	0.1228	0.0032
ntree = 100	0.0770	0.0014

- What happens with the mean error rate when the number of trees in the random forest grows? Why?
- The third dataset represents a slightly more complicated classification problem than the first one. Still, you should get better performance for it when using sufficient trees in the random forest. Explain why you get better performance.

Answer

2 Assignment 2. MIXTURE MODELS

Your task is to implement the EM algorithm for Bernoulli mixture model. Please use the R template below to solve the assignment. Then, use your implementation to show what happens when your mixture model has too few and too many clusters, i.e. set $M = 2, 3, 4$ and compare results. Please provide a short explanation as well. A Bernoulli mixture model is

$$p(\mathbf{x}) = \sum_{m=1}^{M} \pi_m Bern(\mathbf{x}|\mu_m)$$

where $\mathbf{x} = (x_1, ..., x_D)$ is a D-dimensional binary random vector, $\pi_m = p(y = m)$ and

$$Bern(\mathbf{x}|\mu_m) = \prod_{d=1}^{D} \mu_{m,d}^{x_d}(1 - \mu_{m,d})^{(1-x_d)}$$

where $\mu_m = (\mu_{m,1}, ..., \mu_{m,D})$ is a D-dimensional vector of probabilities. As usual, the log likelihood of the dataset $\{\mathbf{x}_i\}_{i=1}^n$ is

$$\sum_{i=1}^{n} \log p(\mathbf{x}_i)$$

Need to check/fix equations for Bernoulli (think pi is correct but check mu and wi).
Finally, in the EM algorithm, the parameter updates for the Bernoulli mixture model are:

$$\hat{\pi}_m = \frac{1}{n} \sum_{1=1}^{n} w_i(m)$$

$$\hat{\mu}_m = \frac{1}{\sum_{1=1}^{n} w_i(m)} \sum_{1=1}^{n} w_i(m)\mathbf{x}_i$$

where $w_i(m) = p(y_i = m|\mathbf{x}_i)$

Training data consists of 1 000 observations in 10 dimensions and are sampled as follows:

```
# Template code
set.seed(1234567890)
max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log lik between two consecutive iterations
n=1000 # number of training points
D=10 # number of dimensions
x <- matrix(nrow=n, ncol=D) # training data

true_pi <- vector(length = 3) # true mixing coefficients
true_mu <- matrix(nrow=3, ncol=D) # true conditional distributions

true_pi=c(1/3, 1/3, 1/3)
true_mu[1,]=c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
```

```
true_mu[2,]=c(0.5,0.4,0.6,0.3,0.7,0.2,0.8,0.1,0.9,0)
true_mu[3,]=c(0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5)

# Producing the training data
for(i in 1:n) {
  m <- sample(1:3, 1, prob=true_pi)
  for(d in 1:D) {
    x[i,d] <- rbinom(1,1,true_mu[m,d])
  }
}
```

The true values for the mean, μ for each of the three clusters are presented in figure X.

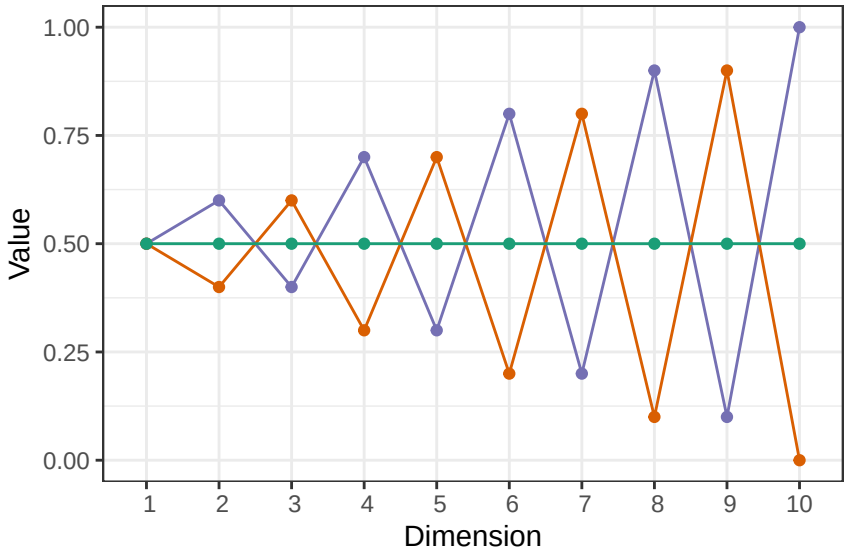

Figure 1: True value of μ.

2.1 EM-algorithm with 3 clusters

The EM-algorithm was implemented first for 3 clusters with the code as follows:

```
# Template code
# Creates empty variables for the EM-algorithm
M=3 # number of clusters
w <- matrix(nrow=n, ncol=M) # weights
pi <- vector(length = M) # mixing coefficients
mu <- matrix(nrow=M, ncol=D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations

# Random initialization of the parameters
pi <- runif(M, 0.49, 0.51)
pi <- pi / sum(pi)

# Random initialization of mu for each cluster.
for(m in 1:M) {
  mu[m,] <- runif(D,0.49,0.51)
}
# End template code

bernoulli <- matrix(nrow=n, ncol=M)

for(it in 1:max_it) {
  # E-step: Computation of the weights
  for (i in 1:1000){
    x_i <- x[i,]
    bernoulli[i,1] <- prod(mu[1,]^x_i * (1-mu[1,])^(1-x_i))
    bernoulli[i,2] <- prod(mu[2,]^x_i * (1-mu[2,])^(1-x_i))
    bernoulli[i,3] <- prod(mu[3,]^x_i * (1-mu[3,])^(1-x_i))

    w[i,1] <- bernoulli[i,1] * pi[1] / sum(bernoulli[i,] * pi)
    w[i,2] <- bernoulli[i,2] * pi[2] / sum(bernoulli[i,] * pi)
    w[i,3] <- bernoulli[i,3] * pi[3] / sum(bernoulli[i,] * pi)
  }

  p_x <- bernoulli[,1]*pi[1] + bernoulli[,2]*pi[2] + bernoulli[,3]*pi[3]
  llik[it] <- sum(log(p_x))
  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")


  # Stop if the log likelihood has not changed significantly
  if (it>1){
    if (abs(llik[it] - llik[it-1]) < min_change){
      # Saves a plot of the last iteration for mu
      plot_data <- data.frame(t(mu))
      p4 <- ggplot(plot_data, aes(x=1:10)) +
        geom_line(aes(y=X3), color="#7570B3") +
```

```
      geom_point(aes(y=X3), color="#7570B3") +
      geom_line(aes(y=X1), color="#D95F02") +
      geom_point(aes(y=X1), color="#D95F02") +
      geom_line(aes(y=X2), color="#1B9E77") +
      geom_point(aes(y=X2), color="#1B9E77") +
      theme_bw() +
      labs(x = "Dimension",
           y = "Value") +
      ylim(0,1) +
      scale_x_discrete(limits=c(1:10))
    # Exits the EM-algorithm
    break
  }
}

# M-step: ML parameter estimation from the data and weights
# Calculate new pi
pi <- 1/1000 * colSums(w)

# Calculates new mu
mu[1,] <- 1/sum(w[,1]) * colSums(w[,1] * x)
mu[2,] <- 1/sum(w[,2]) * colSums(w[,2] * x)
mu[3,] <- 1/sum(w[,3]) * colSums(w[,3] * x)


# Saves a plot of iteration 1 for mu
if(it == 1){
  plot_data <- data.frame(t(mu))
  p1 <- ggplot(plot_data, aes(x=1:10)) +
    geom_line(aes(y=X3), color="#7570B3") +
    geom_point(aes(y=X3), color="#7570B3") +
    geom_line(aes(y=X1), color="#D95F02") +
    geom_point(aes(y=X1), color="#D95F02") +
    geom_line(aes(y=X2), color="#1B9E77") +
    geom_point(aes(y=X2), color="#1B9E77") +
    theme_bw() +
    labs(x = "Dimension",
         y = "Value") +
    ylim(0,1) +
    scale_x_discrete(limits=c(1:10))
}
# Saves a plot of iteration 9 for mu
if(it == 9){
  plot_data <- data.frame(t(mu))
  p2 <- ggplot(plot_data, aes(x=1:10)) +
    geom_line(aes(y=X3), color="#7570B3") +
    geom_point(aes(y=X3), color="#7570B3") +
    geom_line(aes(y=X1), color="#D95F02") +
    geom_point(aes(y=X1), color="#D95F02") +
```

```
      geom_line(aes(y=X2), color="#1B9E77") +
      geom_point(aes(y=X2), color="#1B9E77") +
      theme_bw() +
      labs(x = "Dimension",
           y = "Value") +
      ylim(0,1) +
      scale_x_discrete(limits=c(1:10))
  }
  # Saves a plot of iteration 17 for mu
  if(it == 17){
    plot_data <- data.frame(t(mu))
    p3 <- ggplot(plot_data, aes(x=1:10)) +
      geom_line(aes(y=X3), color="#7570B3") +
      geom_point(aes(y=X3), color="#7570B3") +
      geom_line(aes(y=X1), color="#D95F02") +
      geom_point(aes(y=X1), color="#D95F02") +
      geom_line(aes(y=X2), color="#1B9E77") +
      geom_point(aes(y=X2), color="#1B9E77") +
      theme_bw() +
      labs(x = "Dimension",
           y = "Value") +
      ylim(0,1) +
      scale_x_discrete(limits=c(1:10))
  }
}
```

```
## iteration:  1 log likelihood:  -6931.482
## iteration:  2 log likelihood:  -6929.074
## iteration:  3 log likelihood:  -6928.081
## iteration:  4 log likelihood:  -6920.57
## iteration:  5 log likelihood:  -6868.29
## iteration:  6 log likelihood:  -6646.505
## iteration:  7 log likelihood:  -6403.476
## iteration:  8 log likelihood:  -6357.743
## iteration:  9 log likelihood:  -6351.637
## iteration:  10 log likelihood:  -6349.59
## iteration:  11 log likelihood:  -6348.513
## iteration:  12 log likelihood:  -6347.809
## iteration:  13 log likelihood:  -6347.284
## iteration:  14 log likelihood:  -6346.861
## iteration:  15 log likelihood:  -6346.506
## iteration:  16 log likelihood:  -6346.2
## iteration:  17 log likelihood:  -6345.934
## iteration:  18 log likelihood:  -6345.699
## iteration:  19 log likelihood:  -6345.492
## iteration:  20 log likelihood:  -6345.309
## iteration:  21 log likelihood:  -6345.147
## iteration:  22 log likelihood:  -6345.003
## iteration:  23 log likelihood:  -6344.875
```

```
## iteration:  24 log likelihood:  -6344.762
## iteration:  25 log likelihood:  -6344.66
## iteration:  26 log likelihood:  -6344.57
```

The EM-algorithm converged after 26 iterations. In figure Y, the values of μ for iterations $1, 9, 17, 26$ are presented for each cluster.

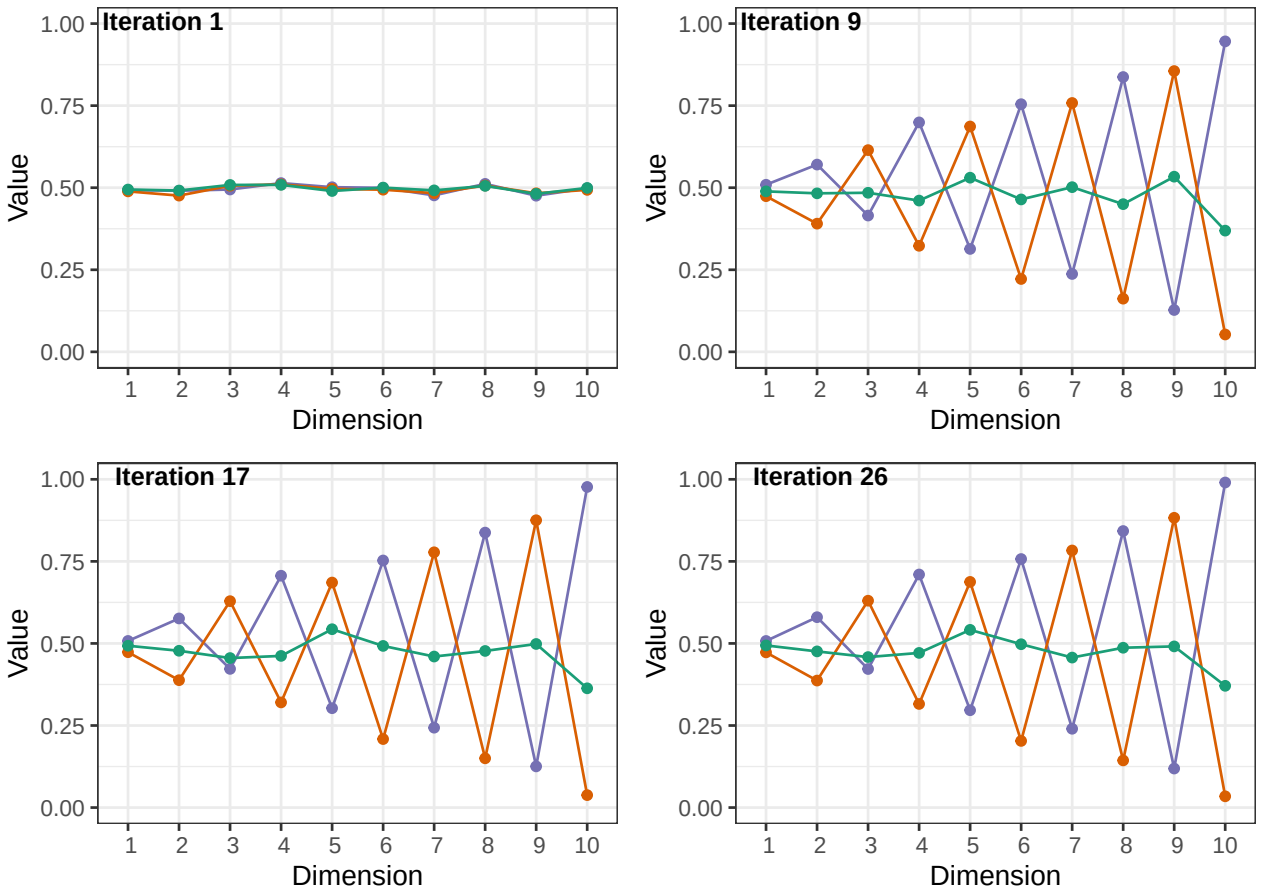

Figure 2: Different values for μ throughout the EM-algoritm with 3 clusters.

From figure Y, the EM-algorithm almost converged after 9 iterations to the three clusters. However, comparing the true values from figure X, the algorithm does not converge exactly to the true values but still close.

2.2 EM-algorithm with 2 clusters

The EM-algorithm for 2 clusters used similar code as 3 clusters, however min change in log likelihood between two consecutive iterations was lowered from 0.1 to 0.05. Without the change, the algorithm converged after 3 iterations with both clusters still having values for μ around 0,5. The values for the log likelihood function are presented as follows:

```
## iteration:  1 log likelihood:  -6930.885
## iteration:  2 log likelihood:  -6929.223
## iteration:  3 log likelihood:  -6929.161
## iteration:  4 log likelihood:  -6928.816
## iteration:  5 log likelihood:  -6926.211
## iteration:  6 log likelihood:  -6906.763
## iteration:  7 log likelihood:  -6790.183
## iteration:  8 log likelihood:  -6499.677
## iteration:  9 log likelihood:  -6373.325
## iteration:  10 log likelihood:  -6364.13
## iteration:  11 log likelihood:  -6363.182
## iteration:  12 log likelihood:  -6362.96
## iteration:  13 log likelihood:  -6362.897
## iteration:  14 log likelihood:  -6362.878
```

The EM-algorithm converged after 14 iterations. In figure Z, the values of μ for iterations $1, 5, 10, 14$ are presented for each cluster.

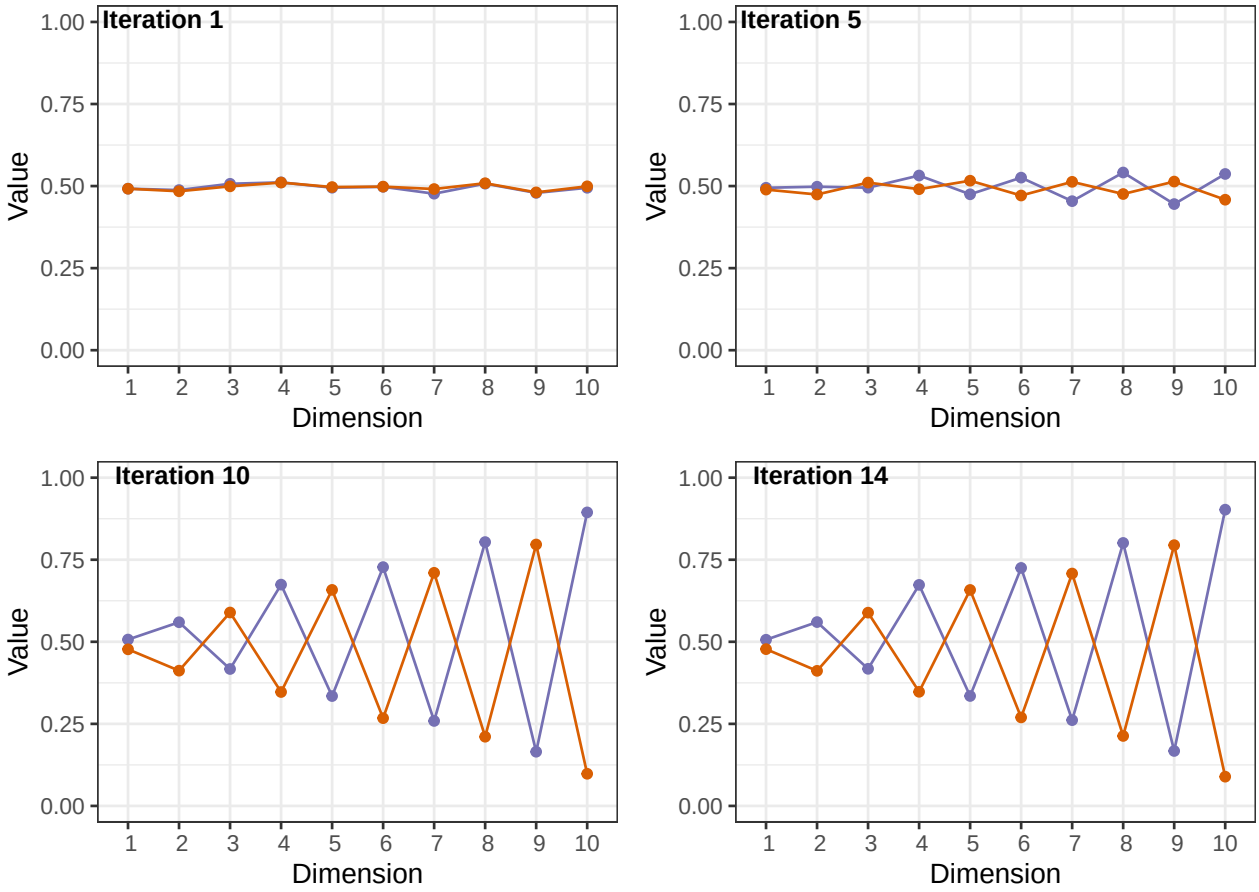

Figure 3: Different values for μ throughout the EM-algoritm with 2 clusters.

From figure Z, the EM-algorithm found the blue and red clusters but the green cluster could not be identified. Examining the probability matrix w a lot of observations had probabilities around 50% for each cluster and these are the observations that are from the green cluster.

2.3 EM-algorithm with 4 clusters

The code for EM-algorithm with 4 clusters had same stopping criterion as the template code and is presented under appendix. The values for the log likelihood function for each iterations of the EM-algorithm are as follows:

```
## iteration:  1 log likelihood:  -6930.838
## iteration:  2 log likelihood:  -6928.641
## iteration:  3 log likelihood:  -6924.748
## iteration:  4 log likelihood:  -6896.25
## iteration:  5 log likelihood:  -6741.896
## iteration:  6 log likelihood:  -6452.658
## iteration:  7 log likelihood:  -6366.493
## iteration:  8 log likelihood:  -6359.764
## iteration:  9 log likelihood:  -6357.876
## iteration:  10 log likelihood:  -6356.372
## iteration:  11 log likelihood:  -6354.86
## iteration:  12 log likelihood:  -6353.31
## iteration:  13 log likelihood:  -6351.776
## iteration:  14 log likelihood:  -6350.33
## iteration:  15 log likelihood:  -6349.03
## iteration:  16 log likelihood:  -6347.908
## iteration:  17 log likelihood:  -6346.968
## iteration:  18 log likelihood:  -6346.196
## iteration:  19 log likelihood:  -6345.566
## iteration:  20 log likelihood:  -6345.055
## iteration:  21 log likelihood:  -6344.637
## iteration:  22 log likelihood:  -6344.293
## iteration:  23 log likelihood:  -6344.008
## iteration:  24 log likelihood:  -6343.768
## iteration:  25 log likelihood:  -6343.563
## iteration:  26 log likelihood:  -6343.387
## iteration:  27 log likelihood:  -6343.233
## iteration:  28 log likelihood:  -6343.097
## iteration:  29 log likelihood:  -6342.975
## iteration:  30 log likelihood:  -6342.864
## iteration:  31 log likelihood:  -6342.762
## iteration:  32 log likelihood:  -6342.668
```

The algorithm converged after 32 iterations and in figure XYZ, the values for μ are presented for iterations $1, 12, 24, 32$.

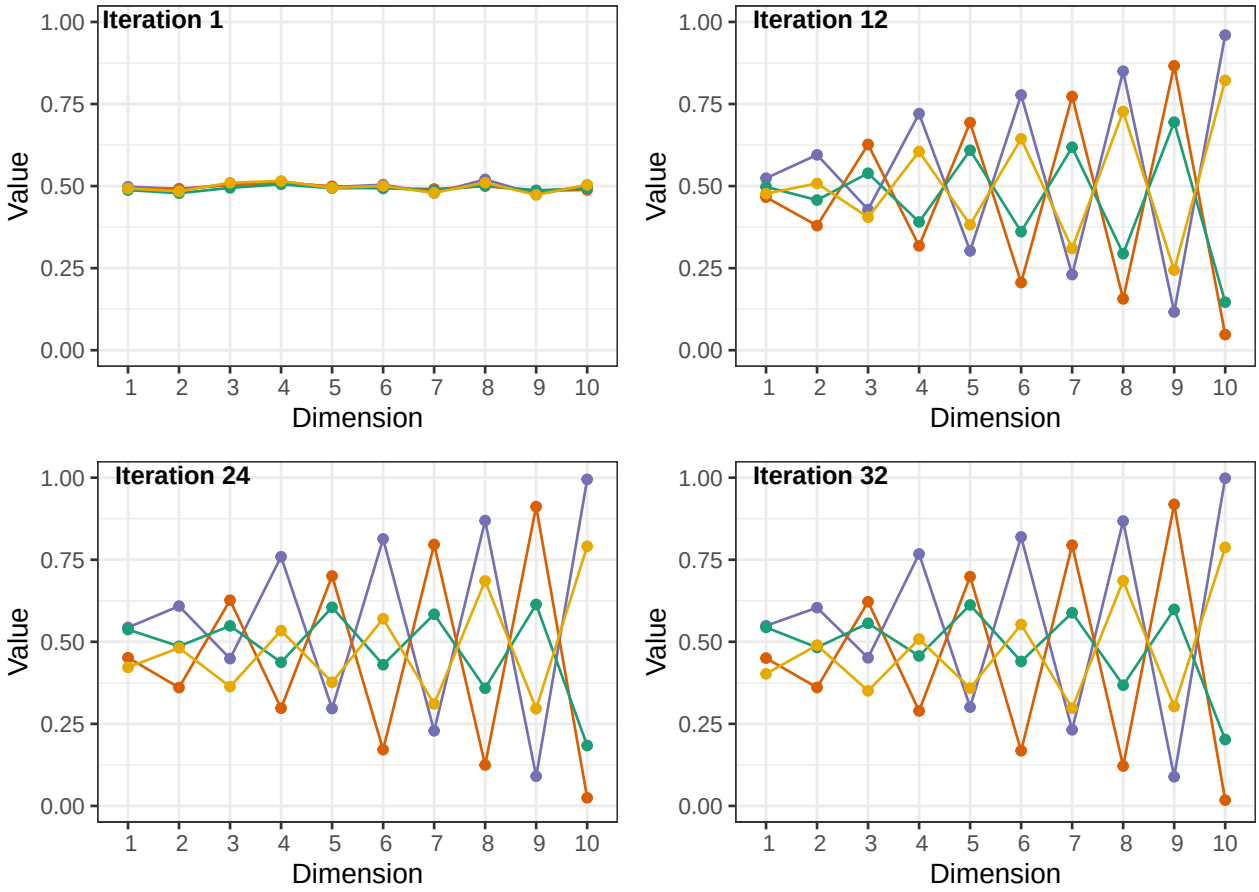

Figure 4: Different values for μ throughout the EM-algoritm with 4 clusters.

In figure XYZ, the mean values of the red and blue clusters are close to the true values in figure X. Except for dimension 3, where the mean value is larger for the blue cluster. The green cluster in figure X was not identified, instead the cluster have been divided into two smaller clusters, green and yellow in figure XYZ.

3 Statement of Contribution

We worked on the assignment individually and later compared and discussed our solutions.

4 Appendix

The code used in this laboration report are summarised in the code as follows:

```
library(ggplot2)
library(tree)
library(knitr)
library(dplyr)
library(ggpubr)
library(randomForest)
knitr::opts_chunk$set(
  echo = TRUE,
  fig.width = 4.5,
  fig.height = 3)

x1 <- runif(100)
x2 <- runif(100)
trdata <- cbind(x1,x2)
y <- as.numeric(x1<x2)
trlabels <- as.factor(y)



set.seed(1234)
x1 <- runif(1000)
x2 <- runif(1000)
tedata <- cbind(x1,x2)
y <- as.numeric(x1<x2)
telabels <- as.factor(y)
#plot(x1 ,x2 , col = (y+1))


# Function to create 1000 training datasets
sim_random_forest <- function(tedata, telabels, assignment, nodesize = 25) {

  misclass_ntree_1 <- c()
  misclass_ntree_10 <- c()
  misclass_ntree_100 <- c()

  for(i in 1:1000) {

      x1 <- runif(100)
      x2 <- runif(100)
```

```
    # Assign y depending on assignment
    if(assignment == 1) {

      trdata <- cbind(x1, x2)
      y <- as.numeric(x1 < x2)
      trlabels <- as.factor(y)

    } else if(assignment == 2) {

      trdata <- cbind(x1, x2)
      y <- as.numeric(x1 < 0.5)
      trlabels <- as.factor(y)

    } else if(assignment == 3) {

      trdata <- cbind(x1, x2)
      y <- as.numeric((x1 < 0.5 & x2 < 0.5) | (x1 > 0.5 & x2 > 0.5))
      trlabels <- as.factor(y)

    }

    # Train data to data.frame
    train <- data.frame(y = trlabels, x1, x2)

    # ntree = 1 ----------------------------------------------------------------
    r1 <-  randomForest(y ~ ., data=train, ntree=1,  nodesize=nodesize, keep.forest=TRUE)
    pred_r1 <-  predict(r1, tedata)
    confusion_r1 <- table(telabels, pred_r1)
    misclass_ntree_1[i] <-  (confusion_r1[1,2] + confusion_r1[2,1]) / sum(confusion_r1)


    # ntree = 10 ---------------------------------------------------------------
    r2 <-  randomForest(y ~ ., data=train, ntree=10,  nodesize=nodesize, keep.forest=TRUE)
    pred_r2 <-  predict(r2, tedata)
    confusion_r2 <- table(telabels, pred_r2)
    misclass_ntree_10[i] <-  (confusion_r2[1,2] + confusion_r2[2,1]) / sum(confusion_r2)


    # ntree = 100 --------------------------------------------------------------
    r3 <-  randomForest(y ~ ., data=train, ntree=100,  nodesize=nodesize, keep.forest=TRUE)
    pred_r3 <-  predict(r3, tedata)
    confusion_r3 <- table(telabels, pred_r3)
    misclass_ntree_100[i] <-  (confusion_r3[1,2] + confusion_r3[2,1]) / sum(confusion_r3)

}

return(list(misclass_ntree_1, misclass_ntree_10, misclass_ntree_100))
```

```
}


set.seed(1234)
x1 <- runif(1000)
x2 <- runif(1000)
tedata <- cbind(x1, x2)
y <- as.numeric(x1 < x2)
telabels <- as.factor(y)

result1 <- sim_random_forest(tedata, telabels, assignment = 1)

r1 <- data.frame(mean = c(mean(result1[[1]]), mean(result1[[2]]), mean(result1[[3]])),
                 variance = c(var(result1[[1]]), var(result1[[2]]), var(result1[[3]])))

rownames(r1) <- c("ntree = 1", "ntree = 10", "ntree = 100")

kable(r1, digits = 4, caption = "Mean and variance for first condition")


set.seed(1234)
x1 <- runif(1000)
x2 <- runif(1000)
tedata <- cbind(x1, x2)
y <- as.numeric(x1 < 0.5)
telabels <- as.factor(y)
#plot(x1, x2, col = (y+1))

result2 <- sim_random_forest(tedata, telabels, assignment = 2)


r2 <- data.frame(mean = c(mean(result2[[1]]), mean(result2[[2]]), mean(result2[[3]])),
                 variance = c(var(result2[[1]]), var(result2[[2]]), var(result2[[3]])))

rownames(r2) <- c("ntree = 1", "ntree = 10", "ntree = 100")

kable(r2, digits = 4, caption = "Mean and variance for second condition")



set.seed(1234)
x1 <- runif(1000)
x2 <- runif(1000)
tedata <- cbind(x1, x2)
y <- as.numeric((x1 < 0.5 & x2 < 0.5) | (x1 > 0.5 & x2 > 0.5))
telabels <- as.factor(y)
#plot(x1, x2, col = (y+1))

result3 <- sim_random_forest(tedata, telabels, assignment = 3, nodesize = 12)
```

```r
r3 <- data.frame(mean = c(mean(result3[[1]]), mean(result3[[2]]), mean(result3[[3]])),
                 variance = c(var(result3[[1]]), var(result3[[2]]), var(result3[[3]])))

rownames(r3) <- c("ntree = 1", "ntree = 10", "ntree = 100")

kable(r3, digits = 4, caption = "Mean and variance for third condition")


# Template code
set.seed(1234567890)
max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log lik between two consecutive iterations
n=1000 # number of training points
D=10 # number of dimensions
x <- matrix(nrow=n, ncol=D) # training data

true_pi <- vector(length = 3) # true mixing coefficients
true_mu <- matrix(nrow=3, ncol=D) # true conditional distributions

true_pi=c(1/3, 1/3, 1/3)
true_mu[1,]=c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
true_mu[2,]=c(0.5,0.4,0.6,0.3,0.7,0.2,0.8,0.1,0.9,0)
true_mu[3,]=c(0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5)

# Producing the training data
for(i in 1:n) {
  m <- sample(1:3, 1, prob=true_pi)
  for(d in 1:D) {
    x[i,d] <- rbinom(1,1,true_mu[m,d])
  }
}
plot_data <- data.frame(t(true_mu))
ggplot(plot_data, aes(x=1:10)) +
  geom_line(aes(y=X1), color="#7570B3") +
  geom_point(aes(y=X1), color="#7570B3") +
  geom_line(aes(y=X2), color="#D95F02") +
  geom_point(aes(y=X2), color="#D95F02") +
  geom_line(aes(y=X3), color="#1B9E77") +
  geom_point(aes(y=X3), color="#1B9E77") +
  theme_bw() +
  labs(x = "Dimension",
       y = "Value") +
  scale_x_discrete(limits=c(1:10))

# Template code
# Creates empty variables for the EM-algorithm
M=3 # number of clusters
w <- matrix(nrow=n, ncol=M) # weights
```

```
pi <- vector(length = M) # mixing coefficients
mu <- matrix(nrow=M, ncol=D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations

# Random initialization of the parameters
pi <- runif(M, 0.49, 0.51)
pi <- pi / sum(pi)

# Random initialization of mu for each cluster.
for(m in 1:M) {
  mu[m,] <- runif(D,0.49,0.51)
}
# End template code

bernoulli <- matrix(nrow=n, ncol=M)

for(it in 1:max_it) {
  # E-step: Computation of the weights
  for (i in 1:1000){
    x_i <- x[i,]
    bernoulli[i,1] <- prod(mu[1,]^x_i * (1-mu[1,])^(1-x_i))
    bernoulli[i,2] <- prod(mu[2,]^x_i * (1-mu[2,])^(1-x_i))
    bernoulli[i,3] <- prod(mu[3,]^x_i * (1-mu[3,])^(1-x_i))

    w[i,1] <- bernoulli[i,1] * pi[1] / sum(bernoulli[i,] * pi)
    w[i,2] <- bernoulli[i,2] * pi[2] / sum(bernoulli[i,] * pi)
    w[i,3] <- bernoulli[i,3] * pi[3] / sum(bernoulli[i,] * pi)
  }

  p_x <- bernoulli[,1]*pi[1] + bernoulli[,2]*pi[2] + bernoulli[,3]*pi[3]
  llik[it] <- sum(log(p_x))
  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")


  # Stop if the log likelihood has not changed significantly
  if (it>1){
    if (abs(llik[it] - llik[it-1]) < min_change){
      # Saves a plot of the last iteration for mu
      plot_data <- data.frame(t(mu))
      p4 <- ggplot(plot_data, aes(x=1:10)) +
        geom_line(aes(y=X3), color="#7570B3") +
        geom_point(aes(y=X3), color="#7570B3") +
        geom_line(aes(y=X1), color="#D95F02") +
        geom_point(aes(y=X1), color="#D95F02") +
        geom_line(aes(y=X2), color="#1B9E77") +
        geom_point(aes(y=X2), color="#1B9E77") +
        theme_bw() +
        labs(x = "Dimension",
             y = "Value") +
```

```
      ylim(0,1) +
      scale_x_discrete(limits=c(1:10))
    # Exits the EM-algorithm
    break
  }
}

# M-step: ML parameter estimation from the data and weights
# Calculate new pi
pi <- 1/1000 * colSums(w)

# Calculates new mu
mu[1,] <- 1/sum(w[,1]) * colSums(w[,1] * x)
mu[2,] <- 1/sum(w[,2]) * colSums(w[,2] * x)
mu[3,] <- 1/sum(w[,3]) * colSums(w[,3] * x)


# Saves a plot of iteration 1 for mu
if(it == 1){
  plot_data <- data.frame(t(mu))
  p1 <- ggplot(plot_data, aes(x=1:10)) +
    geom_line(aes(y=X3), color="#7570B3") +
    geom_point(aes(y=X3), color="#7570B3") +
    geom_line(aes(y=X1), color="#D95F02") +
    geom_point(aes(y=X1), color="#D95F02") +
    geom_line(aes(y=X2), color="#1B9E77") +
    geom_point(aes(y=X2), color="#1B9E77") +
    theme_bw() +
    labs(x = "Dimension",
         y = "Value") +
    ylim(0,1) +
    scale_x_discrete(limits=c(1:10))
}
# Saves a plot of iteration 9 for mu
if(it == 9){
  plot_data <- data.frame(t(mu))
  p2 <- ggplot(plot_data, aes(x=1:10)) +
    geom_line(aes(y=X3), color="#7570B3") +
    geom_point(aes(y=X3), color="#7570B3") +
    geom_line(aes(y=X1), color="#D95F02") +
    geom_point(aes(y=X1), color="#D95F02") +
    geom_line(aes(y=X2), color="#1B9E77") +
    geom_point(aes(y=X2), color="#1B9E77") +
    theme_bw() +
    labs(x = "Dimension",
         y = "Value") +
    ylim(0,1) +
    scale_x_discrete(limits=c(1:10))
}
```

```
  # Saves a plot of iteration 17 for mu
  if(it == 17){
    plot_data <- data.frame(t(mu))
    p3 <- ggplot(plot_data, aes(x=1:10)) +
      geom_line(aes(y=X3), color="#7570B3") +
      geom_point(aes(y=X3), color="#7570B3") +
      geom_line(aes(y=X1), color="#D95F02") +
      geom_point(aes(y=X1), color="#D95F02") +
      geom_line(aes(y=X2), color="#1B9E77") +
      geom_point(aes(y=X2), color="#1B9E77") +
      theme_bw() +
      labs(x = "Dimension",
           y = "Value") +
      ylim(0,1) +
      scale_x_discrete(limits=c(1:10))
  }
}
ggarrange(p1, p2, p3, p4,
          labels=c("Iteration 1","Iteration 9","Iteration 17", "Iteration 26"),
          font.label = list(size = 10),
          hjust = -0.9,
          vjust = 2)
# EM-algorithm for 2 clusters
min_change <- 0.05 # min change in log lik between two consecutive iterations
M=2 # number of clusters
w <- matrix(nrow=n, ncol=M) # weights
pi <- vector(length = M) # mixing coefficients
mu <- matrix(nrow=M, ncol=D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations


# Random initialization of the parameters
pi <- runif(M, 0.49, 0.51)
pi <- pi / sum(pi)

# Random initialization of mu for each cluster.
for(m in 1:M) {
  mu[m,] <- runif(D,0.49,0.51)
}

bernoulli <- matrix(nrow=n, ncol=M)

for(it in 1:max_it) {
  # E-step: Computation of the weights
  for (i in 1:1000){
    x_i <- x[i,]
    bernoulli[i,1] <- prod(mu[1,]^x_i * (1-mu[1,])^(1-x_i))
    bernoulli[i,2] <- prod(mu[2,]^x_i * (1-mu[2,])^(1-x_i))
```

```
  w[i,1] <- bernoulli[i,1] * pi[1] / sum(bernoulli[i,] * pi)
  w[i,2] <- bernoulli[i,2] * pi[2] / sum(bernoulli[i,] * pi)
}

p_x <- bernoulli[,1]*pi[1] + bernoulli[,2]*pi[2]
llik[it] <- sum(log(p_x))
cat("iteration: ", it, "log likelihood: ", llik[it], "\n")


# Stop if the log likelihood has not changed significantly
if (it>1){
  if (abs(llik[it] - llik[it-1]) < min_change){
    # Saves a plot of the last iteration for mu
    plot_data <- data.frame(t(mu))
    p8 <- ggplot(plot_data, aes(x=1:10)) +
      geom_line(aes(y=X2), color="#7570B3") +
      geom_point(aes(y=X2), color="#7570B3") +
      geom_line(aes(y=X1), color="#D95F02") +
      geom_point(aes(y=X1), color="#D95F02") +
      theme_bw() +
      labs(x = "Dimension",
           y = "Value") +
      ylim(0,1) +
      scale_x_discrete(limits=c(1:10))
    # Exits the EM-algorithm
    break
  }
}

# M-step: ML parameter estimation from the data and weights
# Calculate new pi
pi <- 1/1000 * colSums(w)

# Calculates new mu
mu[1,] <- 1/sum(w[,1]) * colSums(w[,1] * x)
mu[2,] <- 1/sum(w[,2]) * colSums(w[,2] * x)


# Saves a plot of iteration 1 for mu
if(it == 1){
  plot_data <- data.frame(t(mu))
  p5 <- ggplot(plot_data, aes(x=1:10)) +
    geom_line(aes(y=X2), color="#7570B3") +
    geom_point(aes(y=X2), color="#7570B3") +
    geom_line(aes(y=X1), color="#D95F02") +
    geom_point(aes(y=X1), color="#D95F02") +
    theme_bw() +
    labs(x = "Dimension",
         y = "Value") +
```

```
      ylim(0,1) +
      scale_x_discrete(limits=c(1:10))
  }
  # Saves a plot of iteration 5 for mu
  if(it == 5){
    plot_data <- data.frame(t(mu))
    p6 <- ggplot(plot_data, aes(x=1:10)) +
      geom_line(aes(y=X2), color="#7570B3") +
      geom_point(aes(y=X2), color="#7570B3") +
      geom_line(aes(y=X1), color="#D95F02") +
      geom_point(aes(y=X1), color="#D95F02") +
      theme_bw() +
      labs(x = "Dimension",
           y = "Value") +
      ylim(0,1) +
      scale_x_discrete(limits=c(1:10))
  }
  # Saves a plot of iteration 9 for mu
  if(it == 10){
    plot_data <- data.frame(t(mu))
    p7 <- ggplot(plot_data, aes(x=1:10)) +
      geom_line(aes(y=X2), color="#7570B3") +
      geom_point(aes(y=X2), color="#7570B3") +
      geom_line(aes(y=X1), color="#D95F02") +
      geom_point(aes(y=X1), color="#D95F02") +
      theme_bw() +
      labs(x = "Dimension",
           y = "Value") +
      ylim(0,1) +
      scale_x_discrete(limits=c(1:10))
  }
}
ggarrange(p5, p6, p7, p8,
          labels=c("Iteration 1","Iteration 5","Iteration 10", "Iteration 14"),
          font.label = list(size = 10),
          hjust = -0.9,
          vjust = 2)
min_change <- 0.1 # min change in log lik between two consecutive iterations
M=4 # number of clusters
w <- matrix(nrow=n, ncol=M) # weights
pi <- vector(length = M) # mixing coefficients
mu <- matrix(nrow=M, ncol=D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations

# Random initialization of the parameters
pi <- runif(M, 0.49, 0.51)
pi <- pi / sum(pi)

# Random initialization of mu for each cluster.
```

```
for(m in 1:M) {
  mu[m,] <- runif(D,0.49,0.51)
}
# End template code

bernoulli <- matrix(nrow=n, ncol=M)

for(it in 1:max_it) {
  # E-step: Computation of the weights
  for (i in 1:1000){
    x_i <- x[i,]
    bernoulli[i,1] <- prod(mu[1,]^x_i * (1-mu[1,])^(1-x_i))
    bernoulli[i,2] <- prod(mu[2,]^x_i * (1-mu[2,])^(1-x_i))
    bernoulli[i,3] <- prod(mu[3,]^x_i * (1-mu[3,])^(1-x_i))
    bernoulli[i,4] <- prod(mu[4,]^x_i * (1-mu[4,])^(1-x_i))

    w[i,1] <- bernoulli[i,1] * pi[1] / sum(bernoulli[i,] * pi)
    w[i,2] <- bernoulli[i,2] * pi[2] / sum(bernoulli[i,] * pi)
    w[i,3] <- bernoulli[i,3] * pi[3] / sum(bernoulli[i,] * pi)
    w[i,4] <- bernoulli[i,4] * pi[4] / sum(bernoulli[i,] * pi)
  }

  p_x <- bernoulli[,1]*pi[1] + bernoulli[,2]*pi[2] + bernoulli[,3]*pi[3] + bernoulli[,4]*pi[4]
  llik[it] <- sum(log(p_x))
  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")


  # Stop if the log likelihood has not changed significantly
  if (it>1){
    if (abs(llik[it] - llik[it-1]) < min_change){
      # Saves a plot of the last iteration for mu
      plot_data <- data.frame(t(mu))
      p12 <- ggplot(plot_data, aes(x=1:10)) +
        geom_line(aes(y=X2), color="#7570B3") +
        geom_point(aes(y=X2), color="#7570B3") +
        geom_line(aes(y=X1), color="#D95F02") +
        geom_point(aes(y=X1), color="#D95F02") +
        geom_line(aes(y=X3), color="#1B9E77") +
        geom_point(aes(y=X3), color="#1B9E77") +
        geom_line(aes(y=X4), color="#E6AB02") +
        geom_point(aes(y=X4), color="#E6AB02") +
        theme_bw() +
        labs(x = "Dimension",
             y = "Value") +
        ylim(0,1) +
        scale_x_discrete(limits=c(1:10))
      # Exits the EM-algorithm
      break
    }
```

```
}

# M-step: ML parameter estimation from the data and weights
# Calculate new pi
pi <- 1/1000 * colSums(w)

# Calculates new mu
mu[1,] <- 1/sum(w[,1]) * colSums(w[,1] * x)
mu[2,] <- 1/sum(w[,2]) * colSums(w[,2] * x)
mu[3,] <- 1/sum(w[,3]) * colSums(w[,3] * x)
mu[4,] <- 1/sum(w[,4]) * colSums(w[,4] * x)

# Saves a plot of iteration 1 for mu
if(it == 1){
  plot_data <- data.frame(t(mu))
  p9 <- ggplot(plot_data, aes(x=1:10)) +
    geom_line(aes(y=X2), color="#7570B3") +
    geom_point(aes(y=X2), color="#7570B3") +
    geom_line(aes(y=X1), color="#D95F02") +
    geom_point(aes(y=X1), color="#D95F02") +
    geom_line(aes(y=X3), color="#1B9E77") +
    geom_point(aes(y=X3), color="#1B9E77") +
    geom_line(aes(y=X4), color="#E6AB02") +
    geom_point(aes(y=X4), color="#E6AB02") +
    theme_bw() +
    labs(x = "Dimension",
         y = "Value") +
    ylim(0,1) +
    scale_x_discrete(limits=c(1:10))
}
# Saves a plot of iteration 9 for mu
if(it == 12){
  plot_data <- data.frame(t(mu))
  p10 <- ggplot(plot_data, aes(x=1:10)) +
    geom_line(aes(y=X2), color="#7570B3") +
    geom_point(aes(y=X2), color="#7570B3") +
    geom_line(aes(y=X1), color="#D95F02") +
    geom_point(aes(y=X1), color="#D95F02") +
    geom_line(aes(y=X3), color="#1B9E77") +
    geom_point(aes(y=X3), color="#1B9E77") +
    geom_line(aes(y=X4), color="#E6AB02") +
    geom_point(aes(y=X4), color="#E6AB02") +
    theme_bw() +
    labs(x = "Dimension",
         y = "Value") +
    ylim(0,1) +
    scale_x_discrete(limits=c(1:10))
}
# Saves a plot of iteration 17 for mu
```

```
  if(it == 24){
    plot_data <- data.frame(t(mu))
    p11 <- ggplot(plot_data, aes(x=1:10)) +
      geom_line(aes(y=X2), color="#7570B3") +
      geom_point(aes(y=X2), color="#7570B3") +
      geom_line(aes(y=X1), color="#D95F02") +
      geom_point(aes(y=X1), color="#D95F02") +
      geom_line(aes(y=X3), color="#1B9E77") +
      geom_point(aes(y=X3), color="#1B9E77") +
      geom_line(aes(y=X4), color="#E6AB02") +
      geom_point(aes(y=X4), color="#E6AB02") +
      theme_bw() +
      labs(x = "Dimension",
           y = "Value") +
      ylim(0,1) +
      scale_x_discrete(limits=c(1:10))
  }
}
ggarrange(p9, p10, p11, p12,
          labels=c("Iteration 1","Iteration 12","Iteration 24", "Iteration 32"),
          font.label = list(size = 10),
          hjust = -0.9,
          vjust = 2)
```