Lab 1 Topic 1 Block 2 Machine Learning

Hugo Knape & Zahra Jalil Pour & Niklas Larsson

11/12/2020

State of contribution

Assigment 1: Ensembled methods

Task A

Generate test data:

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.0.3
```

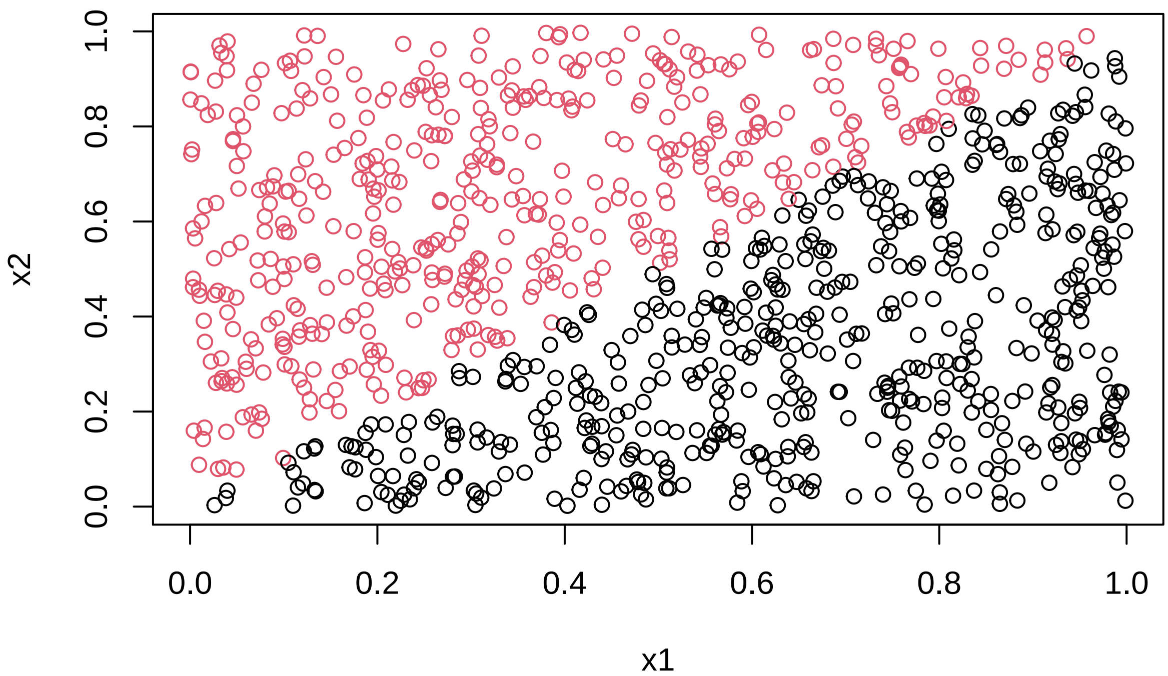
```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
set.seed(1234)

x1<-runif(1000)
x2<-runif(1000)
tedataA<-cbind(x1,x2)
y<-as.numeric(x1<x2)
telabelsA<-as.factor(y)
plot(x1,x2,col=(y+1))
```

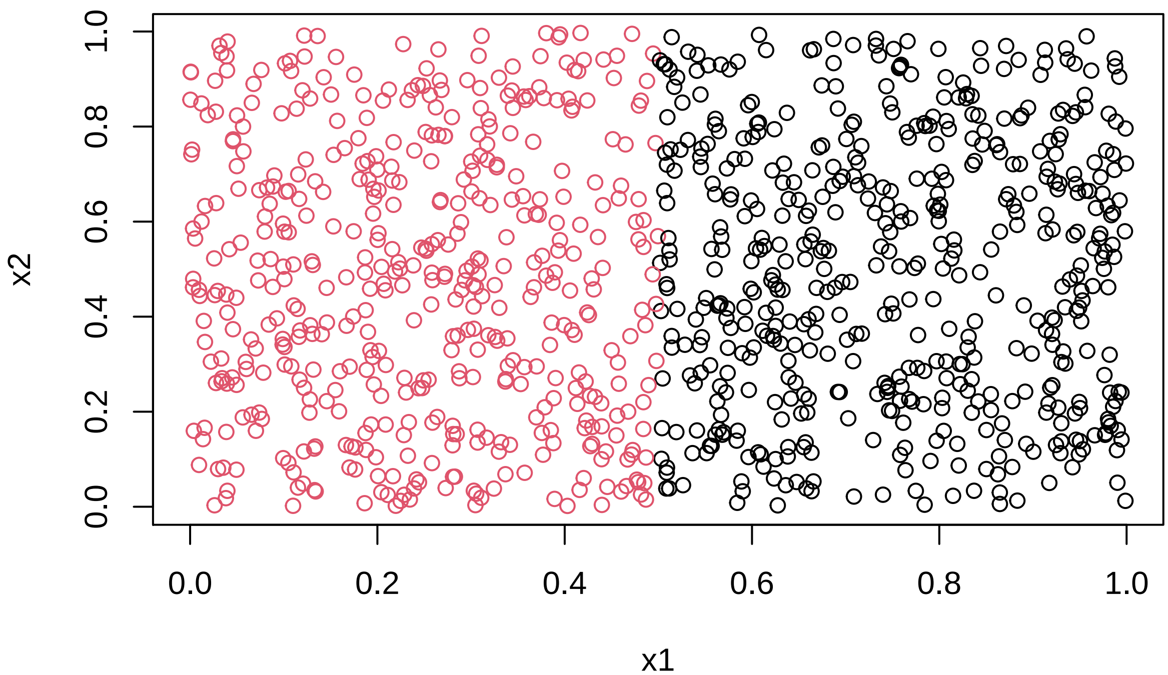



```
forestsize = c(1,10,100)
results = matrix(0,100,3)

for (i in 1:3){
  ntree = forestsize[i]

  for(j in 1:100){
    x1<-runif(100)
    x2<-runif(100)
    trdata<-cbind(x1,x2)
    y<-as.numeric(x1<x2)
    trlabels<-as.factor(y)

    fitA = randomForest(trdata, trlabels, ntree=ntree, nodesize = 25, keep.forest = TRUE)
    predA = predict(fitA, tedataA)
    results[j,i] = sum(as.numeric(predA != telabelsA))/length(telabelsA)
  }
}

resdataA = matrix(0,3,2)
dimnames(resdataA) = list(c("[1]","[10]","[100]"),c("Mean","Variance"))
for(i in 1:3){
  resdataA[i,1] = mean(results[,i])
  resdataA[i,2] = var(results[,i])
}
```

Task B

New data generation

```
set.seed(1234)

x1<-runif(1000)
x2<-runif(1000)
tedataB<-cbind(x1,x2)
y<-as.numeric(x1<0.5)
telabelsB<-as.factor(y)
plot(x1,x2,col=(y+1))
```



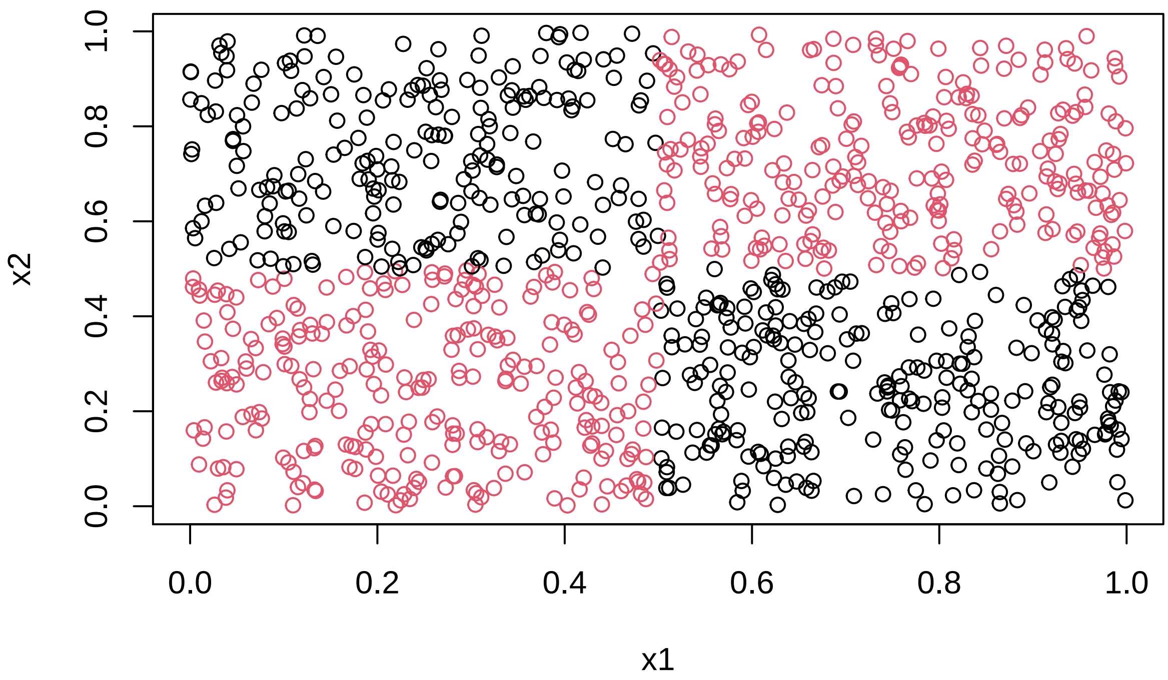

```
forestsize = c(1,10,100)
results = matrix(0,100,3)

for (i in 1:3){
  ntree = forestsize[i]

  for(j in 1:100){
    x1<-runif(100)
    x2<-runif(100)
    trdata<-cbind(x1,x2)
    y<-as.numeric(x1<0.5)
```

```
    trlabels<-as.factor(y)

    fitB = randomForest(trdata, trlabels, ntree=ntree, nodesize = 25, keep.forest = TRUE)
    predB = predict(fitB, tedataB)
    results[j,i] = sum(as.numeric(predB != telabelsB))/length(telabelsB)
  }
}

resdataB = matrix(0,3,2)
dimnames(resdataB) = list(c("[1]","[10]","[100]"),c("Mean","Variance"))
for(i in 1:3){
  resdataB[i,1] = mean(results[,i])
  resdataB[i,2] = var(results[,i])
}
```

Task C

New data generation

```
set.seed(1234)

x1<-runif(1000)
x2<-runif(1000)
tedataC<-cbind(x1,x2)
y<-as.numeric((x1<0.5 & x2<0.5) | (x1>0.5 & x2>0.5))
telabelsC<-as.factor(y)
plot(x1,x2,col=(y+1))
```




```
forestsize = c(1,10,100)
results = matrix(0,100,3)

for (i in 1:3){
  ntree = forestsize[i]

  for(j in 1:100){
    x1<-runif(100)
    x2<-runif(100)
    trdata<-cbind(x1,x2)
    y<-as.numeric( (x1<0.5 & x2<0.5) | (x1>0.5 & x2>0.5) )
    trlabels<-as.factor(y)

    fitC = randomForest(trdata, trlabels, ntree=ntree, nodesize = 12, keep.forest = TRUE)
    predC = predict(fitC, tedataC)
    results[j,i] = sum(as.numeric(predC != telabelsC))/length(telabelsC)
  }
}

resdataC = matrix(0,3,2)
dimnames(resdataC) = list(c("[1]","[10]","[100]"),c("Mean","Variance"))
for(i in 1:3){
  resdataC[i,1] = mean(results[,i])
  resdataC[i,2] = var(results[,i])
}
```

Task D

Question A

Question: What happens with the mean and variance of the error rate when the number of trees in the random forest grows ?

As seen in all three cases the mean error rate decreases with an increasing number of trees in the forest. The same applies for the variance of the error except for case 1 where the variance increased just slightly between the 10- and 100-trees models.

```
print("Task A results:", quote = FALSE)
```

```
## [1] Task A results:
```

```
print(resdataA)
```

```
##          Mean     Variance
## [1]   0.20471 0.0034907130
## [10]  0.13291 0.0008548302
## [100] 0.11047 0.0009660900
```

```
print("Task B results:", quote = FALSE)
```

```
## [1] Task B results:
```

```
print(resdataB)
```

```
##          Mean     Variance
## [1]   0.09324 0.0195394570
## [10]  0.01463 0.0003040334
## [100] 0.00692 0.0001089228
```

```
print("Task B results:", quote = FALSE)
```

```
## [1] Task B results:
```

```
print(resdataC)
```

```
##          Mean    Variance
## [1]   0.23886 0.011984303
## [10]  0.11890 0.003328333
## [100] 0.07728 0.001224567
```

Question B

Due to the node size parameter. Using a smaller minimum node size allows the tree to grow more, in other words the model can divide the data into more specific sections where it can label data. Using smaller node size will need more computation power and could potentially overfit to the data while too large size does the opposite.

Question C

The variance is a measure which tells how much the resulting misclassification are deviating from the mean error. Having a lower variance gives a higher certainty regarding the mean error rate.

Assignment 2: Mixture models

```
EM_algo <- function(k_num){
set.seed(1234567890)
max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log likelihood between two consecutive EM 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(n in 1:N) {
  k <- sample(1:3,1,prob=true_pi)
  for(d in 1:D) {
    x[n,d] <- rbinom(1,1,true_mu[k,d])
  }
}

K <- k_num# number of guessed components
z <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow=K, ncol=D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations

# Random initialization of the paramters
pi <- runif(K,0.49,0.51)
pi <- pi / sum(pi)
for(k in 1:K) {
  mu[k,] <- runif(D,0.49,0.51)
}

#pi
#mu

for(it in 1:max_it) {
  #plot(mu[1,], type="o", col="blue", ylim=c(0,1))
  #points(mu[2,], type="o", col="red")
  #points(mu[3,], type="o", col="green")
  #points(mu[4,], type="o", col="yellow")
```

```
  Sys.sleep(0.5)

  # E-step: Computation of the fractional component assignments
  for (j in 1:k) {
    for (i in 1:n) {
      z[i,j] <- pi[j]*prod((mu[j,]^x[i,])*((1-mu[j,])^(1-x[i,])))
    }
  }
  for(j in 1:nrow(z)) {
    z[j,] <- z[j,]/sum(z[j,])
  }

  #Log likelihood computation.
  for(i in 1:n ){
    for(j in 1:k){
      llik[it] <- llik[it] + z[i,j] * (log(pi[j]) + sum(x[i,] * log(mu[j,]) + (1- x[i,])*log(1- mu[j,]))
    }
  }

  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the lok likelihood has not changed significantly
  if(it > 1 ){
    if(( abs(llik[it]- llik[it-1])  < min_change)){
      break
    }
  }
  #M-step: ML parameter estimation from the data and fractional component assignments
  row_sum_z <- c(rep(NA, ncol(z)))
  for (i in 1:ncol(z)) {
    row_sum_z[i] <- sum(z[,i])
  }
  pi <- row_sum_z/N
  mu <- t(z) %*% x /row_sum_z

}
return(list(pi = pi))
}
```

For the E-step for mixtures of multivariate Bernoulli distributions we compute:

$$p\left(z_{nk} \mid \boldsymbol{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi}\right)=\frac{p\left(\boldsymbol{z}_{nk}, \boldsymbol{x}_n \mid \boldsymbol{\mu}, \boldsymbol{\pi}\right)}{\sum_k p\left(z_{nk}, \boldsymbol{x}_n \mid \boldsymbol{\mu}, \boldsymbol{\pi}\right)}=\frac{\pi_k p\left(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k\right)}{\sum_k \pi_k p\left(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k\right)}$$

for all n and k

where

$$p(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k)=\prod_i \mu_{ki}^{x_i}\left(1-\mu_{ki}\right)^{(1-x_i)}$$

The code is:

```r
# E-step: Computation of the fractional component assignments
for (j in 1:k) {
  for (i in 1:n) {
    z[i,j] <- pi[j]*prod((mu[j,]^x[i,])*((1-mu[j,])^(1-x[i,])))
  }
}
for(j in 1:nrow(z)) {
  z[j,] <- z[j,]/sum(z[j,])
}
```

For computing the Log likelihood we use:

$$\sum_n \sum_k p\left(z_{nk} \mid \boldsymbol{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi}\right)\left[\log \pi_k + \sum_i \left[x_{ni} \log \mu_{ki} + \left(1 - x_{ni}\right) \log\left(1 - \mu_{ki}\right)\right]\right]$$

The code is:

```r
#Log likelihood computation.
for(i in 1:n ){
  for(j in 1:k){
    llik[it] <- llik[it] + z[i,j] * (log(pi[j]) + sum(x[i,] * log(mu[j,]) + (1- x[i,])*log(1- mu[j,])
  }
}
```

ML parameter estimation from the data and fractional component assignments we use:

$$\pi_k^{ML} = \frac{\sum_n p\left(z_{nk} \mid \boldsymbol{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi}\right)}{N}$$
$$\mu_{ki}^{ML} = \frac{\sum_n x_{ni} p\left(z_{nk} \mid \boldsymbol{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi}\right)}{\sum_n p\left(z_{nk} \mid \boldsymbol{x}_n, \boldsymbol{\mu}, \boldsymbol{\pi}\right)}$$

The code is:

```r
#M-step: ML parameter estimation from the data and fractional component assignments
row_sum_z <- c(rep(NA, ncol(z)))
for (i in 1:ncol(z)) {
  row_sum_z[i] <- sum(z[,i])
}
```

```r
EM_algo(2)
```

```
## iteration:  1 log likelihood:  -7623.873
## iteration:  2 log likelihood:  -7621.944
## iteration:  3 log likelihood:  -7620.533
## iteration:  4 log likelihood:  -7609.638
## iteration:  5 log likelihood:  -7532.2
## iteration:  6 log likelihood:  -7173.56
## iteration:  7 log likelihood:  -6661.821
## iteration:  8 log likelihood:  -6520.028
## iteration:  9 log likelihood:  -6503.563
## iteration:  10 log likelihood:  -6499.807
## iteration:  11 log likelihood:  -6498.296
```

```
## iteration:  12 log likelihood:  -6497.535
## iteration:  13 log likelihood:  -6497.12
## iteration:  14 log likelihood:  -6496.883
## iteration:  15 log likelihood:  -6496.745
## iteration:  16 log likelihood:  -6496.662
```

```
## $pi
## [1] 0.4981919 0.5018081
```

When K is equal to 2 the EM-algorithm stops after 16 iterations and the pi values are very close to each other. In this case we miss one true pi, which maybe can be seen as under fitting.

```
EM_algo(3)
```

```
## iteration:  1 log likelihood:  -8029.723
## iteration:  2 log likelihood:  -8027.183
## iteration:  3 log likelihood:  -8024.696
## iteration:  4 log likelihood:  -8005.631
## iteration:  5 log likelihood:  -7877.606
## iteration:  6 log likelihood:  -7403.513
## iteration:  7 log likelihood:  -6936.919
## iteration:  8 log likelihood:  -6818.582
## iteration:  9 log likelihood:  -6791.377
## iteration:  10 log likelihood:  -6780.713
## iteration:  11 log likelihood:  -6774.958
## iteration:  12 log likelihood:  -6771.261
## iteration:  13 log likelihood:  -6768.606
## iteration:  14 log likelihood:  -6766.535
## iteration:  15 log likelihood:  -6764.815
## iteration:  16 log likelihood:  -6763.316
## iteration:  17 log likelihood:  -6761.967
## iteration:  18 log likelihood:  -6760.727
## iteration:  19 log likelihood:  -6759.572
## iteration:  20 log likelihood:  -6758.491
## iteration:  21 log likelihood:  -6757.475
## iteration:  22 log likelihood:  -6756.521
## iteration:  23 log likelihood:  -6755.625
## iteration:  24 log likelihood:  -6754.784
## iteration:  25 log likelihood:  -6753.996
## iteration:  26 log likelihood:  -6753.26
## iteration:  27 log likelihood:  -6752.571
## iteration:  28 log likelihood:  -6751.928
## iteration:  29 log likelihood:  -6751.328
## iteration:  30 log likelihood:  -6750.768
## iteration:  31 log likelihood:  -6750.246
## iteration:  32 log likelihood:  -6749.758
## iteration:  33 log likelihood:  -6749.304
## iteration:  34 log likelihood:  -6748.88
## iteration:  35 log likelihood:  -6748.484
## iteration:  36 log likelihood:  -6748.114
## iteration:  37 log likelihood:  -6747.767
## iteration:  38 log likelihood:  -6747.444
## iteration:  39 log likelihood:  -6747.14
```

```
## iteration:  40 log likelihood:  -6746.856
## iteration:  41 log likelihood:  -6746.589
## iteration:  42 log likelihood:  -6746.338
## iteration:  43 log likelihood:  -6746.102
## iteration:  44 log likelihood:  -6745.88
## iteration:  45 log likelihood:  -6745.67
## iteration:  46 log likelihood:  -6745.472
## iteration:  47 log likelihood:  -6745.285
## iteration:  48 log likelihood:  -6745.108
## iteration:  49 log likelihood:  -6744.939
## iteration:  50 log likelihood:  -6744.78
## iteration:  51 log likelihood:  -6744.627
## iteration:  52 log likelihood:  -6744.483
## iteration:  53 log likelihood:  -6744.344
## iteration:  54 log likelihood:  -6744.212
## iteration:  55 log likelihood:  -6744.086
## iteration:  56 log likelihood:  -6743.964
## iteration:  57 log likelihood:  -6743.848
## iteration:  58 log likelihood:  -6743.736
## iteration:  59 log likelihood:  -6743.628
## iteration:  60 log likelihood:  -6743.524
## iteration:  61 log likelihood:  -6743.423
## iteration:  62 log likelihood:  -6743.326
```

```
## $pi
## [1] 0.3259592 0.3044579 0.3695828
```

When K is equal to 3 the EM-algorithm stops after 62 iterations and the pi values are pretty close to each other.

```r
EM_algo(4)
```

```
## iteration:  1 log likelihood:  -8317.187
## iteration:  2 log likelihood:  -8314.81
## iteration:  3 log likelihood:  -8312.256
## iteration:  4 log likelihood:  -8292.606
## iteration:  5 log likelihood:  -8159.059
## iteration:  6 log likelihood:  -7666.637
## iteration:  7 log likelihood:  -7196.701
## iteration:  8 log likelihood:  -7061.15
## iteration:  9 log likelihood:  -7018.948
## iteration:  10 log likelihood:  -6999.971
## iteration:  11 log likelihood:  -6989.735
## iteration:  12 log likelihood:  -6983.5
## iteration:  13 log likelihood:  -6979.315
## iteration:  14 log likelihood:  -6976.279
## iteration:  15 log likelihood:  -6973.932
## iteration:  16 log likelihood:  -6972.026
## iteration:  17 log likelihood:  -6970.415
## iteration:  18 log likelihood:  -6969.009
## iteration:  19 log likelihood:  -6967.751
## iteration:  20 log likelihood:  -6966.598
## iteration:  21 log likelihood:  -6965.517
```

```
## iteration:  22 log likelihood:  -6964.48
## iteration:  23 log likelihood:  -6963.457
## iteration:  24 log likelihood:  -6962.415
## iteration:  25 log likelihood:  -6961.313
## iteration:  26 log likelihood:  -6960.098
## iteration:  27 log likelihood:  -6958.703
## iteration:  28 log likelihood:  -6957.042
## iteration:  29 log likelihood:  -6955.01
## iteration:  30 log likelihood:  -6952.485
## iteration:  31 log likelihood:  -6949.342
## iteration:  32 log likelihood:  -6945.475
## iteration:  33 log likelihood:  -6940.834
## iteration:  34 log likelihood:  -6935.458
## iteration:  35 log likelihood:  -6929.501
## iteration:  36 log likelihood:  -6923.217
## iteration:  37 log likelihood:  -6916.917
## iteration:  38 log likelihood:  -6910.896
## iteration:  39 log likelihood:  -6905.381
## iteration:  40 log likelihood:  -6900.502
## iteration:  41 log likelihood:  -6896.299
## iteration:  42 log likelihood:  -6892.745
## iteration:  43 log likelihood:  -6889.776
## iteration:  44 log likelihood:  -6887.313
## iteration:  45 log likelihood:  -6885.273
## iteration:  46 log likelihood:  -6883.583
## iteration:  47 log likelihood:  -6882.178
## iteration:  48 log likelihood:  -6881.007
## iteration:  49 log likelihood:  -6880.024
## iteration:  50 log likelihood:  -6879.196
## iteration:  51 log likelihood:  -6878.494
## iteration:  52 log likelihood:  -6877.895
## iteration:  53 log likelihood:  -6877.383
## iteration:  54 log likelihood:  -6876.941
## iteration:  55 log likelihood:  -6876.56
## iteration:  56 log likelihood:  -6876.228
## iteration:  57 log likelihood:  -6875.939
## iteration:  58 log likelihood:  -6875.687
## iteration:  59 log likelihood:  -6875.465
## iteration:  60 log likelihood:  -6875.27
## iteration:  61 log likelihood:  -6875.098
## iteration:  62 log likelihood:  -6874.947
## iteration:  63 log likelihood:  -6874.813
## iteration:  64 log likelihood:  -6874.694
## iteration:  65 log likelihood:  -6874.59
## iteration:  66 log likelihood:  -6874.497
```

```
## $pi
## [1] 0.1614155 0.1383613 0.3609912 0.3392319
```

When K is equal to 4 the EM-algorithm stops after 16 iterations and the pi values are pretty far from each other. In this task we have one extra pi, which maybe can be seen as over fitting.

Assignment 3

Appendix:

```
knitr::opts_chunk$set(echo = TRUE)
library(kknn)
library(ggplot2)

library(randomForest)
set.seed(1234)

x1<-runif(1000)
x2<-runif(1000)
tedataA<-cbind(x1,x2)
y<-as.numeric(x1<x2)
telabelsA<-as.factor(y)
plot(x1,x2,col=(y+1))


forestsize = c(1,10,100)
results = matrix(0,100,3)

for (i in 1:3){
  ntree = forestsize[i]

  for(j in 1:100){
    x1<-runif(100)
    x2<-runif(100)
    trdata<-cbind(x1,x2)
    y<-as.numeric(x1<x2)
    trlabels<-as.factor(y)

    fitA = randomForest(trdata, trlabels, ntree=ntree, nodesize = 25, keep.forest = TRUE)
    predA = predict(fitA, tedataA)
    results[j,i] = sum(as.numeric(predA != telabelsA))/length(telabelsA)
  }
}

resdataA = matrix(0,3,2)
dimnames(resdataA) = list(c("[1]","[10]","[100]"),c("Mean","Variance"))
for(i in 1:3){
  resdataA[i,1] = mean(results[,i])
  resdataA[i,2] = var(results[,i])
}

set.seed(1234)

x1<-runif(1000)
x2<-runif(1000)
tedataB<-cbind(x1,x2)
y<-as.numeric(x1<0.5)
telabelsB<-as.factor(y)
```

```
plot(x1,x2,col=(y+1))

forestsize = c(1,10,100)
results = matrix(0,100,3)

for (i in 1:3){
  ntree = forestsize[i]

  for(j in 1:100){
    x1<-runif(100)
    x2<-runif(100)
    trdata<-cbind(x1,x2)
    y<-as.numeric(x1<0.5)
    trlabels<-as.factor(y)

    fitB = randomForest(trdata, trlabels, ntree=ntree, nodesize = 25, keep.forest = TRUE)
    predB = predict(fitB, tedataB)
    results[j,i] = sum(as.numeric(predB != telabelsB))/length(telabelsB)
  }
}

resdataB = matrix(0,3,2)
dimnames(resdataB) = list(c("[1]","[10]","[100]"),c("Mean","Variance"))
for(i in 1:3){
  resdataB[i,1] = mean(results[,i])
  resdataB[i,2] = var(results[,i])
}

set.seed(1234)

x1<-runif(1000)
x2<-runif(1000)
tedataC<-cbind(x1,x2)
y<-as.numeric((x1<0.5 & x2<0.5) | (x1>0.5 & x2>0.5))
telabelsC<-as.factor(y)
plot(x1,x2,col=(y+1))

forestsize = c(1,10,100)
results = matrix(0,100,3)

for (i in 1:3){
  ntree = forestsize[i]

  for(j in 1:100){
    x1<-runif(100)
    x2<-runif(100)
    trdata<-cbind(x1,x2)
    y<-as.numeric( (x1<0.5 & x2<0.5) | (x1>0.5 & x2>0.5) )
    trlabels<-as.factor(y)

    fitC = randomForest(trdata, trlabels, ntree=ntree, nodesize = 12, keep.forest = TRUE)
    predC = predict(fitC, tedataC)
    results[j,i] = sum(as.numeric(predC != telabelsC))/length(telabelsC)
```

```
  }
}

resdataC = matrix(0,3,2)
dimnames(resdataC) = list(c("[1]","[10]","[100]"),c("Mean","Variance"))
for(i in 1:3){
  resdataC[i,1] = mean(results[,i])
  resdataC[i,2] = var(results[,i])
}

print("Task A results:", quote = FALSE)
print(resdataA)

print("Task B results:", quote = FALSE)
print(resdataB)

print("Task B results:", quote = FALSE)
print(resdataC)
EM_algo <- function(k_num){
set.seed(1234567890)
max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log likelihood between two consecutive EM 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(n in 1:N) {
  k <- sample(1:3,1,prob=true_pi)
  for(d in 1:D) {
    x[n,d] <- rbinom(1,1,true_mu[k,d])
  }
}

K <- k_num# number of guessed components
z <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow=K, ncol=D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations

# Random initialization of the paramters
pi <- runif(K,0.49,0.51)
pi <- pi / sum(pi)
for(k in 1:K) {
  mu[k,] <- runif(D,0.49,0.51)
}
```

```r
#pi
#mu

for(it in 1:max_it) {
  #plot(mu[1,], type="o", col="blue", ylim=c(0,1))
  #points(mu[2,], type="o", col="red")
  #points(mu[3,], type="o", col="green")
  #points(mu[4,], type="o", col="yellow")
  Sys.sleep(0.5)

  # E-step: Computation of the fractional component assignments
  for (j in 1:k) {
    for (i in 1:n) {
      z[i,j] <- pi[j]*prod((mu[j,]^x[i,])*((1-mu[j,])^(1-x[i,])))
    }
  }
  for(j in 1:nrow(z)) {
    z[j,] <- z[j,]/sum(z[j,])
  }

  #Log likelihood computation.
  for(i in 1:n ){
    for(j in 1:k){
      llik[it] <- llik[it] + z[i,j] * (log(pi[j]) + sum(x[i,] * log(mu[j,]) + (1- x[i,])*log(1- mu[j,])
    }
  }

  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the lok likelihood has not changed significantly
  if(it > 1 ){
    if(( abs(llik[it]- llik[it-1])  < min_change)){
      break
    }
  }
  #M-step: ML parameter estimation from the data and fractional component assignments
  row_sum_z <- c(rep(NA, ncol(z)))
  for (i in 1:ncol(z)) {
    row_sum_z[i] <- sum(z[,i])
  }
  pi <- row_sum_z/N
  mu <- t(z) %*% x /row_sum_z

}
return(list(pi = pi))
}
  # E-step: Computation of the fractional component assignments
  for (j in 1:k) {
    for (i in 1:n) {
      z[i,j] <- pi[j]*prod((mu[j,]^x[i,])*((1-mu[j,])^(1-x[i,])))
    }
  }
  for(j in 1:nrow(z)) {
```

```
    z[j,] <- z[j,]/sum(z[j,])
  }
  #Log likelihood computation.
  for(i in 1:n ){
    for(j in 1:k){
      llik[it] <- llik[it] + z[i,j] * (log(pi[j]) + sum(x[i,] * log(mu[j,]) + (1- x[i,])*log(1- mu[j,]))
    }
  }


  #M-step: ML parameter estimation from the data and fractional component assignments
  row_sum_z <- c(rep(NA, ncol(z)))
  for (i in 1:ncol(z)) {
    row_sum_z[i] <- sum(z[,i])
  }
EM_algo(2)
EM_algo(3)
EM_algo(4)
```