# Introduction to Machine Learning

Lab 2 Block 2

*Anton Persson, Emil Klasson Svensson, Mattias Karlsson, Rasmus Holm*

*2016-12-09*

## Contents

# Assignment 1a

Assumptions:

$$\mathrm{E}\left[\epsilon^b(x)\right] = 0,$$
$$\forall_{i,j}, i \neq j : \mathrm{E}\left[\epsilon^i(x)\epsilon^j(x)\right] = 0$$

Prove:

$$\mathrm{E}_X\left[(f_{bag}(x) - h(x))^2\right] = \frac{1}{B}\left[\frac{1}{B}\sum_b \mathrm{E}_X\left[(f^b(x) - h(x))^2\right]\right]$$

We know:

$$f^b(x) = h(x) + \epsilon^b(x)$$
$$f_{\text{bag}}(x) = \frac{1}{B}\sum_b f^b(x)$$

Proof:

$$\mathrm{E}_X\left[(f_{bag}(x) - h(x))^2\right] =$$
$$\mathrm{E}_X\left[(\frac{1}{B}\sum_b f^b(x) - h(x))^2\right] =$$
$$\mathrm{E}_X\left[(\frac{1}{B}\sum_b \epsilon^b(x))^2\right] =$$
$$\frac{1}{B^2}\mathrm{E}_X\left[(\epsilon^1(x))^2 + 2\epsilon^1(x)\epsilon^2(x) + \cdots + 2\epsilon^{b-1}(x)\epsilon^b(x) + (\epsilon^b(x))^2\right] =$$
$$\frac{1}{B^2}\left(\mathrm{E}_X\left[(\epsilon^1(x))^2\right] + 2\mathrm{E}_X\left[\epsilon^1(x)\epsilon^2(x)\right] + \cdots + 2\mathrm{E}_X\left[\epsilon^{b-1}(x)\epsilon^b(x)\right] + \mathrm{E}_X\left[(\epsilon^b(x))^2\right]\right) =$$
$$\frac{1}{B^2}\sum_b \mathrm{E}_X\left[(\epsilon^b(x))^2\right] =$$
$$\frac{1}{B^2}\sum_b \mathrm{E}_X\left[(f^b(x) - h(x))^2\right] =$$
$$\frac{1}{B}\left[\frac{1}{B}\sum_b \mathrm{E}_X\left[(f^b(x) - h(x))^2\right]\right]$$

# Assignment 2a

## 1

An estimated upperbound of the squared error of the bagging regression tree using 2/3 of the data set as training data and 1/3 for testing.

```
set.seed(1234567890)
id <- sample(1:n, floor(n* (2/3)), replace = FALSE)
fatTrain <- fatbody[id,]
fatTest <- fatbody[-id,]

n_T <- 74
n_Te <- 36

bagger <- function(train, B){
    df <- data.frame("MSE" = 1:B)
    for( i in 1:B){
        treeData <- train[sample(1:n_T, replace = TRUE),]
        regTree <- tree(Bodyfat_percent ~ ., data = treeData)
        df$MSE[i] <- mean((predict(regTree,fatTest)-fatTest[,3])^2)
    }

    baggyMSE <- mean(df$MSE)
    return(baggyMSE)
}

bagger(fatTrain,100)
#> [1] 37.11309
```

## 2

An estimated upperbound of the squared error of the bagging regression tree using 3-fold cross-validation.

```
set.seed(1234567890)
tree_count <- 100
fold_count <- 3
test_errors <- matrix(0, nrow=tree_count, ncol=fold_count)

folds <- suppressWarnings(split(1:nrow(fatbody), f=1:fold_count))

for (j in 1:fold_count) {
    train <- fatbody[-folds[[j]],]
    test <- fatbody[folds[[j]],]

    for (i in 1:tree_count) {
        newdata <- train[sample(nrow(train), replace=TRUE),]
        fit <- tree(Bodyfat_percent ~ ., data=newdata, split="deviance")

        test_error <- mean((predict(fit, test) - test$Bodyfat)^2)
        test_errors[i, j] <- test_error
    }
}
```

```
mean(test_errors)
#> [1] 40.19377
```

### 3

The resulting bagging regression tree and its predicted value where data is the complete data set and newdata is the data to predict. We would return this result for both the techniques used to estimate the upperbound of the squared error.

```
bagging.regtrees <- function(formula, data, newdata, b) {
    predictions <- matrix(0, nrow=nrow(newdata), ncol=b)
    trees <- list()

    for (i in 1:b) {
        bootstrap_sample <- data[sample(nrow(data), replace=TRUE),]
        fit <- tree(formula, data=bootstrap_sample, split="deviance")
        trees[[i]] <- fit
        predictions[, i] <- predict(fit, newdata)
    }

    list(trees=trees, predictions=rowMeans(predictions))
}
```

## Assignment 2b

The plot below shows the three true multivariate Bernoulli distributions from which the data set have been generated.

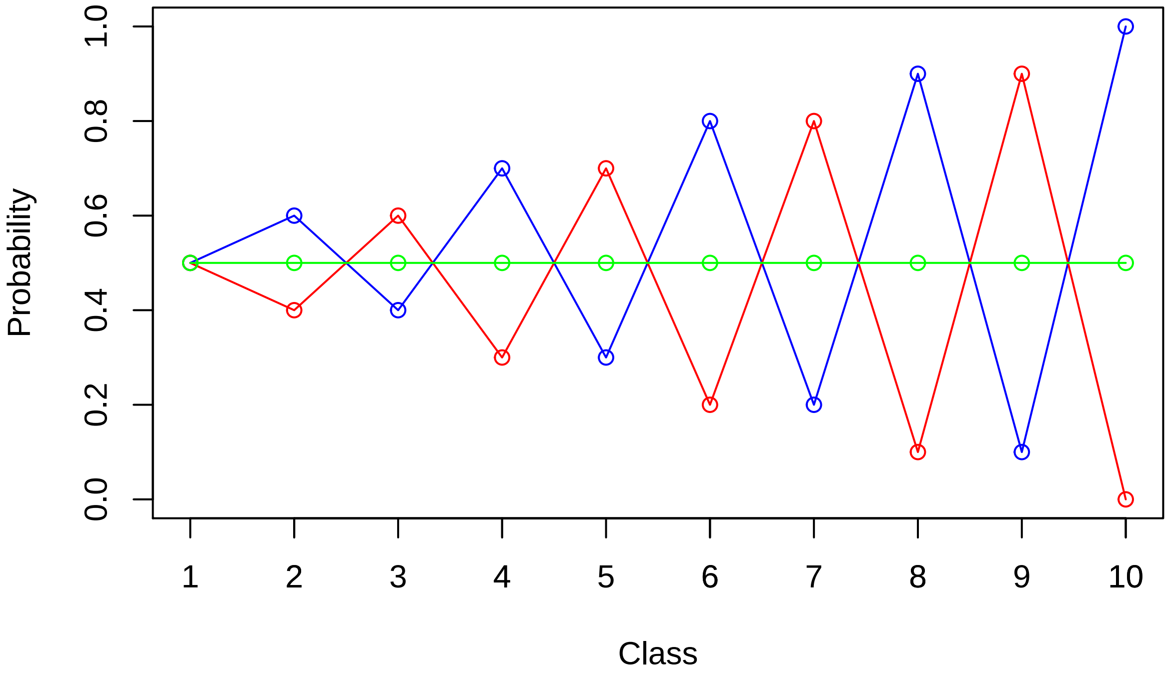


*Figure 1: The true probabilities of the multivariate Bernoulli distributions.*

The plot below shows two multivariate Bernoulli distributions estimated by the expectation-maximization (EM) algorithm. We can see that the multivariate Bernoulli with equal probabilities for each class has not affected EM particular much in order to find the other two distributions. This is probably because equal probabilities even out in the long run, i.e. the noise from that distribution is approximately equal for both sides of the coin.

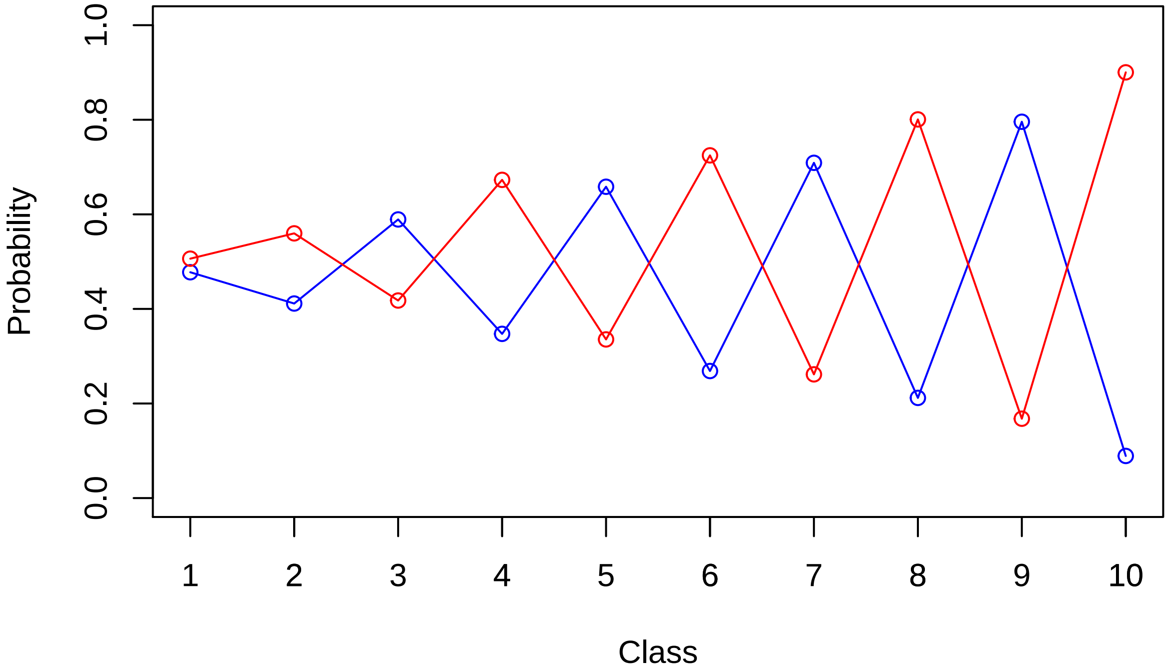


*Figure 2: The estimated probabilities of the multivariate Bernoulli distributions.*

The prior probabilities for each distribution.

```
#> [1] 0.497125 0.502875
```

The plot below shows the log-likelihood versus the number of iterations.

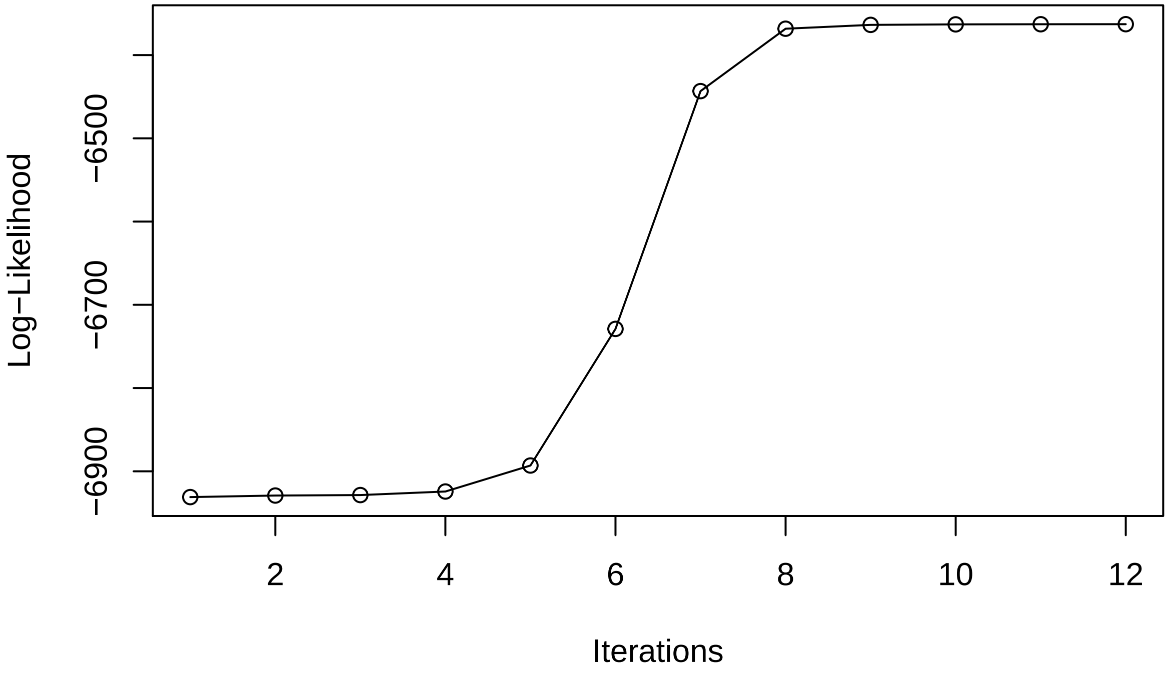


*Figure 3: The log-likelihood versus the number of iterations.*

The plot below shows three multivariate Bernoulli distributions estimated by the EM algorithm. The distributions found are pretty similar to the true ones with exceptation of the uniform one which have been influenced by the other two distributions and/or bad luck on the coin flips that have resulted in seemingly unfair coins.

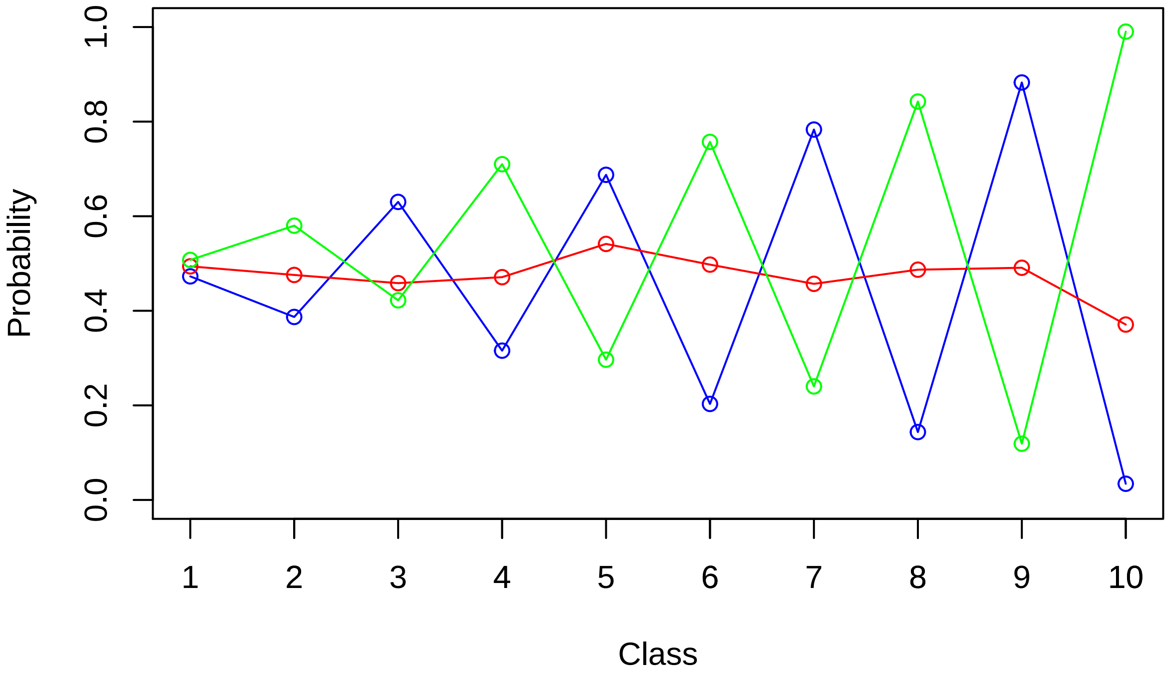


*Figure 4: The estimated probabilities of the multivariate Bernoulli distributions.*

The prior probabilities for each distribution.

```
#> [1] 0.3416794 0.2690298 0.3892909
```

The plot below shows the log-likelihood versus the number of iterations.

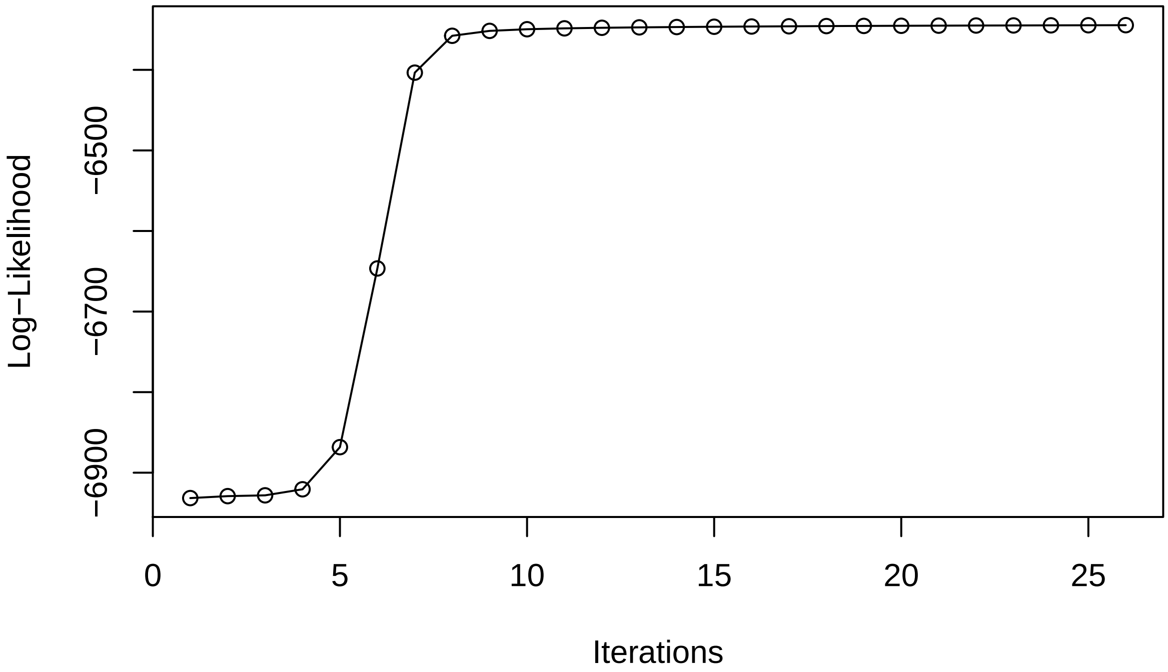


*Figure 5: The log-likelihood versus the number of iterations.*

The plot below shows four multivariate Bernoulli distributions estimated by the EM algorithm. The blue and red curves are quite chaotic that do not resemble any of the true ones but taking the average would approximate the multivariate Bernoulli distribution with uniform parameters pretty well. So the EM algorithm have basically modelled two distributions based on the noise from the uniform one which is not surprising given that there are only three true distributions and that one is the most unpredictable.

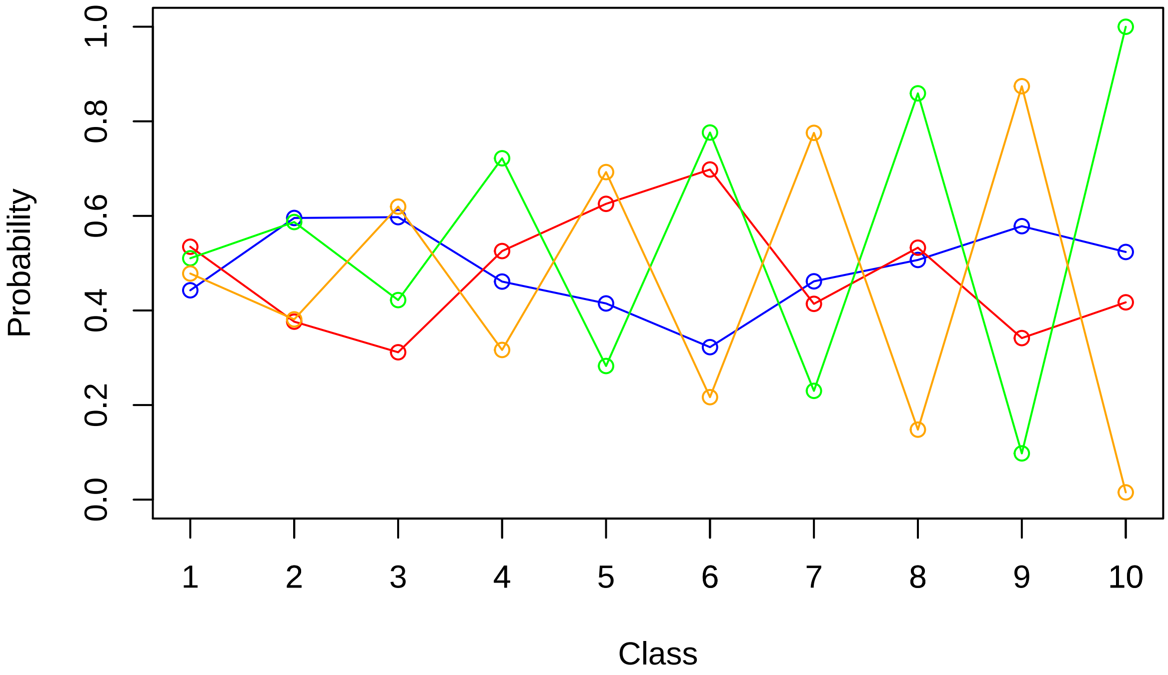


*Figure 6: The estimated probabilities of the multivariate Bernoulli distributions.*

The prior probabilities for each distribution.

```
#> [1] 0.1547196 0.1418652 0.3514089 0.3520062
```

The plot below shows the log-likelihood versus the number of iterations.

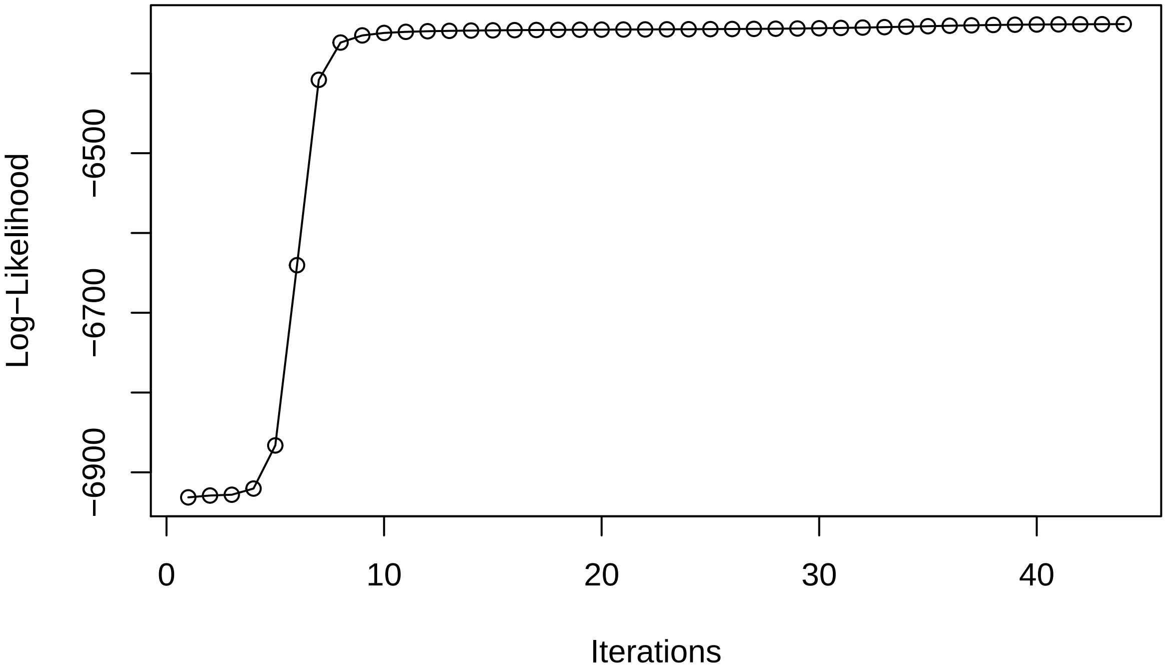


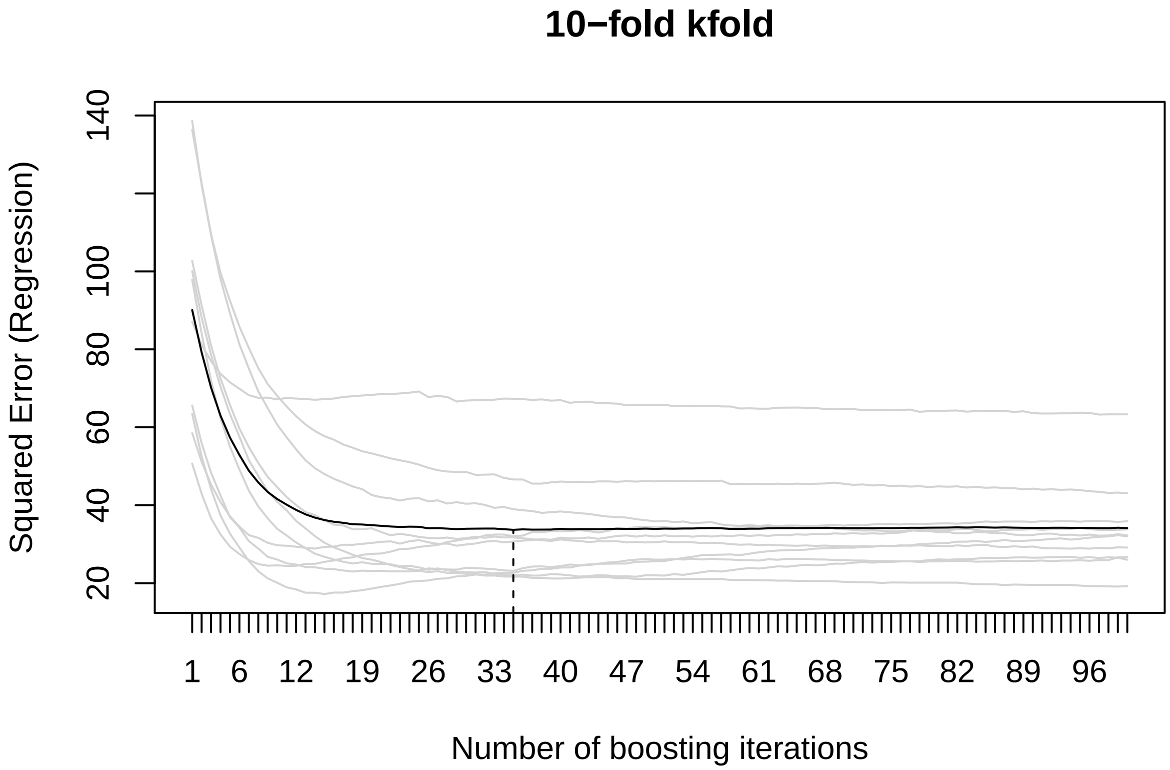
*Figure 7: The log-likelihood versus the number of iterations.*

We can see that the EM algorithm have almost found the maximum log-likelihood after roughly 10 iterations for all settings in this experiment.

## Assignment 3a

### 1

The number of iterations in boosting is a hyper-parameter and the plot below shows estimated squared errors using 10-fold cross-validation as the number of iterations increases from 1 to 100. The gray curves are the mean squared errors from each validation set and the black curve shows the mean of the those errors. The optimal seems to be around 35 iterations.



### 2

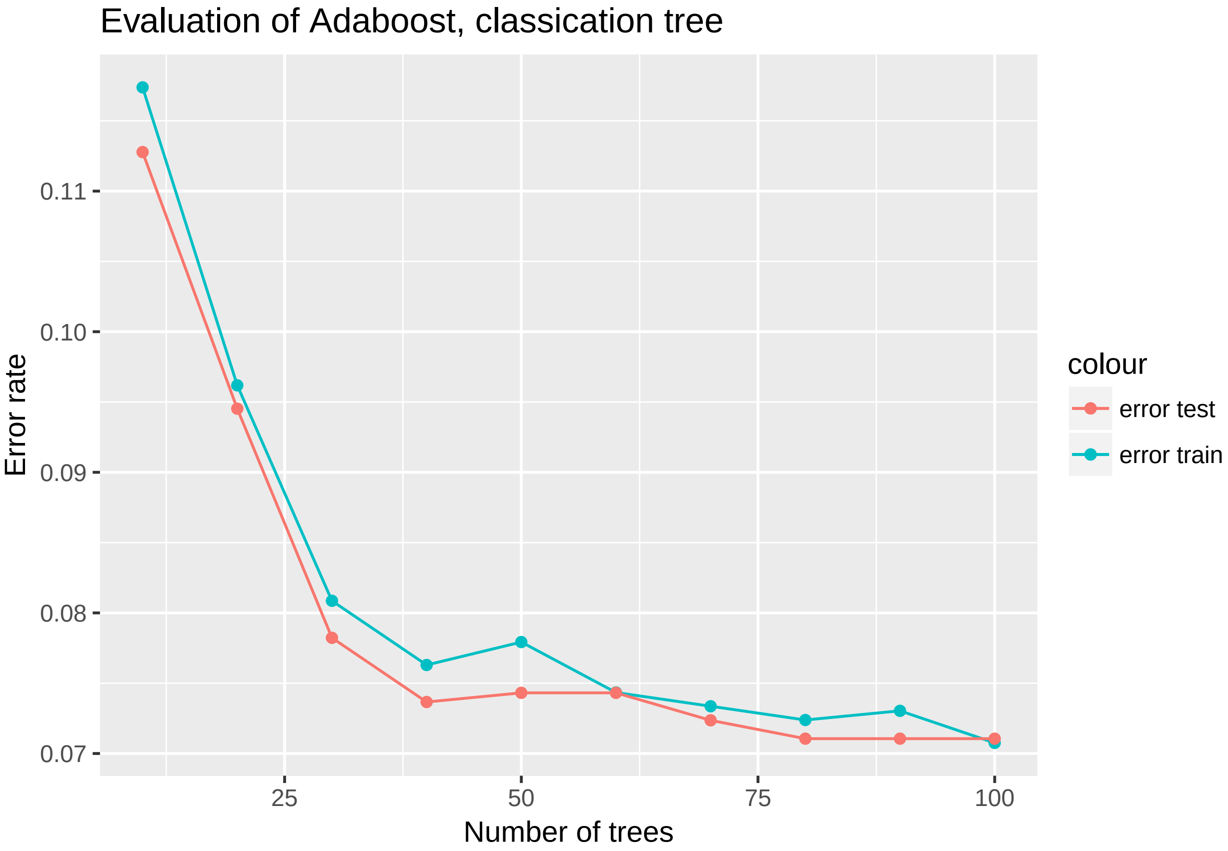
Here we used 2/3 of the data for training and 1/3 as test data and created a boosting regression tree using the optimal number of iterations, i.e. number of trees, found above. The squared errors for the two data sets were the following.

```
#> SSE for traning: 1634.085
#> SSE for test: 867.6893
```

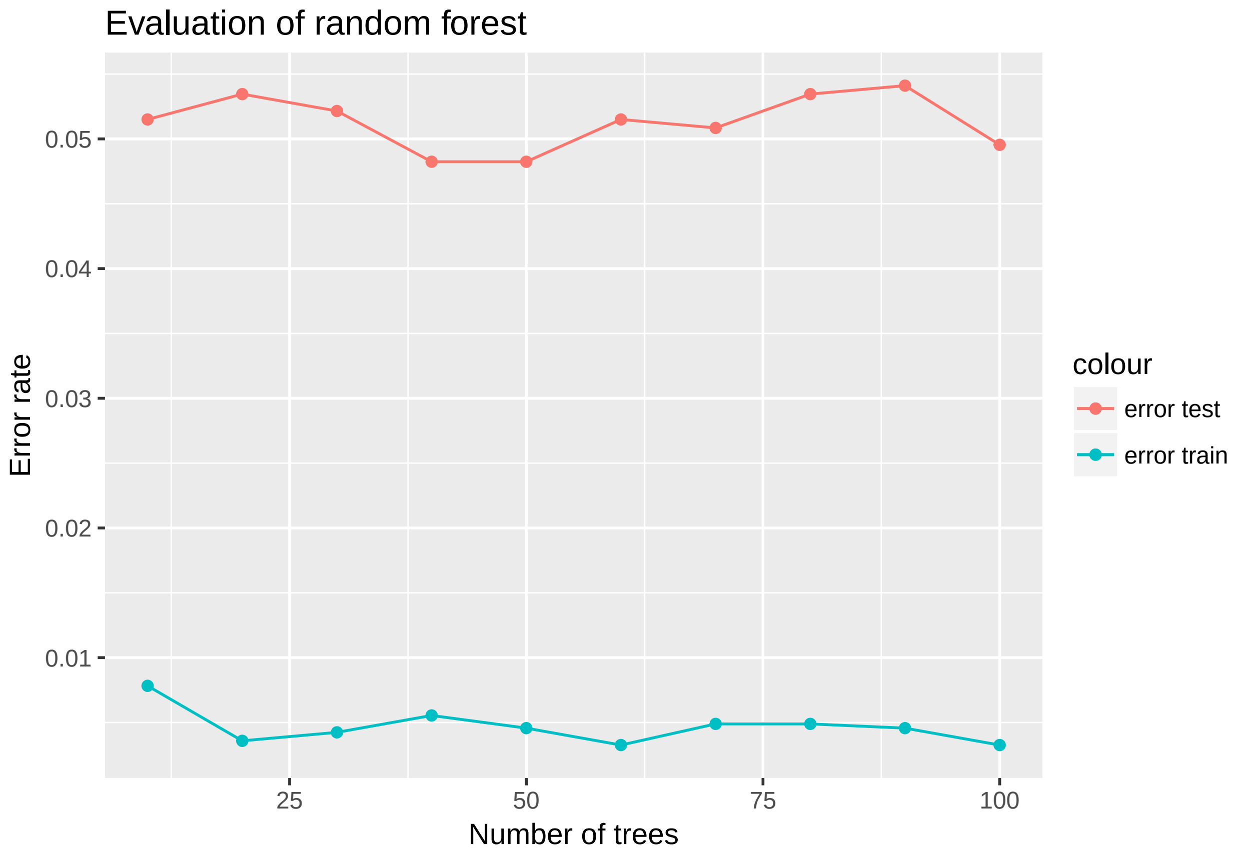
# Assignment 4a

In this exercise we have used Adaboost classification trees and random forests to evauluate their performance on spam data. The data set have been divided into two parts, 2/3 for training and 1/3 as test data.

The performance of the Adaboost classification trees can be seen below. We can see that the optimal would be roughly 40 trees by the elbow technique because after that it barely decreases as the number of trees grows. At 80 trees the test error seems to halt so if you want to push the limit and create a substantially more complex model it may be preferable to use 80 trees. However, since the error on the test data stops monotonically decrease after 40 trees that is the recommended choice.



The performance of the random forest can be seen below. Same as above, the test error seem to stop monotonically decreasing after 40 trees and that should be the prefered choice. We can see that the train error barely moves as the number of trees increases after 20 trees so the model have almost fit the training data perfectly with 20 trees.


Evaluation of random forest
Error rate
0.05
0.04
0.03
0.02
0.01
25
50
75
100
Number of trees
colour
error test
error train

# Appendix

## Code for Assignment 2a

```r
fatbody <- read.csv("../data/bodyfatregression.csv", sep = ";", de = ",")
n <- 110

set.seed(1234567890)
id <- sample(1:n, floor(n* (2/3)), replace = FALSE)
fatTrain <- fatbody[id,]
fatTest <- fatbody[-id,]

n_T <- 74
n_Te <- 36

bagger <- function(train, B){
    df <- data.frame("MSE" = 1:B)
    for( i in 1:B){
        treeData <- train[sample(1:n_T, replace = TRUE),]
        regTree <- tree(Bodyfat_percent ~ ., data = treeData)
        df$MSE[i] <- mean((predict(regTree,fatTest)-fatTest[,3])^2)
    }

    baggyMSE <- mean(df$MSE)
    return(baggyMSE)
}

bagger(fatTrain,100)

set.seed(1234567890)
tree_count <- 100
fold_count <- 3
test_errors <- matrix(0, nrow=tree_count, ncol=fold_count)

folds <- suppressWarnings(split(1:nrow(fatbody), f=1:fold_count))

for (j in 1:fold_count) {
    train <- fatbody[-folds[[j]],]
    test <- fatbody[folds[[j]],]

    for (i in 1:tree_count) {
        newdata <- train[sample(nrow(train), replace=TRUE),]
        fit <- tree(Bodyfat_percent ~ ., data=newdata, split="deviance")

        test_error <- mean((predict(fit, test) - test$Bodyfat)^2)
        test_errors[i, j] <- test_error
    }
}

mean(test_errors)

bagging.regtrees <- function(formula, data, newdata, b) {
    predictions <- matrix(0, nrow=nrow(newdata), ncol=b)
    trees <- list()

    for (i in 1:b) {
        bootstrap_sample <- data[sample(nrow(data), replace=TRUE),]
        fit <- tree(formula, data=bootstrap_sample, split="deviance")
        trees[[i]] <- fit
        predictions[, i] <- predict(fit, newdata)
    }

    list(trees=trees, predictions=rowMeans(predictions))
}
```

## Code for Assignment 2b

```
x_given_mu <- function(x, mu) {
    x_mu <- matrix(1, nrow=nrow(x), ncol=nrow(mu))

    for (n in 1:N) {
        for (k in 1:K) {
            for (i in 1:D) {
                prob <-  mu[k, i]^x[n, i] * (1 - mu[k, i])^(1 - x[n, i])
                x_mu[n, k] <- x_mu[n, k] * prob
            }
        }
    }

    x_mu
}

expectation.step <- function(x, x_given_mu, pi) {
    z <- matrix(nrow=nrow(x), ncol=length(pi))

    for (n in 1:N) {
        denominator <- sum(pi * x_given_mu[n,])

        for (k in 1:K) {
            nominator <- pi[k] * x_given_mu[n, k]

            z[n, k] <- nominator / denominator
        }
    }

    z
}

loglikelihood <- function(x, x_given_mu, pi) {
    llik <- 0
    for (n in 1:N) {
        inner_summation <- 0
        for (k in 1:K) {
            inner_summation <- inner_summation + pi[k] * x_given_mu[n, k]
        }
        llik <- llik + log(inner_summation)
    }

    llik
}


maximization.step <- function(x, z) {
    pi <- vector(length=ncol(z))
    mu <- matrix(nrow=ncol(z), ncol=ncol(x))

    for (k in 1:K) {
        pi[k] <- sum(z[, k]) / nrow(x)
    }

    for (k in 1:K) {
        denominator <- sum(z[, k])
        for (i in 1:D) {
            nominator <- sum(x[, i] * z[, k])
            mu[k, i] <- nominator / denominator
        }
    }

    list(pi=pi, mu=mu)
}

EM <- function(N, D, K, max_it, min_change, true_pi, true_mu) {

    ## Producing the training data
    x <- matrix(nrow=N, ncol=D)

    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])
        }
    }

    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)
    }

    for(it in 1:max_it) {
        x_mu <- x_given_mu(x, mu)
        ## E-step: Computation of the fractional component assignments
        z <- expectation.step(x, x_mu, pi)

        ## Log likelihood computation.
        llik[it] <- loglikelihood(x, x_mu, pi)

        ## Stop if the lok likelihood has not changed significantly
        if (it > 1 && abs(llik[it] - llik[it-1]) < min_change) break

        ## M-step: ML parameter estimation from the data and fractional component assignments
        result <- maximization.step(x, z)
        pi <- result$pi
        mu <- result$mu
    }

    list(pi=pi, mu=mu, llik=llik, it=it)
}

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
K <- 3 # number of guessed components

## true mixing coefficients
true_pi <- vector(length=3)
true_pi <- c(1/3, 1/3, 1/3)

## true conditional distributions
true_mu <- matrix(nrow=3, ncol=D)
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)
plot(true_mu[1,], type="o", col="blue", ylim=c(0,1),
     xlab="Class", ylab="Probability")
axis(side=1, at=c(1:D))
points(true_mu[2,], type="o", col="red")
points(true_mu[3,], type="o", col="green")
set.seed(1234567890)

K <- 2
result <- EM(N, D, K, max_it, min_change, true_pi, true_mu)
mu <- result$mu
pi <- result$pi
llik <- result$llik
it <- result$it
plot(mu[1,], type="o", col="blue", ylim=c(0,1),
     xlab="Class", ylab="Probability")
axis(side=1, at=c(1:D))
points(mu[2,], type="o", col="red")
plot(llik[1:it], type="o", xlab="Iterations",
     ylab="Log-Likelihood")
```

```
set.seed(1234567890)

K <- 3
result <- EM(N, D, K, max_it, min_change, true_pi, true_mu)
mu <- result$mu
pi <- result$pi
llik <- result$llik
it <- result$it
plot(mu[1,], type="o", col="blue", ylim=c(0,1),
     xlab="Class", ylab="Probability")
axis(side=1, at=c(1:D))
points(mu[2,], type="o", col="red")
points(mu[3,], type="o", col="green")
plot(llik[1:it], type="o", xlab="Iterations",
     ylab="Log-Likelihood")
set.seed(1234567890)

K <- 4
result <- EM(N, D, K, max_it, min_change, true_pi, true_mu)
mu <- result$mu
pi <- result$pi
llik <- result$llik
it <- result$it
plot(mu[1,], type="o", col="blue", ylim=c(0,1),
     xlab="Class", ylab="Probability")
axis(side=1, at=c(1:D))
points(mu[2,], type="o", col="red")
points(mu[3,], type="o", col="green")
points(mu[4,], type="o", col="orange")
plot(llik[1:it], type="o", xlab="Iterations",
     ylab="Log-Likelihood")
```

## Code for Assignment 3a

```
library(mboost)

BFR <- read.csv2("B2lab2/bodyfatregression.csv")
set.seed(1234567890)
m <- blackboost(Bodyfat_percent ~ Waist_cm + Weight_kg, data = BFR)

cvf <- cv(model.weights(m), type = "kfold")
cvm <- cvrisk(m, folds = cvf, grid = 1:100)
plot(cvm)

set.seed(1234567890)
m2 <- blackboost(Bodyfat_percent ~ Waist_cm + Weight_kg, data = train,
                 control=boost_control(mstop=mstop(cvm)))

mstop(m2)
cvf2 <- cv(model.weights(m2), type = "kfold")
cvm2 <- cvrisk(m2, folds = cvf2, grid = 1:100)
```

```r
m2.train <- sum( (predict(m2,train) - train$Bodyfat_percent)^2)
m2.test <- sum( (predict(m2,test) - test$Bodyfat_percent)^2)
cat("SSE for traning:",m2.train,"\n SSE for test:",m2.test)
```

## Code for Assignment 4a

```r
library(mboost)
library(randomForest)
library(ggplot2)

spam <- read.csv2("../data/spambase.csv")
spam$Spam<-as.factor(spam$Spam)
set.seed(1234567890)
spam_samplad<-spam[sample(1:nrow(spam)), ]
spam_tr<-spam_samplad[1:round((2/3)*nrow(spam)), ]
spam_te<-spam_samplad[-(1:round((2/3)*nrow(spam))), ]

sekvens<-seq(10,100, 10)
training_errors<-integer()
test_errors<-integer()
index<-1
for (i in sekvens){
  modellen_ct<-blackboost(Spam~., data=spam_tr, family=AdaExp(),  control=boost_control(mstop=i))

  tejbell_train<-table(pred=predict(modellen_ct, newdata= spam_tr, type="class"), truth=spam_tr$Spam)
  training_errors[index]<-1-sum(diag(tejbell_train))/sum(tejbell_train)

  tejbell_test<-table(pred=predict(modellen_ct, newdata= spam_te, type="class"), truth=spam_te$Spam)
  test_errors[index]<-1-sum(diag(tejbell_test))/sum(tejbell_test)
  index<-index+1
}

plotredo_ct<-data.frame(cbind(sekvens,training_errors, test_errors))

ggplot(data=plotredo_ct)+geom_point(aes(x=sekvens, y=training_errors, col="error train"))+
  geom_line(aes(x=sekvens, y=training_errors, col="error train"))+
  geom_point(aes(x=sekvens, y=test_errors, col="error test"))+
  geom_line(aes(x=sekvens, y=test_errors, col="error test"))+xlab("Number of trees")+
  ylab("Error rate")+ggtitle("Evaluation of Adaboost, classication tree")

sekvens<-seq(10,100, 10)
training_errors_rf<-integer()
test_errors_rf<-integer()
index<-1
for (i in sekvens){
  modellen_rf<-randomForest(Spam ~ ., data=spam_tr, ntree=i, norm.votes=FALSE)

  tr_tab<-table(predict(modellen_rf, newdata= spam_tr, type="class"), spam_tr$Spam)
  training_errors_rf[index]<-1-sum(diag(tr_tab))/sum(tr_tab)

  test_tab<-table(predict(modellen_rf, newdata= spam_te, type="class"), spam_te$Spam)
  test_errors_rf[index]<-1-sum(diag(test_tab))/sum(test_tab)
  index<-index+1
}

plotredo_rf<-data.frame(cbind(sekvens,training_errors_rf, test_errors_rf))

ggplot(data=plotredo_rf)+geom_point(aes(x=sekvens, y=training_errors_rf, col="error train"))+
  geom_line(aes(x=sekvens, y=training_errors_rf, col="error train"))+
  geom_point(aes(x=sekvens, y=test_errors_rf, col="error test"))+
  geom_line(aes(x=sekvens, y=test_errors_rf, col="error test"))+xlab("Number of trees")+
    ylab("Error rate")+ggtitle("Evaluation of random forest")
```

## Contributions

We divided the work into two parts and discussed/compiled the results in pairs. Then we all discussed our findings together as a whole group and checked that everyone had similar/understood the results.