Introduction to Machine Learning

Lab 2 Block 2

Rasmus Holm

2016-12-07

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.

```r
set.seed(1234567890)
tree_count <- 100
test_errors <- rep(0, tree_count)

train_idx <- sample(nrow(data), floor(nrow(data) * (2 / 3)))
train <- data[train_idx,]
test <- data[-train_idx,]

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

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

mean(test_errors)
#> [1] 37.20882
```

2

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

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

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

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

    for (i in 1:tree_count) {
        newdata <- train[sample(nrow(train), replace=TRUE),]
        fit <- tree(Bodyfat ~ ., 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. I would return this result for both the techniques used to estimate the upperbound of the squared error.

```r
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.

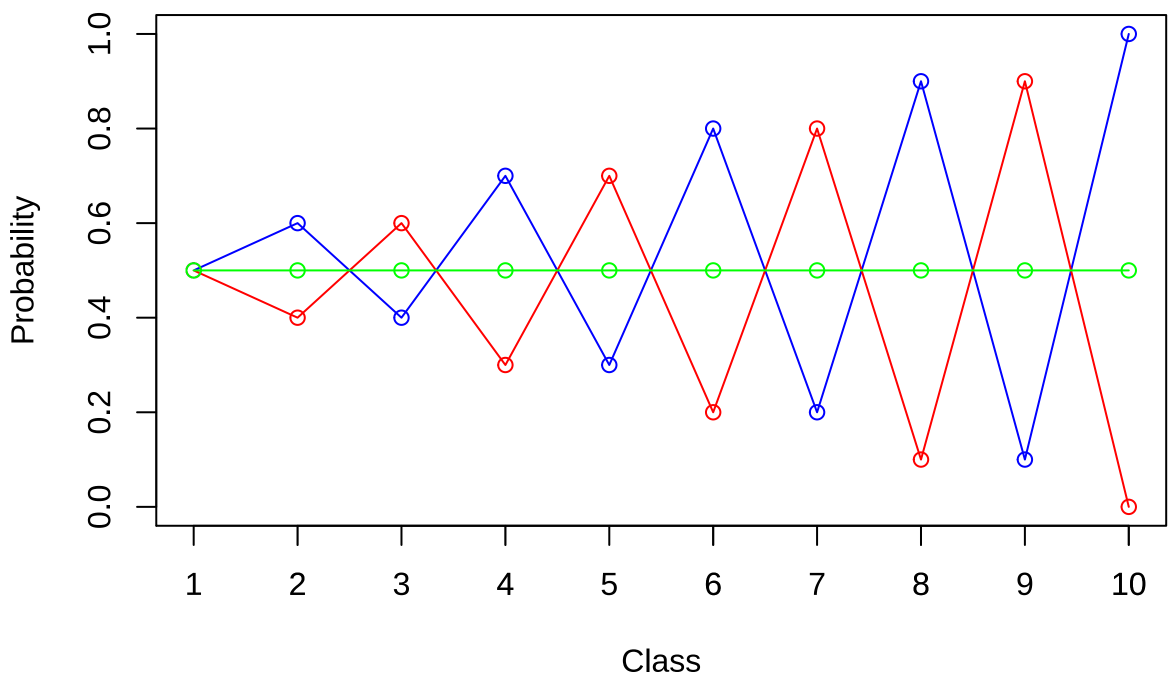


Figure 1: The true probabilities of the multivariate Bernoulli distributions.

The plot below shows two multivariate Bernoulli distributions estimated by the expectation-maximization (EM) algorithm.

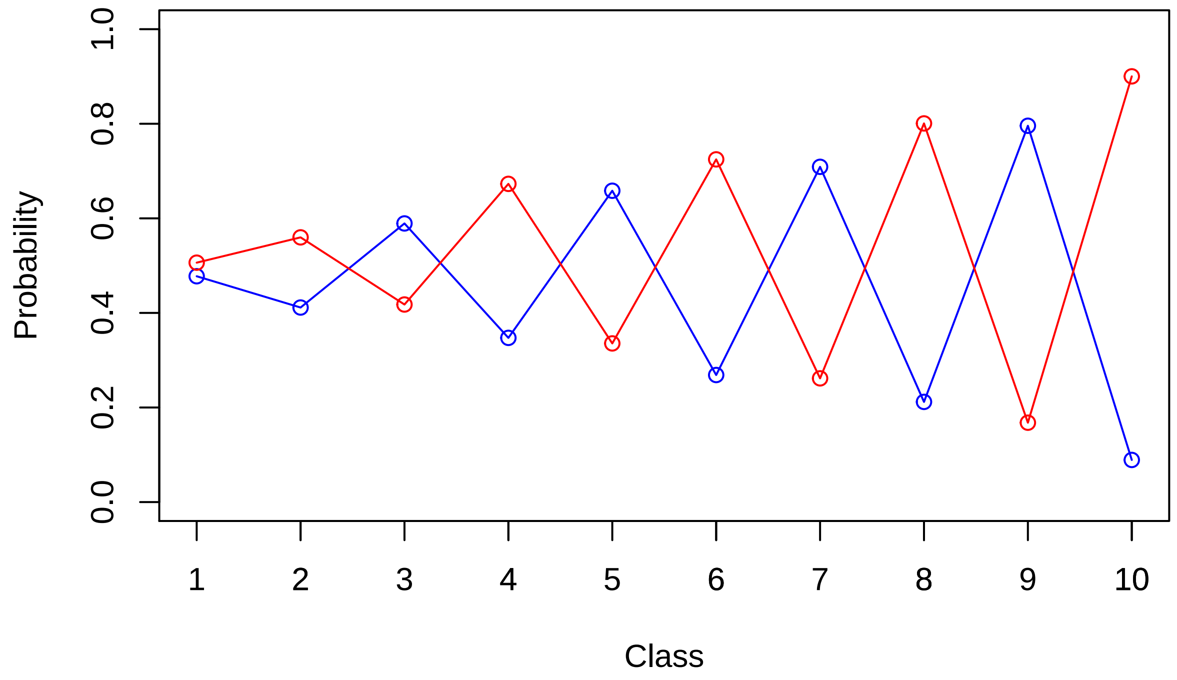


Figure 2: The estimated probabilities of the multivariate Bernoulli distributions.

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

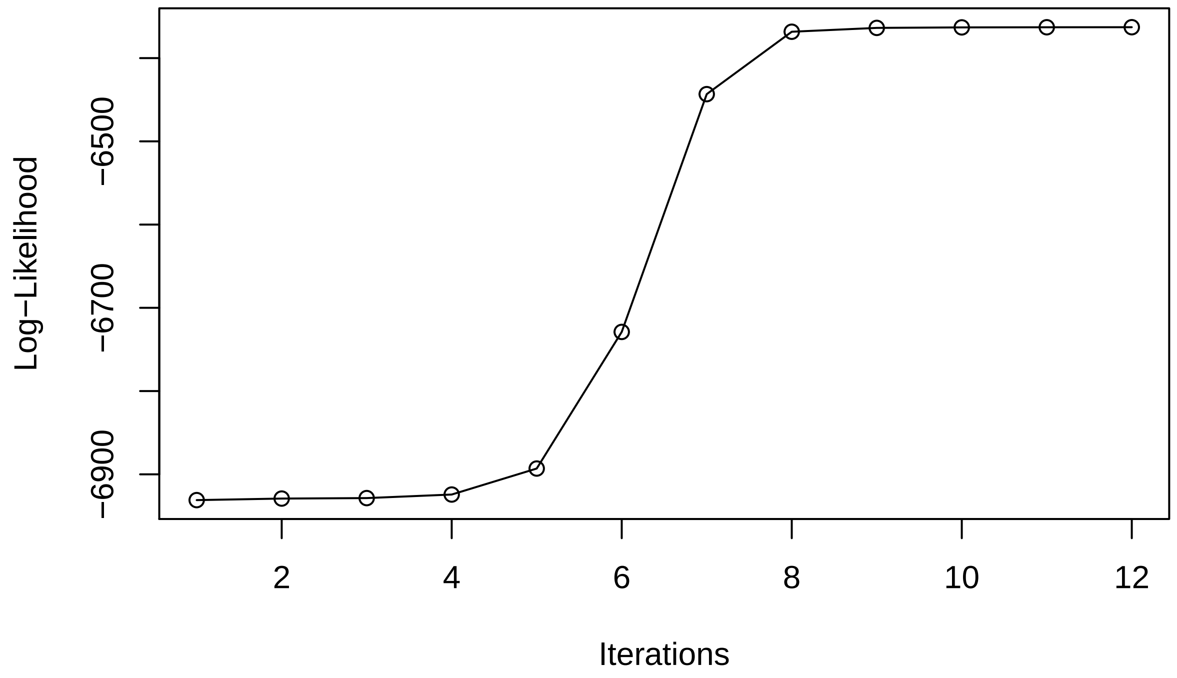


Figure 3: The log-likelihood versus the number of iterations.

The plot below shows three multivariate Bernoulli distributions estimated by the expectation-maximization (EM) algorithm.

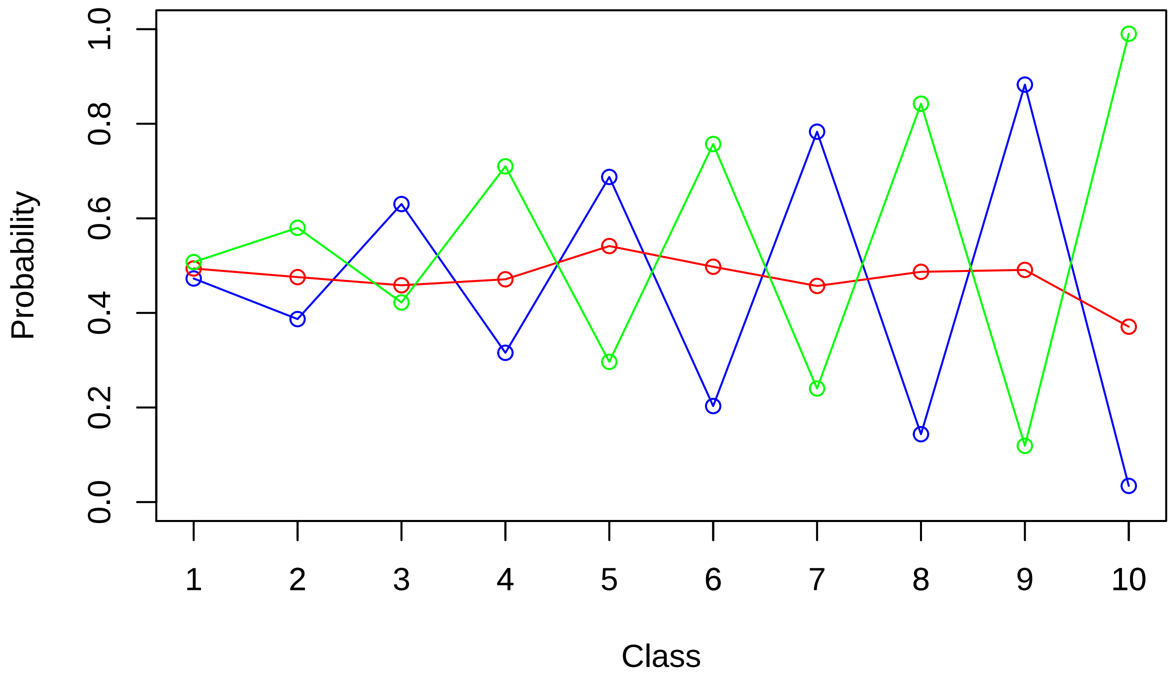


Figure 4: The estimated probabilities of the multivariate Bernoulli distributions.

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

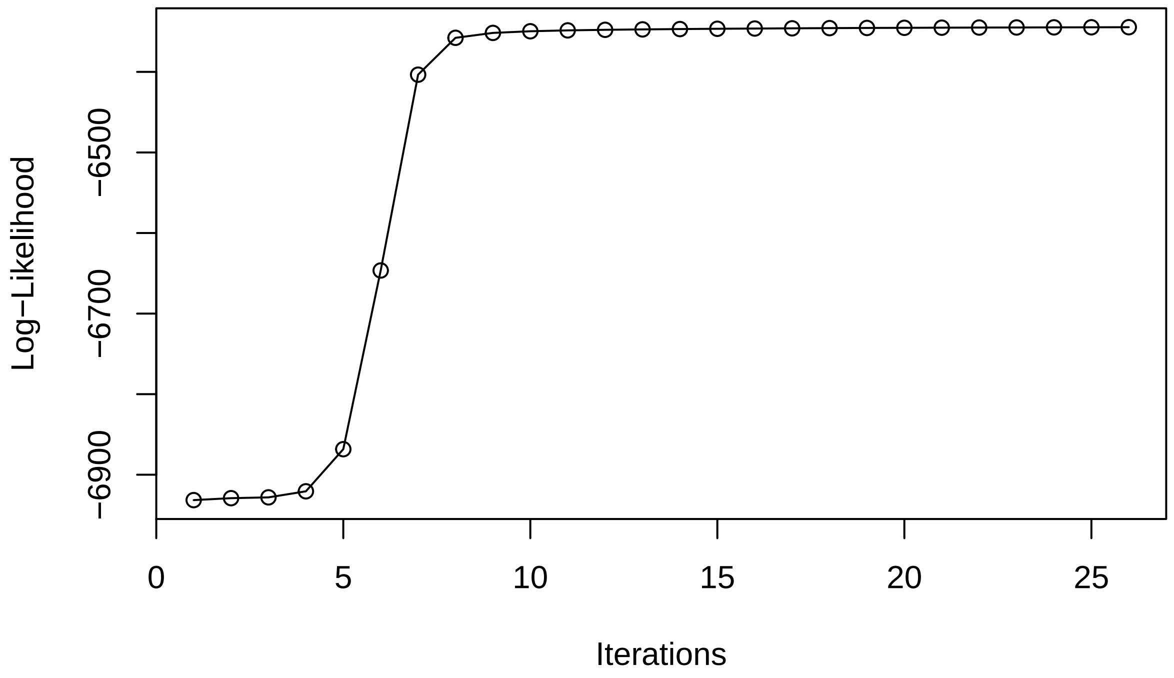


Figure 5: The log-likelihood versus the number of iterations.

The plot below shows four multivariate Bernoulli distributions estimated by the expectation-maximization (EM) algorithm.

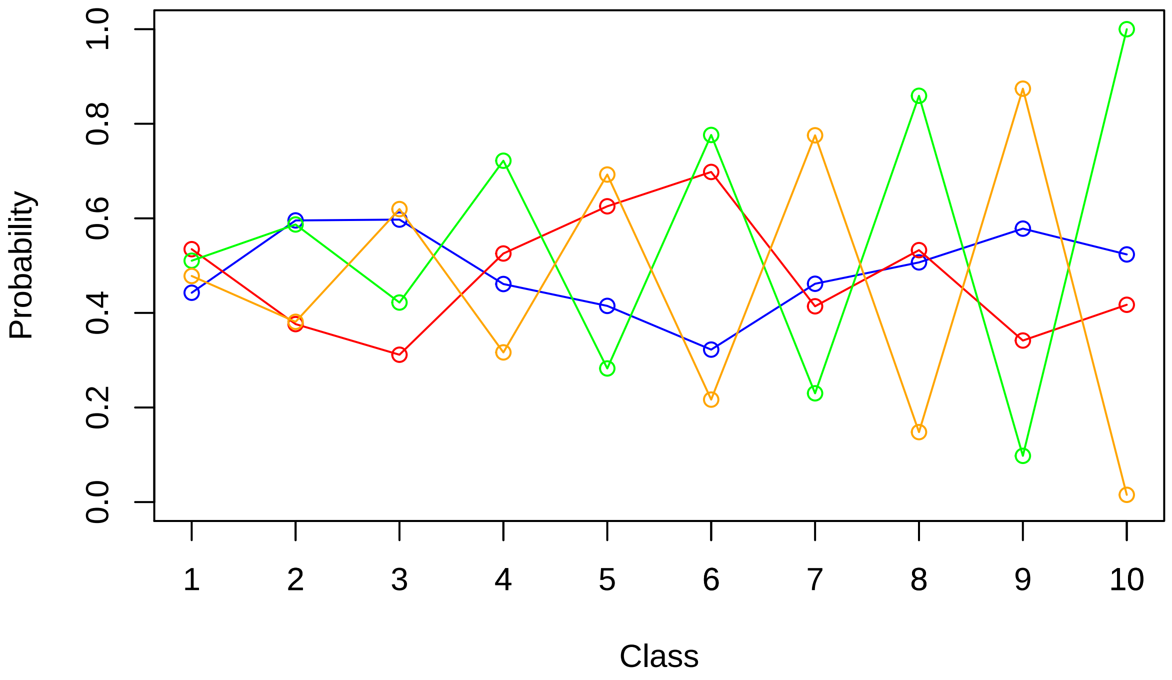


Figure 6: The estimated probabilities of the multivariate Bernoulli distributions.

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

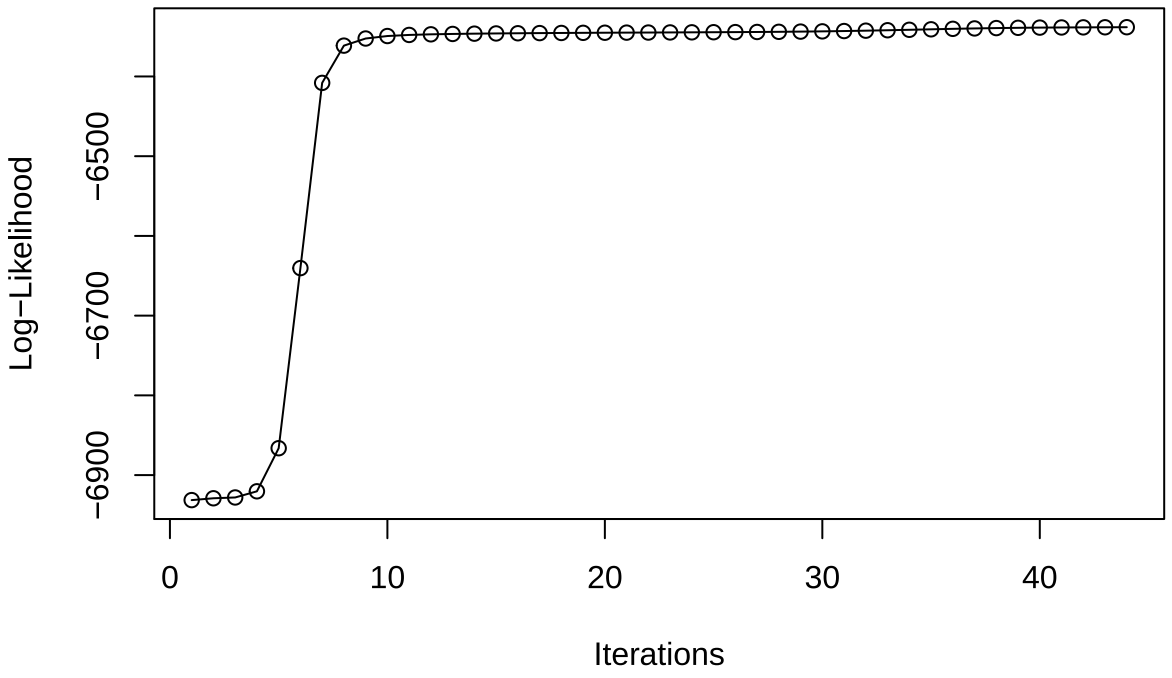
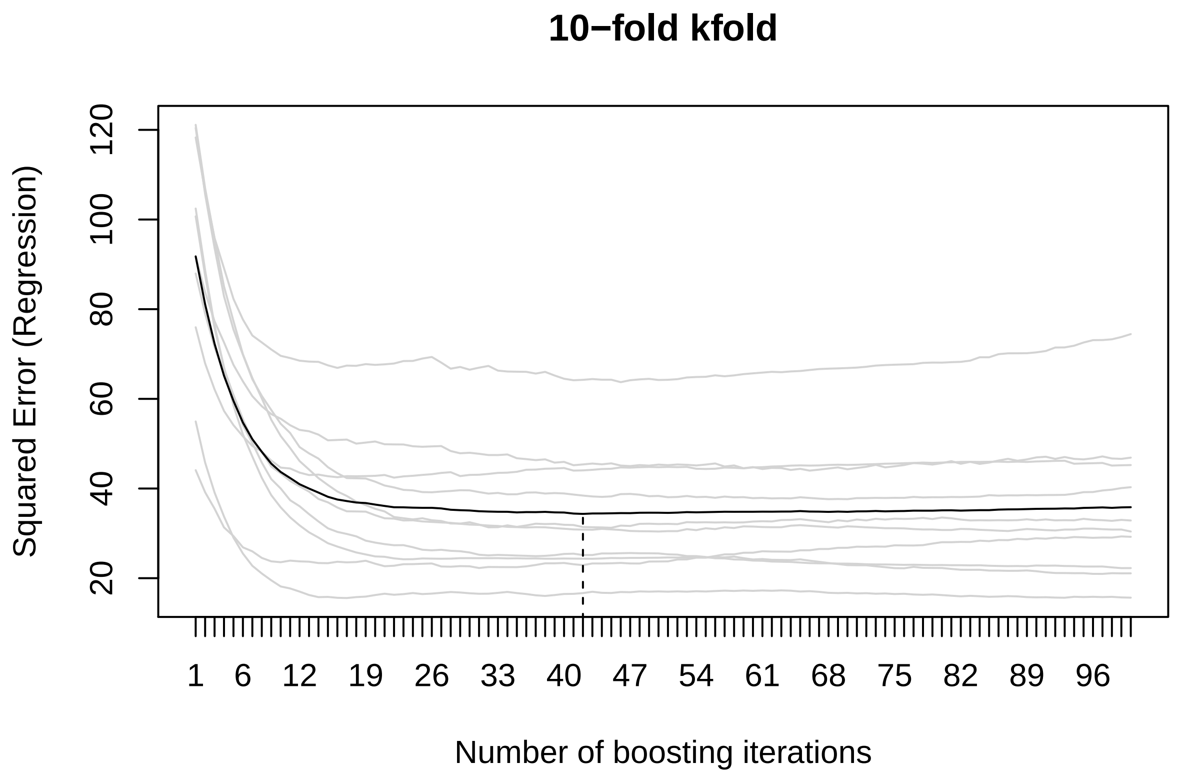


Figure 7: The log-likelihood versus the number of iterations.

Assignment 3a

1

In boosting the number of iterations 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 curve 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 42 iterations.



2

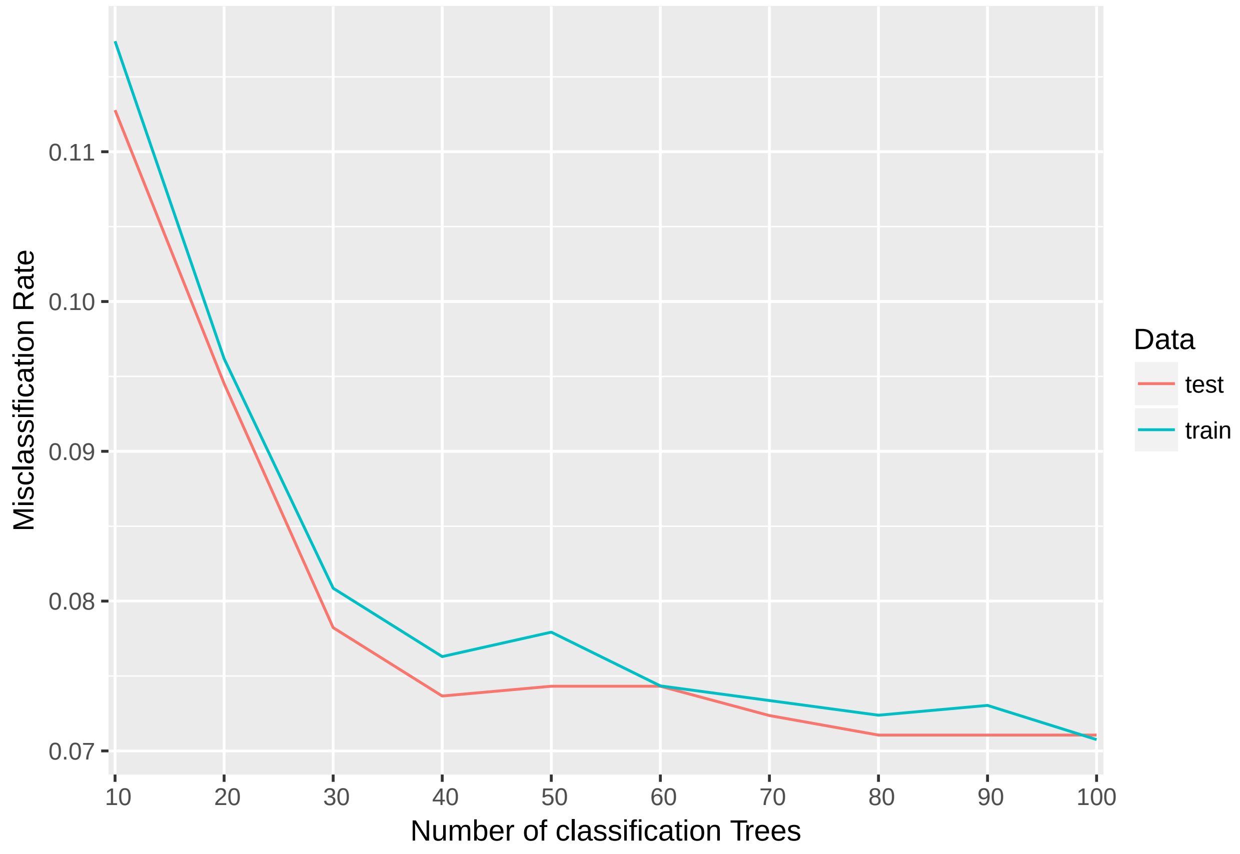
Here I used 2/3 of the data for training and 1/3 as test data and created a boosting regression tree using the optimal number of iterations, i.e. the number of trees, found above. The squared errors for the two data sets were the following.

```
#> Test Error: 988.823293772931
#> Train Error: 1433.94146700376
```

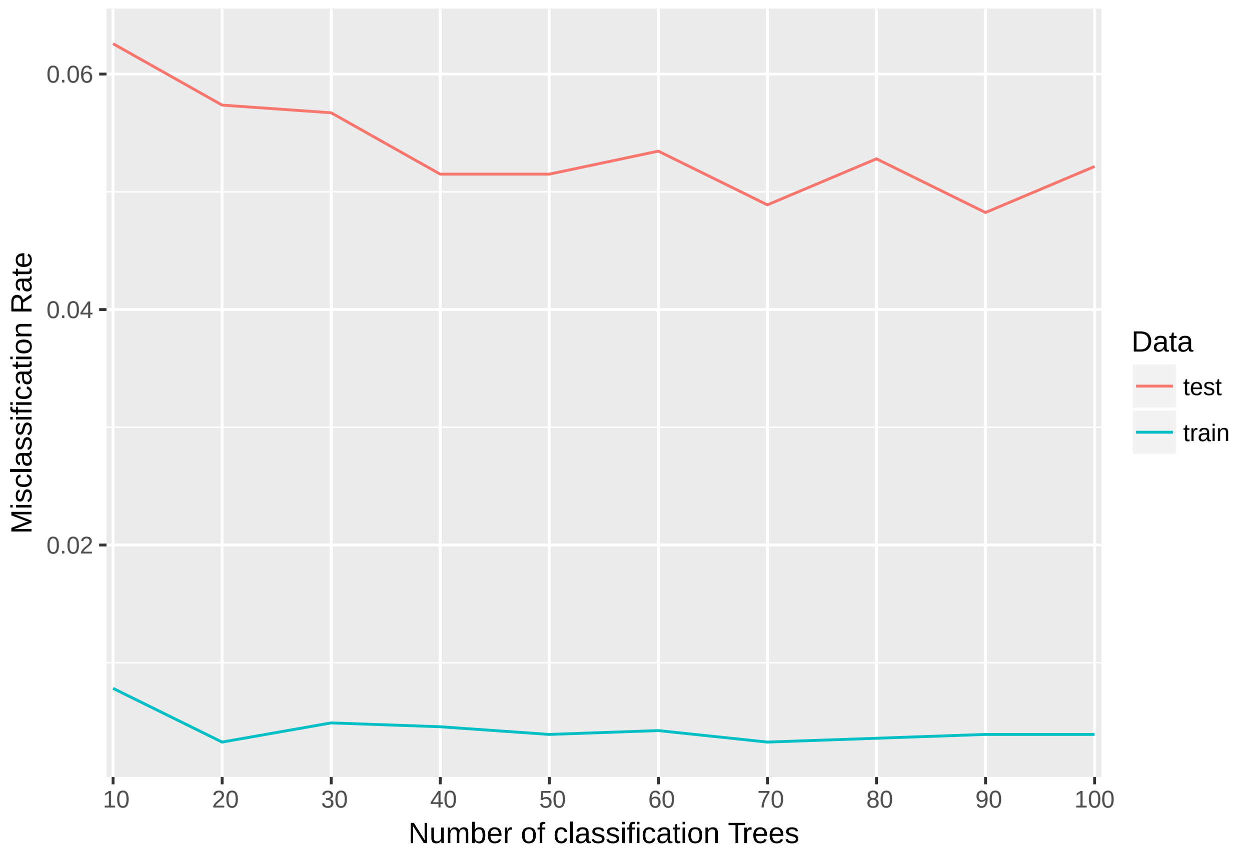
Assignment 4a

In this exercise I 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 then 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 choice I would choose.



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.



0.06
0.04
0.02
Misclassification Rate
10
20
30
40
50
60
70
80
90
100
Number of classification Trees
Data
test
train

Appendix

Code for Assignment 1b

```
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)
```

```r
        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")
```

Code for Assignment 2a

```r
library(tree)

data <- read.csv2("../data/bodyfatregression.csv")
names(data) <- c("Waist", "Weight", "Bodyfat")
set.seed(1234567890)
tree_count <- 100
test_errors <- rep(0, tree_count)

train_idx <- sample(nrow(data), floor(nrow(data) * (2 / 3)))
train <- data[train_idx,]
test <- data[-train_idx,]

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

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

mean(test_errors)
```

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

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

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

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

```r
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
}
```

```r
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)
```

```r
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

```r
library(mboost)

data <- read.csv2("../data/bodyfatregression.csv")

fit <- blackboost(Bodyfat_percent ~ Waist_cm + Weight_kg, data=data)

cvf <- cv(model.weights(fit), type="kfold")
cvm <- cvrisk(fit, folds=cvf, grid=1:100)
plot(cvm)
set.seed(1234567890)
train_idx <- sample(nrow(data), floor(nrow(data) * (2 / 3)))
train <- data[train_idx,]
test <- data[-train_idx,]

fit <- blackboost(Bodyfat_percent ~ Waist_cm + Weight_kg, data=train,
                  control=boost_control(mstop=mstop(cvm)))
test_error <- sum((predict(fit, test) - test$Bodyfat_percent)^2)
train_error <- sum((predict(fit, train) - train$Bodyfat_percent)^2)

cat(paste("Test Error:", test_error, "\n"))
cat(paste("Train Error:", train_error, "\n"))
```

Code for Assignment 4a

```r
library(mboost)
library(randomForest)
library(ggplot2)
library(reshape2)

data <- read.csv2("../data/spambase.csv")
data$Spam <- as.factor(data$Spam)

set.seed(1234567890)
train_idx <- sample(nrow(data), floor(nrow(data) * (2 / 3)))
train <- data[train_idx,]
test <- data[-train_idx,]
tree_counts <- seq(10, 100, by=10)
test_errors <- rep(0, length(tree_counts))
train_errors <- rep(0, length(tree_counts))

for (i in 1:length(tree_counts)) {
    fit <- blackboost(Spam ~ ., data=train, family=AdaExp(),
                      control=boost_control(mstop=tree_counts[i]))
    test_error <- 1 - (sum(predict(fit, test, type="class") == test$Spam) / nrow(test))
    train_error <- 1 - (sum(predict(fit, train, type="class") == train$Spam) / nrow(train))
    test_errors[i] <- test_error
    train_errors[i] <- train_error
}
plot_data <- data.frame(Trees=tree_counts, test=test_errors, train=train_errors)
plot_data <- melt(plot_data, id="Trees", value.name="Error", variable.name="Data")
```

```r
ggplot(plot_data) +
    xlab("Number of classification Trees") +
    ylab("Misclassification Rate") +
    geom_line(aes(x=Trees, y=Error, color=Data)) +
    scale_x_discrete(limits=tree_counts)
test_errors <- rep(0, length(tree_counts))
train_errors <- rep(0, length(tree_counts))

for (i in 1:length(tree_counts)) {
    fit <- randomForest(Spam ~ ., data=train, ntree=tree_counts[i])
    test_error <- 1 - (sum(predict(fit, test, type="class") == test$Spam) / nrow(test))
    train_error <- 1 - (sum(predict(fit, train, type="class") == train$Spam) / nrow(train))
    test_errors[i] <- test_error
    train_errors[i] <- train_error
}
plot_data <- data.frame(Trees=tree_counts, test=test_errors, train=train_errors)
plot_data <- melt(plot_data, id="Trees", value.name="Error", variable.name="Data")

ggplot(plot_data) +
    xlab("Number of classification Trees") +
    ylab("Misclassification Rate") +
    geom_line(aes(x=Trees, y=Error, color=Data)) +
    scale_x_discrete(limits=tree_counts)
```