Statistical_Learning_Exercise_4

2022-05-23

Exercise 1

```
set.seed(123)

y <- rnorm(10,0,3)

mean(y)
```

```
## [1] 0.2238769
```

```
library(dplyr)

bootstrap <- function(data,B,fun){
  samples <- replicate(B, sample(data, length(data), replace = TRUE))
  summary(apply(samples,2,fun))
}
bootstrap(y,1000,mean)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.0375 -0.4393  0.1470  0.1870  0.7802  2.8301
```

The mean estimate from bootstrapping is closer to zero than the mean estimate directly from the sample. Bootstrapping decreases the difference between the estimate and the true mean of the data generating process, which is 0.\ The value that the bootstrap estimate would have if not Monte Carlo sampling would be used to approximate the estimate, but if the estimate based on the true bootstrap distribution would be determined is the least squares estimate.

Exercise 2

We assume the following data generating process

$$Y = X + \epsilon,$$

where $X \sim N(0,1)$ and $\epsilon \sim N(0,1)$.

In addition, 20 covariates Z_1,...,Z_20 are given with

$$Z_i \sim \sqrt{0.9}X + \epsilon_{Z_i},$$

where $\epsilon_{Z_i} \sim N(0, 0.1)$.

Thereof, we draw a training data with 30 observations and test data with 10,000 observations.

```r
X_train <- rnorm(30)
epsilon_train <- rnorm(30)
Y_train <- X_train + epsilon_train

covariates <- function(Z){
    Z <- numeric(20)
    X <-  rnorm(20)
    for(i in 1:20){
    Z[i] <- sqrt(0.9) * X[i] + rnorm(1, 0, sqrt(0.1))
    }
    return(Z)
}

Z_train <- as.data.frame(t(replicate(30, covariates(Z))))
train <- as.data.frame(cbind(Y_train, X_train, Z_train))
colnames(train) <- c("Y", "X", paste("Z_",1:20))

X_test <- rnorm(10000)
epsilon_test <- rnorm(10000)
Y_test <- X_test + epsilon_test
Z_test <- as.data.frame(t(replicate(10000, covariates(Z))))
test <- as.data.frame(cbind(Y_test, X_test, Z_test))
colnames(test) <- c("Y", "X", paste("Z_",1:20))
```

Then we sample 100 bootstrap samples of size 30 from the training data by drawing with replacement and estimate a regression tree, the null model with predicted value equal to the observed empirical mean of Y and the linear model with X and all the Zs as regressors.

```r
bootstrap_trees <- list() # Creating empty vectors for the results
bootstrap_null <- list()
bootstrap_lm <- list()
bootstrap_AIC <- list()
predict_trees <- list()
predict_null <- list()
predict_lm <- list()
predict_AIC <- list()


for (i in 1:100){ #for 100 samples...
    obs <- sample(1:30, replace = TRUE) #...draw 30 samples with replacement
    bootstrap_trees[[i]] <- rpart(Y ~ ., data = train[obs,],
                                  method = "anova", control = list(cp = 0.0001))
    bootstrap_null[[i]] <- lm(Y ~ 1, data = train[obs,])
    bootstrap_lm[[i]] <- lm(Y ~ ., data = train[obs,])
    bootstrap_AIC[[i]] <- stepAIC(bootstrap_null[[i]], direction = "both",
                                  scope = list(upper = bootstrap_lm[[i]],
                                  lower = bootstrap_null[[i]], k = 2, trace = FALSE))
    predict_trees[[i]] <- predict(bootstrap_trees[[i]], newdata = test)
    predict_null[[i]] <- predict(bootstrap_null[[i]], newdata = test)
    predict_lm[[i]] <- predict(bootstrap_lm[[i]], newdata = test)
    predict_AIC[[i]] <- predict(bootstrap_AIC[[i]], newdata = test)
}
```

In a next step, we calculate the averages for each of the predicted values of the test data set.

```
matrix_averages <- matrix(nrow = 10000, ncol = 100) #we will unlist the predictions
#for the test data set for each bootstrap (n=100) and put them into a matrix

averages <- function(matrix){
                matrix_tree <- matrix #create empty matrices for the unlisted predictions
                matrix_null <- matrix
                matrix_lm <- matrix
                matrix_AIC <- matrix
            for (i in 1:100){
                matrix_tree[,i] <- as.vector(unlist(predict_trees[[i]]))
                matrix_null[,i] <- as.vector(unlist(predict_null[[i]]))
                matrix_lm[,i] <- as.vector(unlist(predict_lm[[i]]))
                matrix_AIC[,i] <- as.vector(unlist(predict_AIC[[i]]))
            }
                average_tree <- c() #create empty vectors for the averages
                average_null <- c()
                average_lm <- c()
                average_AIC <- c()
            for (i in 1:10000){
                average_tree[i] <- mean(matrix_tree[i,])#mean prediction of
                                                        #item 1 of test data set
                average_null[i] <- mean(matrix_null[i,])
                average_lm[i] <- mean(matrix_lm[i,])
                average_AIC[i] <- mean(matrix_AIC[i,])
            }
            means_compared <- cbind(test[,1], average_tree, average_null, average_lm, average_AIC)
            colnames(means_compared) <- c("Y", "average_tree", "average_null",
                                          "average_lm", "average_AIC")
            return(means_compared)
}

means_compared <- averages(matrix_averages)
head(means_compared)
```

```
##               Y average_tree average_null average_lm average_AIC
## [1,]  0.0598123   0.04080666    0.2364895  -91.41455  -0.5455142
## [2,] -0.9973311   0.06256425    0.2364895  -51.17295   0.5053492
## [3,]  0.3864299   0.05997415    0.2364895   11.31671  -0.5138846
## [4,]  1.3361844  -0.09885110    0.2364895  -39.22939  -0.6238672
## [5,] -0.7791384   0.18562612    0.2364895  -23.13971   1.1192581
## [6,] -2.0482334   0.02242296    0.2364895   26.84165  -0.5091689
```

Then we determine the mean squared error of the four bagged model estimators on the test sample of size 10,000.

```
MSE <- function(data){
    error_tree <- 1/10000 * sum((data[, "Y"] - data[, "average_tree"])^2)
    error_null <- 1/10000 * sum((data[, "Y"] - data[, "average_null"])^2)
    error_lm <- 1/10000 * sum((data[, "Y"] - data[, "average_lm"])^2)
    error_AIC <- 1/10000 * sum((data[, "Y"] - data[, "average_AIC"])^2)
    errors <- cbind(error_tree, error_null, error_lm, error_AIC)
    colnames(errors) <- c("MSE_tree", "MSE_null", "MSE_lm", "MSE_AIC")
    return(errors)
```

```
}

MSE(means_compared)
```

```
##      MSE_tree MSE_null   MSE_lm  MSE_AIC
## [1,] 1.913061 2.026921 2081.796 2.759401
```

Exercise 3

We use a random forest to fit a predictive model for the probability of survival to hospital discharge of ICU patients.

```
set.seed(544)
ICU <- aplore3::icu
names(ICU)
```

```
##  [1] "id"     "sta"    "age"    "gender" "race"   "ser"    "can"    "crn"
##  [9] "inf"    "cpr"    "sys"    "hra"    "pre"    "type"   "fra"    "po2"
## [17] "ph"     "pco"    "bic"    "cre"    "loc"
```

```
ICU <- ICU[,-c(1)] #drop variable id
head(ICU) #our dependent variable is "sta" with values "died" and "lived"
```

```
##     sta age gender  race      ser can crn inf cpr sys hra pre      type fra
## 1  Died  87 Female White Surgical  No  No Yes  No  80  96  No Emergency Yes
## 2 Lived  27 Female White  Medical  No  No Yes  No 142  88  No Emergency  No
## 3 Lived  59   Male White  Medical  No  No  No  No 112  80 Yes Emergency  No
## 4 Lived  77   Male White Surgical  No  No  No  No 100  70  No  Elective  No
## 5  Died  76 Female White Surgical  No  No Yes  No 128  90 Yes Emergency  No
## 6 Lived  54   Male White  Medical  No  No Yes  No 142 103  No Emergency Yes
##     po2      ph   pco   bic    cre     loc
## 1 <= 60  < 7.25  > 45 >= 18 <= 2.0 Nothing
## 2  > 60 >= 7.25 <= 45 >= 18 <= 2.0 Nothing
## 3  > 60 >= 7.25 <= 45 >= 18 <= 2.0 Nothing
## 4  > 60 >= 7.25 <= 45 >= 18 <= 2.0 Nothing
## 5  > 60 >= 7.25 <= 45 >= 18 <= 2.0 Nothing
## 6  > 60 >= 7.25 <= 45 >= 18 <= 2.0 Nothing
```

```
rf <- randomForest(sta ~ ., data = ICU, importance = TRUE) #ntree is set to 500 as default
#for classification, #mtry is equal to sqrt(p) (p = number of variables) as a default

rf
```

```
##
## Call:
##  randomForest(formula = sta ~ ., data = ICU, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
```

```
##         OOB estimate of  error rate: 16.5%
## Confusion matrix:
##       Lived Died class.error
## Lived   153    7     0.04375
## Died     26   14     0.65000
```

We have an error rate (OOB) of 16%. In order to see whether the default choice of 500 bootstraps iterations is enough, we look at the error rates for each iteration.

```
head(rf$err.rate)
```

```
##            OOB     Lived      Died
## [1,] 0.2500000 0.1428571 0.6250000
## [2,] 0.2644628 0.1304348 0.6896552
## [3,] 0.2451613 0.1138211 0.7500000
## [4,] 0.2415730 0.1134752 0.7297297
## [5,] 0.2486486 0.1283784 0.7297297
## [6,] 0.2328042 0.1066667 0.7179487
```

```
error_rates <- data.frame(
    Number_Trees = rep(1:500, times = 3),
    Error_Type = rep(c("OOB", "Lived", "Died"), each = 500),
    Error = c(rf$err.rate[, "OOB"], rf$err.rate[, "Lived"], rf$err.rate[, "Died"])
)

ggplot(data = error_rates, aes(x = Number_Trees, y = Error)) +
    geom_line(aes(color = Error_Type))
```

From the plot, we see that already at about 50 iterations, the OBB error rate and the misclassification rate for "Lived" seem to stabilise. The misclassification rate for "died" still gets a bit lower with more than 400 iterations. Whereas misclassifications for "lived" are very low, those for "died" are very high. The OOB lies in the middle.

Thus, the default of 500 iterations would not be necessary regarding improvement of OOB and misclassification rate for "lived". However, such a high number of iterations could still improve the "died" classification rate a bit. We also tried out 1000 iterations, which does not improve the error rates.

Next, we look at the influence of varying the hyperparameter m on the out-of-bag error obtained and select a suitable value.

```
errors_mtry <- c()

for (i in 1:19){
    rf_2 <- randomForest(sta ~ ., data = ICU, mtry = i)
    errors_mtry[i] <- rf_2$err.rate[nrow(rf_2$err.rate), 1] #OBB error after
    #500 iterations (500th row, first column)
}
errors_mtry
```

```
##  [1] 0.200 0.165 0.165 0.150 0.160 0.155 0.160 0.180 0.155 0.155 0.175 0.175
## [13] 0.165 0.170 0.170 0.170 0.170 0.155 0.175
```

```
which.min(errors_mtry)
```

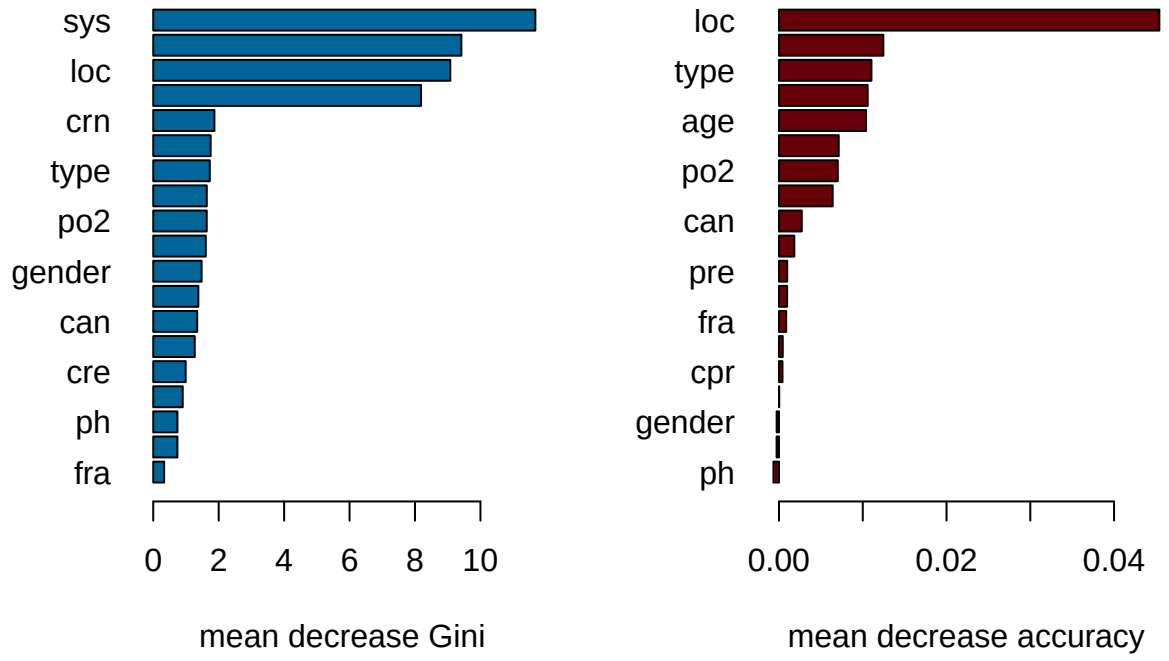
```
## [1] 4
```

For $m = 4$, we get the lowest error rate. The default value was also 4 and we see here that this is the best choice. Last, we inspect the variable importance measures i.e. compare the mean decrease Gini and the mean decrease accuracy measures.

```
options(digits = 2)
head(importance(rf, scale = FALSE))
```

```
##            Lived     Died MeanDecreaseAccuracy MeanDecreaseGini
## age     0.010964  0.00832              0.01039             9.42
## gender -0.000220 -0.00112             -0.00028             1.48
## race   -0.000078 -0.00150             -0.00029             0.74
## ser     0.013135  0.00045              0.01059             1.76
## can     0.004272 -0.00300              0.00272             1.34
## crn     0.007899  0.00417              0.00713             1.87
```

```
MDG <- importance(rf)[, "MeanDecreaseGini"]
MDA <- importance(rf, scale = FALSE)[, "MeanDecreaseAccuracy"]
par(mfrow = c(1, 2))
barplot(MDG[order(MDG)], horiz = TRUE, las = 1,
  xlab = "mean decrease Gini", col = "#006699")
barplot(MDA[order(MDA)], horiz = TRUE, las = 1,
  xlab = "mean decrease accuracy", col = "#660009")
```



```r
MDG[order(MDG)[15:19]] #5 variables with highest MDG
```

```
##  crn  hra  loc  age  sys
##  1.9  8.2  9.1  9.4 11.7
```

```r
head(ICU[, c("crn", "hra", "loc", "age", "sys")])
```

```
##   crn hra     loc age sys
## 1  No  96 Nothing  87  80
## 2  No  88 Nothing  27 142
## 3  No  80 Nothing  59 112
## 4  No  70 Nothing  77 100
## 5  No  90 Nothing  76 128
## 6  No 103 Nothing  54 142
```

```r
class(ICU$crn) #history of chronic renal failure (yes or no)
```

```
## [1] "factor"
```

```r
class(ICU$hra) #hear rate at ICU admission
```

```
## [1] "integer"
```

```r
class(ICU$loc) #Level of consciousness at ICU admission (3 levels)
```

```
## [1] "factor"
```

```r
class(ICU$age) #age in years
```

```
## [1] "integer"
```

```r
class(ICU$sys) #Systolic blood pressure at ICU admission
```

```
## [1] "integer"
```

```r
MDA[order(MDA)[15:19]]
```

```
##   age   ser  type   sys   loc
## 0.010 0.011 0.011 0.012 0.045
```

```r
head(ICU[, c("age", "ser", "type", "sys", "loc")])
```

```
##   age      ser      type sys     loc
## 1  87 Surgical Emergency  80 Nothing
## 2  27  Medical Emergency 142 Nothing
## 3  59  Medical Emergency 112 Nothing
## 4  77 Surgical  Elective 100 Nothing
## 5  76 Surgical Emergency 128 Nothing
## 6  54  Medical Emergency 142 Nothing
```

```
class(ICU$ser) #Service at ICU admission (1: Medical, 2: Surgical)
```

```
## [1] "factor"
```

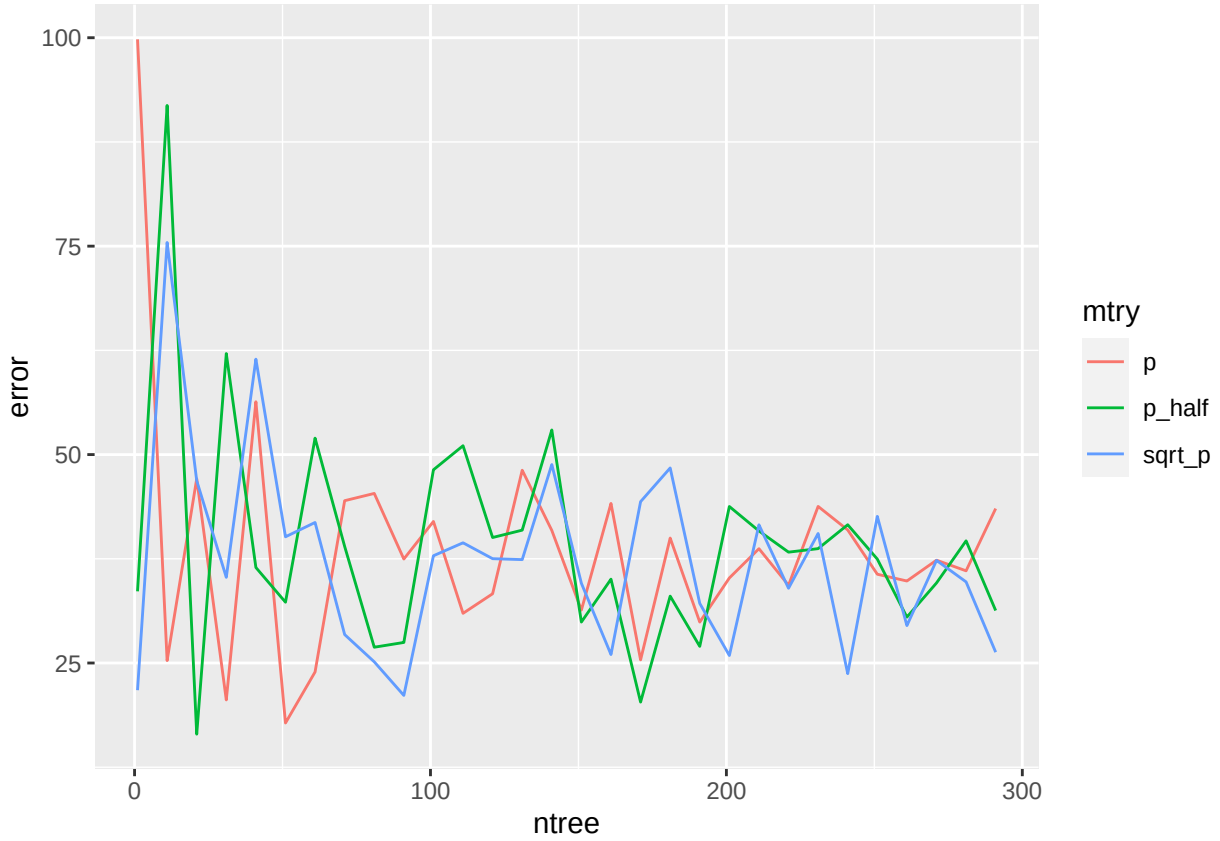
```
class(ICU$type)#Type of admission (1: Elective, 2: Emergency)
```

```
## [1] "factor"
```

For the most important variables regarding MDG, we have both integer and factor variables which are relatively important. However, we see that integer variables show the highest importance. This is in line with what we learned in the course – MDG might give worse (biased – if the predictor values vary in their measurement scales) results for binary/categorical variables compared to continuous ones. In general, the MDG is not as stable and reliable when it comes to predictor variables varying in their scale of measurement or their number of categories. Regarding MDA, the (by far) most important variable is a factor variable.

Exercise 4

```
library(MASS)
library(randomForest)
set.seed(1)
train.ind <- sample(seq_len(nrow(Boston)),250)
train <- data.frame(Boston[train.ind,])
test <- data.frame(Boston[-train.ind,])
p <- ncol(Boston)-1 #number of predictors

ntree <- function(x,y,train,test){
  rf <- randomForest(medv ~ ., data = train, ntree = x, mtry = y)
  rf_pred <- predict(rf, newdata = test[,-which(names(test)=="medv")])
  error <- sum(test$medv-rf_pred)^2/nrow(test)
  error
}
errors <- data.frame( ntree = seq(1,300,10),
                      p = unlist(lapply(seq(1,300,10),
                          function(x) ntree(x,p,train,test))),
                      p_half = unlist(lapply(seq(1,300,10),
                          function(x) ntree(x,p/2,train,test))),
                      sqrt_p = unlist(lapply(seq(1,300,10),
                          function(x) ntree(x,sqrt(p),train,test))))
library(ggplot2)
library(reshape2)
errors_plot <- melt(errors, id.vars = "ntree")
ggplot(errors_plot, aes(x=ntree, y=value, group=variable, color=variable)) +
       geom_line() + labs(x="ntree",y="error", color="mtry")
```

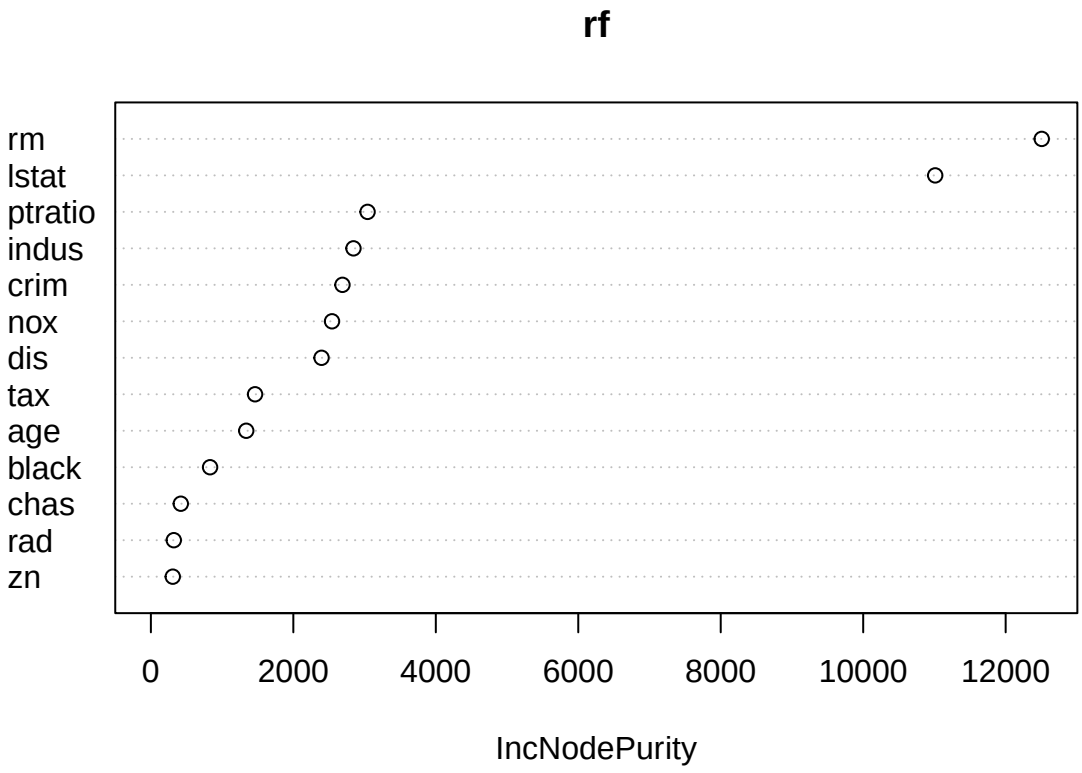


`ntree` is the number of trees to grow. This number should not be set to too small in order to ensure that every input row gets predicted at least a few times. `mtry` is the number of variables randomly sampled as candidates at each split. In the plot we observe a very high error rate in all cases for low `ntree` values. The errors stabilize at values of around 100. \ We chose a suitable model and asses variable importance. First we plot a variable importance plot which represents the mean decrease in node impurity. Then we can also plot partial response plots.

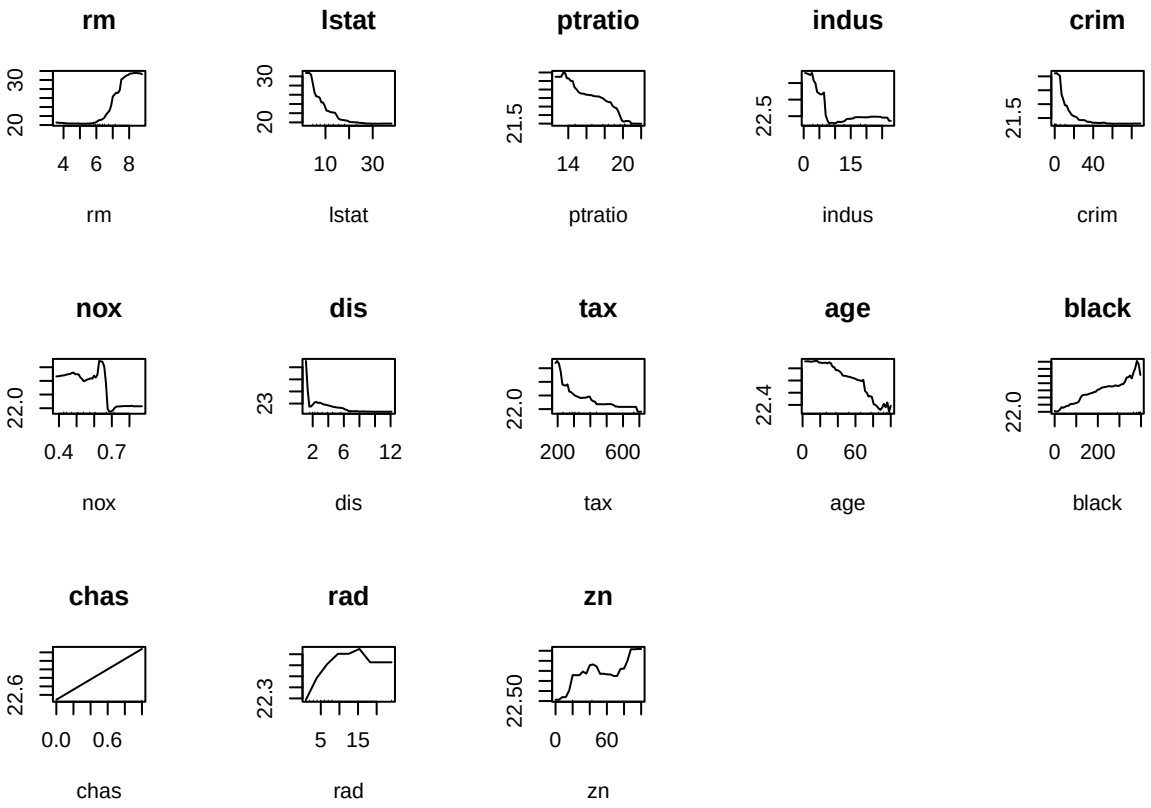
```
rf <- randomForest(medv ~ ., data = Boston, mtry = sqrt(p), ntree = 100)
print(rf)
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data = Boston, mtry = sqrt(p),      ntree = 100)
##                Type of random forest: regression
##                      Number of trees: 100
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 10
##                     % Var explained: 88
```

```
varImpPlot(rf)
```




```
importanceOrder=order(-rf$importance)
names=rownames(rf$importance)[importanceOrder][1:13]
par(mfrow = c(3,5))
for (name in names) partialPlot(rf, Boston, eval(name), main=name, xlab=name)
```



Exercise 5

In the following, we analyze the performance of the variable importance measures for random forests using a simulation study.

- Assume that there are four predictor variables which have the following distributions:

$$X_1 \sim \mathcal{N}(0,1), X_2 \sim U(0,1),$$

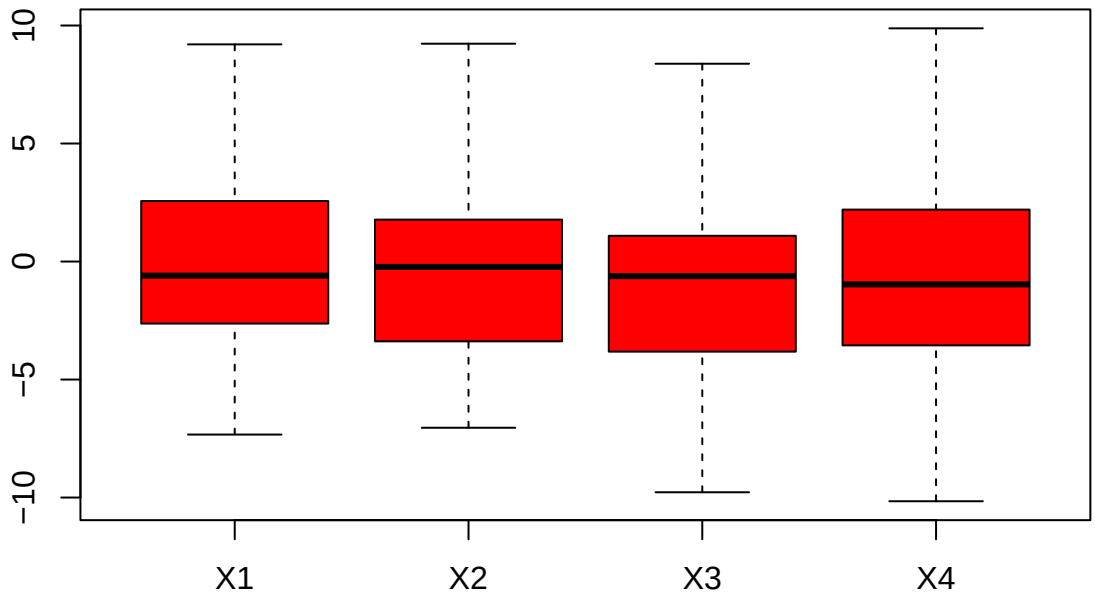
$$X_3 \sim M(1,(0.5,0.5)), X_4 \sim M(1,(0.2,0.2,0.2,0.2,0.2))$$

  This means we have two continuous variables which follow either a standard normal or a standard uniform distribution $U(0,1)$ and two categorical variables with balanced categories with either 2 or 5 categories, i.e., $M(N,\pi)$ is the multinomial distribution for N trials and success probability vector π.

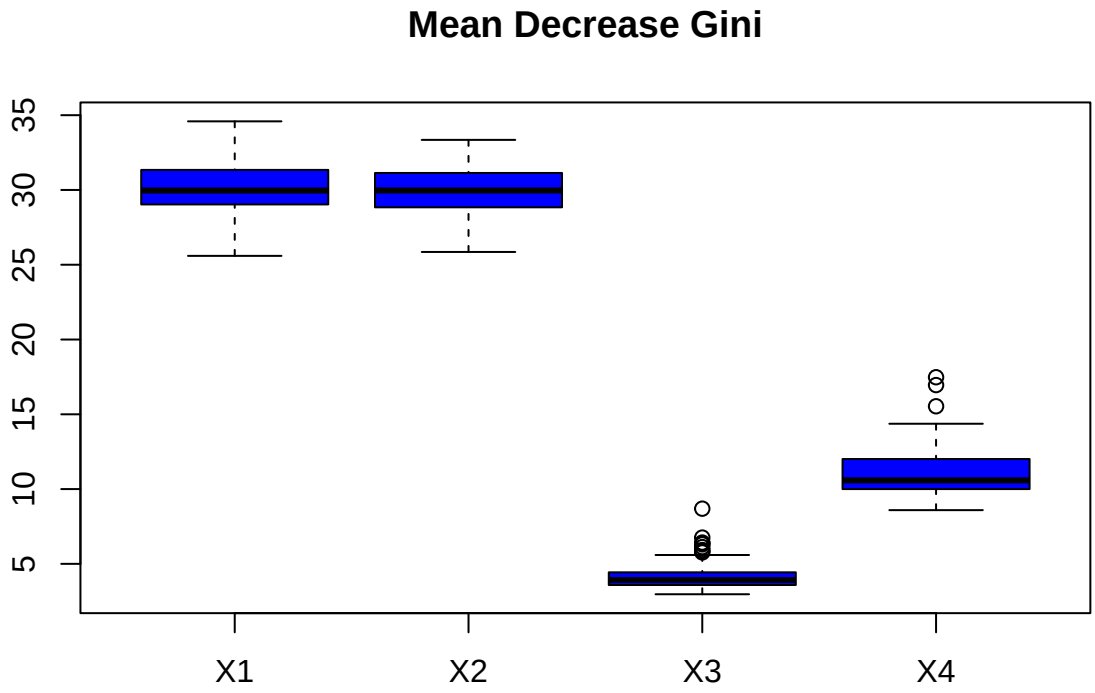
- The dependent variable y is assumed to be a binary categorical variable with equal-sized classes.
- Set the sample size to $N = 200$.
- Generate 100 datasets for each setting and fit a random forest to each dataset and determine the mean decrease Gini and mean decrease accuracy values for each of the predictor variables. Suitably visualize the results and interpret them.

```
library("randomForest")
set.seed(1234)
```

```r
N <- 200
rep <- 100
pred_types <- 4
decrease_acc_gini <- matrix(nrow=rep,ncol=8)

for(j in 1:rep){
    X1 <- rnorm(N)
    X2 <- runif(N)
    X3 <- rmultinom(N, size = 1, prob = c(0.5,0.5))
    X3_vec <- numeric(N)
    for(i in 1:200){
      X3_vec[i] <- which(X3[,i]==1)
    }
    X4 <- rmultinom(N, size = 1, prob = c(0.2, 0.2, 0.2, 0.2, 0.2))
    X4_vec <- numeric(N)
    for(i in 1:200){
      X4_vec[i] <- which(X4[,i]==1)
    }
    X <- cbind(X1,X2,X3_vec,X4_vec)
    Y <- sample(x=c(-1,1),size=N,prob=c(0.5,0.5),replace=TRUE)
    data <- as.data.frame(cbind(Y,X))
    rf <- randomForest(Y ~ .,data=data,importance=TRUE)
    #note that the documentation of randomForest says that importance type is
    #either 1 or 2, specifying the type of importance measure (1=mean decrease in
    #accuracy, 2=mean decrease in node impurity). For classification,
    #the node impurity is measured by the Gini index. So we do not need to
    #change any defaults in varImpPlot
    decrease_acc_gini[j,1:4] <- importance(rf)[,1]
    decrease_acc_gini[j,5:8] <- importance(rf)[,2]

}

boxplot(decrease_acc_gini[,1],decrease_acc_gini[,2],decrease_acc_gini[,3],decrease_acc_gini[,4],xaxt="n
axis(1,                         # Define x-axis manually
     at = 1:4,
     labels = c("X1","X2","X3","X4"))
```




```
boxplot(decrease_acc_gini[,5],decrease_acc_gini[,6],decrease_acc_gini[,7],decrease_acc_gini[,8],xaxt="n
axis(1,                          # Define x-axis manually
     at = 1:4,
     labels = c("X1","X2","X3","X4"))
```



Whereas the boxplots for the Mean Decrease Accuracy look rather homogeneous, the Mean Decrease Gini has a lot less variance and clearly favors two of the explanatory variables. This is due to the fact that random forests also use the oob samples to construct a different variable-importance measure, apparently to measure the prediction strength of each variable. The decrease in accuracy as a result of this permuting is averaged over all trees, and is used as a measure of the importance of a specific variable in the random forest. The randomization effectively voids the effect of a variable.

Exercise 6

In the following the influence of the ratio of relevant to irrelevant predictor variables on the performance is assessed for random forests with $m = \sqrt{p}$.

- A binary dependent variable is generated by

$$P(Y = 1|X) = q + (1 - 2q) \cdot 1\Big(\sum_{j=1}^{J} X_j > J/2\Big),$$

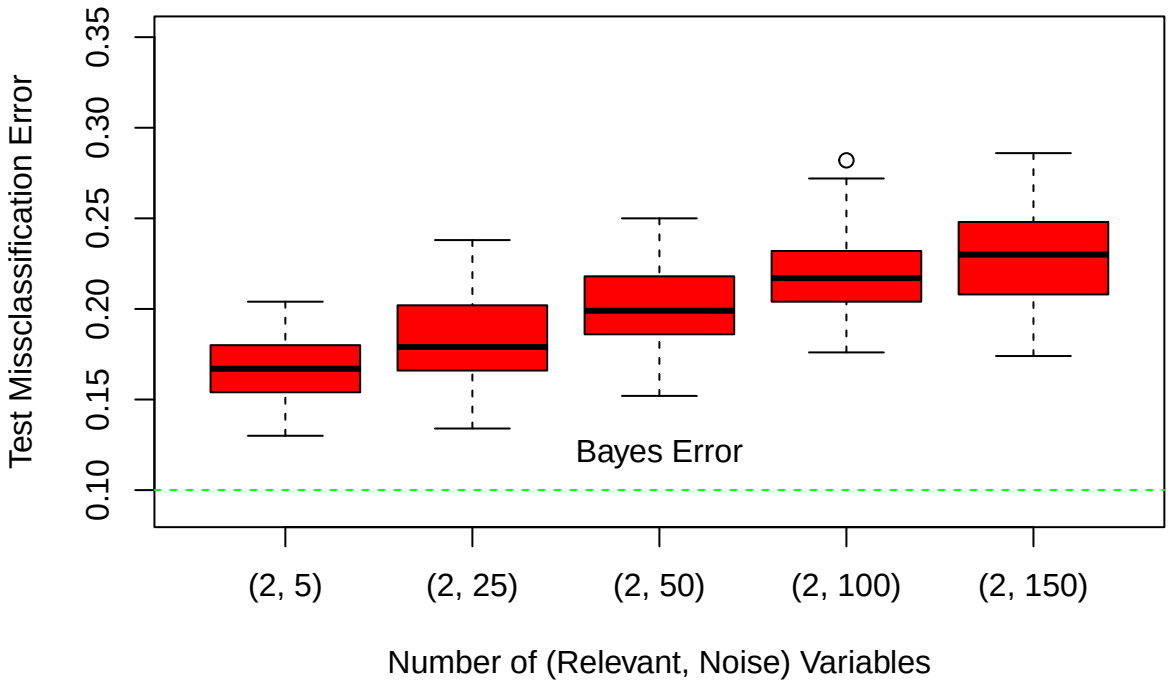
  where $X \sim U[0, 1]^p, 0 \leq q \leq 1/2$.

- Two predictor variables are assumed to be informative, i.e., $J = 2$. The number of noise predictor variables is varied between $\{5, 25, 50, 100, 150\}$. The value for q is set to 0.1 to obtain a Bayes error rate of 0.1.

- The training sample size is $N = 300$ and the test sample size is 500.

- Fit random forests with $m = \sqrt{p}$ and visualize the test misclassification rates obtained for 50 repetitions. Interpret the results.

```r
library("randomForest")
library("uniformly")
set.seed(1234)

q <- 0.1
n_informative <- 2
n_uninformative <- c(5, 25, 50, 100, 150)
rep <- 50
misscl <- matrix(nrow=rep,ncol=length(n_uninformative))

for(i in 1:rep){
  j <- 1
  for(d in n_uninformative){
    x_train <- runif_in_cube(300, d+2, O = rep(0.5, d+2), r = 0.5)
    indicator <- rep(0,300)
    index <- which(rowSums(x_train[,1:2])>1)
    indicator[index] <- 1
    P_Y <- q+(1-2*q)*indicator
    Y <- rbinom(300,1,P_Y)
    train_data <- as.data.frame(cbind(Y,x_train))
    train_data$Y <- as.factor(train_data$Y)
    x_test <- runif_in_cube(500, d+2, O = rep(0.5, d+2), r = 0.5)
    indicator_test <- rep(0,500)
    index_test <- which(rowSums(x_test[,1:2])>1)
    indicator_test[index_test] <- 1
    P_y_test <- q+(1-2*q)*indicator_test
    y_test <- rbinom(500,1,P_y_test)
    test_data <- as.data.frame(cbind(y_test,x_test))
    test_data$y_test <- as.factor(test_data$y_test)
    #nothing to be specified for mtry since default is floor(sqrt(ncol(x)))
    class <- randomForest(Y ~ .,data=train_data)
    predicted <- predict(class,test_data)
    misscl[i,j] <- mean(y_test != predicted)
    j <- j+1
  }
}

boxplot(misscl[,1],misscl[,2],misscl[,3],misscl[,4],misscl[,5],col="red",ylab="Test Missclassification R
abline(a=0.1,b=0,col="green",lty="dashed")
text(x = 3, y = 0.12, # Coordinates
     label = "Bayes Error")
```




```
misscl_ratios <- rbind(c(2/5,2/25,2/50,2/100,2/250),colMeans(misscl))
diff(misscl_ratios[1,])/misscl_ratios[1,][-length(misscl_ratios[1,])]
```

```
## [1] -0.8 -0.5 -0.5 -0.6
```

```
diff(misscl_ratios[2,])/misscl_ratios[2,][-length(misscl_ratios[2,])]
```

```
## [1] 0.093 0.103 0.088 0.053
```

As the ratio of unimportant variables to important variables increases, the box of the missclassification error is shifted upwards for small $m = \sqrt{p}$. This is due to the fact that the signal to noise ratio is heavily affected. At each split the chance is small that the relevant variables will be selected. The hyper-geometric probability that a relevant variable will be selected at any split by a random forest tree is:

```
dhyper(1, 2, 5, floor(sqrt(7)), log = FALSE)
```

```
## [1] 0.48
```

```
dhyper(1, 2, 25, floor(sqrt(27)), log = FALSE)
```

```
## [1] 0.31
```

```r
dhyper(1, 2, 50, floor(sqrt(52)), log = FALSE)
```

```
## [1] 0.24
```

```r
dhyper(1, 2, 100, floor(sqrt(102)), log = FALSE)
```

```
## [1] 0.18
```

```r
dhyper(1, 2, 150, floor(sqrt(152)), log = FALSE)
```

```
## [1] 0.15
```