Lab 13: Tree-Based Methods

Donggyun Kim
27008257

4/24/2018

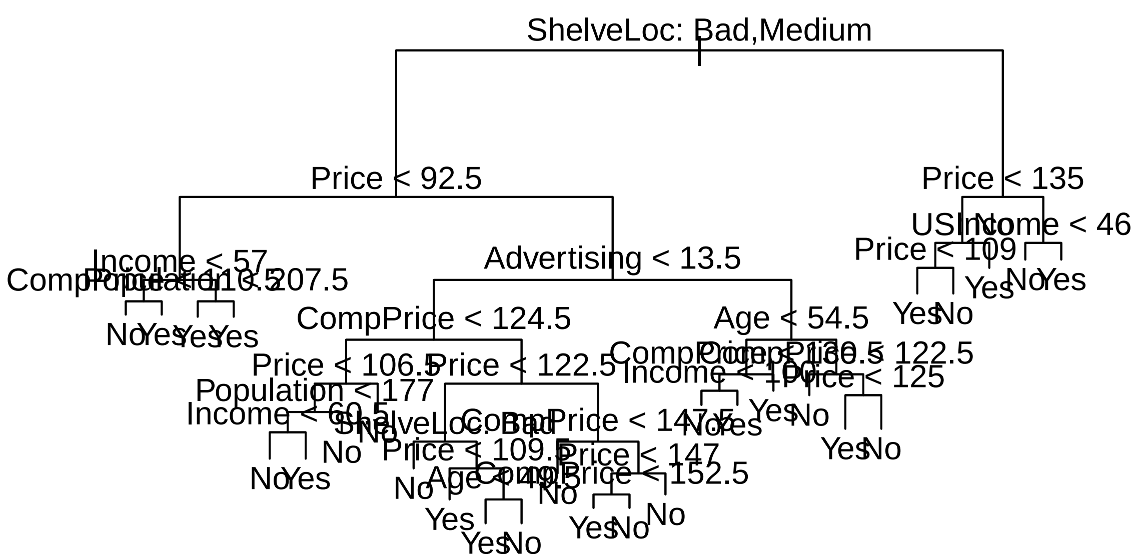
```r
library(ISLR)
attach(Carseats)
High <- ifelse(Sales <= 8, "No", "Yes")
carseats <- data.frame(Carseats, High)
```

Decision Trees

```r
library(tree)
tree_carseats <- tree(High ~ . -Sales, carseats)

summary(tree_carseats)
```

```
##
## Classification tree:
## tree(formula = High ~ . - Sales, data = carseats)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Income"      "CompPrice"   "Population"
## [6] "Advertising" "Age"         "US"
## Number of terminal nodes:  27
## Residual mean deviance:  0.4575 = 170.7 / 373
## Misclassification error rate: 0.09 = 36 / 400
```

```r
plot(tree_carseats)
text(tree_carseats, pretty = 0)
```




```r
tree_carseats
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##   1) root 400 541.500 No ( 0.59000 0.41000 )
##     2) ShelveLoc: Bad,Medium 315 390.600 No ( 0.68889 0.31111 )
##       4) Price < 92.5 46  56.530 Yes ( 0.30435 0.69565 )
##         8) Income < 57 10  12.220 No ( 0.70000 0.30000 )
##          16) CompPrice < 110.5 5   0.000 No ( 1.00000 0.00000 ) *
##          17) CompPrice > 110.5 5   6.730 Yes ( 0.40000 0.60000 ) *
##         9) Income > 57 36  35.470 Yes ( 0.19444 0.80556 )
##          18) Population < 207.5 16  21.170 Yes ( 0.37500 0.62500 ) *
##          19) Population > 207.5 20   7.941 Yes ( 0.05000 0.95000 ) *
##       5) Price > 92.5 269 299.800 No ( 0.75465 0.24535 )
##        10) Advertising < 13.5 224 213.200 No ( 0.81696 0.18304 )
##          20) CompPrice < 124.5 96  44.890 No ( 0.93750 0.06250 )
##            40) Price < 106.5 38  33.150 No ( 0.84211 0.15789 )
##              80) Population < 177 12  16.300 No ( 0.58333 0.41667 )
##               160) Income < 60.5 6   0.000 No ( 1.00000 0.00000 ) *
##               161) Income > 60.5 6   5.407 Yes ( 0.16667 0.83333 ) *
##              81) Population > 177 26   8.477 No ( 0.96154 0.03846 ) *
##            41) Price > 106.5 58   0.000 No ( 1.00000 0.00000 ) *
##          21) CompPrice > 124.5 128 150.200 No ( 0.72656 0.27344 )
##            42) Price < 122.5 51  70.680 Yes ( 0.49020 0.50980 )
##              84) ShelveLoc: Bad 11   6.702 No ( 0.90909 0.09091 ) *
##              85) ShelveLoc: Medium 40  52.930 Yes ( 0.37500 0.62500 )
##               170) Price < 109.5 16   7.481 Yes ( 0.06250 0.93750 ) *
##               171) Price > 109.5 24  32.600 No ( 0.58333 0.41667 )
##                 342) Age < 49.5 13  16.050 Yes ( 0.30769 0.69231 ) *
##                 343) Age > 49.5 11   6.702 No ( 0.90909 0.09091 ) *
##            43) Price > 122.5 77  55.540 No ( 0.88312 0.11688 )
##              86) CompPrice < 147.5 58  17.400 No ( 0.96552 0.03448 ) *
##              87) CompPrice > 147.5 19  25.010 No ( 0.63158 0.36842 )
##               174) Price < 147 12  16.300 Yes ( 0.41667 0.58333 )
##                 348) CompPrice < 152.5 7   5.742 Yes ( 0.14286 0.85714 ) *
##                 349) CompPrice > 152.5 5   5.004 No ( 0.80000 0.20000 ) *
##               175) Price > 147 7   0.000 No ( 1.00000 0.00000 ) *
##        11) Advertising > 13.5 45  61.830 Yes ( 0.44444 0.55556 )
##          22) Age < 54.5 25  25.020 Yes ( 0.20000 0.80000 )
##            44) CompPrice < 130.5 14  18.250 Yes ( 0.35714 0.64286 )
##              88) Income < 100 9  12.370 No ( 0.55556 0.44444 ) *
##              89) Income > 100 5   0.000 Yes ( 0.00000 1.00000 ) *
##            45) CompPrice > 130.5 11   0.000 Yes ( 0.00000 1.00000 ) *
##          23) Age > 54.5 20  22.490 No ( 0.75000 0.25000 )
##            46) CompPrice < 122.5 10   0.000 No ( 1.00000 0.00000 ) *
##            47) CompPrice > 122.5 10  13.860 No ( 0.50000 0.50000 )
##              94) Price < 125 5   0.000 Yes ( 0.00000 1.00000 ) *
##              95) Price > 125 5   0.000 No ( 1.00000 0.00000 ) *
##     3) ShelveLoc: Good 85  90.330 Yes ( 0.22353 0.77647 )
##       6) Price < 135 68  49.260 Yes ( 0.11765 0.88235 )
##        12) US: No 17  22.070 Yes ( 0.35294 0.64706 )
##          24) Price < 109 8   0.000 Yes ( 0.00000 1.00000 ) *
##          25) Price > 109 9  11.460 No ( 0.66667 0.33333 ) *
##        13) US: Yes 51  16.880 Yes ( 0.03922 0.96078 ) *
##       7) Price > 135 17  22.070 No ( 0.64706 0.35294 )
```

```
##        14) Income < 46 6   0.000 No ( 1.00000 0.00000 ) *
##        15) Income > 46 11  15.160 Yes ( 0.45455 0.54545 ) *
```

Random Forests

```r
library(randomForest)
set.seed(1991)
n <- nrow(carseats)
index <- sample(n, 0.8 * n)
training <- carseats[index, ]
test <- carseats[-index, ]
RF_training <- randomForest(High ~ . - Sales, training, importance = TRUE)
RF_predict <- predict(RF_training, newdata = test)
tbl <- table(test$High, RF_predict)
# test error rate
1 - sum(diag(tbl)) / sum(tbl)
```

```
## [1] 0.2125
```

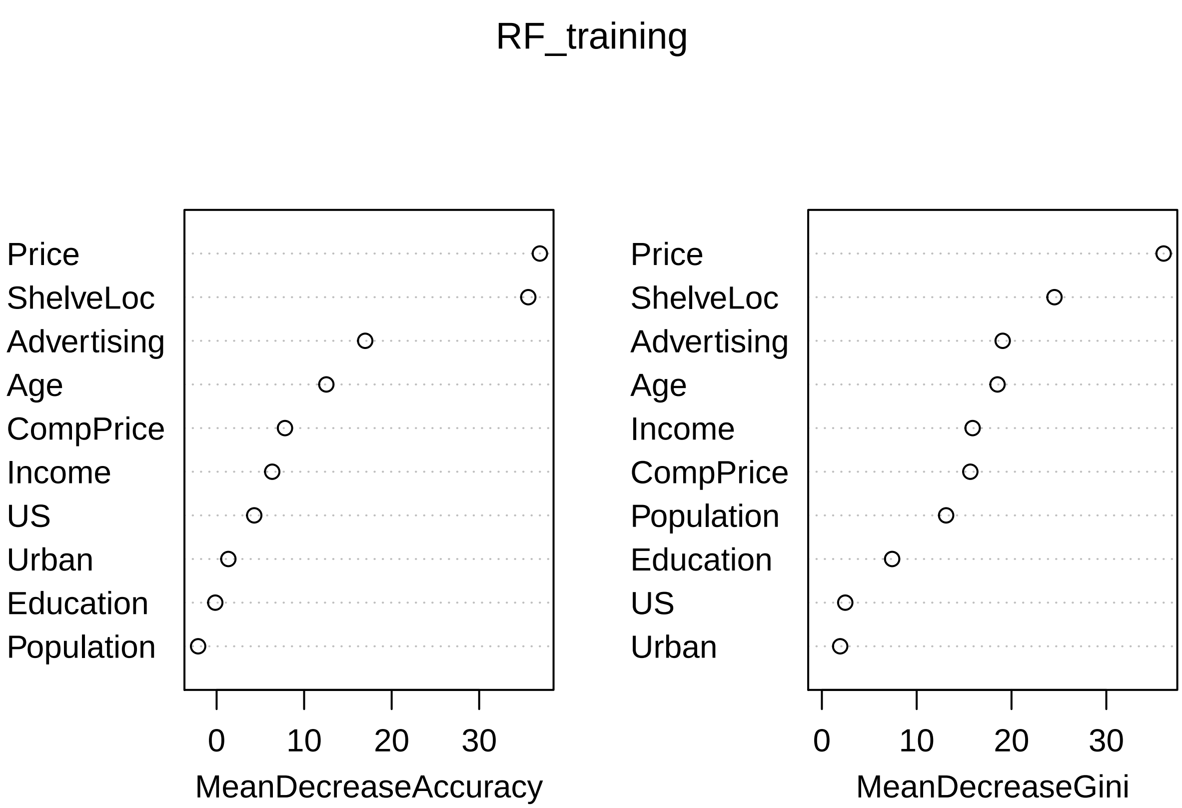
```r
# oob error rate
1 - sum(diag(RF_training$confusion)) / sum(RF_training$confusion)
```

```
## [1] 0.1822402
```

```r
importance(RF_training)
```

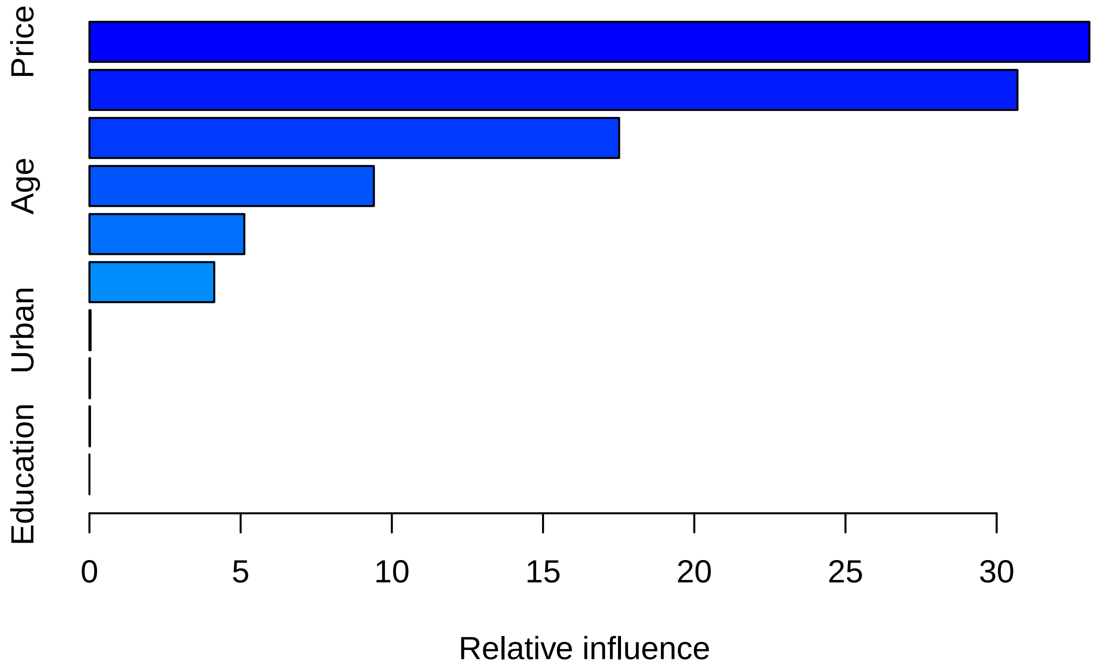
```
##                     No        Yes MeanDecreaseAccuracy MeanDecreaseGini
## CompPrice    7.3032375  4.0546146            7.8125775        15.655874
## Income       4.0001415  5.4058053            6.3534488        15.898380
## Advertising  8.3304240 16.0936016           16.9770839        19.070433
## Population  -3.3939716  0.7557087           -2.1127816        13.104987
## Price       30.1305130 27.7149086           36.9356272        36.045189
## ShelveLoc   27.9951836 29.7587251           35.6092447        24.531438
## Age          7.9805915 10.8930464           12.5376477        18.523661
## Education   -0.6947404  0.3735257           -0.1544433         7.410901
## Urban        0.3050675  1.9108170            1.3407577         1.931511
## US           0.8077455  4.1845106            4.2992776         2.465214
```

```r
varImpPlot(RF_training)
```



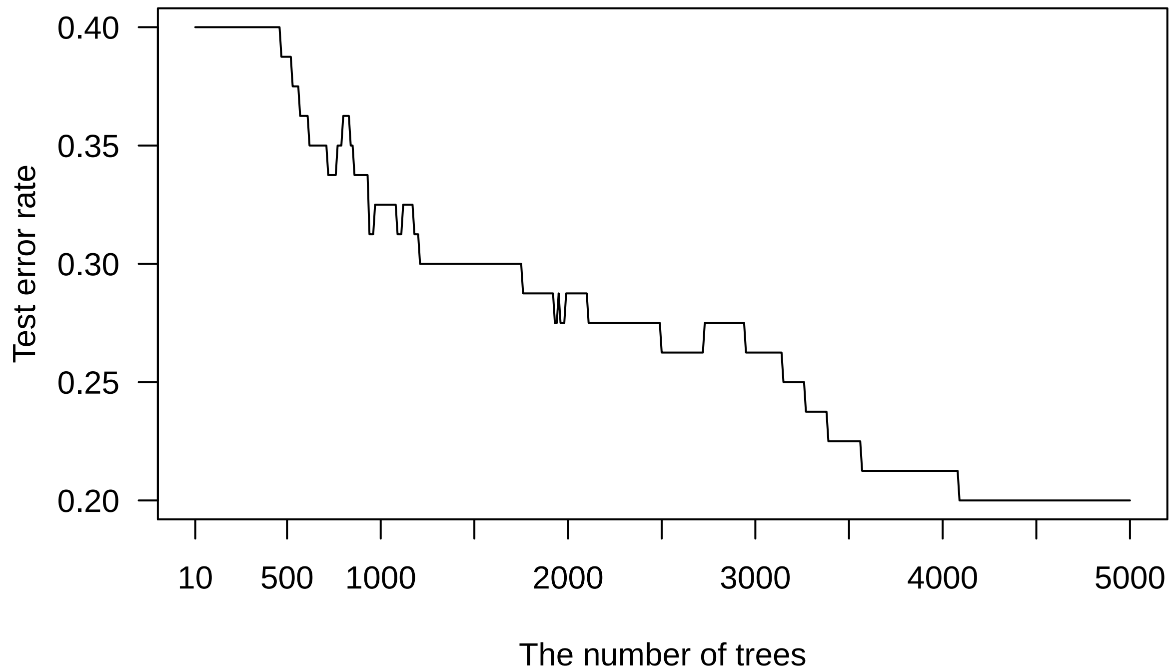
Boosted Trees

```
library(gbm)
B <- 5000
High <- ifelse(Sales <= 8, 0, 1)
carseats <- data.frame(Carseats, High)
training <- carseats[index, ]
test <- carseats[-index, ]
boosting_training <- gbm(High ~ . - Sales, data = training,
                         distribution = "bernoulli", n.trees = B)

summary(boosting_training)
```

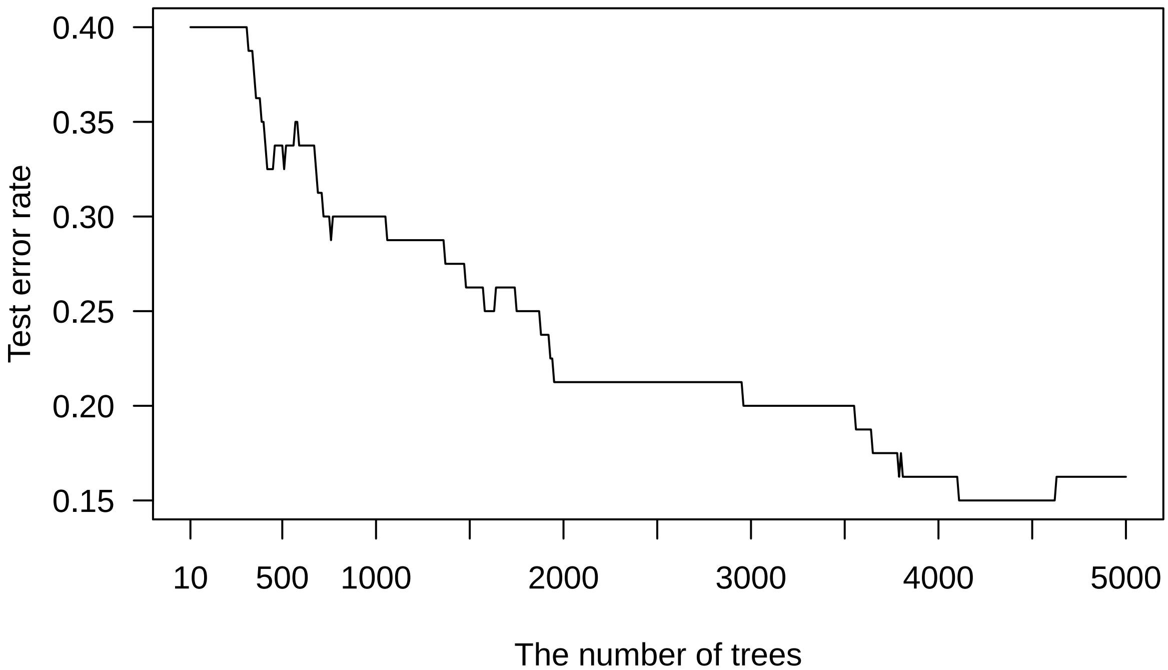



```
##                      var     rel.inf
## Price              Price 33.06618005
## ShelveLoc      ShelveLoc 30.68783059
## Advertising Advertising 17.51610373
## Age                  Age  9.40537181
## Income            Income  5.12220623
## CompPrice      CompPrice  4.12656668
## Urban              Urban  0.03793815
## US                    US  0.01928665
## Population    Population  0.01851611
## Education      Education  0.00000000
```

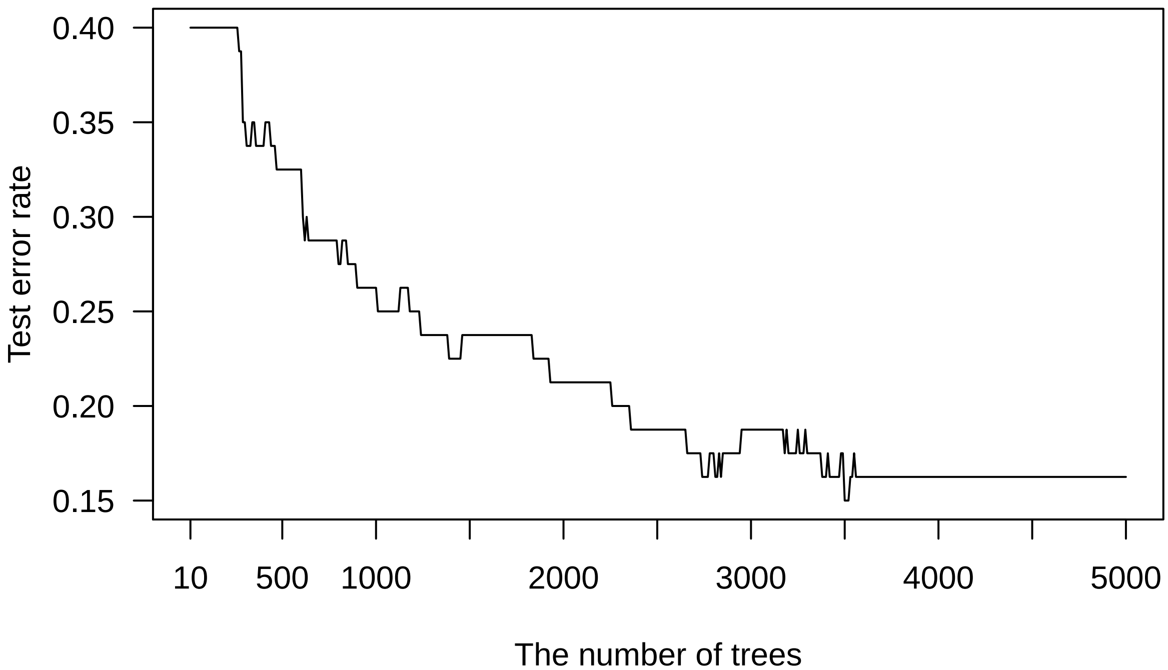
```
b <- seq(from = 10, to = 5000, by = 10)

boosting_predict <- predict(boosting_training, newdata = test,
                            n.trees = b,
                            type = "response")
yhat_boosting <- matrix(0, nrow = nrow(boosting_predict),
                        ncol = ncol(boosting_predict))

yhat_boosting[boosting_predict >= 0.5] <- 1

error_rate <- numeric(length(b))
for (i in seq_along(b)) {
  error_rate[i] <- 1 - sum(yhat_boosting[, i] == test$High) / nrow(yhat_boosting)
}

plot(b, error_rate, type = "l", ylab = "Test error rate",
     xlab = "The number of trees", las = 1, xaxt = "n")
axis(1, c(10, seq(from = 500, to = 5000, by = 500)))
```



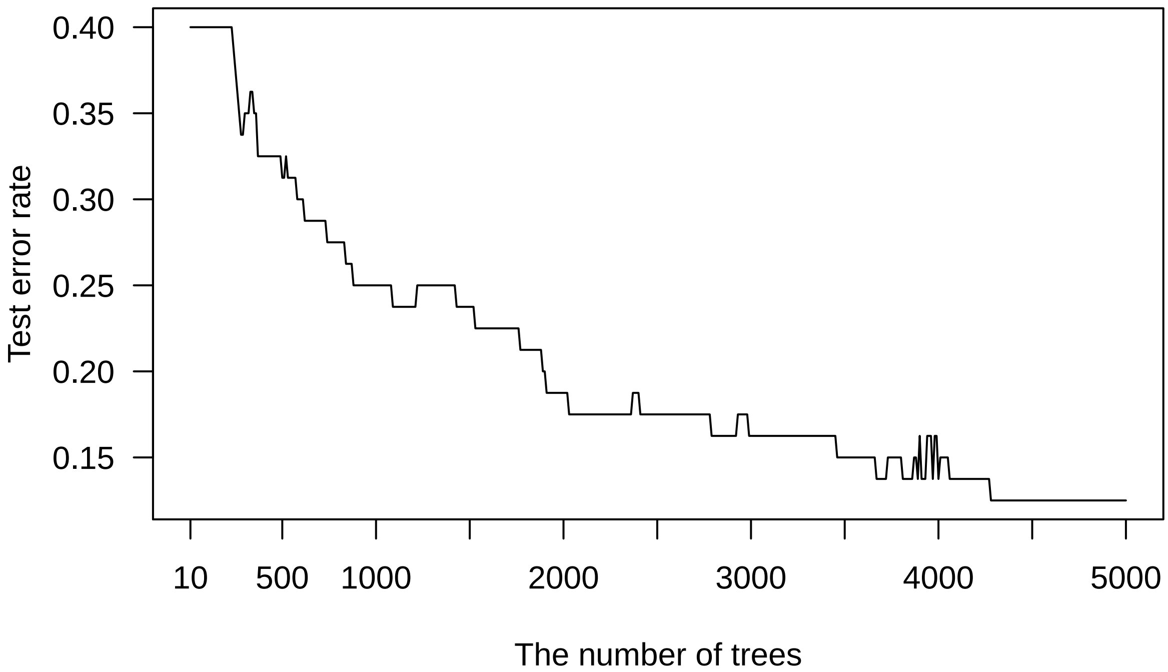

```r
for (j in 2:4) {
  boosting_training <- gbm(High ~ . - Sales, data = training,
                         distribution = "bernoulli", n.trees = B,
                         interaction.depth = j)

  boosting_predict <- predict(boosting_training, newdata = test,
                            n.trees = b,
                            type = "response")

  yhat_boosting <- matrix(0, nrow = nrow(boosting_predict),
                        ncol = ncol(boosting_predict))

  yhat_boosting[boosting_predict >= 0.5] <- 1

  error_rate <- numeric(length(b))

  for (i in seq_along(b)) {
    error_rate[i] <- 1 - sum(yhat_boosting[, i] == test$High) / nrow(yhat_boosting)
  }

  plot(b, error_rate, type = "l", ylab = "Test error rate",
       xlab = "The number of trees", las = 1, xaxt = "n")
  axis(1, c(10, seq(from = 500, to = 5000, by = 500)))
}
```



0.40
0.35
0.30
0.25
0.20
0.15
Test error rate
10
500
1000
2000
3000
4000
5000
The number of trees



0.40
0.35
0.30
0.25
0.20
0.15
Test error rate
10
500
1000
2000
3000
4000
5000
The number of trees


Test error rate
0.40
0.35
0.30
0.25
0.20
0.15
10
500
1000
2000
3000
4000
5000
The number of trees