TreeBasedMethods

Chase Enzweiler

11/20/2017

Tree Based Methods

```
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 3.3.2
```

```
library(tree)
library(randomForest)
```

```
## randomForest 4.6-12
```

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

```
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 3.3.2
```

```
## Loading required package: survival
```

```
## Warning: package 'survival' was built under R version 3.3.2
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 3.3.2
```

```
## Loading required package: splines
```

```
## Loading required package: parallel
```

```
## Loaded gbm 2.1.3
```

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

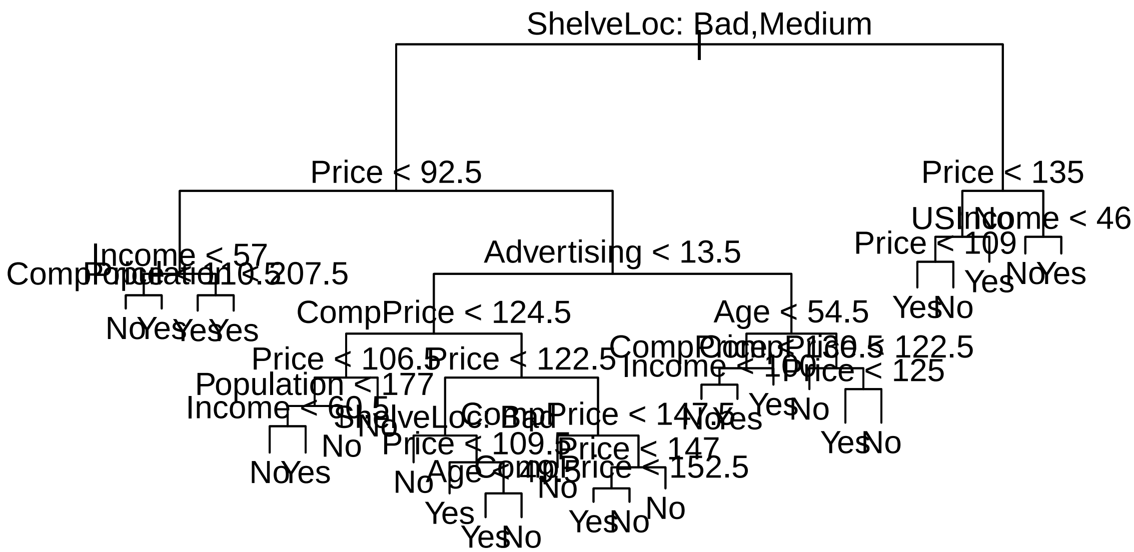
fit a decision tree. describe the output of summary, plot, text, and display

```
# fit the decision tree
tree_carseats <- tree(High ~ . -Sales, data = 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)
```



Looking at the summary of our tree we can find our overall misclassification error rate for our 400 observations, which gives us a good training misclassification rate. It also gives the the amount of terminal nodes which means our tree was split many times. From our tree we see that the first split is of shelf location which means shelf location is a good indicator of sales as is price which is the next split in our tree.

```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 ) *
```

The output of tree_carseats shows us all the splits for each node and which nodes are terminal nodes, it also gives us proportions of the responses in each node.

Random Forests

```r
# create training data using 80% observations
samp <- sample(1:dim(carseats)[1], size =  floor(.8 * dim(carseats)[1]))
train <- carseats[samp,]

test <- carseats[-samp,]

# fit random forest on the training data, High on all but sales
forest <- randomForest(High ~ . - Sales, data = train, xtest = test[,-c(1,12) ], ytest = test[,12], impo

# we can compute the test error rate using forest$test
head(forest$test$err.rate, 30)
```

```
##          Test          No       Yes
##  [1,] 0.3250 0.28947368 0.3571429
##  [2,] 0.3625 0.26315789 0.4523810
```

```
##   [3,] 0.2375 0.13157895 0.3333333
##   [4,] 0.2375 0.18421053 0.2857143
##   [5,] 0.2750 0.15789474 0.3809524
##   [6,] 0.2625 0.18421053 0.3333333
##   [7,] 0.2125 0.13157895 0.2857143
##   [8,] 0.2625 0.15789474 0.3571429
##   [9,] 0.2250 0.15789474 0.2857143
##  [10,] 0.2250 0.18421053 0.2619048
##  [11,] 0.2625 0.15789474 0.3571429
##  [12,] 0.2375 0.15789474 0.3095238
##  [13,] 0.2125 0.15789474 0.2619048
##  [14,] 0.2250 0.15789474 0.2857143
##  [15,] 0.1875 0.10526316 0.2619048
##  [16,] 0.1875 0.10526316 0.2619048
##  [17,] 0.2125 0.10526316 0.3095238
##  [18,] 0.2125 0.10526316 0.3095238
##  [19,] 0.1875 0.07894737 0.2857143
##  [20,] 0.1875 0.07894737 0.2857143
##  [21,] 0.1875 0.07894737 0.2857143
##  [22,] 0.2125 0.10526316 0.3095238
##  [23,] 0.1875 0.07894737 0.2857143
##  [24,] 0.2125 0.07894737 0.3333333
##  [25,] 0.1875 0.07894737 0.2857143
##  [26,] 0.2000 0.07894737 0.3095238
##  [27,] 0.2000 0.07894737 0.3095238
##  [28,] 0.1875 0.05263158 0.3095238
##  [29,] 0.2000 0.07894737 0.3095238
##  [30,] 0.2000 0.07894737 0.3095238
```

```r
# get the oob error rate
head(forest$err.rate, 30)
```

```
##              OOB        No       Yes
##   [1,] 0.2857143 0.2133333 0.3921569
##   [2,] 0.2487047 0.1652893 0.3888889
##   [3,] 0.2941176 0.1959459 0.4555556
##   [4,] 0.2771536 0.2000000 0.4019608
##   [5,] 0.3286713 0.2824859 0.4036697
##   [6,] 0.2956811 0.2204301 0.4173913
##   [7,] 0.2754098 0.2157895 0.3739130
##   [8,] 0.3022508 0.2435233 0.3983051
##   [9,] 0.2779553 0.2010309 0.4033613
##  [10,] 0.2651757 0.2010309 0.3697479
##  [11,] 0.2420382 0.1701031 0.3583333
##  [12,] 0.2215190 0.1487179 0.3388430
##  [13,] 0.2555205 0.1785714 0.3801653
##  [14,] 0.2288401 0.1624365 0.3360656
##  [15,] 0.2507837 0.1776650 0.3688525
##  [16,] 0.2507837 0.1675127 0.3852459
##  [17,] 0.2476489 0.1624365 0.3852459
##  [18,] 0.2351097 0.1472081 0.3770492
##  [19,] 0.2319749 0.1370558 0.3852459
##  [20,] 0.2413793 0.1472081 0.3934426
##  [21,] 0.2413793 0.1522843 0.3852459
##  [22,] 0.2351097 0.1472081 0.3770492
```

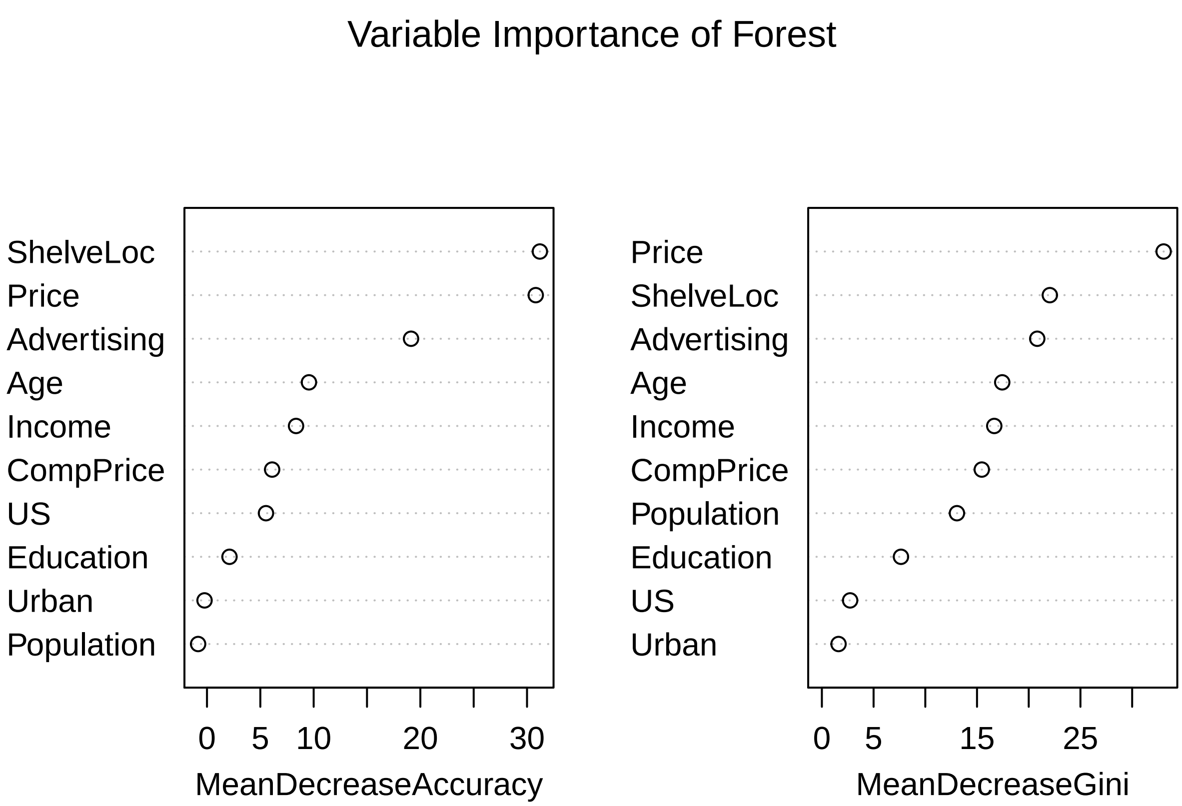
```
## [23,] 0.2445141 0.1421320 0.4098361
## [24,] 0.2375000 0.1515152 0.3770492
## [25,] 0.2468750 0.1515152 0.4016393
## [26,] 0.2375000 0.1464646 0.3852459
## [27,] 0.2375000 0.1414141 0.3934426
## [28,] 0.2375000 0.1515152 0.3770492
## [29,] 0.2375000 0.1464646 0.3852459
## [30,] 0.2437500 0.1616162 0.3770492
```

compared to the test error rates the out of box error rates are overall higher than the test error.

```r
# view the importance of each variable
importance(forest)
```

```
##                     No         Yes MeanDecreaseAccuracy MeanDecreaseGini
## CompPrice    7.0354274  0.95171426            6.1071683        15.460958
## Income       5.0809498  7.15747344            8.3495081        16.667539
## Advertising 10.3716325 16.86409330           19.1275852        20.824070
## Population  -1.2500342  0.19294144           -0.8314643        13.059987
## Price       22.1821575 26.18501787           30.8171623        33.045180
## ShelveLoc   24.0190599 26.03330158           31.2051712        22.041078
## Age          7.7330749  5.36820896            9.5584406        17.437742
## Education    2.1110777  0.63974637            2.1099724         7.647948
## Urban       -0.2146927 -0.07884738           -0.2304757         1.611492
## US           1.1022491  5.69818658            5.5301502         2.731150
```

```r
# create a visualization with varImpPlot()
varImpPlot(forest, main = "Variable Importance of Forest")
```



The variables that are most important are Price and ShelveLoc, they have the greatest decreases in mean accuracy and mean gini.

Boosted Trees

Compute the test error rate for boosted trees

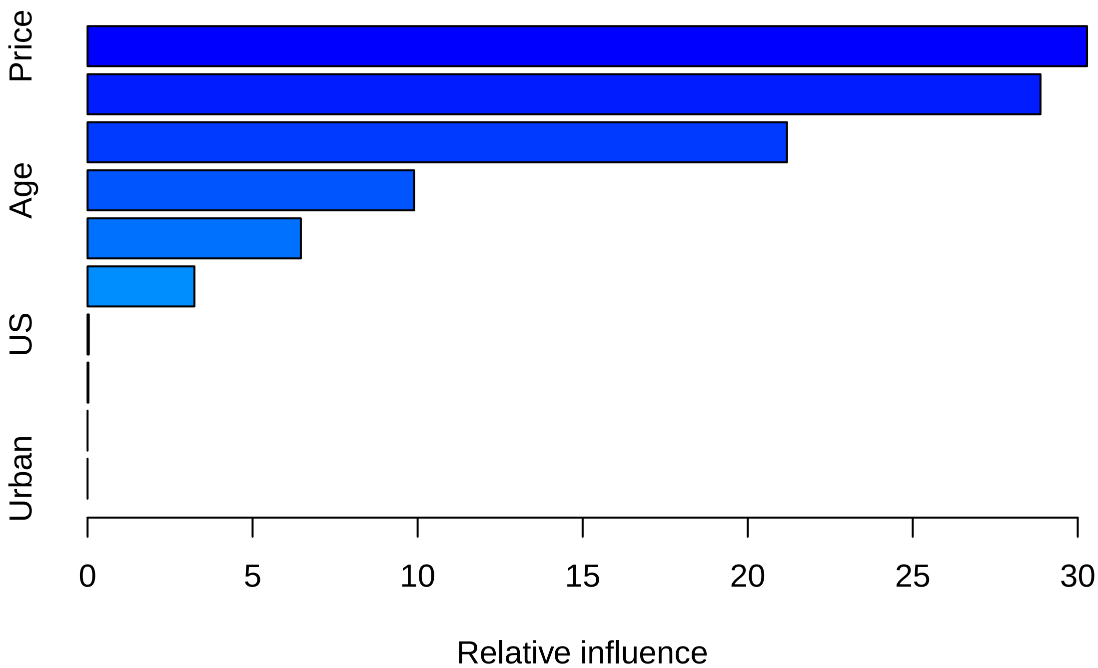
```
# gbm package needs response in [0,1]
train$High <- as.integer(train$High) - 1
test$High <- as.integer(test$High) - 1
```

fit the boosted tree

```
# fit boosted trees B = 5000, cutoff = .5
boosted_tree <- gbm(High ~ . -Sales, distribution = "bernoulli" , data = train, n.trees = 5000)
```

run summary fo the train boosted trees

```
summary(boosted_tree)
```



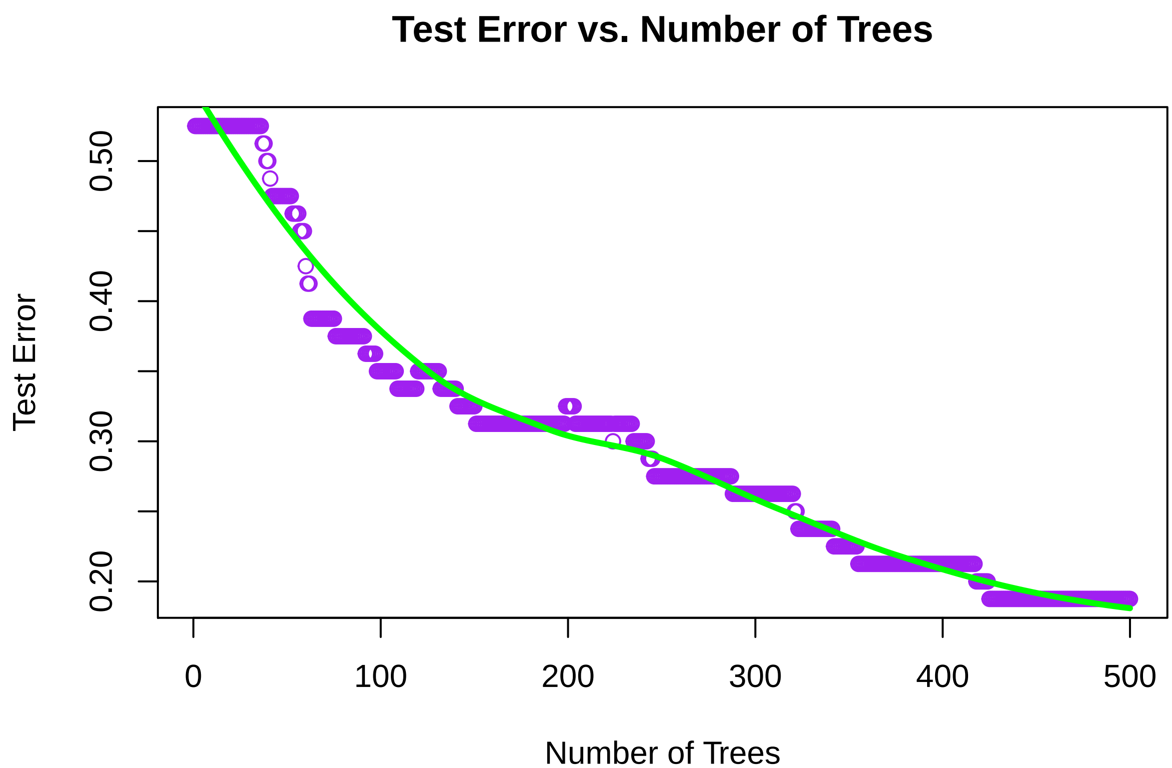

```
##                     var     rel.inf
## Price             Price 30.28151549
## ShelveLoc     ShelveLoc 28.87179702
## Advertising Advertising 21.19074267
## Age                 Age  9.89309292
## Income           Income  6.46141737
## CompPrice     CompPrice  3.23853766
## US                   US  0.03699901
## Population   Population  0.02589785
## Education     Education  0.00000000
## Urban             Urban  0.00000000
```

The most important variables are ShelveLoc and Price. predict and comput the test error rate

```r
# predict with n.trees [10, 20, ..., 5000]
boosted_predict <- predict(boosted_tree, newdata = test, n.trees = seq(10, 5000, by = 10), type = "respo

# convert the probabilities to same type 1 or 0 as High in training and test, using cutoff .5
class_predictions <- ifelse(boosted_predict <.5, 0, 1)

# calculate test error rate for each number of tree iteration
test_error_rates <- class_predictions == test$High

# test error is 1 - test accuracy
test_error_rates <- 1 - colMeans(test_error_rates)
```

plot the test error rates against the number of trees

```r
plot(test_error_rates, xlab = "Number of Trees", ylab = "Test Error", main = "Test Error vs. Number of T
lo <- loess(test_error_rates ~ seq(10, 5000, by = 10))
lines(predict(lo), col = "green", lwd = 3)
```



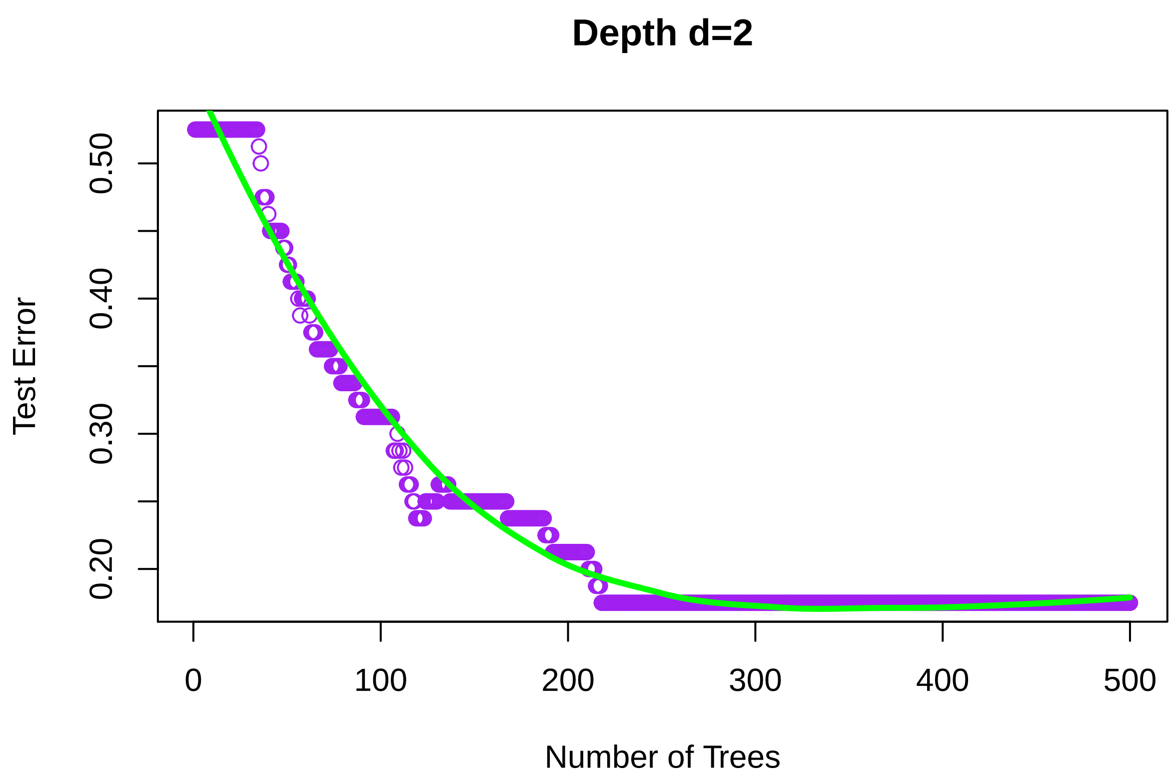
redo the last part with different interaction depths

```r
# interaction depth 2,3,4

test_error_by_depth <- list()

for (i in 2:4){

  boosted_tree_loop <- gbm(High ~ . -Sales, distribution = "bernoulli" , data = train, n.trees = 5000, i

  loop_predict <- predict(boosted_tree_loop, newdata = test, n.trees = seq(10, 5000, by = 10), type = "r

```

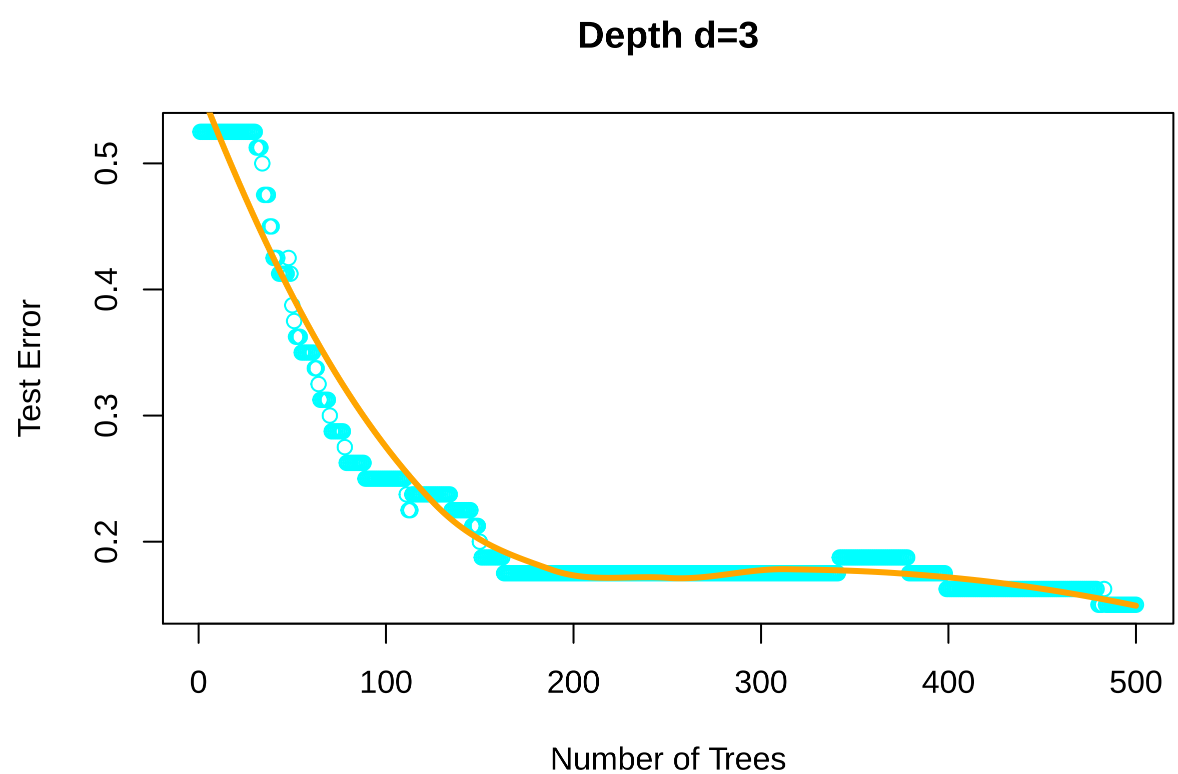
```r
  # convert the probabilities to same type 1 or 0 as High in training and test, using cutoff .5
  class_predictions_loop <- ifelse(loop_predict <.5, 0, 1)

  # calculate test error rate for each number of tree iteration
  test_error_rates_loop <- class_predictions_loop == test$High

  # test error is 1 - test accuracy
  test_error_rates_loop <- 1 - colMeans(test_error_rates_loop)

  test_error_by_depth[[i-1]] <- test_error_rates_loop

}
```

plot test error rates against number of trees given different interaction depths

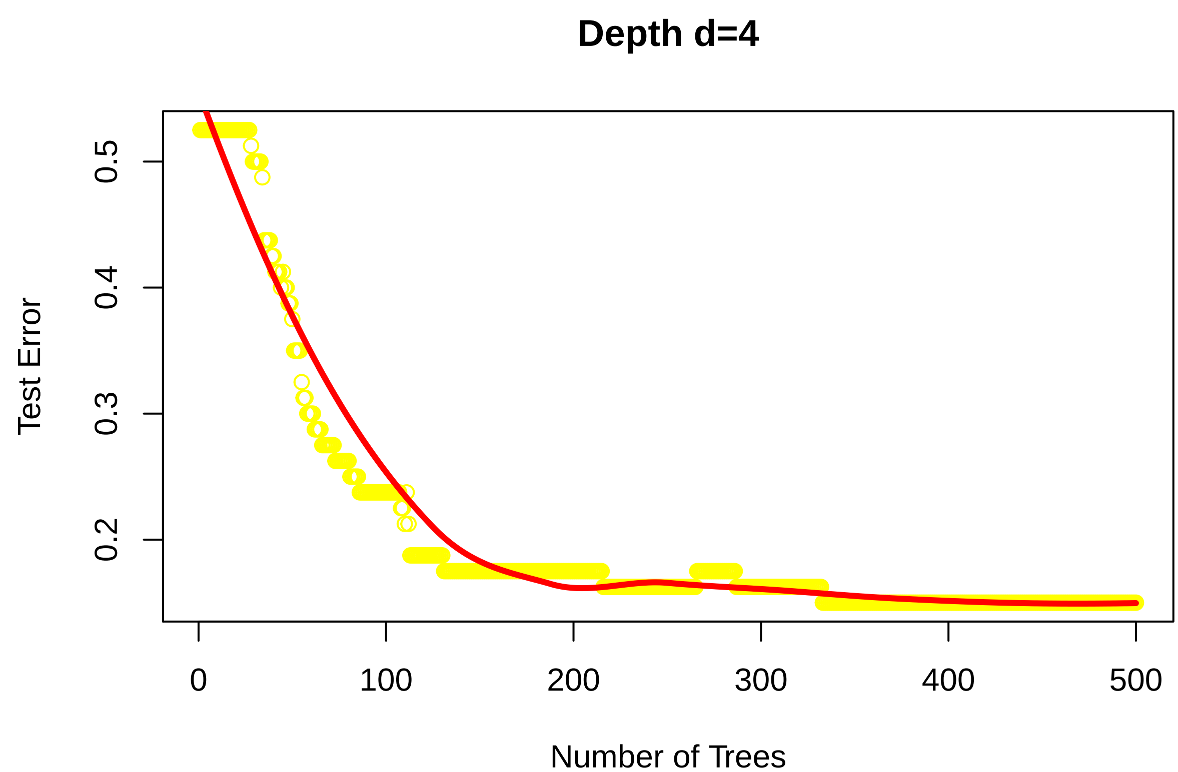
```r
# ineraction depth 2
plot(test_error_by_depth[[1]], xlab = "Number of Trees", ylab = "Test Error", main = "Depth d=2", col =
lo <- loess(test_error_by_depth[[1]] ~ seq(10, 5000, by = 10))
lines(predict(lo), col = "green", lwd = 3)
```




```r
#interaction depth 3
plot(test_error_by_depth[[2]], xlab = "Number of Trees", ylab = "Test Error", main = "Depth d=3", col =
lo <- loess(test_error_by_depth[[2]] ~ seq(10, 5000, by = 10))
lines(predict(lo), col = "orange", lwd = 3)
```




```
# interaction depth 4
plot(test_error_by_depth[[3]], xlab = "Number of Trees", ylab = "Test Error", main = "Depth d=4", col =
lo <- loess(test_error_by_depth[[3]] ~ seq(10, 5000, by = 10))
lines(predict(lo), col = "red", lwd = 3)
```



There are only slight differences in our error curves, All of their test error generally derease as the number of trees used increase. With depth 4 it seams to have the lowest test error rate overall.