lab12

shichenh

11/18/2017

```
library(ISLR)
library(tree)
library(randomForest)
library(gbm)
library(ggplot2)
```

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

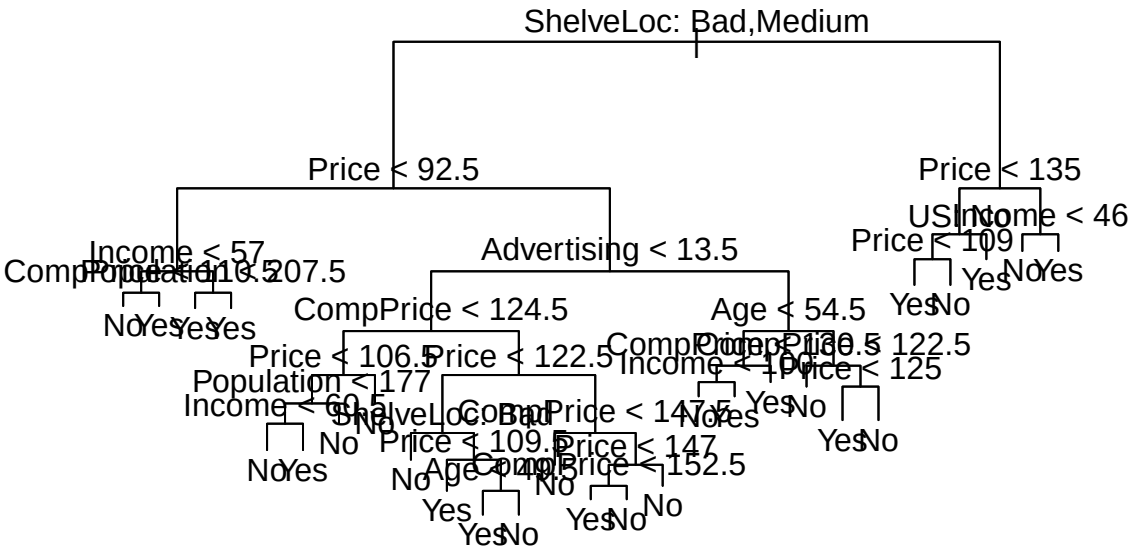
```
tree_carseats <- tree(High ~.-Sales, data=carseats)
```

```
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
```

We have a 9% training misclassification error.

```
# show plot
plot(tree_carseats)
text(tree_carseats, pretty=0)
```



Seems like price is a very strong tree divider(consistently separates categories in many depths). ShelveLoc is also a strong predictor since it is a the top of the tree.

```
# display
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 ) *
```

Asumming the first number () at the end of each root is the accuracy, some roots are performing very bad and some are very good.

Random Forest

```
set.seed(123)
train_index <- sample(1:nrow(carseats), nrow(carseats)*0.8)
train <- carseats[train_index,]
test <- carseats[-train_index,]

rf_mod <- randomForest(High ~.-Sales, data=train, importance = T)
rf_mod
```

```
##
## Call:
##  randomForest(formula = High ~ . - Sales, data = train, importance = T)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 18.12%
## Confusion matrix:
##      No Yes class.error
## No  178  17  0.08717949
## Yes  41  84  0.32800000
```

```
mean(predict(rf_mod, data=test) != test$High)
```

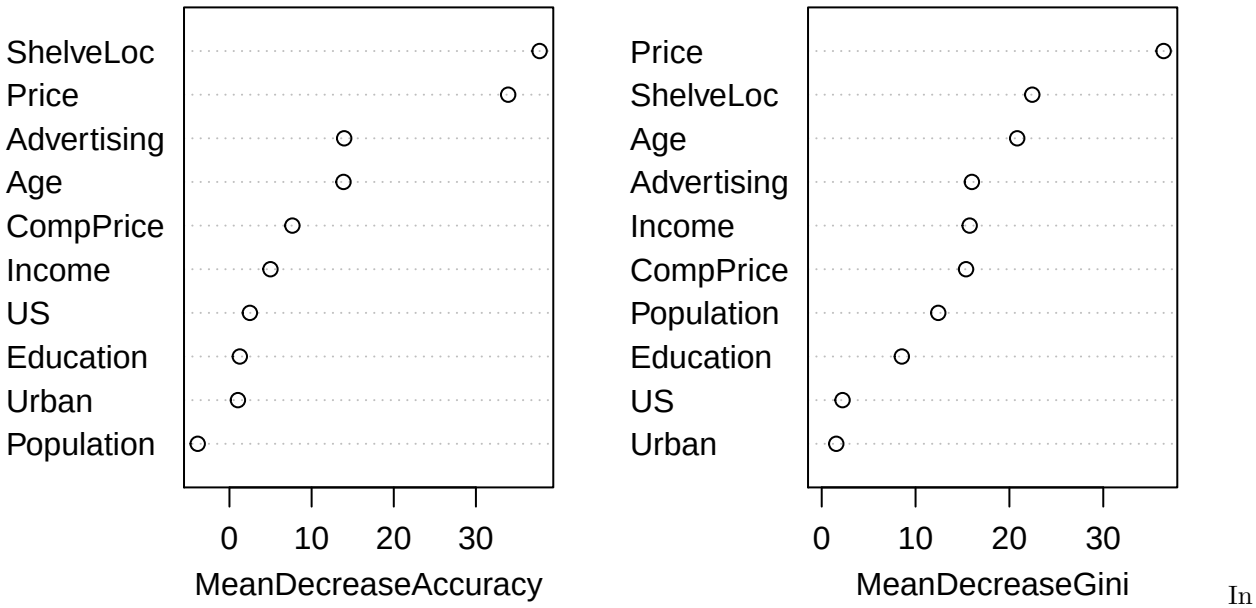
```
## [1] 0.515625
```

The test error is significantly greater than the OOB error.

```
randomForest::importance(rf_mod)
```

```
##                    No        Yes MeanDecreaseAccuracy MeanDecreaseGini
## CompPrice    5.879234  4.9611995             7.657238        15.377608
## Income       2.434138  4.9492465             4.989317        15.766094
## Advertising  6.165743 13.0656296            13.963248        16.000137
## Population  -1.995887 -3.7236726            -3.867340        12.418919
## Price       28.038553 24.4483964            33.926733        36.432319
## ShelveLoc   29.491577 28.5385791            37.751263        22.435012
## Age          9.973905 10.7506078            13.878122        20.830463
## Education    1.212527  0.6770653             1.259273         8.536295
## Urban        1.633113 -0.5864025             1.028939         1.551099
## US          -2.287862  4.9097965             2.483681         2.218444
```

```
varImpPlot(rf_mod)
```

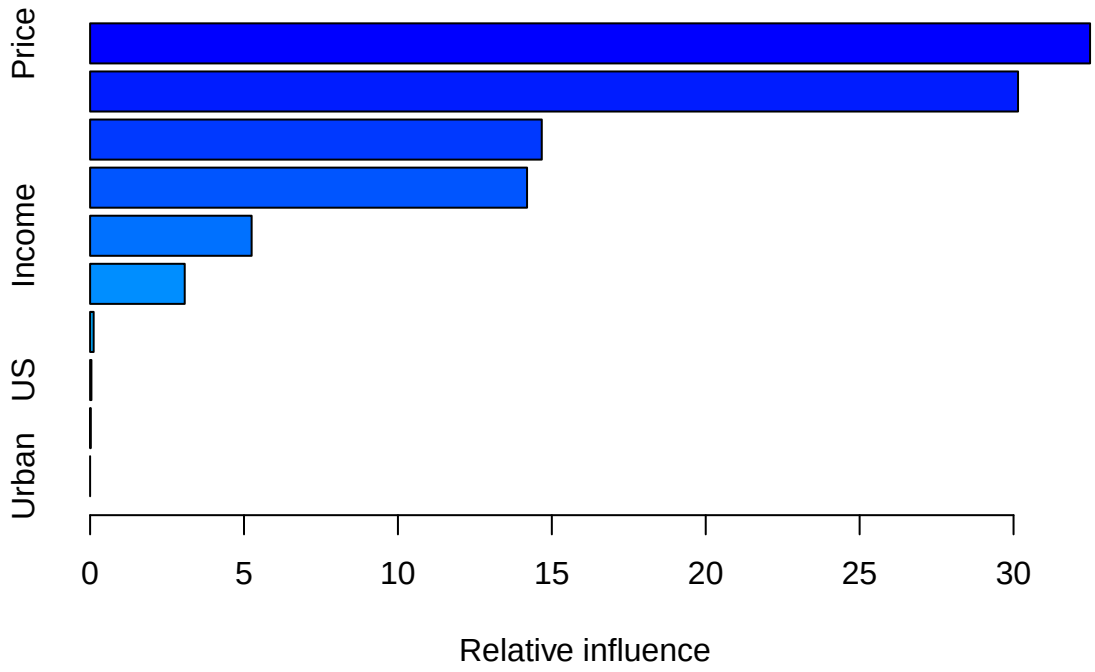


In seed 123, the two most important variables are Price and ShelveLoc.

Boosted Trees

```
set.seed(123)
btree <- gbm(Sales > 8 ~.-Sales-High,
             distribution = "bernoulli",
             n.tree = 5000,
             data=train)
test_error = predict(btree, newdata=test, n.trees=5000, type = "response")
```

```
summary(btree)
```

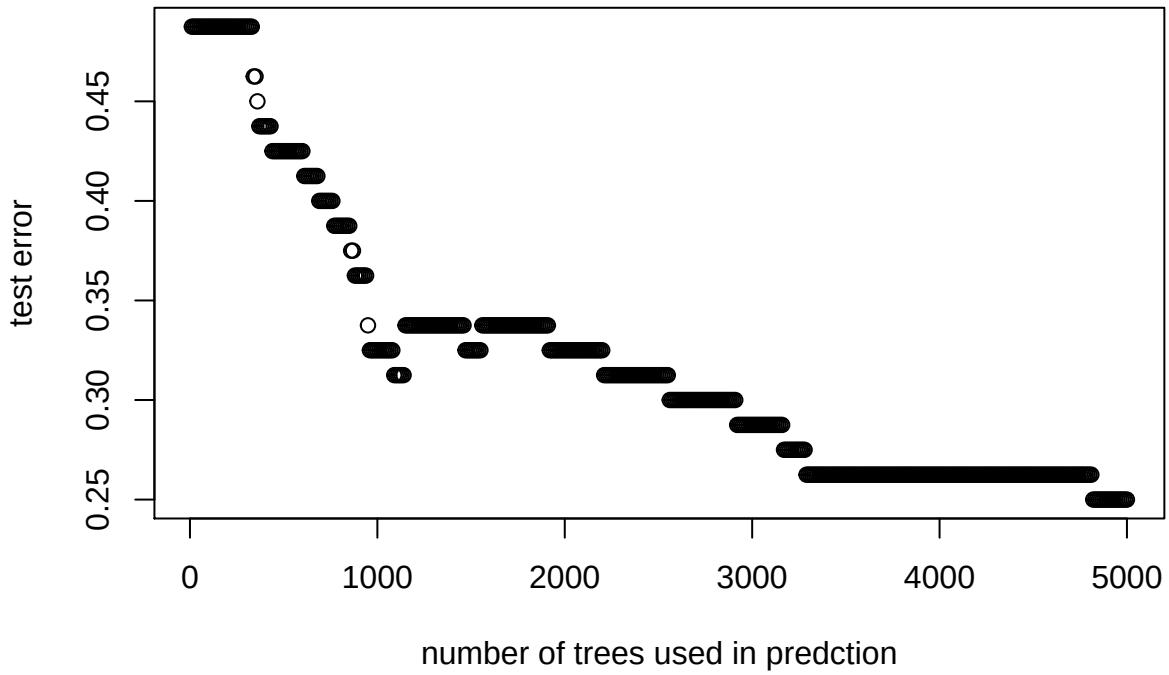



```
##                     var     rel.inf
## Price             Price 32.48577998
## ShelveLoc     ShelveLoc 30.14598518
## Advertising Advertising 14.67335862
## Age                 Age 14.19498058
## Income           Income  5.24792476
## CompPrice     CompPrice  3.07391307
## Education     Education  0.11555577
## US                   US  0.04270746
## Population   Population  0.01979459
## Urban             Urban  0.00000000
```

In seed 123, the two most important variable are Price and Shelveloc.

```
test_rates <- numeric()
ntrees <- seq(10, 5000, 10)
for (i in 1:length(ntrees)) {
  pred = predict(btree, newdata=test, n.trees=ntrees[i], type = "response")
  test_rates[i] = mean((pred >= 0.5) != (test$Sales > 8))
  if (i %% 500 == 0){
    #print(paste(i, "iterations completed"))
  }
}
```

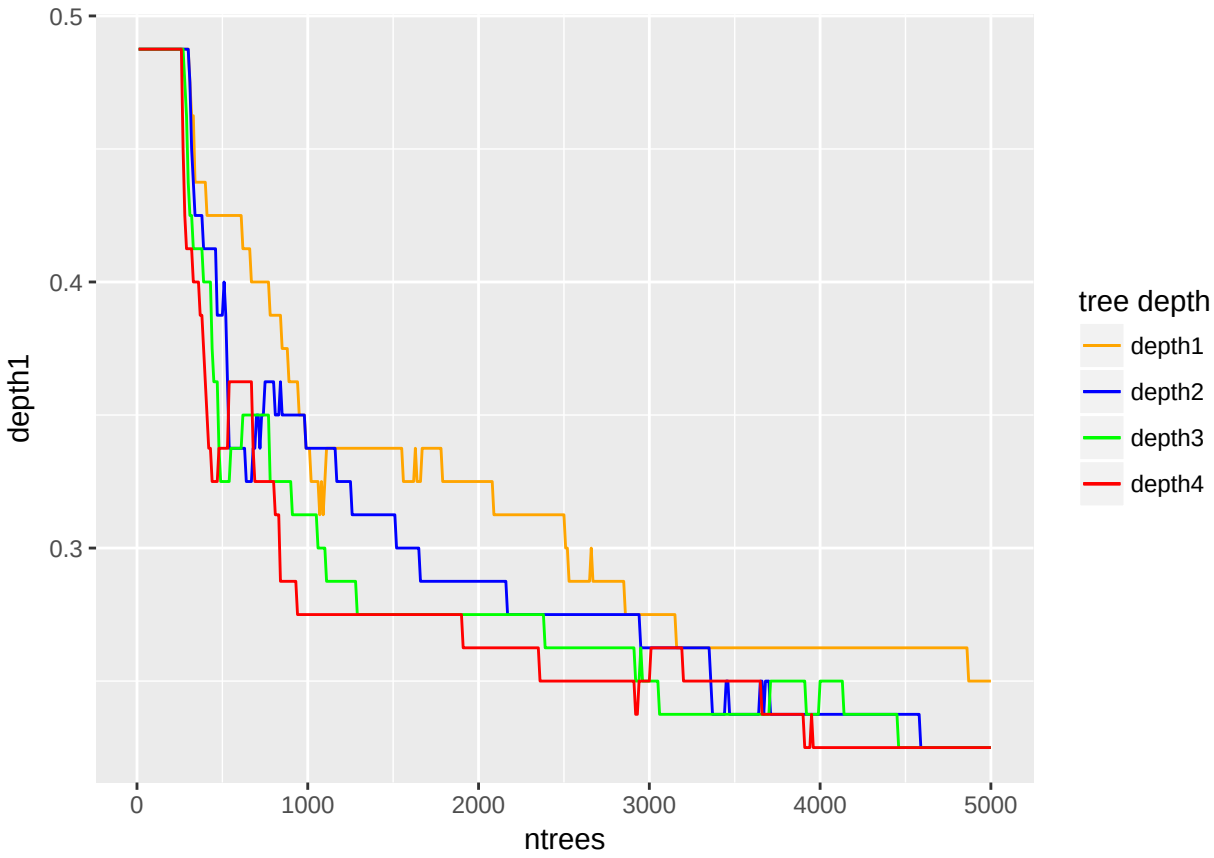
```
plot(ntrees, test_rates, xlab="number of trees used in predction", ylab="test error")
```




```r
# make the matrix for test error for different depths
test_rates_depth <- matrix(rep(-1, 4*length(ntrees)), nrow=4)
for (depth in 1:4) {
  # make a new forest for each depth
  temp.btree = gbm(Sales > 8 ~.-Sales-High,
                   distribution = "bernoulli",
                   n.tree = 5000,
                   interaction.depth=depth,
                   data=train)
  # run the predictions with different number of trees
  for (ntree in 1:length(ntrees)){
    temp.pred = predict(temp.btree, newdata=test, n.trees=ntrees[ntree], type = "response")
    test_rates_depth[depth, ntree] = mean((temp.pred >= 0.5) != (test$Sales > 8))
  }
  # just to check how many models I need to make
  #print(paste(depth, "forests completed"))
}
```

```r
test_rates_depth <- data.frame(t(test_rates_depth))
colnames(test_rates_depth) <- paste0("depth",c(1:4))

ggplot(test_rates_depth) +
  geom_line(aes(ntrees, depth1, colour = "depth1")) +
  geom_line(aes(ntrees, depth2, colour = "depth2")) +
  geom_line(aes(ntrees, depth3, colour = "depth3")) +
  geom_line(aes(ntrees, depth4, colour = "depth4")) +
  scale_color_manual(name="tree depth", values=c(depth1="orange", depth2="blue", depth3="green", depth4=
```



In general, the higher the depth(from 1 to 4), the lower the maximum error rate is. Also the error rate converges to maximum error rate as we use more trees to make the prediction. The convergent rate goes fast at the beginning then slows down after 1000 trees.