

Lab 07: Bagging and Random Forests

PSTAT 131/231, Spring 2018

Learning Objectives

- Bagged trees and random forest by `randomForest()`
- Variable importance by `importance()` and `varImpPlot()`
- Boosting by `gbm()`

In Lab 04 - Decision Trees, we used classification trees to analyze the Carseats data set. In this data, Sales is a continuous variable, and so we begin by recoding it as a binary variable. We use the `ifelse()` function to create a variable, called *High*, which takes on a value of *Yes* if the Sales variable exceeds the median of Sales, and takes a value *No* otherwise.

```
library(dplyr)
#install.packages("randomForest")
library(randomForest)
#install.packages("gbm")
library(gbm)
library(ISLR)
library(tree)
```

Let's have a glance at the data.

```
attach(Carseats)
Carseats = Carseats %>%
  mutate(High=as.factor(ifelse(Sales <= median(Sales), "No", "Yes"))) %>%
  select(-Sales)

# Check the structure of above data frame we just created
glimpse(Carseats)
```

```
## Observations: 400
## Variables: 11
## $ CompPrice    <dbl> 138, 111, 113, 117, 141, 124, 115, 136, 132, 132, ...
## $ Income       <dbl> 73, 48, 35, 100, 64, 113, 105, 81, 110, 113, 78, 9...
## $ Advertising  <dbl> 11, 16, 10, 4, 3, 13, 0, 15, 0, 0, 9, 4, 2, 11, 11...
## $ Population   <dbl> 276, 260, 269, 466, 340, 501, 45, 425, 108, 131, 1...
## $ Price        <dbl> 120, 83, 80, 97, 128, 72, 108, 120, 124, 124, 100,...
## $ ShelveLoc    <fctr> Bad, Good, Medium, Medium, Bad, Bad, Medium, Good...
## $ Age          <dbl> 42, 65, 59, 55, 38, 78, 71, 67, 76, 76, 26, 50, 62...
## $ Education    <dbl> 17, 10, 12, 14, 13, 16, 15, 10, 10, 17, 10, 13, 18...
## $ Urban        <fctr> Yes, Yes, Yes, Yes, Yes, No, Yes, Yes, No, No, No...
## $ US           <fctr> Yes, Yes, Yes, Yes, No, Yes, No, Yes, No, Yes, Ye...
## $ High         <fctr> Yes, Yes, Yes, No, No, Yes, No, Yes, No, No, Yes,...
```

1. Review of a single tree model

As usual, we split the data into training and test set.

```
# Sample 250 observations as training data
set.seed(3)
```

```
train = sample(1:nrow(Carseats), 250)
train.carseats = Carseats[train,]
```

```
# The rest as test data
test.carseats = Carseats[-train,]
```

As a review, we use 10-fold CV to select the best tree size and prune the original large tree to this target number. We calculate the test error rate for future comparison.

```
tree.carseats = tree(High ~ ., data = Carseats, subset = train)
summary(tree.carseats)
```

```
##
## Classification tree:
## tree(formula = High ~ ., data = Carseats, subset = train)
## Variables actually used in tree construction:
## [1] "Price"          "US"              "CompPrice"      "ShelveLoc"      "Advertising"
## [6] "Age"            "Income"
## Number of terminal nodes: 21
## Residual mean deviance: 0.4589 = 105.1 / 229
## Misclassification error rate: 0.112 = 28 / 250
```

```
# 10-fold CV for selecting best tree size
tree.cv = cv.tree(tree.carseats, FUN=prune.misclass, K=10)
```

```
# Best size
best.cv = min(tree.cv$size[tree.cv$dev==min(tree.cv$dev)])
best.cv
```

```
## [1] 10
```

```
# Prune the tree to the optimal size
tree.prune = prune.misclass(tree.carseats, best=best.cv)
```

```
# Test error for tree.prune
tree.err = table(treePred=predict(tree.prune, newdata=test.carseats, type="class"),
                  truth=test.carseats$High)
test.tree.err = 1 - sum(diag(tree.err))/sum(tree.err)
test.tree.err
```

```
## [1] 0.2466667
```

2. Bagging

The test error rate for the best-size pruned tree is 0.2467. In the following, we apply bagging and random forests to the Carseats data, using the randomForest package in R and compare the same metric for bagged tree and random forest. Note that the exact results obtained in this section may depend on the version of R and the version of the randomForest package installed on your computer. Recall that bagging is simply a special case of a random forest with $m = p$. Therefore, the randomForest() function can be used to perform both random forests and bagging. We perform bagging as follows:

```
bag.carseats = randomForest(High ~ ., data=train.carseats, mtry=10, importance=TRUE)
bag.carseats
```

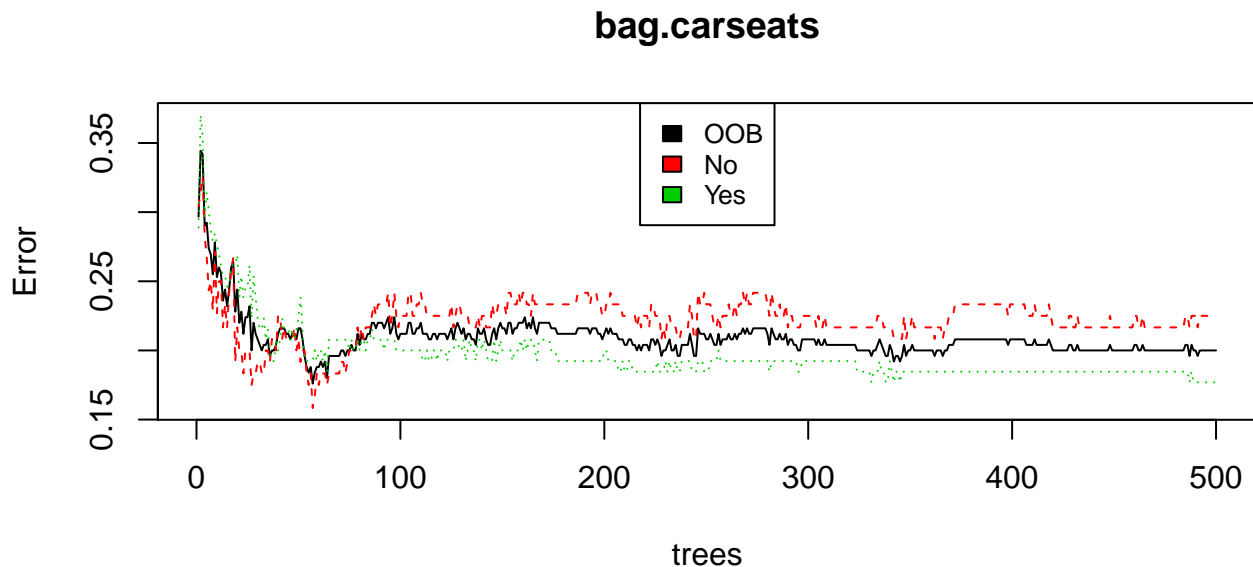
```
##
## Call:
```

```
## randomForest(formula = High ~ ., data = train.carseats, mtry = 10, importance = TRUE)
##           Type of random forest: classification
##           Number of trees: 500
## No. of variables tried at each split: 10
##
##           OOB estimate of  error rate: 20%
## Confusion matrix:
##      No Yes class.error
## No  93  27  0.2250000
## Yes 23 107  0.1769231

# equivalently, you can do
# bag.carseats = randomForest(High ~ ., data=Carseats, subset=train, mtry=4, importance=TRUE)
```

The argument `mtry=10` indicates that 10 predictors should be considered for each split of the tree -in other words, that bagging should be done. The argument `importance=TRUE` tells whether independent variable importance in bagged trees should be assessed.

```
plot(bag.carseats)
legend("top", colnames(bag.carseats$err.rate), col=1:4, cex=0.8, fill=1:4)
```



How well does this bagged model perform on the test set?

```
yhat.bag = predict(bag.carseats, newdata = test.carseats)

# Confusion matrix
bag.err = table(pred = yhat.bag, truth = test.carseats$High)
test.bag.err = 1 - sum(diag(bag.err))/sum(bag.err)
test.bag.err
```

```
## [1] 0.2
```

The test set error rate associated with the bagged classification tree is 0.2, 4.67% lower than that obtained using an optimally-pruned single tree (0.2467). You may consider this a minor improvement, however there are many cases that the improvement could be as half. We could change the number of trees grown by `randomForest()` using the `ntree` argument. For simplicity of output, we set the following code chunk option as `eval=FALSE`.

```

bag.carseats = randomForest(High ~ ., data=train.carseats, mtry=10, ntree=700, importance=TRUE)
yhat.bag = predict (bag.carseats, newdata = test.carseats)

# Confusion matrix
bag.err = table(pred = yhat.bag, truth = test.carseats$High)
test.bag.err = 1 - sum(diag(bag.err))/sum(bag.err)
test.bag.err

```

3. Random Forests

Growing a random forest proceeds in exactly the same way, except that we use a smaller value of the `mtry` argument. By default, `randomForest()` uses $p/3$ variables when building a random forest of regression trees, and \sqrt{p} variables when building a random forest of classification trees. Here we use `mtry = 3`.

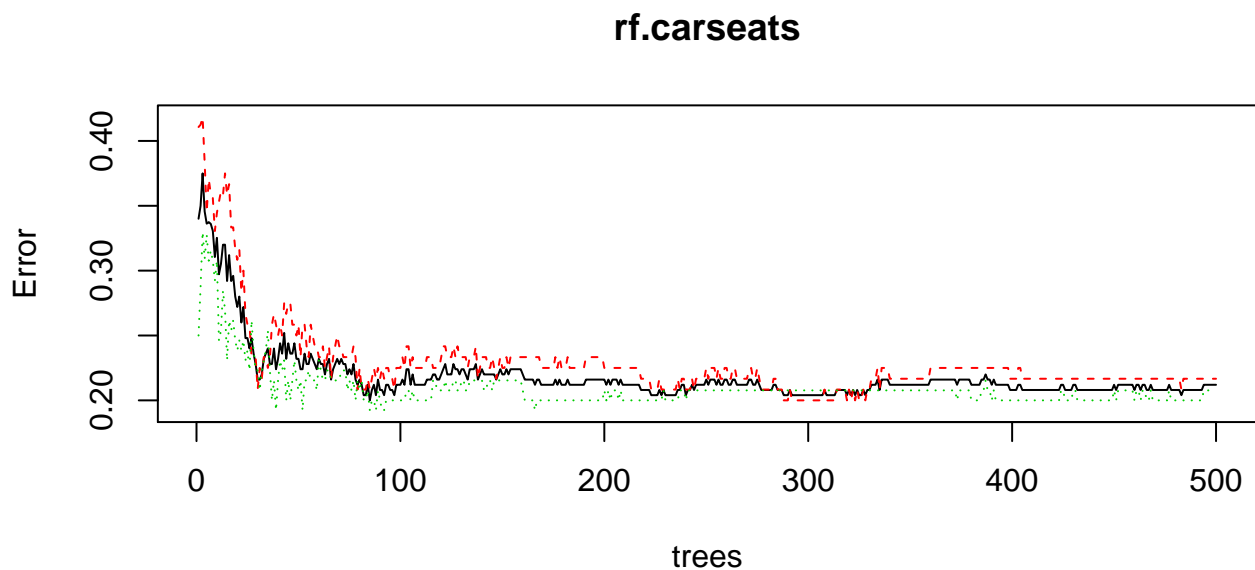
```

rf.carseats = randomForest(High ~ ., data=train.carseats, mtry=3, ntree=500, importance=TRUE)
rf.carseats

##
## Call:
## randomForest(formula = High ~ ., data = train.carseats, mtry = 3,          ntree = 500, importance = TR
##              Type of random forest: classification
##              Number of trees: 500
## No. of variables tried at each split: 3
##
##              OOB estimate of  error rate: 21.2%
## Confusion matrix:
##      No Yes class.error
## No  94  26   0.2166667
## Yes 27 103   0.2076923

plot(rf.carseats)

```



```

yhat.rf = predict (rf.carseats, newdata = test.carseats)

# Confusion matrix

```

```
rf.err = table(pred = yhat.rf, truth = test.carseats$High)
test.rf.err = 1 - sum(diag(rf.err))/sum(rf.err)
test.rf.err
```

```
## [1] 0.186667
```

The test set error rate is 0.1867; this indicates that random forests yielded an improvement over bagging in this case.

Using the importance() function, we can view the importance of each importance() variable.

```
importance(rf.carseats)
```

##		No	Yes	MeanDecreaseAccuracy	MeanDecreaseGini
##	CompPrice	3.151355	1.989371	3.611065	13.758996
##	Income	2.084306	5.612889	5.548553	14.251814
##	Advertising	9.038012	13.224980	15.949961	14.025503
##	Population	-4.270508	-3.088536	-5.243950	10.476878
##	Price	28.274627	28.991627	37.346530	32.432844
##	ShelveLoc	21.959490	22.228576	28.142472	14.997104
##	Age	4.890296	5.247278	7.050210	14.998370
##	Education	-1.479957	-2.402804	-2.617748	6.385125
##	Urban	-1.144061	-1.157302	-1.449926	1.341353
##	US	-1.751939	4.198982	2.020854	1.473375

Variable importance plot is also a useful tool and can be plotted using varImpPlot() function. By default, top 10 variables are selected and plotted based on Model Accuracy and Gini value. We can also get a plot with decreasing order of importance based on Model Accuracy and Gini value.

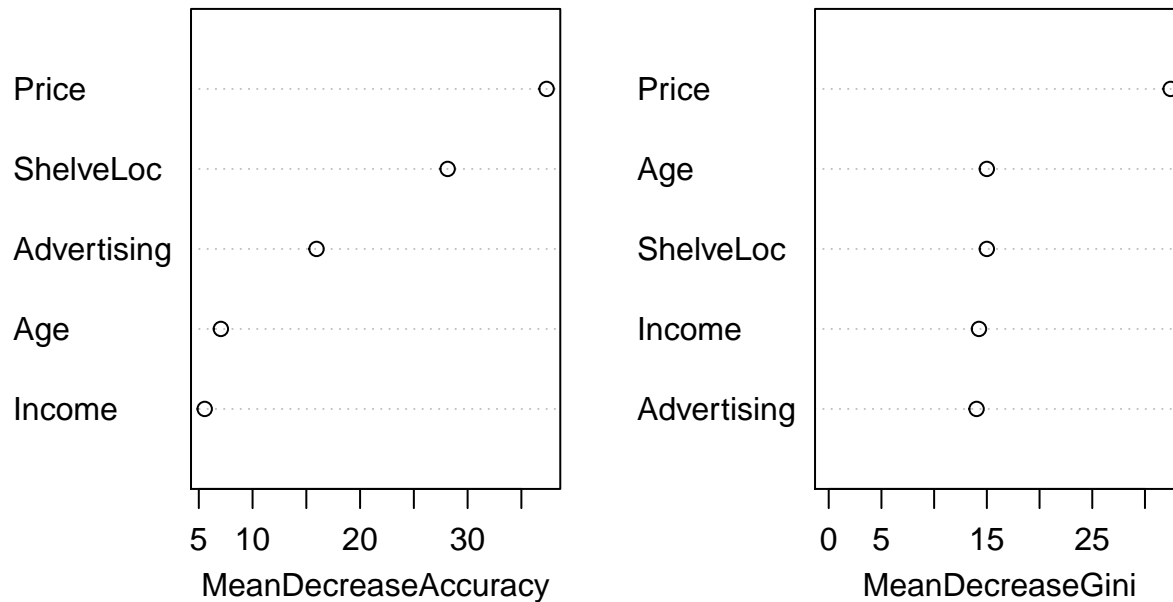
```
varImpPlot(rf.carseats)
```

rf.carseats



```
varImpPlot(rf.carseats, sort=T, main="Variable Importance for rf.carseats", n.var=5)
```

Variable Importance for rf.carseats



The results indicate that across all of the trees considered in the random forest, the price is by far the most important variable in terms of Model Accuracy and Gini index.

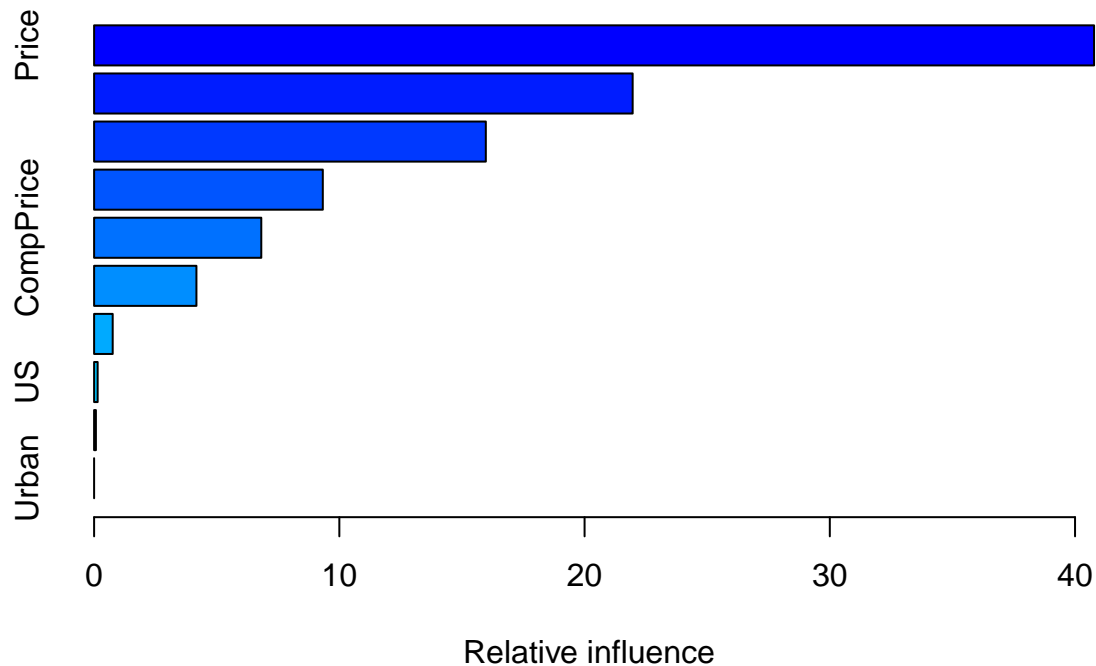
4. Boosting

Here we use the `gbm` package, and within it the `gbm()` function, to fit boosted classification trees to the Carseats data set. To use `gbm()`, we have to guarantee that the response variable is coded as $\{0, 1\}$ instead of two levels. We run `gbm()` with the option `distribution="bernoulli"` since this is a binary classification problem; if it were a regression problem, we would use `distribution="gaussian"`. The argument `n.trees=500` indicates that we want 500 trees, and the option `interaction.depth=4` limits the depth of each tree. The argument `shrinkage` is the learning rate or step-size reduction in every step of the boosting. Its default value is 0.001.

```
set.seed(1)
boost.carseats = gbm(ifelse(High=="Yes",1,0)~., data=train.carseats,
                     distribution="bernoulli", n.trees=500, interaction.depth=4)
```

The `summary()` function produces a relative influence plot and also outputs the relative influence statistics.

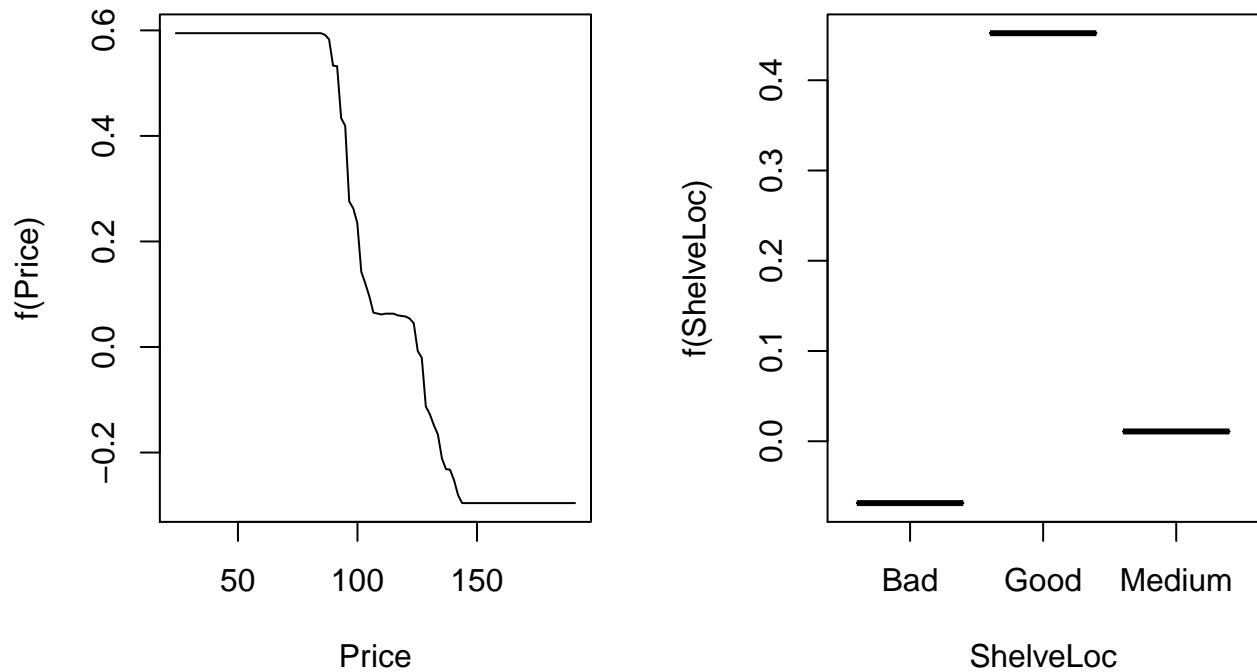
```
summary(boost.carseats)
```



```
##           var      rel.inf
## Price      Price 40.77518327
## ShelveLoc  ShelveLoc 21.96034564
## Advertising Advertising 15.97340122
## Age        Age 9.32590223
## CompPrice  CompPrice 6.81678196
## Income     Income 4.17206033
## Population Population 0.75832087
## US         US 0.14492721
## Education  Education 0.07307725
## Urban      Urban 0.00000000
```

We see that Price and ShelveLoc are by far the most important variables. We can also produce partial dependence plots for these variables. These plots illustrate the marginal effect of the selected variables on the response after integrating out the other variables.

```
par(mfrow =c(1,2))
plot(boost.carseats ,i="Price")
plot(boost.carseats ,i="ShelveLoc")
```



We now use the boosted model to predict High on the test set:

```
yhat.boost = predict(boost.carseats, newdata = test.carseats, n.trees=500)

# Confusion matrix
boost.err = table(pred = yhat.rf, truth = test.carseats$High)
test.boost.err = 1 - sum(diag(boost.err))/sum(boost.err)
test.boost.err
```

```
## [1] 0.1866667
```

The test error rate obtained is 0.1867; similar to the test error rate for random forests and superior to that for bagging.