# Stats 101C HW5

Anna Piskun

11/20/2020

From Section 8.4 of the textbook:

- Problem 8.4.2
- Problem 8.4.5
- Problem 8.4.8

## Problem 8.4.2

**It is mentioned in Section 8.2.3 that boosting using depth-one trees (or stumps) leads to an additive model: that is, a model of the following form. Explain why this is the case. You can begin with (8.12) in Algorithm 8.2.**

Generally speaking, boosting creates a sequence of trees, each one building on the previous one. Earlier trees are small, and the next tree is created with the residuals of the previous tree. The parameter d represents the number of splits in each tree. If d = 1, then each tree is only split on one predictor. Looking at the algorithm, we set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training data. For b=1,..., B, we repeat the following steps: Fit a regression tree $\hat{f}_b(x)$ with d splits, in this case 1 split, to the training data. Next, update $\hat{f}$ by adding in a shrunken version of the new tree and update the residuals. This leads us to the final boosted model, which is found by summing up all of these shrunken models giving us $\hat{f}(x) =$

$$\sum_{b=1}^{B} \lambda \hat{f}_b(x)$$

. Since we know that each $\hat{f}_b$ is split only on one predictor variable and we add the shrunken versions of them in order to find our final model, we can conclude that boosting using depth-one trees leads to an additive model.

## Problem 8.4.5

**Suppose we produce ten bootstrapped samples from a data set containing red and green classes. We then apply a classification tree to each bootstrapped sample and, for a specific value of X, produce 10 estimates of P(Class is Red|X): 0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, and 0.75. There are two common ways to combine these results together into a single class prediction. One is the majority vote approach discussed in this chapter. The second approach is to classify based on the average probability. In this example, what is the final classification under each of these two approaches?**

Bagging can improve predictions for many regression methods, and it is particularly useful for decision trees. Given a training data, bagging averages the predictions from decision trees over a collection of bootstrap samples. Using the majority vote approach, for a given test observation we can record the class predicted by each of the B trees by taking a majority vote. This means that the overall prediction is the most commonly occurring majority class among the B predictions. Using the estimates given to us, we see that the majority vote approach gives us a final classification of red as there are 6 estimates for **red** (0.55, 0.6, 0.6, 0.65, 0.7, 0.75 are all greater than 0.5) vs. 4 estimates for green (0.1, 0.15, 0.2, 0.2 are all less than 0.5). The second approach to classify based on the average probability simply averages all 10 of the estimates given to us. We find that this average is 0.45 which is less than 0.5, and gives us a final classification of **green**.

## Problem 8.4.8

8. In the lab, a classification tree was applied to the Carseats data set after converting Sales into a qualitative response variable. Now we will seek to predict Sales using regression trees and related approaches, treating the response as a quantitative variable.

```
library(ISLR)
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

```
attach(Carseats)
```

### (a) Split the data set into a training set and a test set.

```
set.seed(123)
trainIndex <- createDataPartition(Carseats$Sales, p = 0.7,
                                  list = FALSE)
car.training <- Carseats[trainIndex,]
car.test <- Carseats[-trainIndex,]
head(car.training)
```

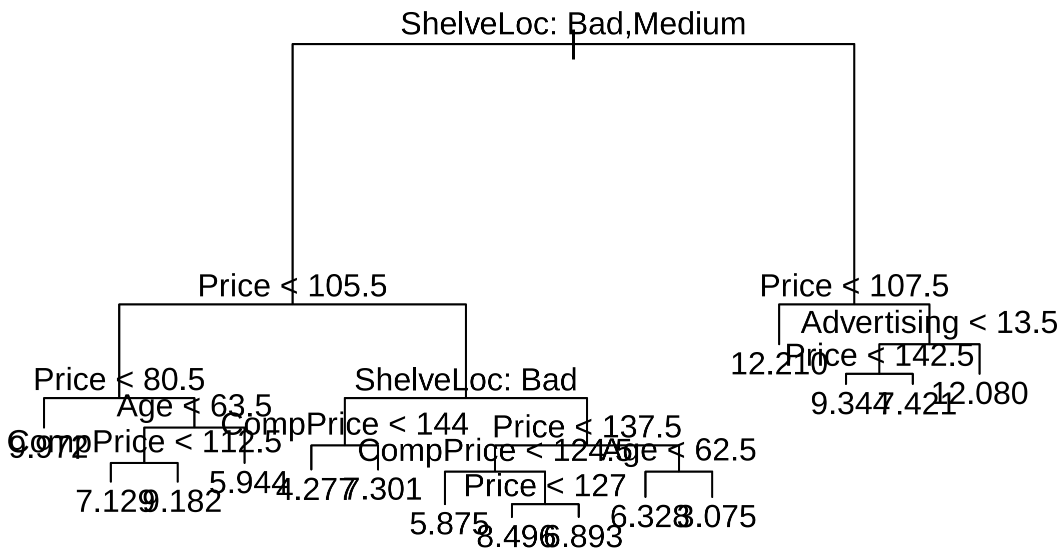
```
##    Sales CompPrice Income Advertising Population Price ShelveLoc Age Education
## 2  11.22       111     48          16        260    83      Good  65        10
## 3  10.06       113     35          10        269    80    Medium  59        12
## 4   7.40       117    100           4        466    97    Medium  55        14
## 6  10.81       124    113          13        501    72       Bad  78        16
## 8  11.85       136     81          15        425   120      Good  67        10
## 11  9.01       121     78           9        150   100       Bad  26        10
##    Urban  US
## 2    Yes Yes
## 3    Yes Yes
## 4    Yes Yes
## 6     No Yes
## 8    Yes Yes
## 11    No Yes
```

### (b) Fit a regression tree to the training set. Plot the tree, and interpret the results. What test MSE do you obtain?

```
library(tree)
tree.model <- tree(Sales~., data = car.training)
summary(tree.model)
```

```
##
## Regression tree:
## tree(formula = Sales ~ ., data = car.training)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Age"         "CompPrice"   "Advertising"
## Number of terminal nodes:  15
## Residual mean deviance:  2.379 = 632.8 / 266
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -4.13900 -1.01800 -0.02316  0.00000  1.02600  3.51500
```

```
plot(tree.model)
text(tree.model, pretty = 0)
```



```
pred.probs.test <- predict(tree.model, car.test)

n <- nrow(Carseats)
n_train <- nrow(car.training)
n_val <- n - n_train

MSE <- sum((car.test$Sales - pred.probs.test)^2)/n_val

MSE
```

```
## [1] 4.638469
```

The test MSE for this model is 4.638469. From our regression tree we find that ShelveLoc serves as our root node. This implies that it is a significant variable, specifically in determining whether a location is "bad" or "medium". Price serves as the next most important node, again signifying its potential significance. The only other variables included in this regression tree despite the fact that I fit a full model are Age, Advertising, and CompPrice. Lastly, we find that there are 15 terminal nodes.

**(c) Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the test MSE?**

```
library(rattle)
```

```
## Loading required package: tibble
```

```
## Loading required package: bitops
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.4.0 Copyright (c) 2006-2020 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
train_control <- trainControl(method="cv", number = 5,
                              classProbs = TRUE,
                              savePredictions = TRUE)
set.seed(123)
treefit <- train(Sales~.,
                 data = car.training, method = 'rpart',
                 trControl = train_control)
```

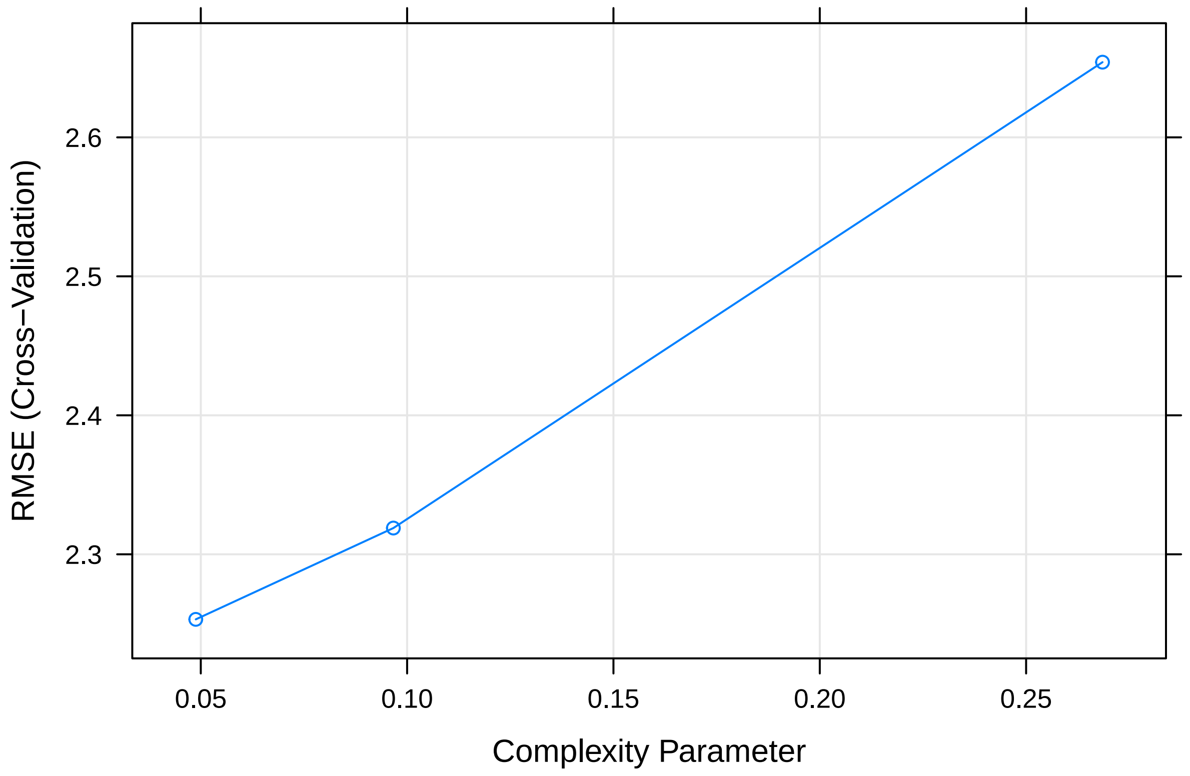
```
## Warning in train.default(x, y, weights = w, ...): cannnot compute class
## probabilities for regression
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```
print(treefit, digits = 2)
```

```
## CART
##
## 281 samples
##  10 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 225, 225, 224, 225, 225
## Resampling results across tuning parameters:
##
##   cp     RMSE  Rsquared  MAE
##   0.049  2.3   0.35      1.8
##   0.097  2.3   0.31      1.9
##   0.269  2.7   0.24      2.1
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was cp = 0.049.
```

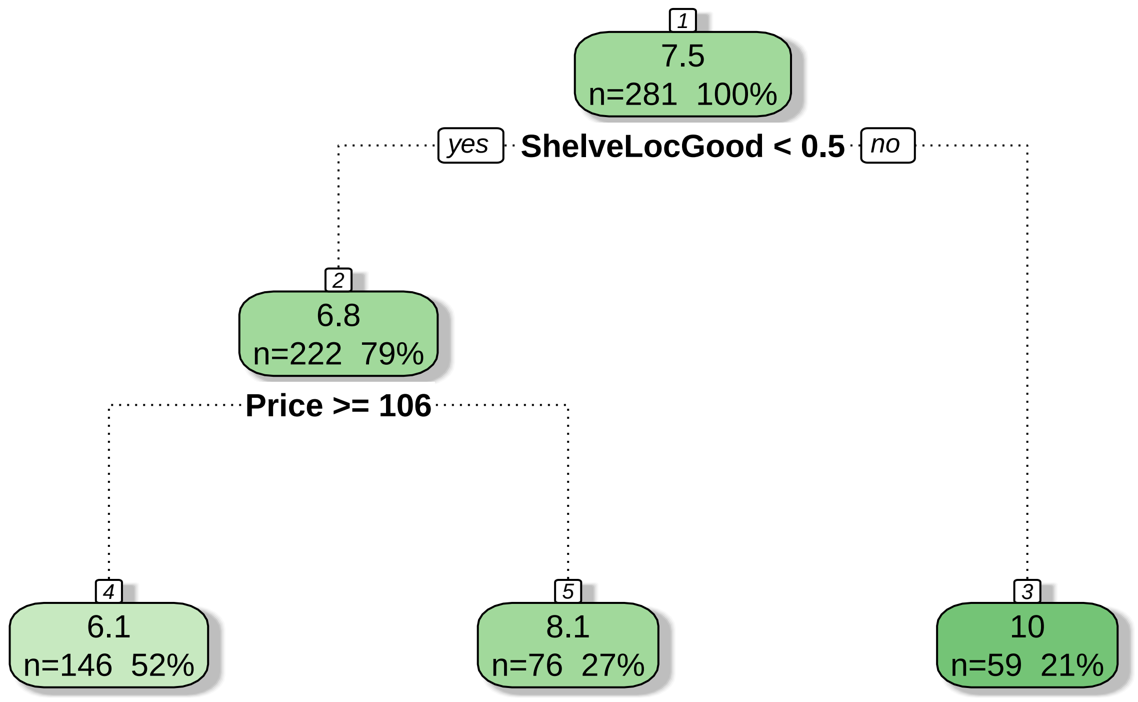
```
plot(treefit)
```



```
treefit$finalModel
```

```
## n= 281
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 281 2161.1110  7.492456
##   2) ShelveLocGood< 0.5 222 1251.2480  6.751622
##     4) Price>=105.5 146  670.6231  6.051712 *
##     5) Price< 105.5 76  371.7074  8.096184 *
##   3) ShelveLocGood>=0.5 59  329.5674 10.280000 *
```

```
fancyRpartPlot(treefit$finalModel)
```



Rattle 2020−Nov−20 23:57:01 annapiskun

```
#we pruned the tree to only include 5 nodes

predTREE <- predict(treefit, newdata = car.test)

MSE.prune <- sum((car.test$Sales - predTREE)^2)/n_val
MSE.prune
```

```
## [1] 5.717196
```

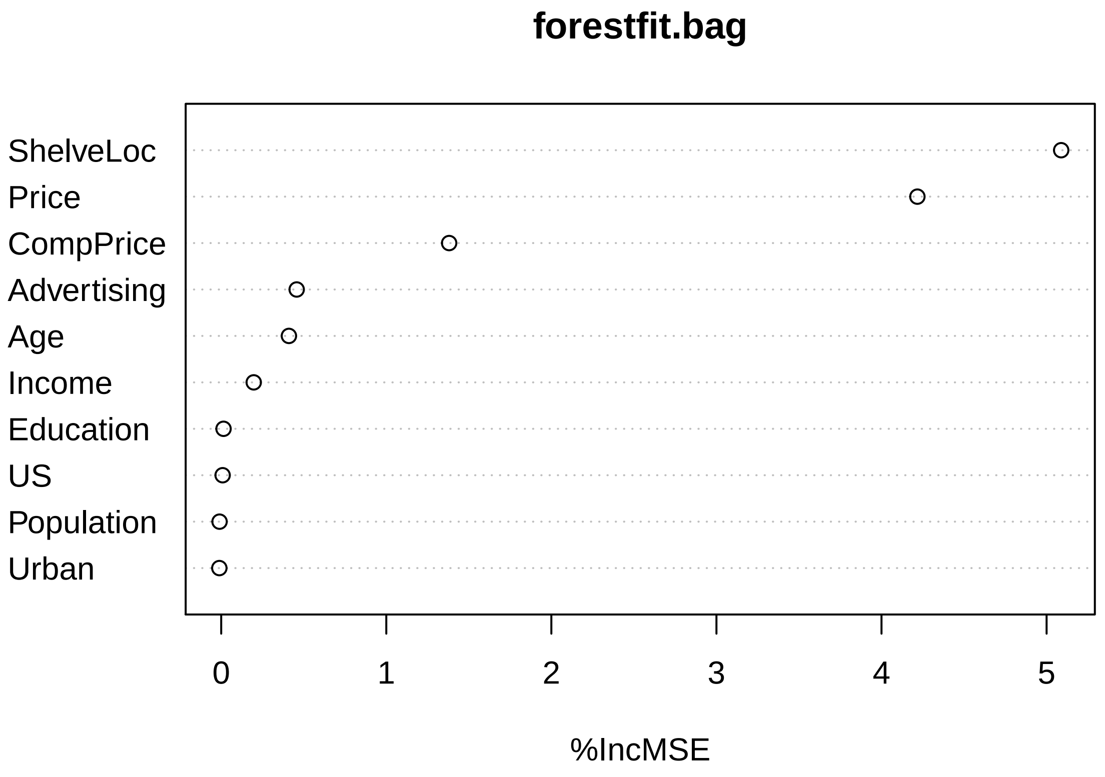
After pruning the tree to only include 5 nodes using cross-validation to determine the optimal level of tree complexity, we find that the test MSE increases to 5.717196. Thus, we see that, in this case, pruning the tree does not improve the test MSE.

**(d) Use the bagging approach in order to analyze this data. What test MSE do you obtain? Use the importance() function to determine which variables are most important.**

```
library(randomForest)
```

```
## randomForest 4.6-14

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:rattle':
##
##     importance

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
set.seed(123)

#using bagging approach means m = p (there are 10 total predictors)

forestfit.bag <- randomForest(Sales~.,
                              data = car.training, mtry = 10,
                              ntree = 500, importance = TRUE)
varImpPlot(forestfit.bag, type = 1, scale = F)
```



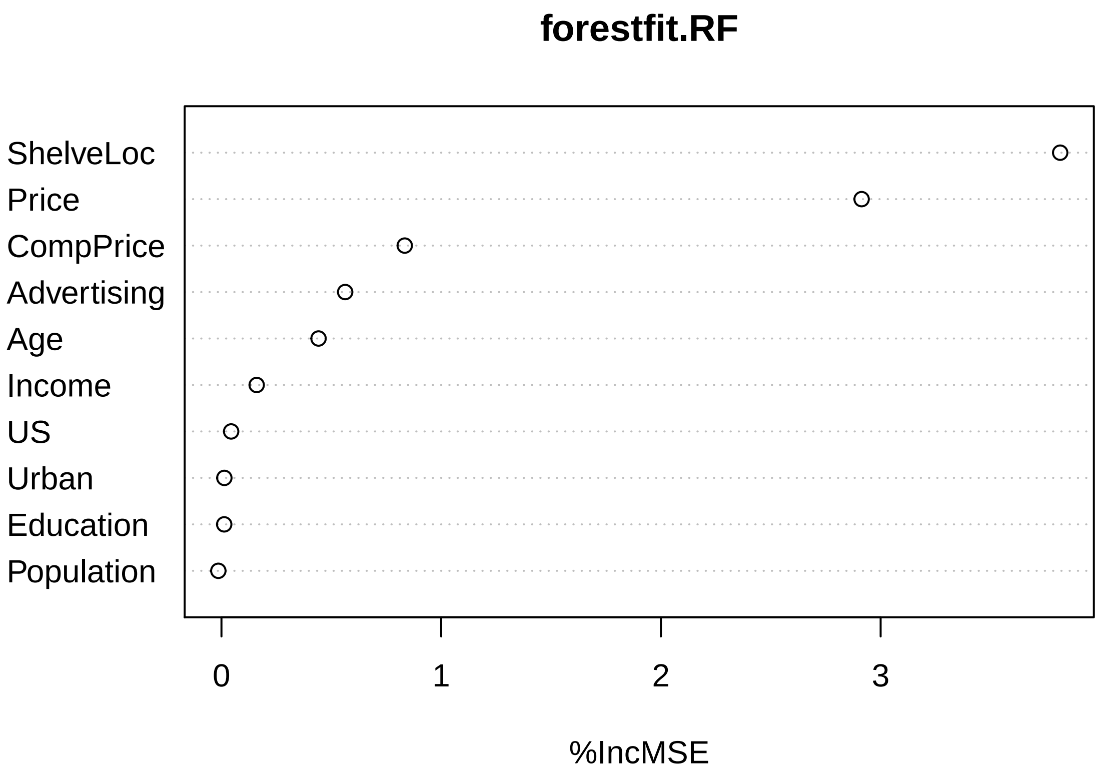
```
predictions <- predict(forestfit.bag, newdata = car.test)

MSE.bag <- sum((car.test$Sales - predictions)^2)/n_val
MSE.bag
```

```
## [1] 2.506079
```

Looking at our plot of variables, ShelveLoc and Price are significantly more important with CompPrice and Advertising following behind. Thus, we find that the price of car seats and the shelving location quality for the car seats are the two most important predictors in our model. The test MSE for this bagged regression tree is 2.506079 which is substantially lower than both our initial regression tree and pruned regression tree.

**(e) Use random forests to analyze this data. What test MSE do you obtain? Use the importance() function to determine which variables are most important. Describe the effect of m, the number of variables considered at each split, on the error rate obtained.**

```
library(randomForest)
set.seed(123)
recommended.mtry <- floor(sqrt(ncol(car.training)))

forestfit.RF <- randomForest(Sales~.,
                       data = car.training, mtry = recommended.mtry,
                       ntree = 500, importance = TRUE)
varImpPlot(forestfit.RF, type = 1, scale = F)
```



```
predRF <- predict(forestfit.RF, newdata = car.test)
MSE.RF <- sum((car.test$Sales - predRF)^2)/n_val
MSE.RF
```

```
## [1] 2.84
```

Using random forests to analyze this data, we obtain a test MSE of 2.84. This test MSE is slightly larger than the earlier test MSE we found using a bagging approach. Thus, we find that random forests do not perform better than bagging in this instance. When using a random forests approach, we go through all possible splits ona random sample of a small number of variables m, where m $<$ p. This way random forests reduce variability further and not so dominant predictors will get a chance to appear by themselves. In the bagging approach we used m = p = 10, but in the random forest approach we used m = 3. Our test MSE increased from bagging to random forest, so as m increased so did our test MSE in this case.