STAT 716 - Class 11 2016-11-26

Vitaly Druker

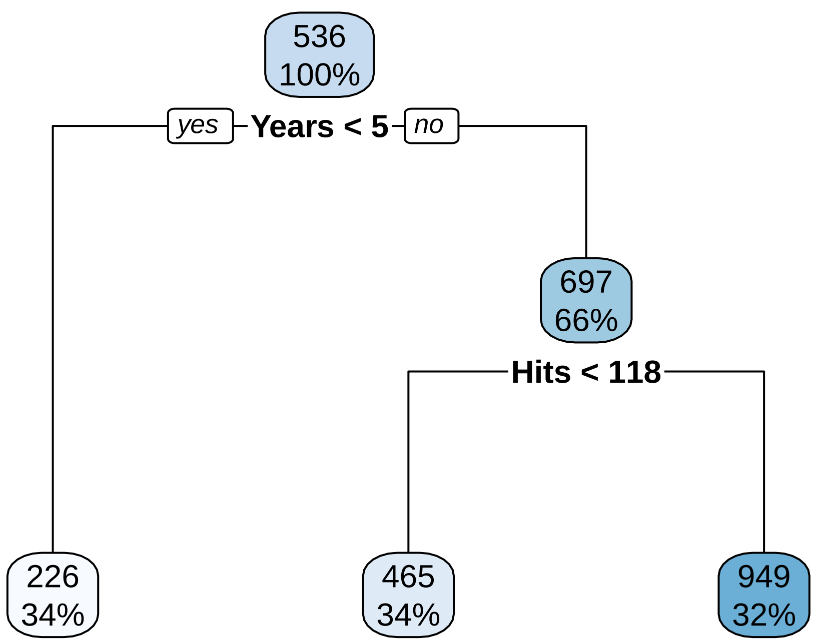
Tree Based Methods

This is a method of splitting up the predictor space into sections.

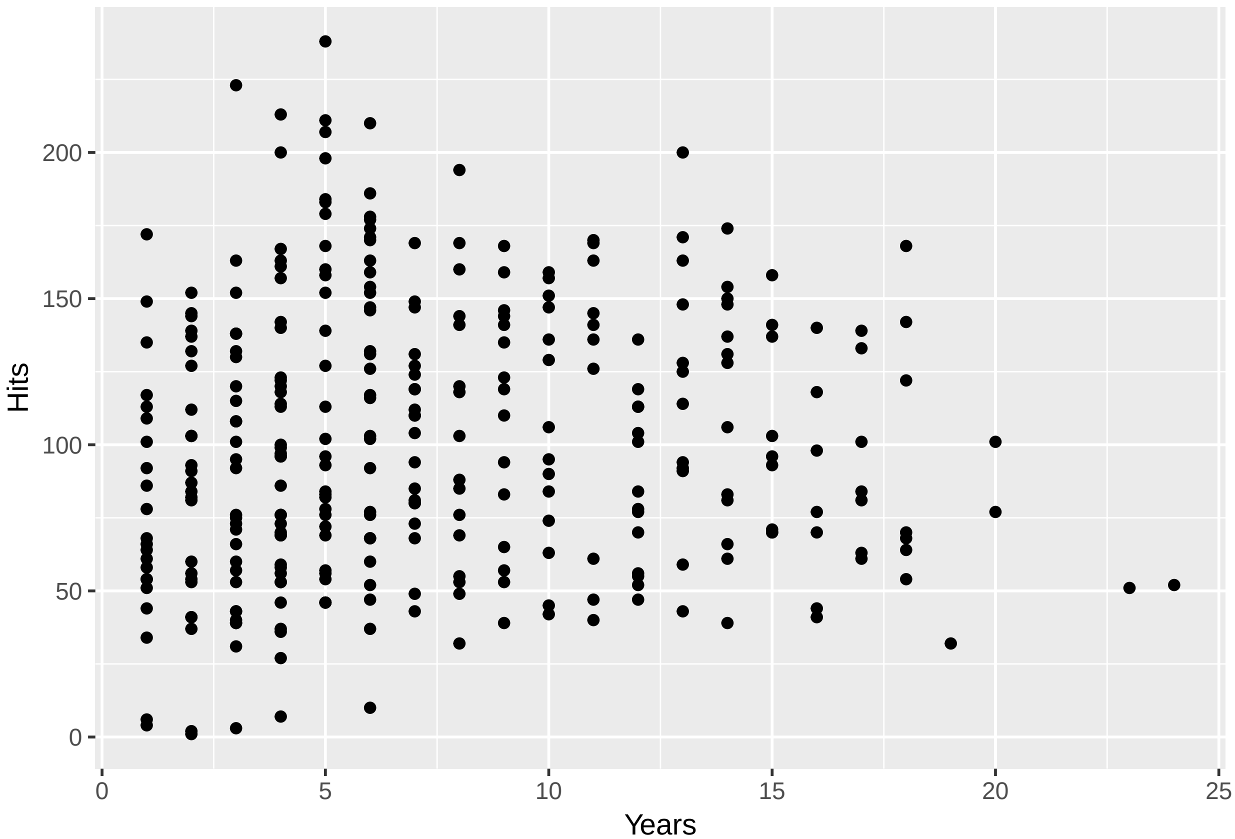
```
library(rpart)
library(rpart.plot)
library(randomForest)
```

```
## randomForest 4.6-14

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

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

```
rpart(Salary ~ Years + Hits, Hitters, control = rpart.control(maxdepth = 2)) %>%
  rpart.plot()
```




```
Hitters %>%
  ggplot(aes(x = Years, y = Hits)) + geom_point()
```



Node - is a decision point

Terminal Node/Leaf - the final end of the tree or the predictor Internal Node

How does prediction work?

How do we make a tree?

1. Divide the predictor space into non overlapping regions. This means over *all* predictors.
2. Make a prediction for everyone that falls into a bucket

Ideally we could look at every single subset of features but that is not computationally feasable (akin to best subset) so we take a top down approach. Using recursive inarty splitting.

1. Pick a predictor and cut into 2 pieces by way of reducing RSS.

We do this by minimizing the joint RSS (it's the same as a step function). What does the RSS look like?

We then split one of those two regions to find the best next split.

This continues until a stoping criteria is reached (there are many different examples of stopping criteria that we will see in the lab)

Unfortunately this is a high variance method - we can't keep going or we will overfit the data.

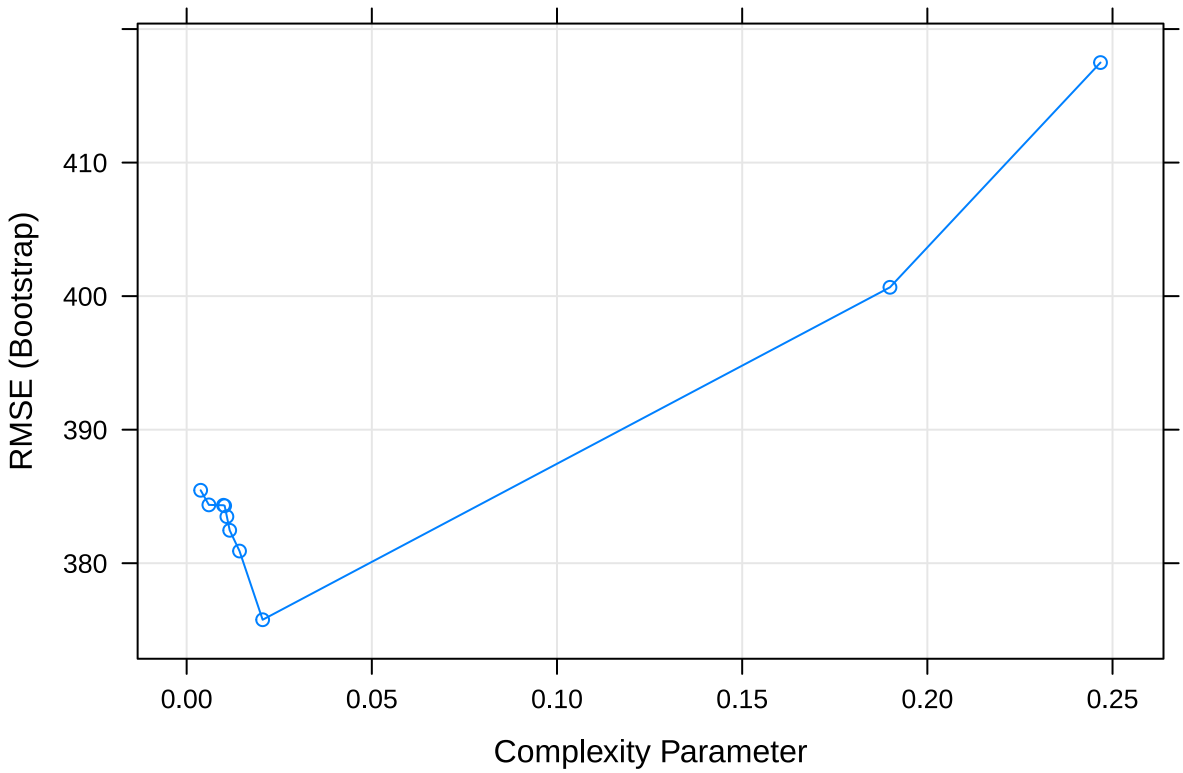
Cost Complexity Pruning

What can we penalize?

Prune back for a value that has a penalty of alpha * |T| (the number of terminal nodes).

```r
d_hit <- Hitters %>%
  select(Salary, Years, Hits) %>%
  filter(complete.cases(.))

train(Salary ~ Years + Hits, d_hit, method = "rpart", tuneLength = 10) %>% plot
```

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



Classification Trees

What error do we use?

Classification Error?

It is not sensative enough Gini Coefficient can be used:

$$G = \sum_{i}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

Cross Entropy

$$\sum_{i}^{K} \hat{p}_{mk} log(\hat{p}_{mk})$$

You can use either Gini or Cross Entropy to build trees - but use classification error to prune them

Benefits

Very easy to explain Nice to display Handle both qual and quant variable easily.

Issues

Hi variability - How does pruning help with variability of the tree? Where is the higher variability?

Dealing with Issues of Trees

Bagging

Bootstrap Aggregation (or bagging) can help methods that have a high variance without loosing too much bias.

OOB Error Estimation - Natural cross validation.

Is the number of bootstraps a tuning parameter?

How do we interpert the models? You can look at the reduction of RSS when a variable is added and average it to see what's most important.

Random Forests

Try to decorrelate the trees by using a random sample of m predictors

$m = \sqrt{p}$

If there is a strong predictor it will show up at the top of each bagged try so we try not to use it every time.

Homework

Read Chapter 8 Chapter 8 question 7

Labs

Basic Trees

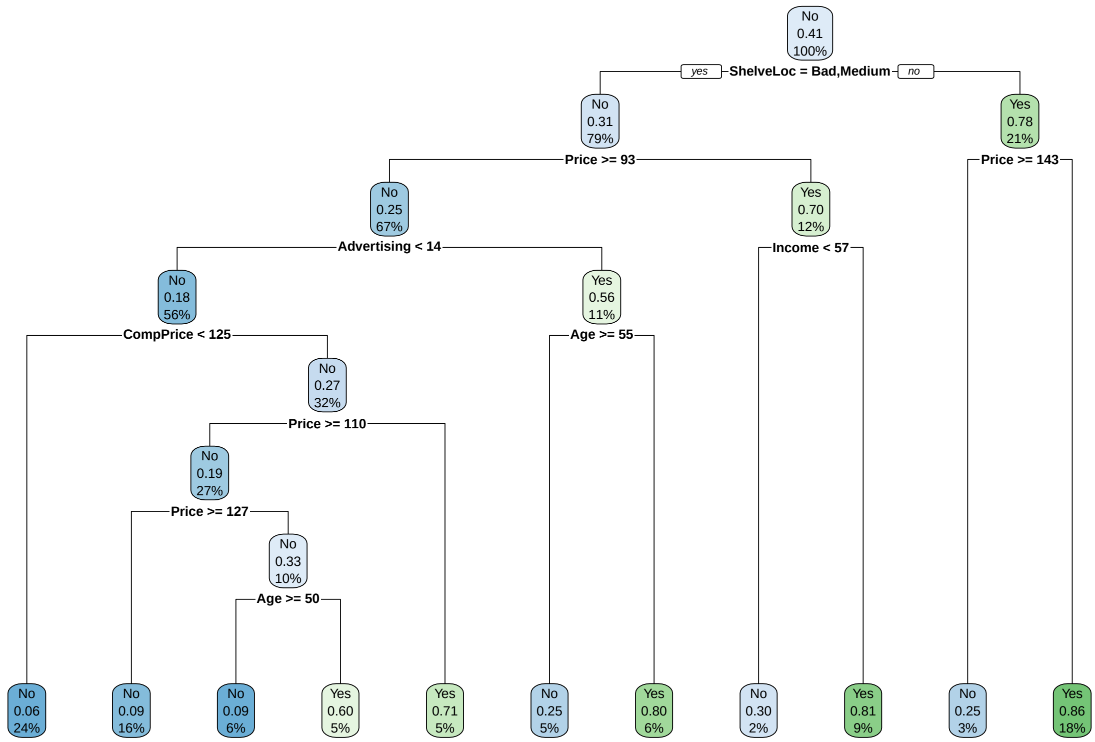
Fitting a regular tree:

I generally use `rpart` instead of `tree`. It works very similarly you can read more here

The largest difference is that you see the parameter cp used

```
d_carseats <- Carseats %>%
  mutate(High = ifelse(Sales > 8, "Yes", "No")) %>%
  mutate(High = as.factor(High))

basic_tree <- rpart(High ~ . - Sales, d_carseats)
```

```
rpart.plot(basic_tree)
```




```
basic_tree$cptable
```

```
##           CP nsplit rel error    xerror       xstd
## 1 0.28658537      0 1.0000000 1.0000000 0.05997967
## 2 0.10975610      1 0.7134146 0.7134146 0.05547692
## 3 0.04573171      2 0.6036585 0.7073171 0.05533684
## 4 0.03658537      4 0.5121951 0.7073171 0.05533684
## 5 0.02743902      5 0.4756098 0.6768293 0.05460552
## 6 0.02439024      7 0.4207317 0.6463415 0.05382112
## 7 0.01219512      8 0.3963415 0.5426829 0.05072258
## 8 0.01000000     10 0.3719512 0.5548780 0.05112415
```

the xerror has not bottomed out yet... let's lower cp

```
set.seed(1298)
full_model <- rpart(High ~ . - Sales, d_carseats, cp = 0.0001, minsplit = 5)
full_model$cptable
```

```
##             CP nsplit  rel error    xerror       xstd
## 1  0.286585366      0 1.00000000 1.0000000 0.05997967
## 2  0.109756098      1 0.71341463 0.7134146 0.05547692
## 3  0.045731707      2 0.60365854 0.6341463 0.05349198
## 4  0.036585366      4 0.51219512 0.6524390 0.05398236
## 5  0.027439024      5 0.47560976 0.6158537 0.05298128
## 6  0.024390244      8 0.39024390 0.5792683 0.05189648
## 7  0.018292683      9 0.36585366 0.5914634 0.05226769
## 8  0.015243902     10 0.34756098 0.5182927 0.04988740
## 9  0.012195122     14 0.28658537 0.5121951 0.04967174
## 10 0.009146341     18 0.23170732 0.5182927 0.04988740
## 11 0.008130081     20 0.21341463 0.5304878 0.05031042
## 12 0.006097561     29 0.14024390 0.5304878 0.05031042
## 13 0.000100000     38 0.08536585 0.5670732 0.05151537
```

```r
pruned_model <- prune(full_model, cp = 0.015243902)

# now let's train/test split...
```

It can be tedious toi rewrite everything with the test train methodology and make it repeatable. There are some different frameworks you can use in R that can improve your workflow

```r
library(modelr)

bestCP <- function(rpart_obj){
  cptable <- rpart_obj$cptable
  best_row <-which.min(cptable[,"xerror"])
  best_cp_sd <- cptable[best_row, "xerror"] + cptable[best_row, "xstd"]
  first_that_passes <- which(cptable[,"xerror"] < best_cp_sd)[1]

  cptable[first_that_passes , "CP"]
}

evalHighTreeModel <- function(model, test_data){
  test_data <- as.data.frame(test_data)
  predictions <- predict(model, test_data, type = "class")
  mean(predictions == test_data$High)
}


d_carseats_model <- d_carseats %>%
  # create a test/train split
  crossv_mc(n = 1, test = 0.5) %>%
  #create the main model off of the train data
  mutate(basic_mod = map(train, ~rpart(High ~ . - Sales, data = .x, cp = 0, minsplit = 5))) %>%
  #which was the best cp? using our function
  mutate(best_cp = map_dbl(basic_mod, bestCP)) %>%
  # prune the model
  mutate(pruned_mod = map2(basic_mod, best_cp, ~prune(.x, cp = .y))) %>%
  #evaluate the model
  mutate(model_eval = map2_dbl(basic_mod, test, ~evalHighTreeModel(.x, .y)))



evalHighTreeModel <- function(model, test_data){
  test_data <- as.data.frame(test_data)
  #have to modify evaluation function because caret standardizes type argument to prob or raw
  predictions <- predict(model, test_data, type = "raw")
  mean(predictions == test_data$High)
}

d_carseats_model_caret <- d_carseats %>%
  # create a test/train split
  crossv_mc(n = 10, test = 0.5) %>%
  #train statement - can be simpler but we tried to make it similar
  # to above
  mutate(basic_mod = map(train, ~train(High ~ . - Sales,
                                       data = as.data.frame(.x),
                                       method = "rpart",
```

```
                                            tuneLength = 5,
                                           trControl = trainControl(method = "cv",
                                                         selectionFunction = "oneSE")))) %>%
  mutate(model_eval = map2_dbl(basic_mod, test, ~evalHighTreeModel(.x, .y)))

d_carseats_model_caret
```

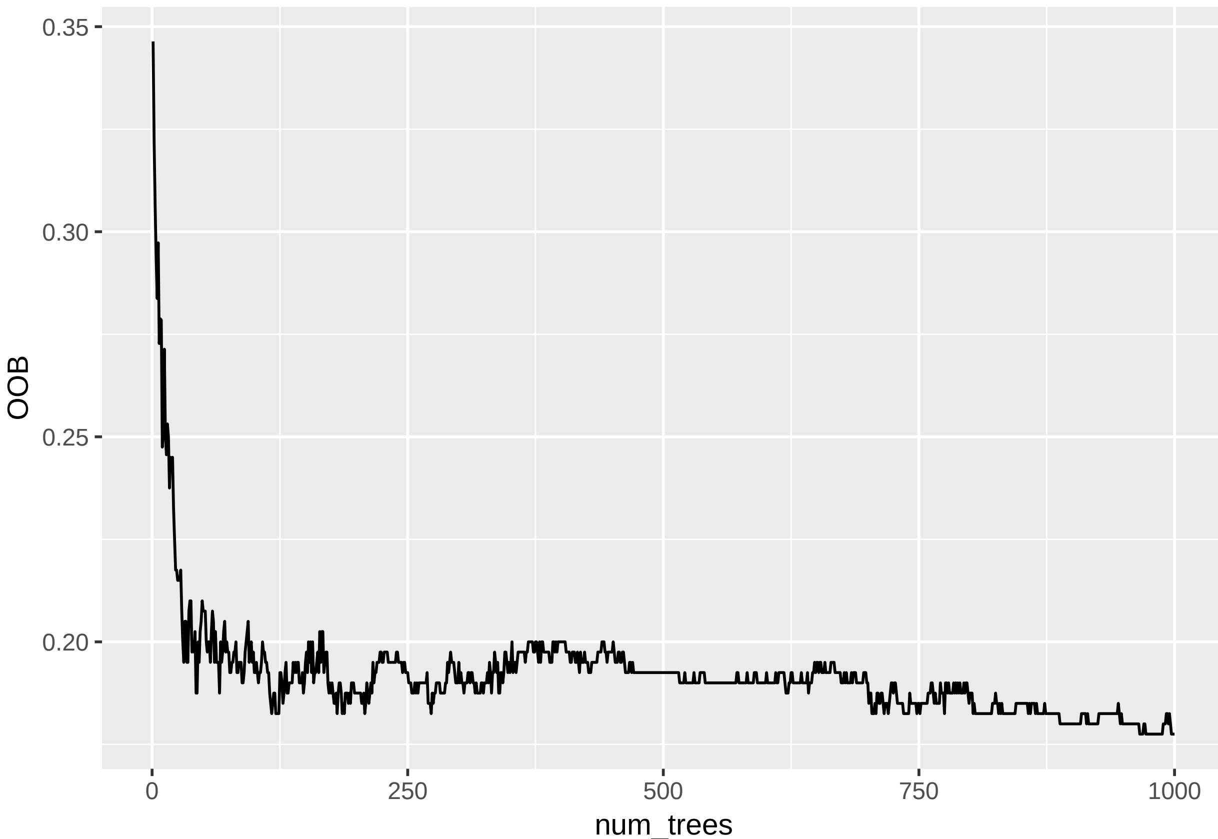
```
## # A tibble: 10 x 5
##    train          test           .id   basic_mod   model_eval
##    <list>         <list>         <chr> <list>           <dbl>
##  1 <S3: resample> <S3: resample> 01    <S3: train>      0.716
##  2 <S3: resample> <S3: resample> 02    <S3: train>      0.751
##  3 <S3: resample> <S3: resample> 03    <S3: train>      0.706
##  4 <S3: resample> <S3: resample> 04    <S3: train>      0.701
##  5 <S3: resample> <S3: resample> 05    <S3: train>      0.721
##  6 <S3: resample> <S3: resample> 06    <S3: train>      0.786
##  7 <S3: resample> <S3: resample> 07    <S3: train>      0.751
##  8 <S3: resample> <S3: resample> 08    <S3: train>      0.701
##  9 <S3: resample> <S3: resample> 09    <S3: train>      0.766
## 10 <S3: resample> <S3: resample> 10    <S3: train>      0.692
```

Random Forests

```
library(randomForest)
set.seed(1238)

ref_mod <- randomForest(High ~ . - Sales, d_carseats)

dim(d_carseats)
```

```
## [1] 400  12
```

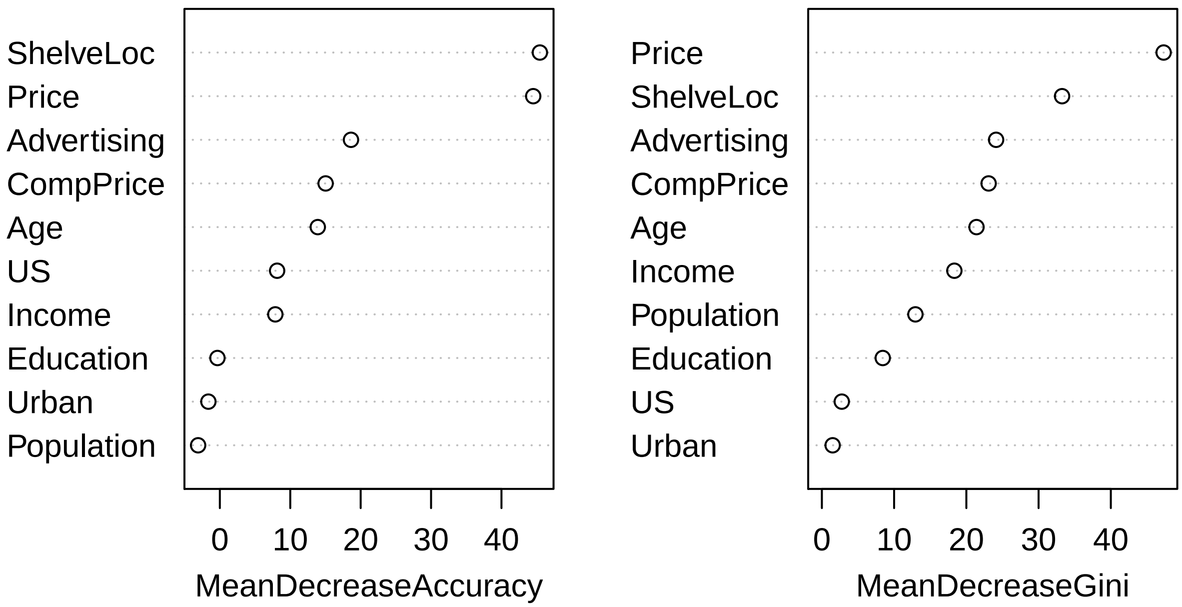
```
ref_mod <- randomForest(High ~ . - Sales, d_carseats, ntree = 1000)


# Error rate by tree number

ref_mod$err.rate %>%
  as.data.frame() %>%
  mutate(num_trees = row_number()) %>%
  ggplot(aes(x = num_trees, y = OOB)) +
  geom_line()
```



Variable Importance Plot

```
randomForest(High ~ . - Sales, d_carseats, mtry = 5, importance = TRUE) %>%
  varImpPlot()
```

.




```
getModelInfo("rf")[[1]]$parameters
```

```
##   parameter   class                         label
## 1      mtry numeric #Randomly Selected Predictors
```

```
train(High ~ . - Sales, d_carseats, method= "rf", trControl = trainControl(method = "cv"))
```

```
## Random Forest
##
## 400 samples
##  11 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 359, 360, 361, 361, 360, 359, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.8202642  0.6188508
##    6    0.8027580  0.5844886
##   11    0.8026360  0.5862315
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
d_carseats_model_caret <- d_carseats %>%
  # create a test/train split
  crossv_mc(n = 10, test = 0.5) %>%
  #train statement - can be simpler but we tried to make it similar
  # to above
```

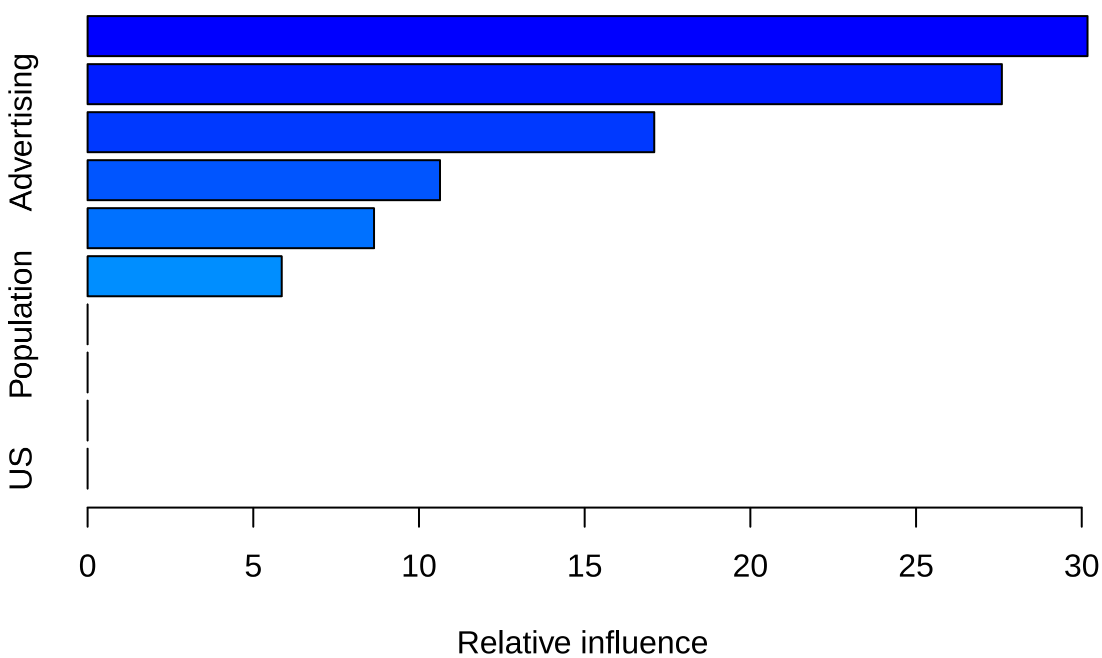
```
  mutate(basic_mod = map(train, ~train(High ~ . - Sales,
                                      data = as.data.frame(.x),
                                      method = "rpart",
                                      tuneLength = 5,
                                     trControl = trainControl(method = "cv",
                                                  selectionFunction = "oneSE")))) %>%
  mutate(model_eval = map2_dbl(basic_mod, test, ~evalHighTreeModel(.x, .y)))
```

Boosting

```
library(gbm)
```

```
## Loaded gbm 2.1.4
```

```
try(
  boost_mod <- gbm(High ~ . - Sales, data = d_carseats, distribution = "bernoulli", verbose = FALSE)
)
d_carseats_gbm <- d_carseats %>%
  mutate(High = as.numeric(High == "Yes"))

boost_mod <- gbm(High ~ . - Sales, data = d_carseats_gbm, distribution = "bernoulli", verbose = FALSE)

summary(boost_mod)
```




```
##                     var   rel.inf
## Price             Price 30.174528
## ShelveLoc     ShelveLoc 27.589793
## Advertising Advertising 17.099489
## CompPrice     CompPrice 10.634090
## Age                 Age  8.643395
## Income           Income  5.858704
## Population   Population  0.000000
```

```
## Education     Education  0.000000
## Urban             Urban  0.000000
## US                   US  0.000000
```

Let's try with caret…

```r
getModelInfo("gbm", regex = F)[[1]]$parameters
```

```
##           parameter   class                   label
## 1           n.trees numeric   # Boosting Iterations
## 2 interaction.depth numeric          Max Tree Depth
## 3         shrinkage numeric               Shrinkage
## 4    n.minobsinnode numeric Min. Terminal Node Size
```

```r
tune_grid <-
  expand.grid(n.trees = c(100, 500),
              interaction.depth = c(1,4),
              n.minobsinnode = 10,
              shrinkage = c(.2, .01)
              )


d_carseats_model_caret <- d_carseats %>%
  # create a test/train split
  crossv_mc(n = 1, test = 0.5) %>%
  #train statement - can be simpler but we tried to make it similar
  # to above
  mutate(basic_mod = map(train, ~train(High ~ . - Sales,
                                       data = as.data.frame(.x),
                                       method = "gbm",
                                       tuneGrid = tune_grid,
                                       verbose = FALSE,
                                      trControl = trainControl(method = "cv",
                                                   selectionFunction = "oneSE")))) %>%
  mutate(model_eval = map2_dbl(basic_mod, test, ~evalHighTreeModel(.x, .y)))

d_carseats_model_caret
```

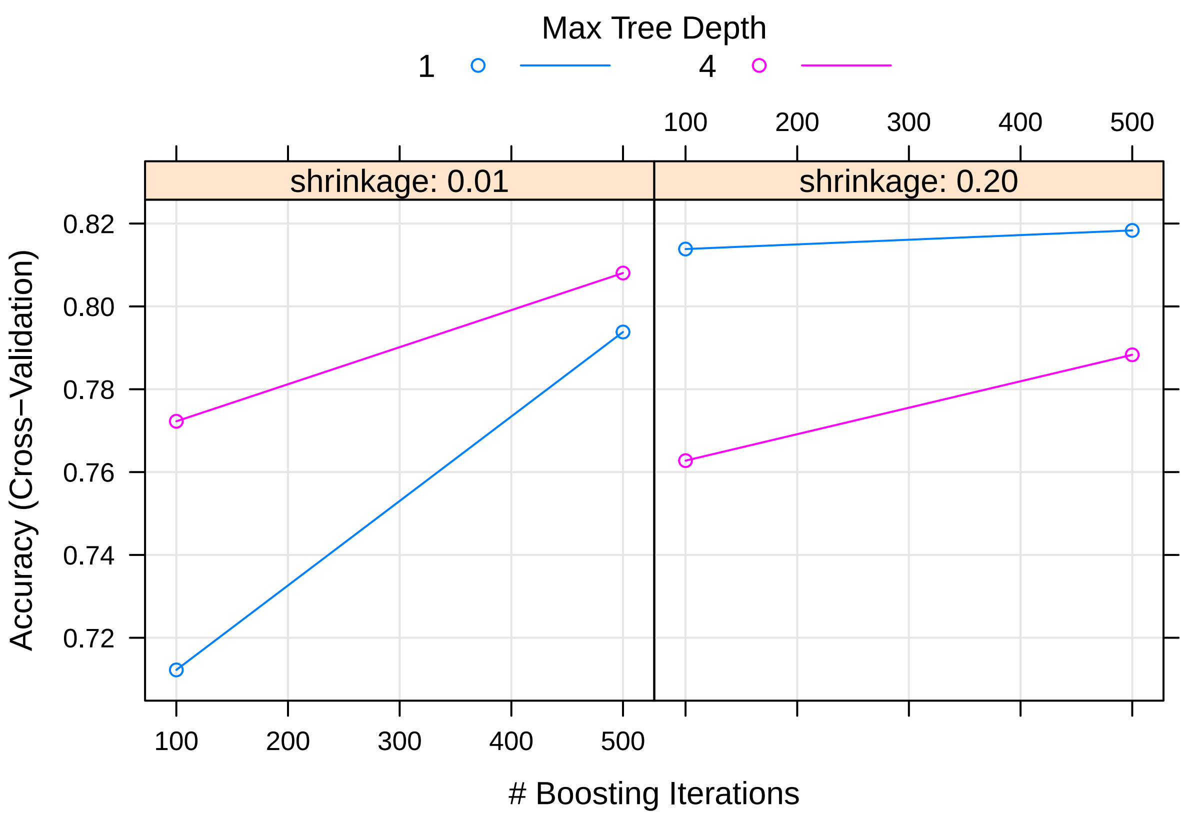
```
## # A tibble: 1 x 5
##   train          test           .id   basic_mod   model_eval
##   <list>         <list>         <chr> <list>           <dbl>
## 1 <S3: resample> <S3: resample> 1     <S3: train>      0.831
```

```r
d_carseats_model_caret$basic_mod[[1]]
```

```
## Stochastic Gradient Boosting
##
## 199 samples
##  11 predictor
##   2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 178, 180, 179, 179, 179, 179, ...
## Resampling results across tuning parameters:
##
##   shrinkage  interaction.depth  n.trees  Accuracy   Kappa
```

```
##   0.01       1                  100      0.7122431  0.4012004
##   0.01       1                  500      0.7938221  0.5789993
##   0.01       4                  100      0.7722682  0.5355571
##   0.01       4                  500      0.8080576  0.6109492
##   0.20       1                  100      0.8138471  0.6246630
##   0.20       1                  500      0.8183459  0.6326855
##   0.20       4                  100      0.7627694  0.5225834
##   0.20       4                  500      0.7883208  0.5711346
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using  the one SE rule.
## The final values used for the model were n.trees = 100,
##  interaction.depth = 1, shrinkage = 0.2 and n.minobsinnode = 10.
```

```
d_carseats_model_caret$basic_mod[[1]] %>% plot()
```




```
tune_grid <-
  expand.grid(n.trees = c(50, 100, 500),
              interaction.depth = c(1,4),
              n.minobsinnode = 10,
              shrinkage = c(.2, .01)
              )


d_carseats_model_caret <- d_carseats %>%
  # create a test/train split
  crossv_mc(n = 1, test = 0.5) %>%
  #train statement - can be simpler but we tried to make it similar
  # to above
```

```
  mutate(basic_mod = map(train, ~train(High ~ . - Sales,
                                       data = as.data.frame(.x),
                                       method = "gbm",
                                       verbose = FALSE,
                                       tuneGrid = tune_grid,
                                      trControl = trainControl(method = "cv")))) %>%
  mutate(model_eval = map2_dbl(basic_mod, test, ~evalHighTreeModel(.x, .y)))

d_carseats_model_caret$basic_mod[[1]] %>% plot
```