Machine Learning 2019: Tree-Based Methods

Sonali Narang

10/28/2019

Tree-Based Methods

Decision tree is a type of supervised learning algorithm that can be used in both regression and classification problems. Tree-based methods works for both categorical and continuous input and output variables.

The Carseats Dataset

400 Observations, 11 variables Response Variable: Sales/High

```
data("Carseats")
carseats = Carseats
head(carseats)
```

```
##   Sales CompPrice Income Advertising Population Price ShelveLoc Age
## 1  9.50       138     73          11        276   120       Bad  42
## 2 11.22       111     48          16        260    83      Good  65
## 3 10.06       113     35          10        269    80    Medium  59
## 4  7.40       117    100           4        466    97    Medium  55
## 5  4.15       141     64           3        340   128       Bad  38
## 6 10.81       124    113          13        501    72       Bad  78
##   Education Urban  US
## 1        17   Yes Yes
## 2        10   Yes Yes
## 3        12   Yes Yes
## 4        14   Yes Yes
## 5        13   Yes  No
## 6        16    No Yes
```

```
#convert quantitative variable Sales into a binary response
High = ifelse(carseats$Sales<=8, "No", "Yes")
carseats = data.frame(carseats, High)

head(carseats)
```

```
##   Sales CompPrice Income Advertising Population Price ShelveLoc Age
## 1  9.50       138     73          11        276   120       Bad  42
## 2 11.22       111     48          16        260    83      Good  65
## 3 10.06       113     35          10        269    80    Medium  59
## 4  7.40       117    100           4        466    97    Medium  55
## 5  4.15       141     64           3        340   128       Bad  38
## 6 10.81       124    113          13        501    72       Bad  78
##   Education Urban  US High
## 1        17   Yes Yes  Yes
## 2        10   Yes Yes  Yes
## 3        12   Yes Yes  Yes
## 4        14   Yes Yes   No
## 5        13   Yes  No   No
## 6        16    No Yes  Yes
```

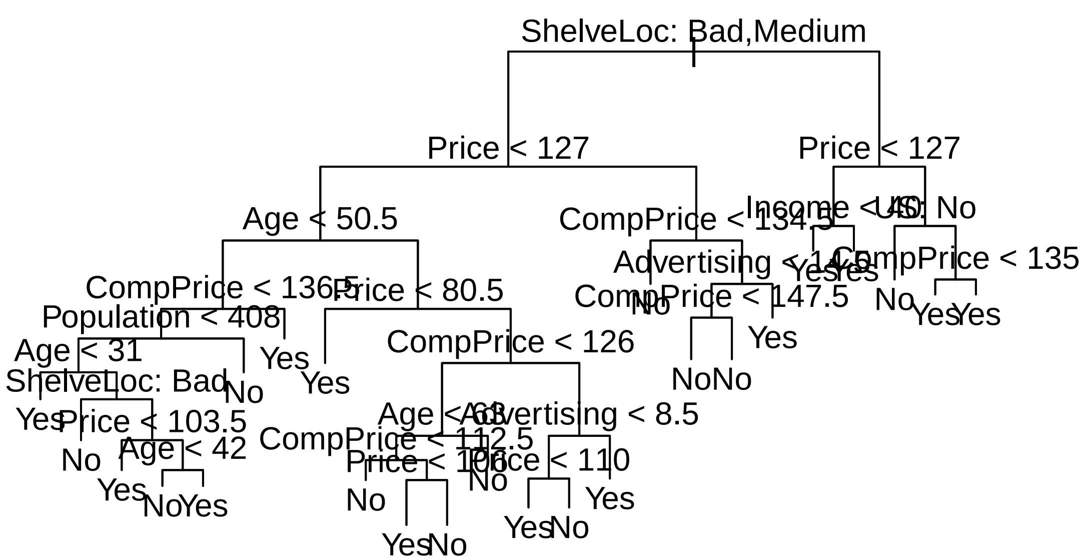
Classification Tree

Input variables (X) can be continuous or categorical. Response variable (Y) is categorical (usually binary): in this case Sales/High.

```r
#set seed to make results reproducible
set.seed(29)

#split data into train and test subset (250 and 150 respectively)
train = sample(1:nrow(carseats), 250)

#Fit train subset of data to model
tree.carseats = tree(High~.-Sales, carseats, subset=train)
summary(tree.carseats)
```

```
##
## Classification tree:
## tree(formula = High ~ . - Sales, data = carseats, subset = train)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Age"         "CompPrice"   "Population"
## [6] "Advertising" "Income"      "US"
## Number of terminal nodes:  24
## Residual mean deviance:  0.3436 = 77.65 / 226
## Misclassification error rate: 0.072 = 18 / 250
```

```r
#Visualize tree
plot(tree.carseats)
text(tree.carseats, pretty=0)
```




```r
#each of the terminal nodes are labeled Yes or No. The variables and the value of the splitting choice

#Use model on test set, predict class labels
tree.pred = predict(tree.carseats, carseats[-train,], type="class")

#Misclassification table to evaluate error
with(carseats[-train,], table(tree.pred, High))
```

```
##          High
## tree.pred No Yes
```

```
##       No  71  20
##       Yes 17  42
```

```r
#Calculate error by summing up the diagonals and dividing by number of total predictions
mc = (71 + 42) / 150
mc
```

```
## [1] 0.7533333
```

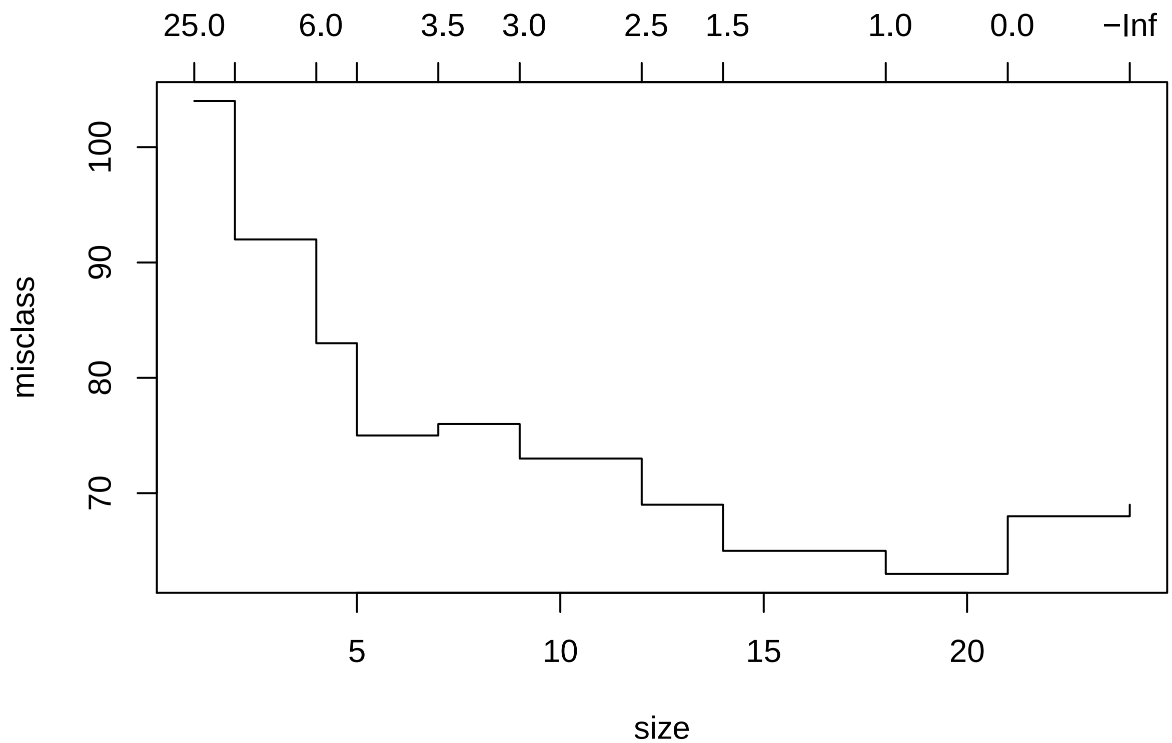
Pruning using cross-validation

Pruning is a method to cut back the tree to prevent over-fitting.

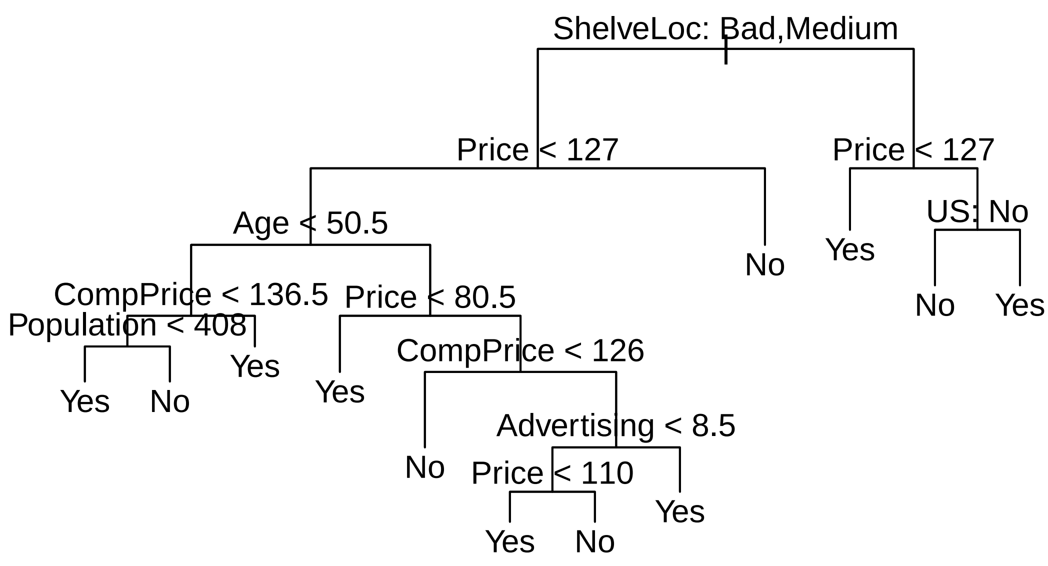
```r
#cross-validation to prune the tree using cv.tree
cv.carseats = cv.tree(tree.carseats, FUN = prune.misclass)

#Sizes of the trees as they were pruned back, the deviances as the pruning proceeded, and cost complexi
cv.carseats
```

```
## $size
##  [1] 24 21 18 14 12  9  7  5  4  2  1
##
## $dev
##  [1]  69  68  63  65  69  73  76  75  83  92 104
##
## $k
##  [1] -Inf  0.0  1.0  1.5  2.5  3.0  3.5  4.0  6.0  7.5 25.0
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```r
#Visualize
plot(cv.carseats)
```




```r
#Prune tree to a size of 12
prune.carseats = prune.misclass(tree.carseats, best = 12)

#Visualize tree
plot(prune.carseats)
text(prune.carseats, pretty=0)
```




```r
#Evaluate on test set
tree.pred = predict(prune.carseats, carseats[-train,], type="class")

#Misclassification
with(carseats[-train,], table(tree.pred, High))
```

```
##          High
## tree.pred No Yes
##       No  66  21
##       Yes 22  41
```

```
#Error
mc_pruning = (66 + 41) / 150
mc_pruning
```

```
## [1] 0.7133333
```

```
##pruning did not increase misclassification error by too much and resulted in a simpler tree!!
```

Pruning did not increase misclassification error by too much and resulted in a simpler tree!!

Decision trees suffer from high variance, meaning if you split the training data into 2 parts at random, and fit a decision tree to both halves, the results that you get could be very different.

Bagging and boosting are technique used to reduce the variance of your predictions.

The Boston Housing Dataset

506 Observations, 14 variables Response Variable: medv (median value of owner-occupied homes for each suburb)

```
data("Boston")
boston = Boston
head(Boston)
```

```
##      crim zn indus chas   nox    rm  age    dis rad tax ptratio  black
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12
##   lstat medv
## 1  4.98 24.0
## 2  9.14 21.6
## 3  4.03 34.7
## 4  2.94 33.4
## 5  5.33 36.2
## 6  5.21 28.7
```

Bagging: Random Forest

Bagging involves creating multiple copies of the original training dataset using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model. Each tree is built on a bootstrapped dataset, independent of the other trees.

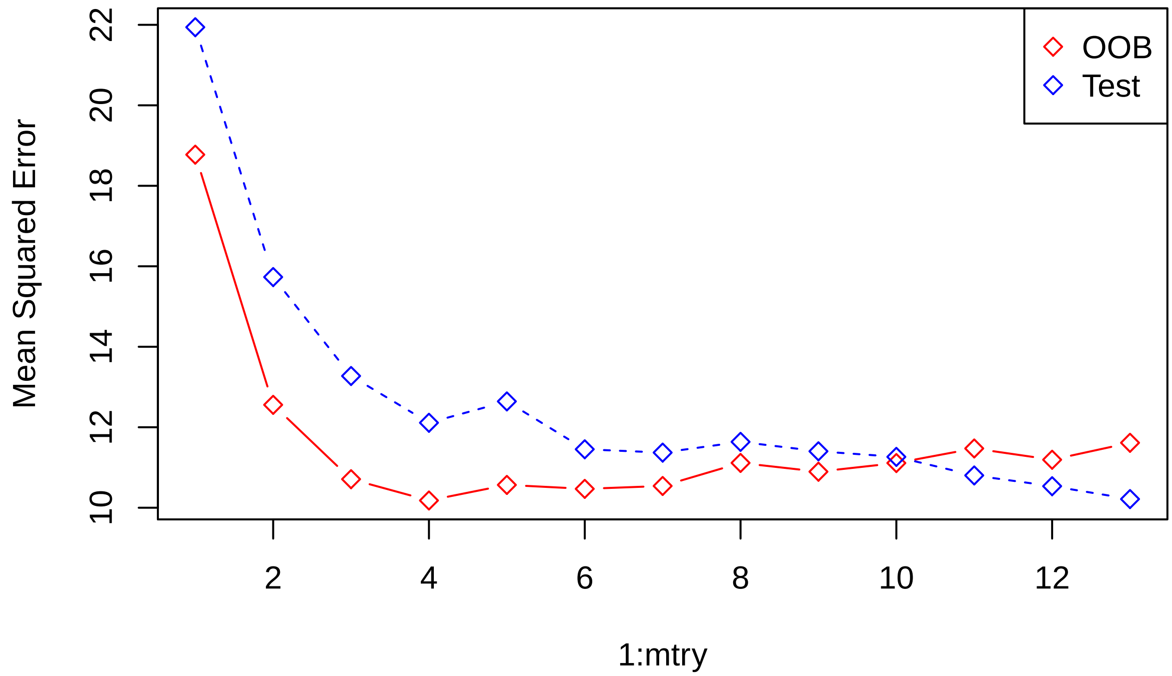
Random Forest: Each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors.

```
#set seed for reproducibility
set.seed(29)

#split into train and test sets (300 and 206 respectively)
train = sample(1:nrow(boston), 300)
```

```r
#fit training subset of data to model
rf.boston = randomForest(medv~., data = boston, subset = train)
rf.boston
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data = boston, subset = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 12.86824
##                     % Var explained: 83.94
```

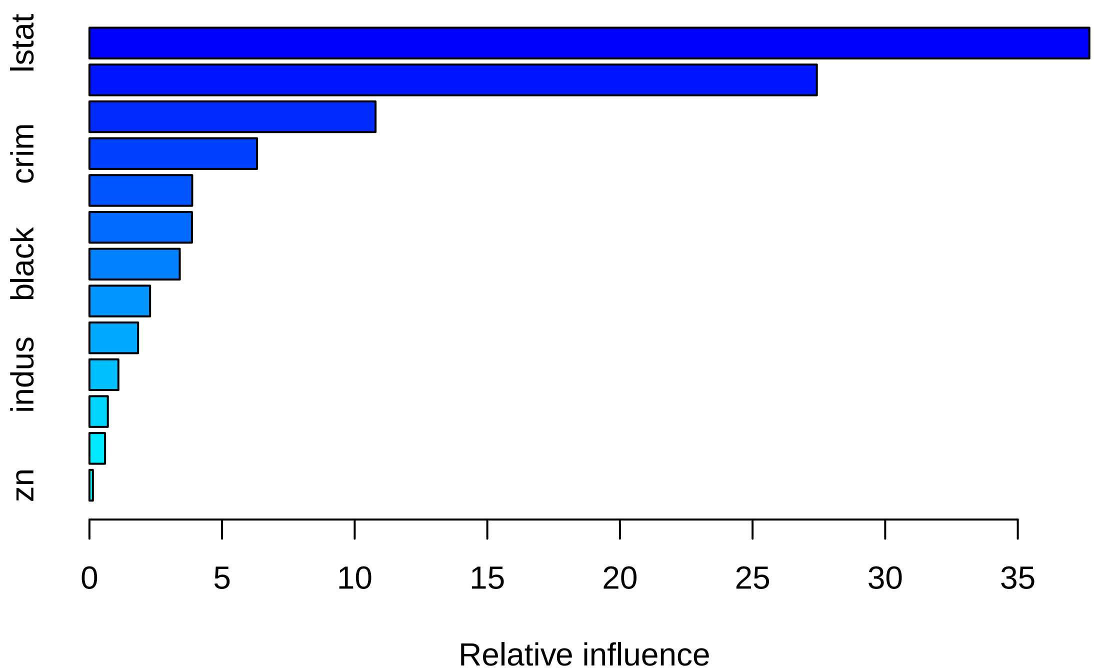
```r
#summary of rf.boston gives information about the number of trees, the mean squared residuals (MSR), an

#No. of variables tried at each split: 4
#Each time the tree comes to split a node, 4 variables would be selected at random, then the split woul

##Lets try a range of mtry (number of variables selected at random at each split)
oob.err = double(13)
test.err = double(13)

#In a loop of mtry from 1 to 13, you first fit the randomForest to the train dataset
for(mtry in 1:13){
  fit = randomForest(medv~., data = boston, subset=train, mtry=mtry, ntree = 350)
  oob.err[mtry] = fit$mse[350] ##extract Mean-squared-error
  pred = predict(fit, boston[-train,]) #predict on test dataset
  test.err[mtry] = with(boston[-train,], mean( (medv-pred)^2 )) #compute test error
}

#Visualize
matplot(1:mtry, cbind(test.err, oob.err), pch = 23, col = c("red", "blue"), type = "b", ylab="Mean Squa
legend("topright", legend = c("OOB", "Test"), pch = 23, col = c("red", "blue"))
```



Boosting

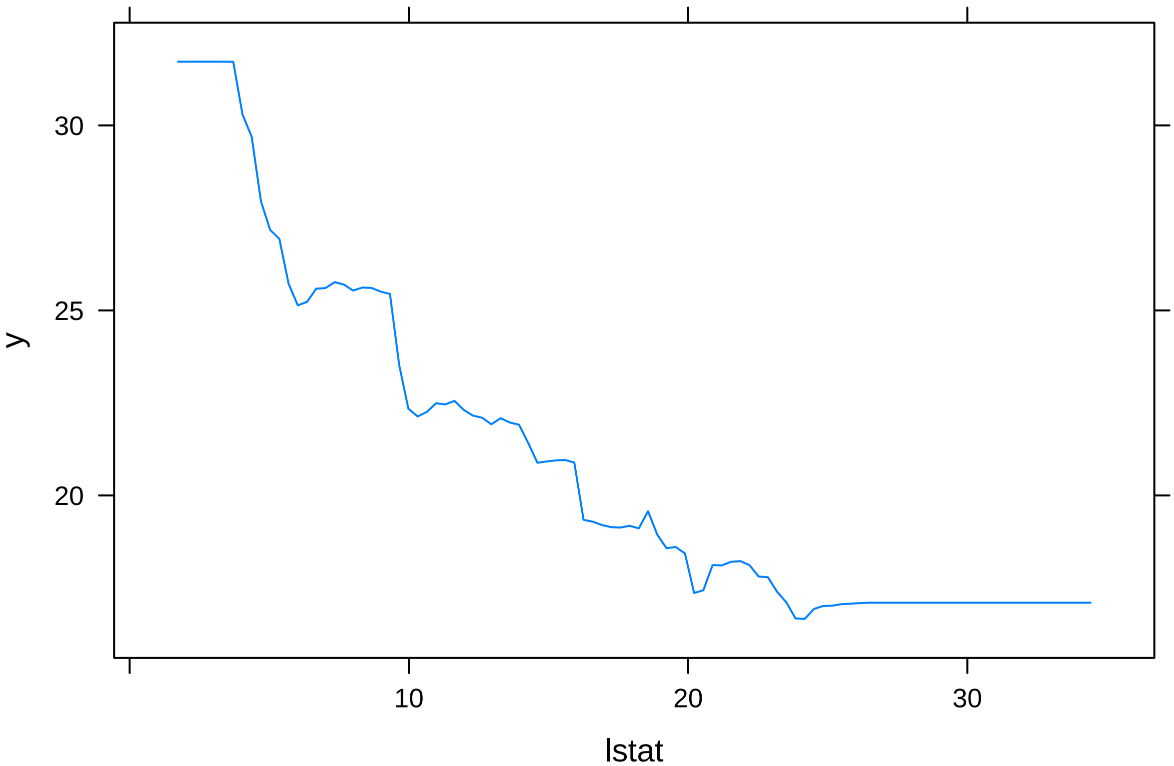
Boosting is another approach to improve the predictions resulting from a decision tree. Trees are grown sequentially: each tree is grown using information from previously grown trees. Each tree is fitted on a modified version of the original dataset.

```
#Gradient Boosting Model
boost.boston = gbm(medv~., data = boston[train,], distribution = "gaussian", n.trees = 10000, shrinkage

#Variable Importance Plot
summary(boost.boston)
```

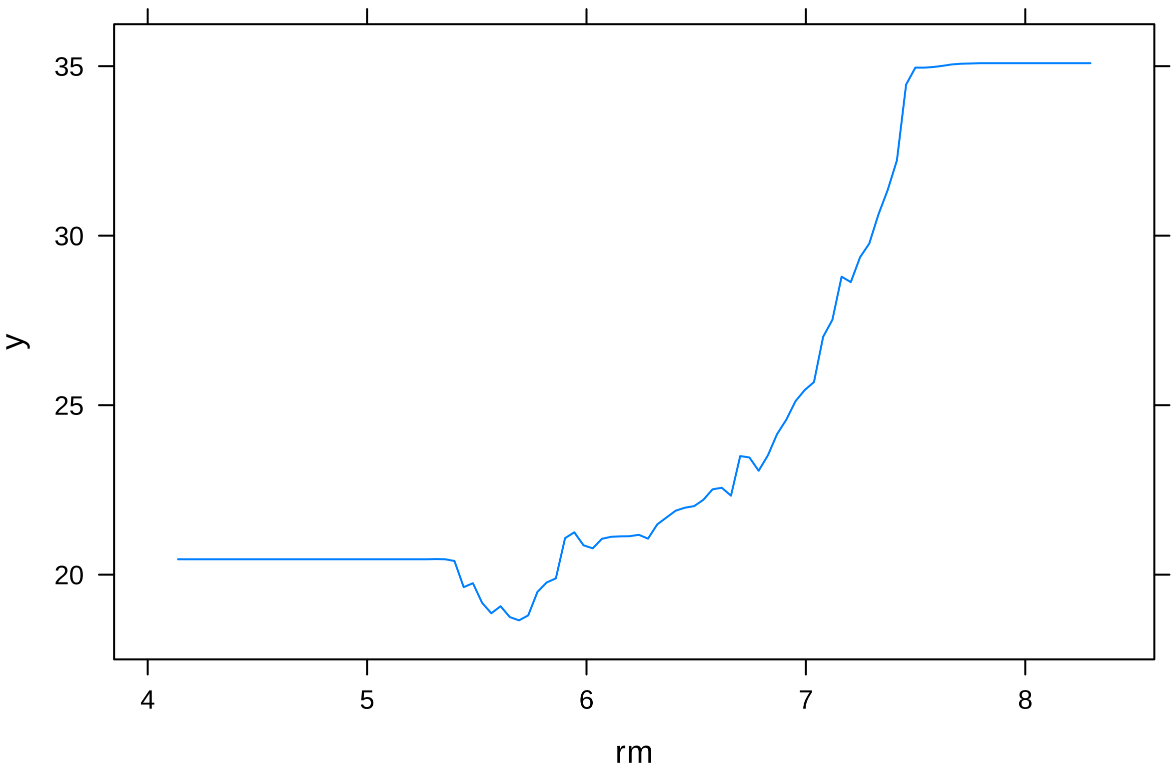


```
##               var    rel.inf
## lstat      lstat 37.6982917
## rm            rm 27.4231506
## dis          dis 10.7870570
## crim        crim  6.3190588
## nox          nox  3.8741970
## age          age  3.8660387
## black      black  3.4055088
## ptratio ptratio  2.2861013
## tax          tax  1.8341376
## indus      indus  1.0918593
## chas        chas  0.6945661
## rad          rad  0.5888764
## zn            zn  0.1311568
```

```r
#Visualize important variables of interest
plot(boost.boston,i="lstat")
```



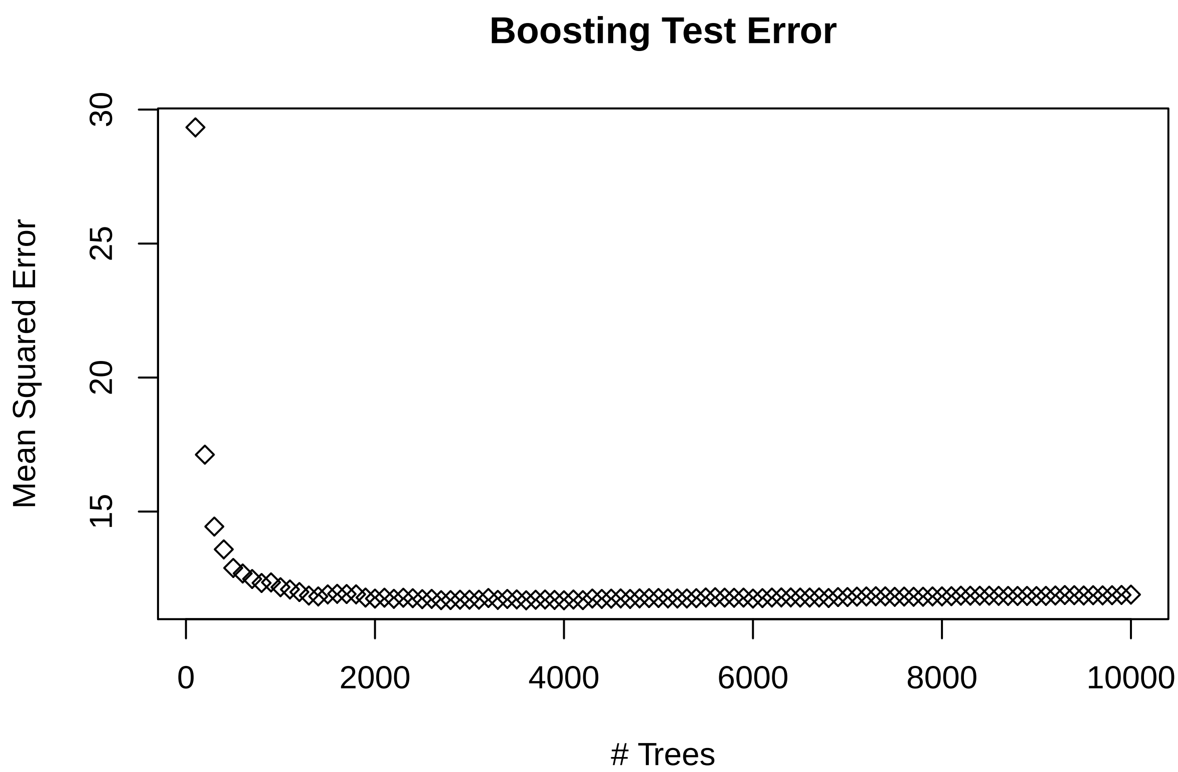

```r
plot(boost.boston,i="rm")
```




```
#Predict on test set
n.trees = seq(from = 100, to = 10000, by = 100)
predmat = predict(boost.boston, newdata = boston[-train,], n.trees = n.trees)
dim(predmat)
```

```
## [1] 206 100
```

```
#Visualize Boosting Error Plot
boost.err = with(boston[-train,], apply( (predmat - medv)^2, 2, mean) )
plot(n.trees, boost.err, pch = 23, ylab = "Mean Squared Error", xlab = "# Trees", main = "Boosting Test
abline(h = min(test.err), col = "red")
```



Homework

1. Attempt a regression tree-based method (not covered in this tutorial) on a reasonable dataset of your choice. Explain the results.

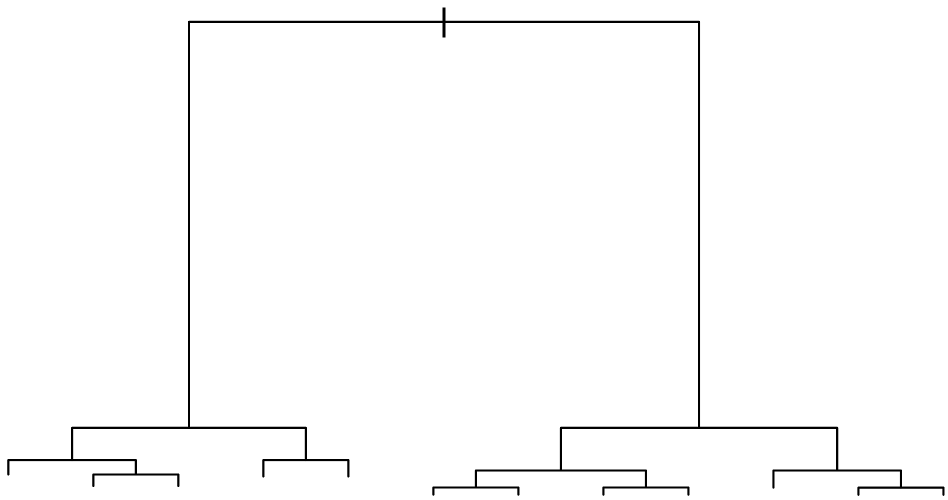
Regression trees can be built for continuous outcomes as well, so the following is a regression tree analysis on the maximum one-hour-avg ozone value in the LA Ozone dataset.

```
##define dataset and explore variables
data(Ozone)
head(Ozone)
```

```
##   V1 V2 V3 V4   V5 V6 V7 V8    V9  V10 V11   V12 V13
## 1  1  1  4  3 5480  8 20 NA    NA 5000 -15 30.56 200
## 2  1  2  5  3 5660  6 NA 38    NA   NA -14    NA 300
## 3  1  3  6  3 5710  4 28 40    NA 2693 -25 47.66 250
## 4  1  4  7  5 5700  3 37 45    NA  590 -24 55.04 100
## 5  1  5  1  5 5760  3 51 54 45.32 1450  25 57.02  60
## 6  1  6  2  6 5720  4 69 35 49.64 1568  15 53.78  60
```

```
##remove missing values
Ozone <- na.omit(Ozone)
```

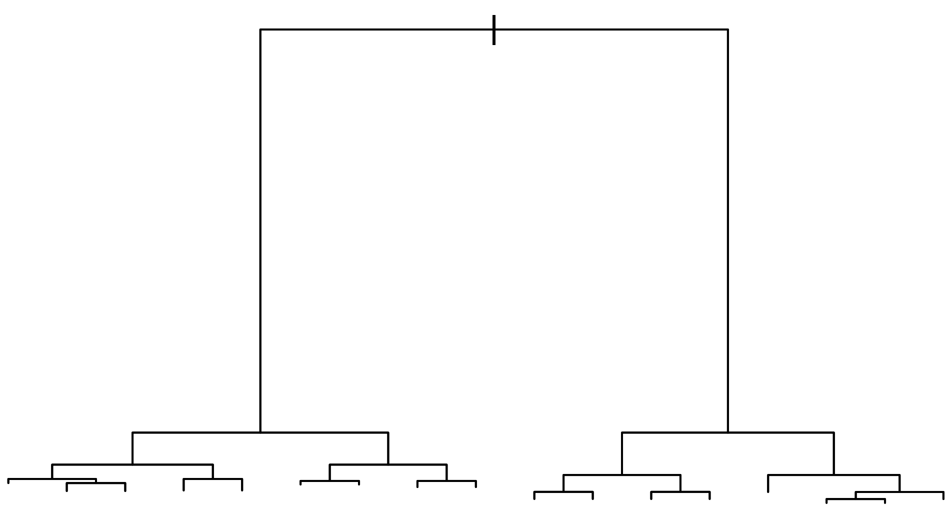
```
##build a regression tree to predict tax based on all other variables in the dataset
tree.model <- tree(V4 ~ ., data=Ozone)
##plot the resulting tree
plot(tree.model)
```



```r
##display residuals to evaluate model quality
summary(tree.model)
```

```
##
## Regression tree:
## tree(formula = V4 ~ ., data = Ozone)
## Variables actually used in tree construction:
## [1] "V9"  "V8"  "V2"  "V1"  "V3"  "V12"
## Number of terminal nodes:  12
## Residual mean deviance:  11.45 = 2187 / 191
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -8.2000 -1.9680 -0.9351  0.0000  1.9120 10.9100
```

```r
##build a second model by adjusting the acceptable error rate reduction from tree growth
tree.model2 <- tree(V4 ~ ., data=Ozone, mindev=0.005)
##plot the resulting tree
plot(tree.model2)
```



```r
##display residuals to evaluate model quality
summary(tree.model2)
```

```
##
```

```
## Regression tree:
## tree(formula = V4 ~ ., data = Ozone, mindev = 0.005)
## Variables actually used in tree construction:
## [1] "V9"  "V8"  "V2"  "V1"  "V12" "V3"
## Number of terminal nodes:  17
## Residual mean deviance:  9.057 = 1685 / 186
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -8.200  -1.700   0.050   0.000   1.333  10.000
```

Decreasing the acceptable error rate reduction in tree growth predictably made for a more complex tree with more nodes. Both chose and used the same set of predictor variables. The mean squared error decreases (11.45 to 9.06) as the tree fits the data more accurately, but results in a model that may be overfit to the training data and suffer in performance on an outside dataset.

2. Attempt both a bagging and boosting method on a reasonable dataset of your choice. Explain the results.

```r
##define dataset and explore variables
data(Ozone)
head(Ozone)
```

```
##   V1 V2 V3 V4   V5 V6 V7 V8    V9  V10 V11   V12 V13
## 1  1  1  4  3 5480  8 20 NA    NA 5000 -15 30.56 200
## 2  1  2  5  3 5660  6 NA 38    NA   NA -14    NA 300
## 3  1  3  6  3 5710  4 28 40    NA 2693 -25 47.66 250
## 4  1  4  7  5 5700  3 37 45    NA  590 -24 55.04 100
## 5  1  5  1  5 5760  3 51 54 45.32 1450  25 57.02  60
## 6  1  6  2  6 5720  4 69 35 49.64 1568  15 53.78  60
```

```r
##remove missing values
Ozone <- na.omit(Ozone)

#set seed for reproducibility
set.seed(29)

#split into train and test sets (100 and 106 respectively)
train = sample(1:nrow(Ozone), 100)

#fit training subset of data to model
rf.ozone = randomForest(V4~., data = Ozone, subset = train)
rf.ozone
```

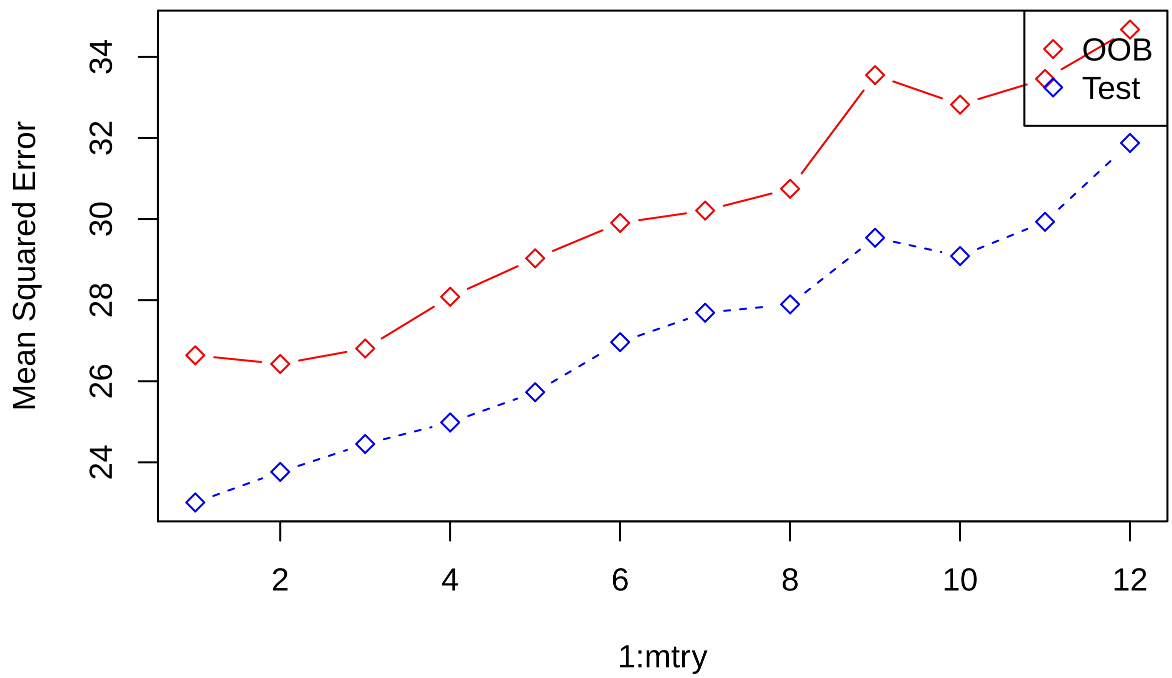
```
##
## Call:
##  randomForest(formula = V4 ~ ., data = Ozone, subset = train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 23.91157
##                     % Var explained: 60.28
```

```r
#No. of variables tried at each split: 4
#Each time the tree comes to split a node, 4 variables would be selected at random, then the split woul

##Restructure dataframe to pull outcome variable to the front
```

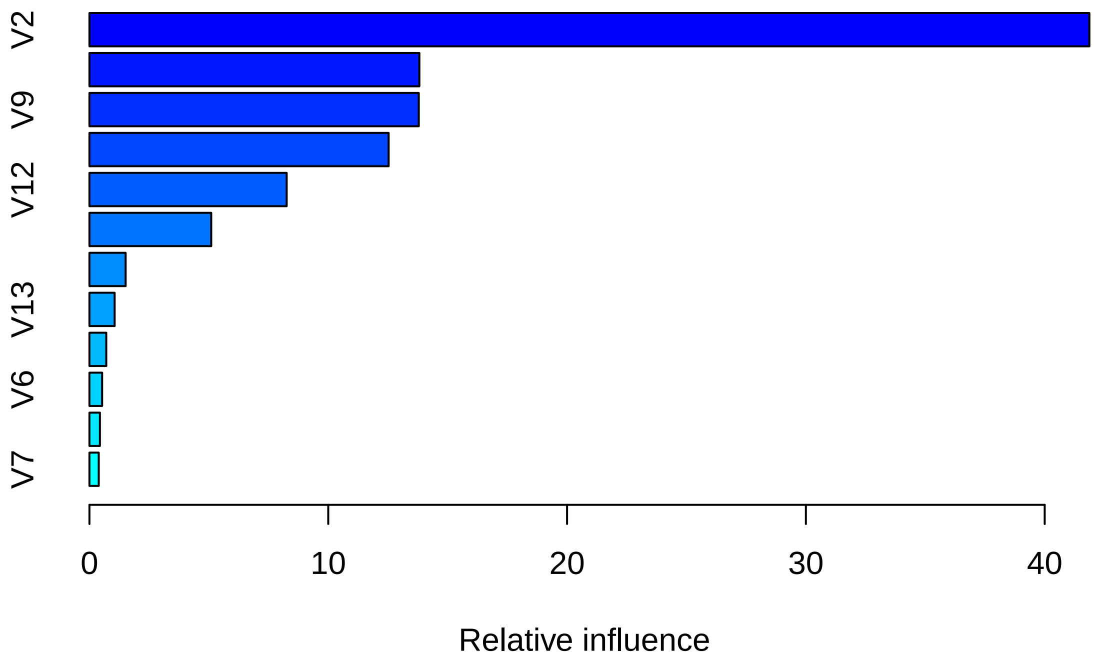
```
col_idx <- grep("V4", names(Ozone))
Ozone <- Ozone[, c((1:ncol(Ozone))[-col_idx], col_idx)]
names(Ozone)
```

```
##  [1] "V1"  "V2"  "V3"  "V5"  "V6"  "V7"  "V8"  "V9"  "V10" "V11" "V12"
## [12] "V13" "V4"
```

```
##Lets try a range of mtry (number of variables selected at random at each split)
oob.err = double(12)
test.err = double(12)

#In a loop of mtry from 1 to 13, you first fit the randomForest to the train dataset
for(mtry in 1:12){
  fit = randomForest(V4~., data = Ozone, subset=train, mtry=mtry, ntree = 350)
  oob.err[mtry] = fit$mse[350] ##extract Mean-squared-error
  pred = predict(fit, Ozone[-train,]) #predict on test dataset
  test.err[mtry] = with(Ozone[-train,], mean( (V4-pred)^2 )) #compute test error
}

#Visualize
matplot(1:mtry, cbind(test.err, oob.err), pch = 23, col = c("red", "blue"), type = "b", ylab="Mean Squa
legend("topright", legend = c("OOB", "Test"), pch = 23, col = c("red", "blue"))
```

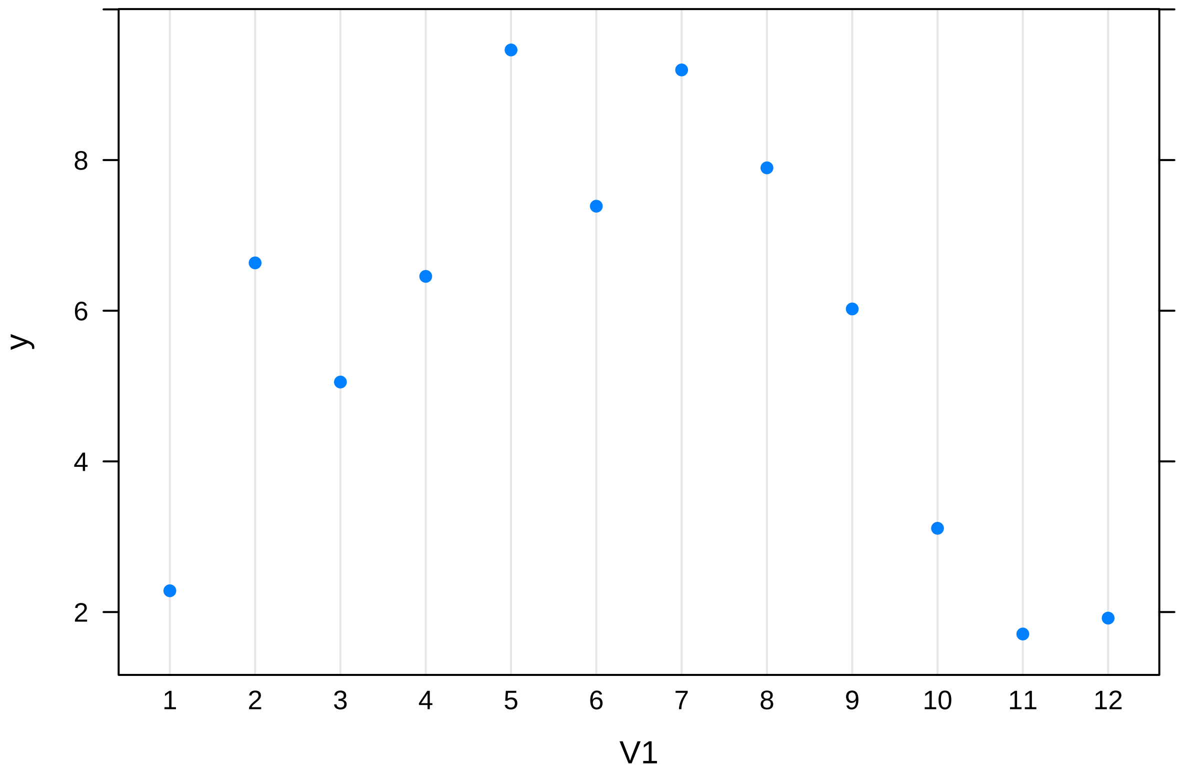



```
#Gradient Boosting Model
boost.ozone = gbm(V4~., data = Ozone[train,], distribution = "gaussian", n.trees = 10000, shrinkage = 0

#Variable Importance Plot
summary(boost.ozone)
```

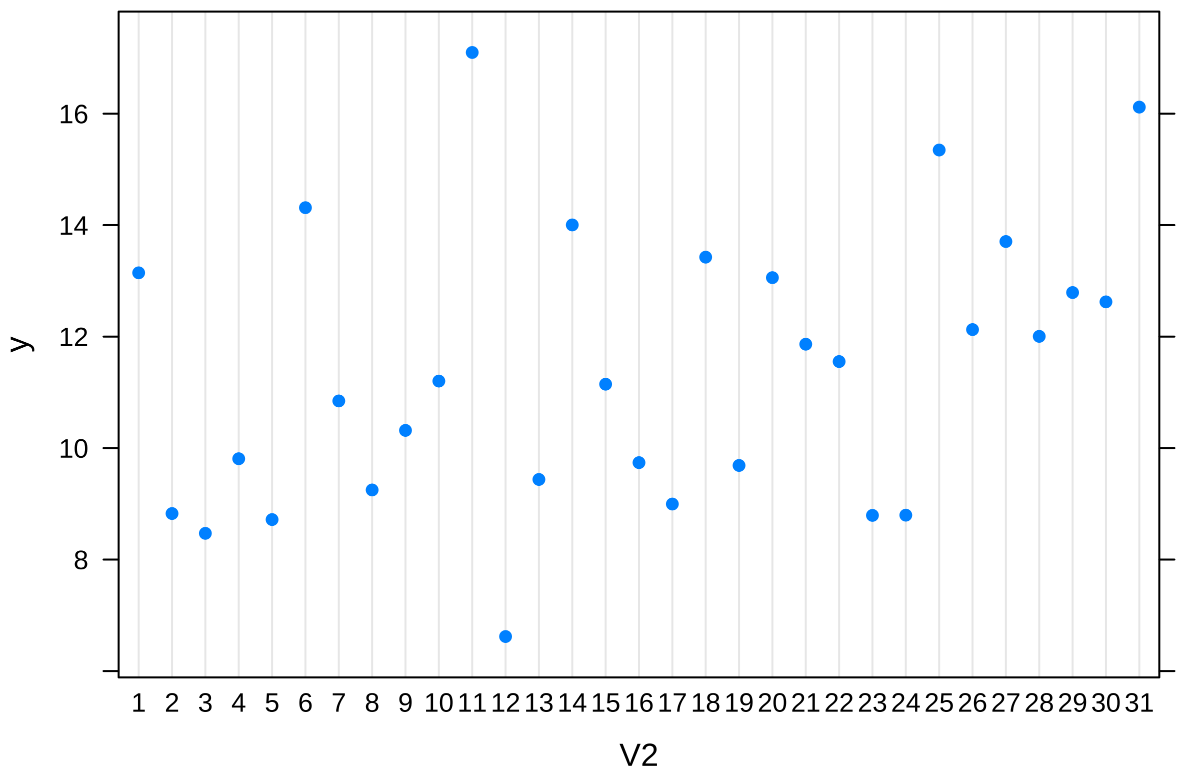



```
##      var    rel.inf
## V2    V2 41.8715912
## V1    V1 13.8153753
## V9    V9 13.7903477
## V8    V8 12.5286831
## V12  V12  8.2582964
## V10  V10  5.1001671
## V3    V3  1.5158141
## V13  V13  1.0549795
## V11  V11  0.7042669
## V6    V6  0.5320559
## V5    V5  0.4385810
## V7    V7  0.3898418
```

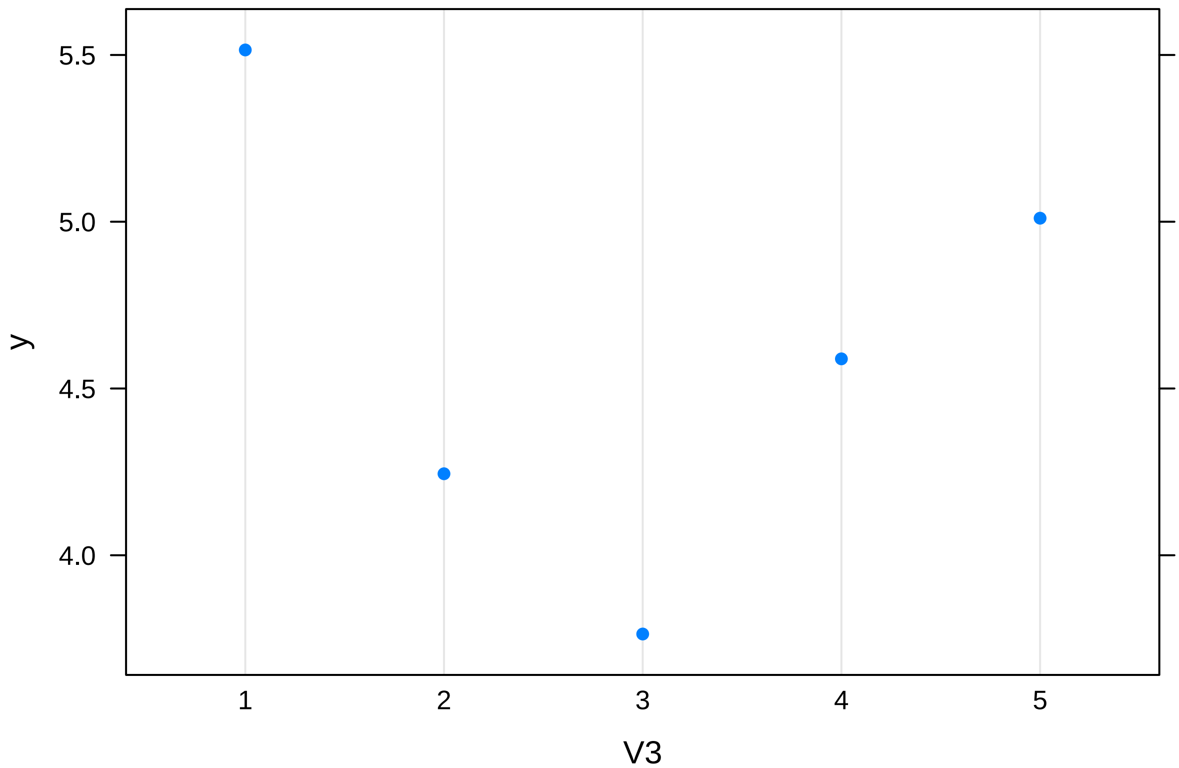
```r
#Visualize important variables of interest
plot(boost.ozone,i="V1")
```




```
plot(boost.ozone,i="V2")
```



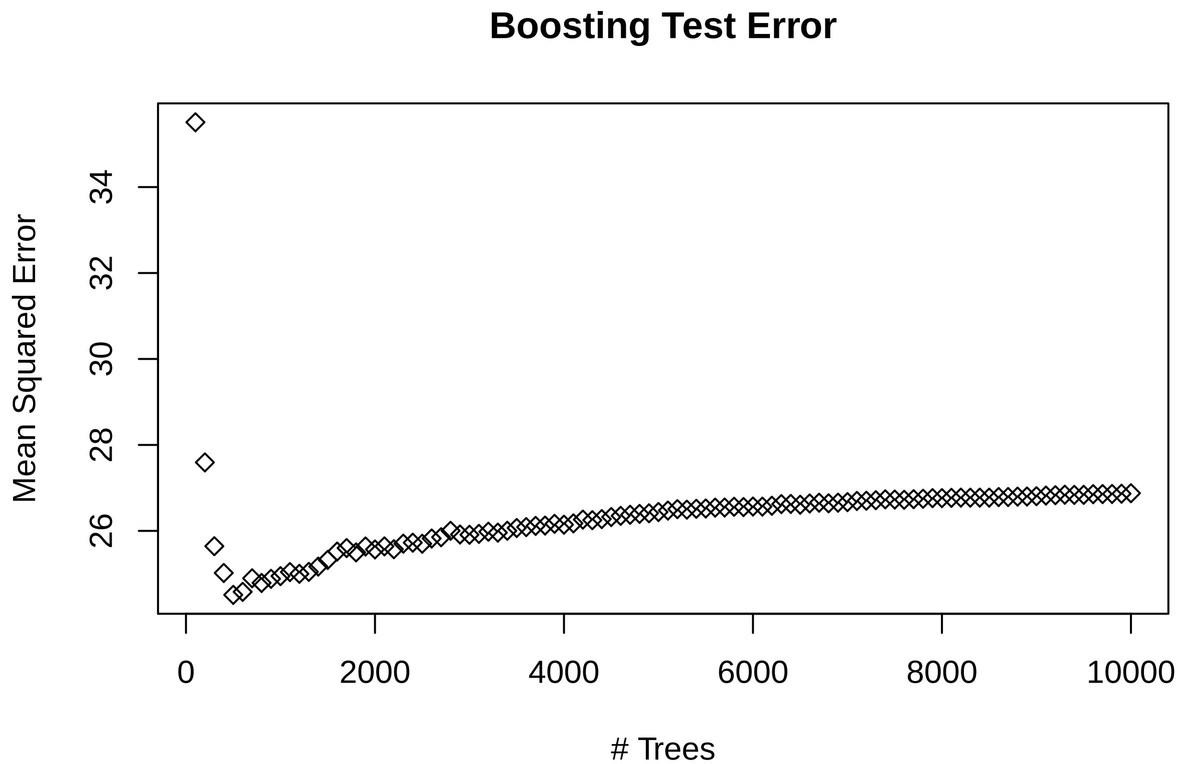

```
plot(boost.ozone,i="V3")
```




```
#Predict on test set
n.trees = seq(from = 100, to = 10000, by = 100)
predmat = predict(boost.ozone, newdata = Ozone[-train,], n.trees = n.trees)
dim(predmat)
```

```
## [1] 103 100
```

```
#Visualize Boosting Error Plot
boost.err = with(Ozone[-train,], apply( (predmat - V4)^2, 2, mean) )
plot(n.trees, boost.err, pch = 23, ylab = "Mean Squared Error", xlab = "# Trees", main = "Boosting Test
```



Both bagging and boosting on this dataset actually resulted in worse model performance than the baseline Random Forest that I created in the first section. The fact that both methods resulted in worse performance is probably indicative that the original model was already optimized, and that the dataset I used could not be reduced in complexity any further due to the variation of the data. The purpose of bagging is to reduce the complexity of models that overfit a training dataset, however, if the model is already optimally complex, then reducing it further results in a less accurate model. Alternatively, the purpose of boosting is to increase the complexity of a model that has high bias, but again, if the model is already optimized, then increasing the complexity will result in worse performance.