lab 2

Awet Tsegay

21/10/2021

K-Nearest Neighbour (KNN) and decision trees

The topics that are briefly covered in this lab are:

1. Fitting Classification Trees
2. Fitting Regression Trees
3. The KNN algorithm

1. Fitting Classification Trees

We first use classification trees to analyze the Carseats data set. This data set is part of the ISLR library. The Carseats dataset contains information about sales of child car seats at 400 different stores (simulated).

```
# help(Carseats) or
# ?Carseats.
library(tree)
```

```
## Warning: package 'tree' was built under R version 4.0.5
```

```
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 4.0.5
```

```
data(Carseats)
names(Carseats)
```

```
##  [1] "Sales"       "CompPrice"   "Income"      "Advertising" "Population"
##  [6] "Price"       "ShelveLoc"   "Age"         "Education"   "Urban"
## [11] "US"
```

Examples of variables are: - Sales (sales in thousands at each location), - CompPrice (price charged by competitor at each location), - Income (community income level in thousands of dollars); overall there are 11 variables.

Our goal is to predict whether the sales at a store are high (defined as more than 8) or low (defined as 8 or less).

Sales is a continuous variable, and so we begin by re-coding 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 8, and takes on a value of No otherwise.

```
attach(Carseats)
High <- ifelse(Sales<=8,"No","Yes")
High <- as.factor(High)
```

(We attach the data set so that we can refer to its variables by their names: without attach(Carseats) we would have to write, e.g., Carseats$Sales instead of Sales.)

The last is to fix a technical problem with the interpretation of High as a non-numerical variable (required by the tree-fitting function below).

Finally, we use the data.frame() function to merge High with the rest of the Carseats data.

```
Carseats <- data.frame(Carseats,High)
```

We now use the tree() function to fit a classification tree in order to predict High using all variables but Sales.

The syntax of the tree() function is standard (similar syntax is used by many other functions for training prediction algorithms).

```
tree.carseats <- tree(High~CompPrice+Income+Advertising+
+ Population+Price+ShelveLoc+Age+Education+Urban+US,Carseats)
```

(Notice that the first + in the second line is R's continuation prompt.) Alternatively, you could use a shorthand.

```
tree.carseats <- tree(High~.-Sales,Carseats)
```

The dot . refers to the set of all variables in the data frame apart from the variable we are predicting. We have removed Sales using the -Sales syntax.

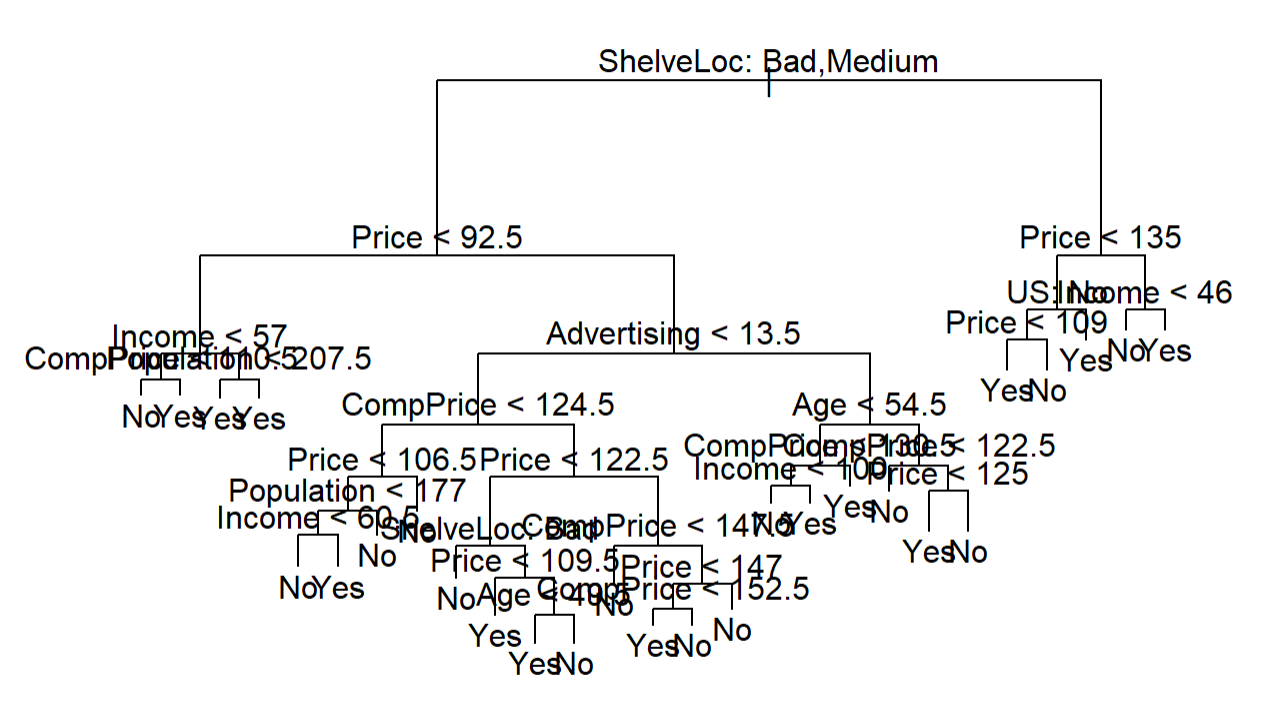
The constructed tree is put into tree.carseats.

```
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 see that the training error rate is 9%.

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




```
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 ) *
```

```
set.seed(2)
train <- sample(1:nrow(Carseats), 200)
Carseats.test <- Carseats[-train,]
High.test <- High[-train]
```

```
tree.carseats <- tree(High~.-Sales, Carseats, subset = train)
tree.pred <- predict(tree.carseats, Carseats.test, type = "class")
table(tree.pred,High.test)
```

```
##          High.test
## tree.pred  No Yes
##       No  104  33
##       Yes  13  50
```

```
(104+50)/200
```

```
## [1] 0.77
```

This approach leads to correct predictions for around 77.0% of the locations in the test data set.

Next, we consider whether pruning the tree might lead to improved results

The function cv.tree() performs cross-validation in order to determine the optimal level of tree complexity; cost complexity pruning is used in order to select a sequence of trees for consideration. We use the argument FUN=prune.misclass in order to indicate that we want the classification error rate to guide the cross-validation and pruning process, rather than the default for the cv.tree() function, which is deviance.

```
set.seed(3)
cv.carseats <- cv.tree(tree.carseats, FUN=prune.misclass)
names(cv.carseats)
```

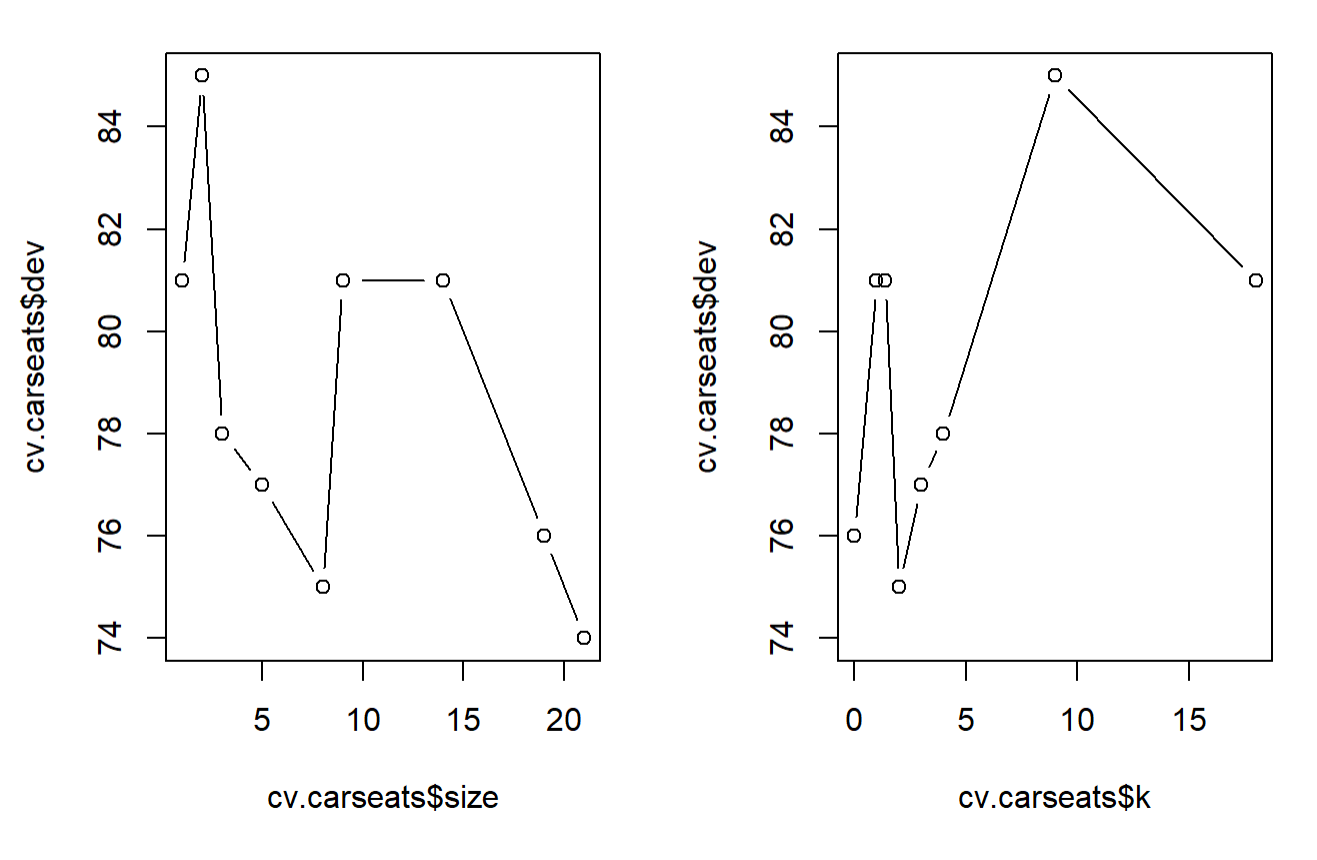
```
## [1] "size"   "dev"    "k"      "method"
```

```
cv.carseats
```

```
## $size
## [1] 21 19 14  9  8  5  3  2  1
##
## $dev
## [1] 74 76 81 81 75 77 78 85 81
##
## $k
## [1] -Inf  0.0  1.0  1.4  2.0  3.0  4.0  9.0 18.0
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

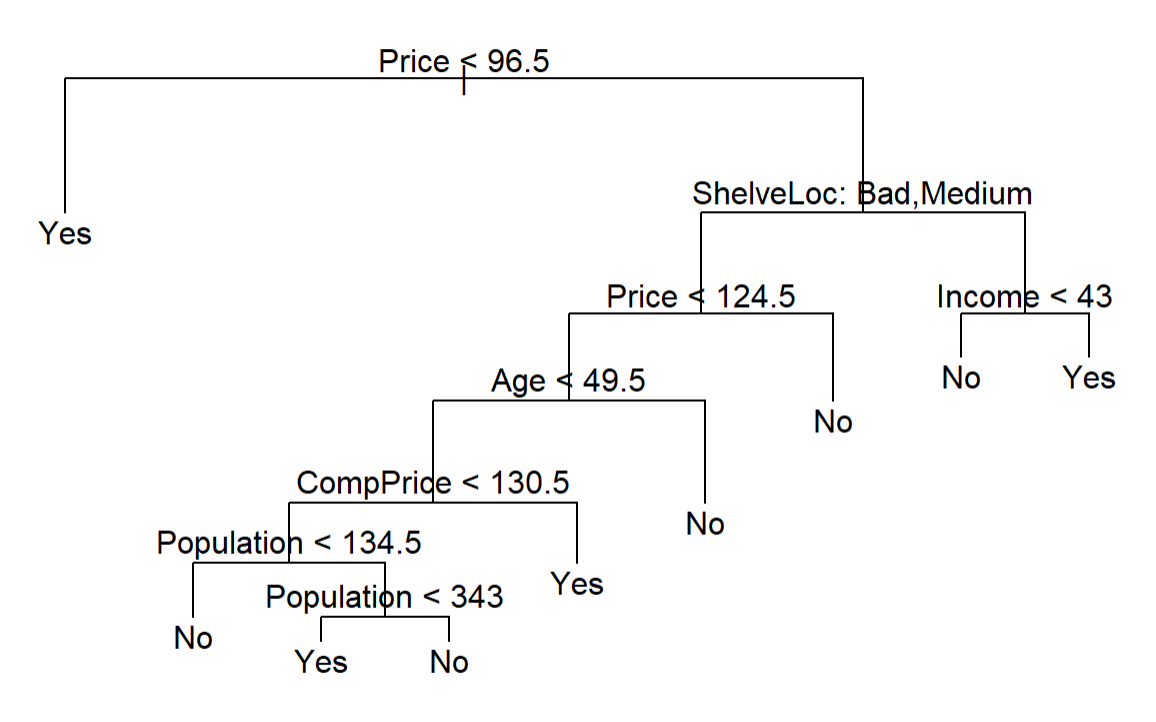
We plot the error rate as a function of both size and k.

```
par(mfrow=c(1,2))
plot(cv.carseats$size, cv.carseats$dev, type="b")
plot(cv.carseats$k, cv.carseats$dev, type="b")
```



We now apply the prune.misclass() function in order to prune the tree to obtain the nine-node tree.

```
prune.carseats <- prune.misclass(tree.carseats,best=9)
plot(prune.carseats)
text(prune.carseats,pretty=0)
```



How well does this pruned tree perform on the test data set? Once again, we apply the predict() function.

```
tree.pred <- predict(prune.carseats, Carseats.test, type="class")
table(tree.pred,High.test)
```

```
##          High.test
## tree.pred No Yes
##       No  97  25
##       Yes 20  58
```

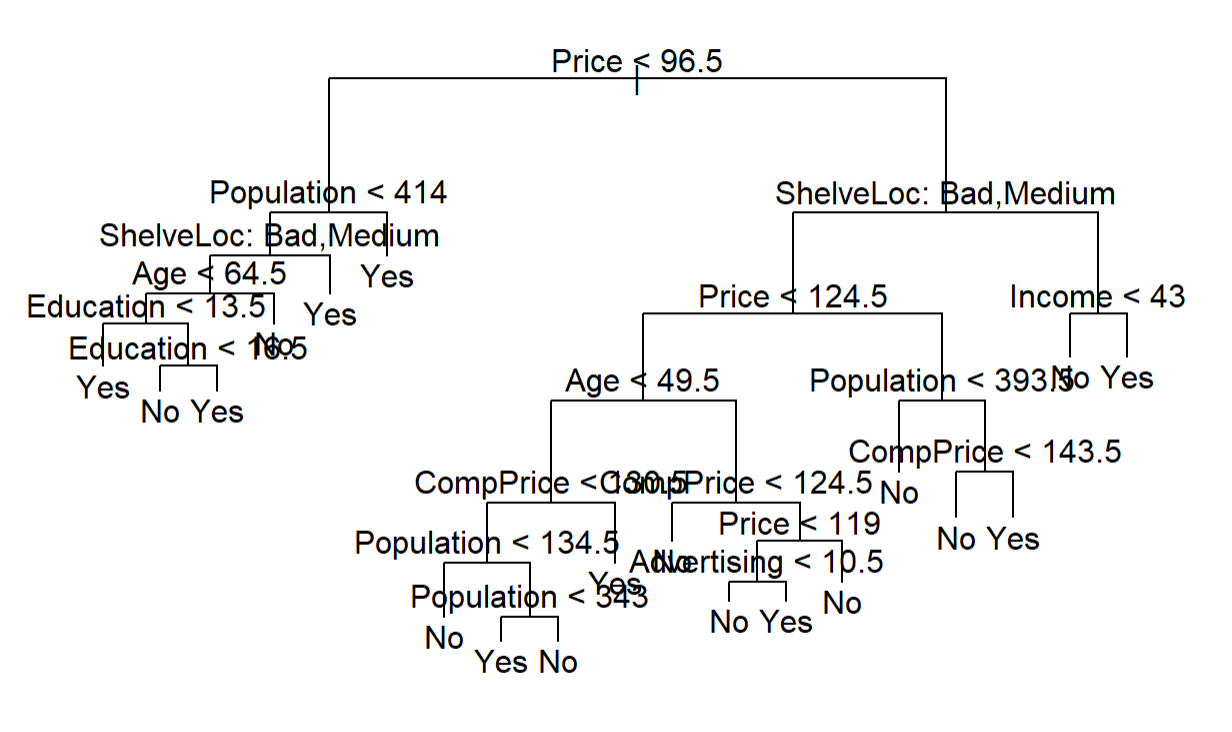
```
(97+58)/200
```

```
## [1] 0.775
```

Now 77.5% of the test observations are correctly classified, so not only has the pruning process produced a more interpretable tree, but it has also improved the classification accuracy.

If we increase the value of best, we obtain a larger pruned tree with lower classification accuracy:

```
prune.carseats <- prune.misclass(tree.carseats, best=15)
plot(prune.carseats)
text(prune.carseats,pretty=0)
```




```
tree.pred <- predict(prune.carseats, Carseats.test, type="class")
```

```
table(tree.pred,High.test)
```

```
##          High.test
## tree.pred  No Yes
##       No  102  30
##       Yes  15  53
```

```
(102 + 53)/200
```

```
## [1] 0.775
```

2. Fitting Regression Trees

In this section we fit a regression tree to the Boston data set.

We will seek to predict medv using 13 attributes.

```
library(MASS)
```

```
## Warning: package 'MASS' was built under R version 4.0.4
```

```
data(Boston)
# fix(Boston)
names(Boston)
```

```
##  [1] "crim"    "zn"      "indus"   "chas"    "nox"     "rm"      "age"
##  [8] "dis"     "rad"     "tax"     "ptratio" "black"   "lstat"   "medv"
```

```
# ?Boston
```

First, we create a training set, and fit the tree to the training data.

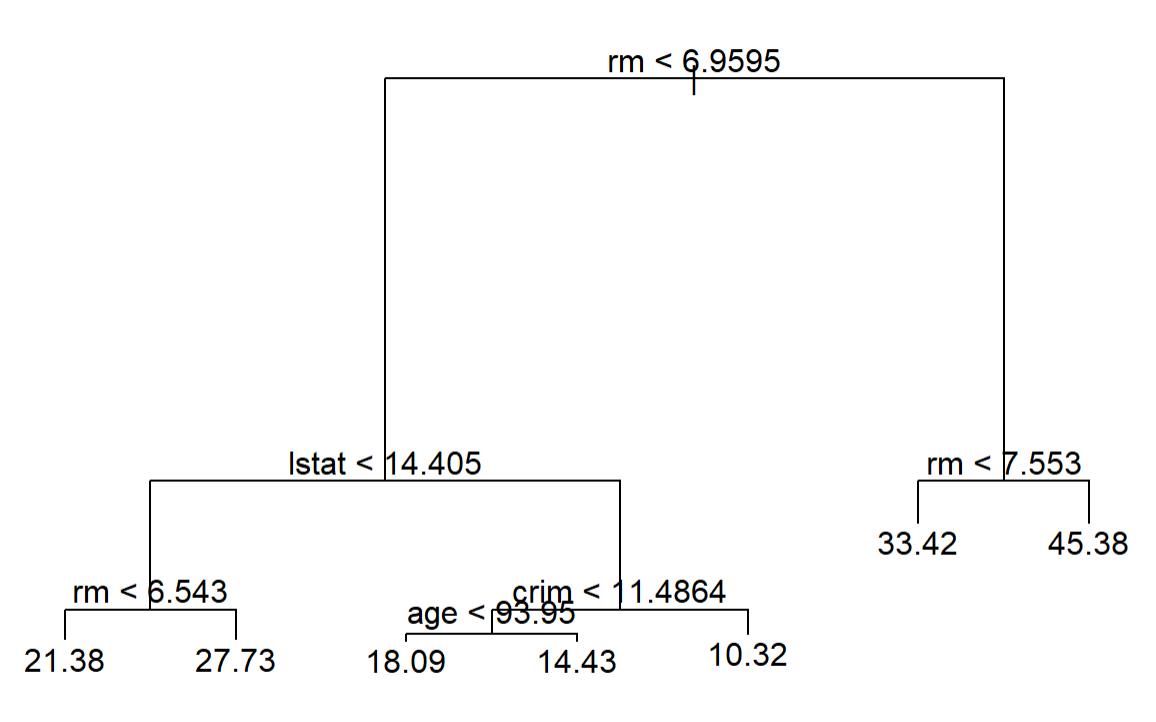
```
set.seed(1)
train <- sample(1:nrow(Boston), nrow(Boston)/2)
tree.boston <- tree(medv ~ ., Boston, subset=train)
summary(tree.boston)
```

```
##
## Regression tree:
## tree(formula = medv ~ ., data = Boston, subset = train)
## Variables actually used in tree construction:
## [1] "rm"    "lstat" "crim"  "age"
## Number of terminal nodes:  7
## Residual mean deviance:  10.38 = 2555 / 246
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -10.1800  -1.7770  -0.1775   0.0000   1.9230  16.5800
```

In the context of a regression tree, the deviance is simply the sum of squared errors for the tree.

We now plot the tree.

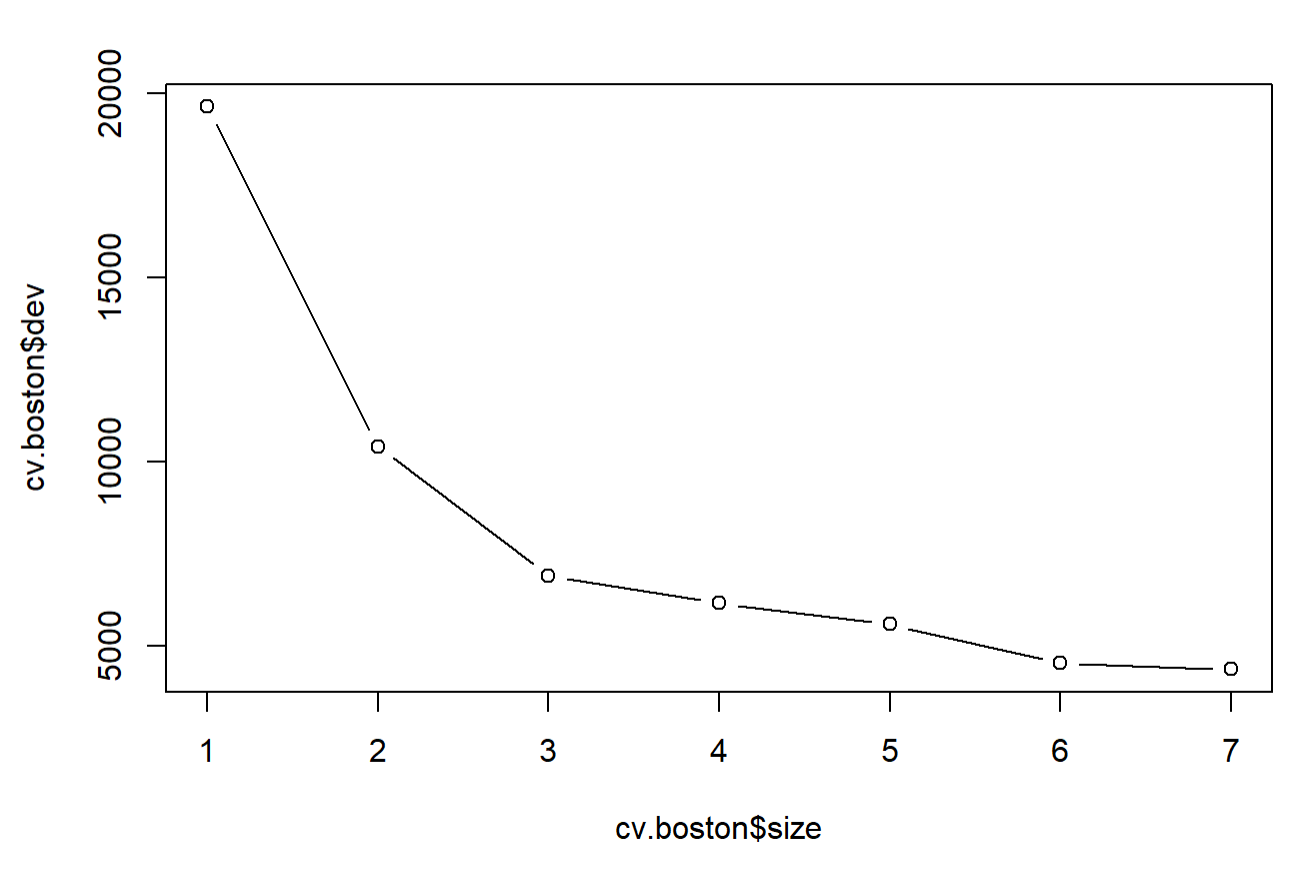
```
plot(tree.boston)
text(tree.boston,pretty=0)
```



The tree predicts a median house price of $46, 380 for larger homes in suburbs in which residents have high socioeconomic status (rm>=7.437 and lstat<9.715).

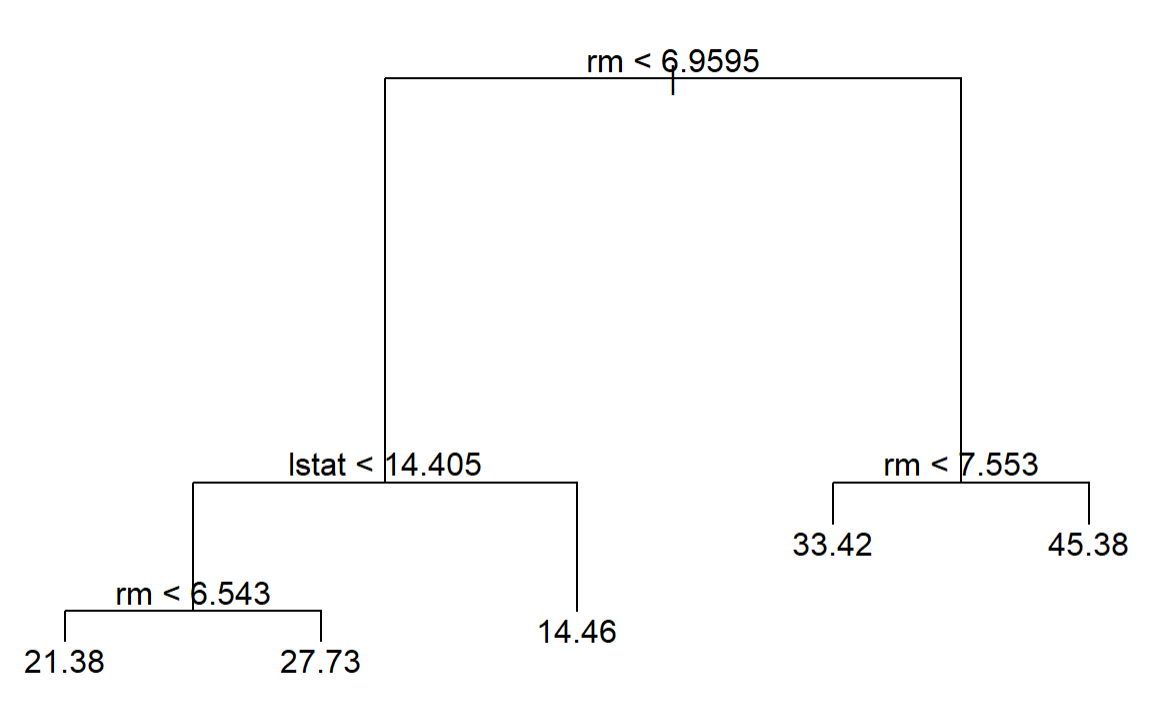
Now we use the cv.tree() function to see whether pruning the tree will improve performance.

```
cv.boston <- cv.tree(tree.boston)
plot(cv.boston$size, cv.boston$dev, type='b')
```



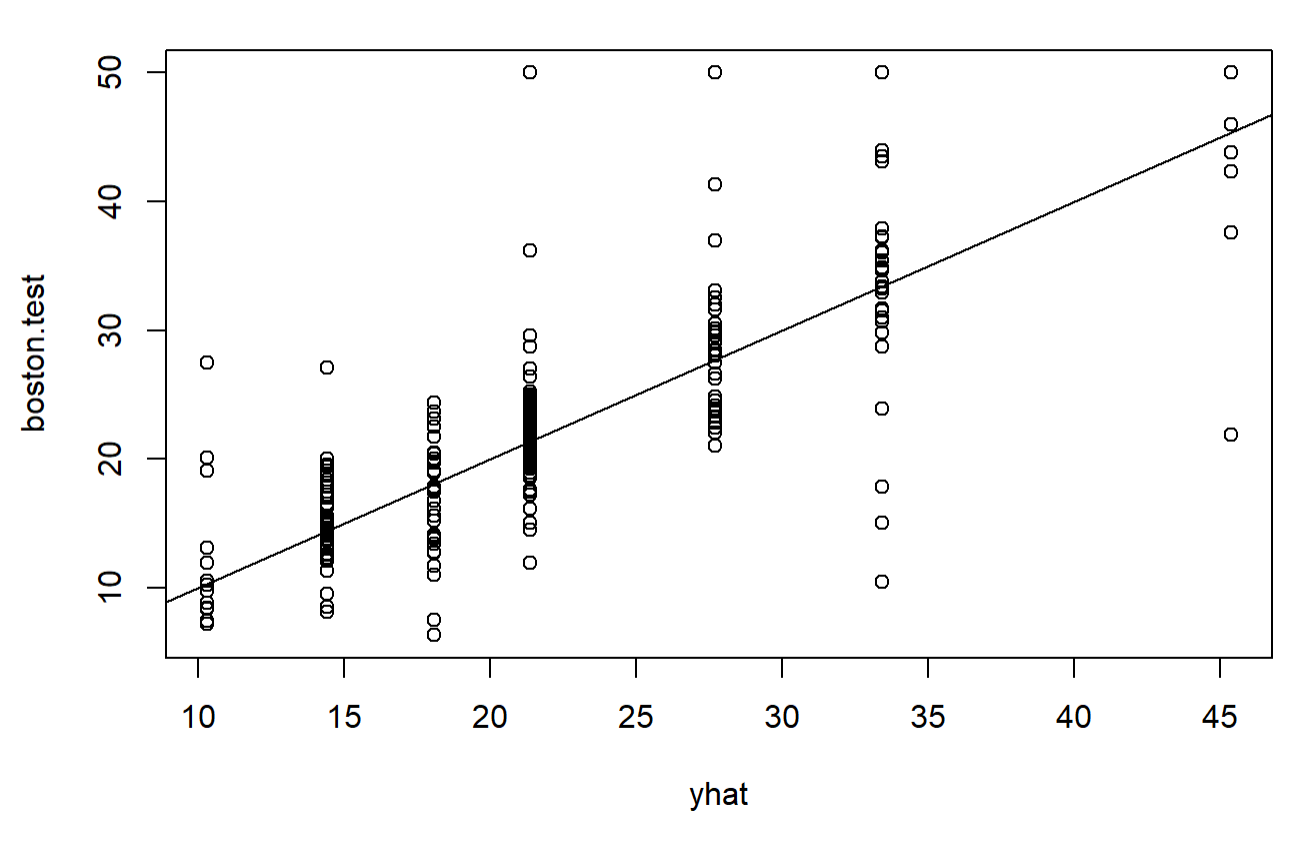
In this case, the most complex tree is selected by cross-validation. However, if we wish to prune the tree, we could do so as follows, using the prune.tree() function:

```
prune.boston <- prune.tree(tree.boston, best=5)
plot(prune.boston)
text(prune.boston, pretty=0)
```



In keeping with the cross-validation results, we use the unpruned tree to make predictions on the test set.

```
yhat <- predict(tree.boston, newdata = Boston[-train,])
boston.test <- Boston[-train,"medv"]
plot(yhat,boston.test)
abline(0,1)
```




```
mean((yhat - boston.test)^2)
```

```
## [1] 35.28688
```

In other words, the test set MSE associated with the regression tree is 25.05. The square root of the MSE is therefore around 5.005, indicating that this model leads to test predictions that are within around $5, 005 of the true median home value for the suburb.

3 K-Nearest Neighbours

```
names(Smarket)
```

```
## [1] "Year"      "Lag1"      "Lag2"      "Lag3"      "Lag4"      "Lag5"
## [7] "Volume"    "Today"     "Direction"
```

```
summary(Smarket)
```

```
##       Year           Lag1                Lag2                Lag3
##  Min.   :2001   Min.   :-4.922000   Min.   :-4.922000   Min.   :-4.922000
##  1st Qu.:2002   1st Qu.:-0.639500   1st Qu.:-0.639500   1st Qu.:-0.640000
##  Median :2003   Median : 0.039000   Median : 0.039000   Median : 0.038500
##  Mean   :2003   Mean   : 0.003834   Mean   : 0.003919   Mean   : 0.001716
##  3rd Qu.:2004   3rd Qu.: 0.596750   3rd Qu.: 0.596750   3rd Qu.: 0.596750
##  Max.   :2005   Max.   : 5.733000   Max.   : 5.733000   Max.   : 5.733000
##       Lag4                Lag5              Volume           Today
##  Min.   :-4.922000   Min.   :-4.92200   Min.   :0.3561   Min.   :-4.922000
##  1st Qu.:-0.640000   1st Qu.:-0.64000   1st Qu.:1.2574   1st Qu.:-0.639500
##  Median : 0.038500   Median : 0.03850   Median :1.4229   Median : 0.038500
##  Mean   : 0.001636   Mean   : 0.00561   Mean   :1.4783   Mean   : 0.003138
##  3rd Qu.: 0.596750   3rd Qu.: 0.59700   3rd Qu.:1.6417   3rd Qu.: 0.596750
##  Max.   : 5.733000   Max.   : 5.73300   Max.   :3.1525   Max.   : 5.733000
##  Direction
##  Down:602
##  Up  :648
##
##
##
##
```

```
library(class)
```

```
## Warning: package 'class' was built under R version 4.0.4
```

```
attach(Smarket)
train.X <- cbind(Lag1,Lag2)[train,]
test.X <- cbind(Lag1,Lag2)[!train,]
train.Direction <- Direction[train]

Direction.2005 <- Direction[!train]
```

Now the knn() function can be used to predict the market's movement for the dates in 2005.

```
set.seed(1234)
knn.pred <- knn(train.X,test.X,train.Direction,k=1)
table(knn.pred,Direction.2005)
```

```
##         Direction.2005
## knn.pred Down Up
##     Down    0  0
##     Up      0  0
```

Below, the analysis is repeated using K = 3.

```
knn.pred <- knn(train.X,test.X,train.Direction,k=3)
table(knn.pred,Direction.2005)
```

```
##         Direction.2005
## knn.pred Down Up
##     Down    0  0
##     Up      0  0
```