```
Variables actually used in tree construction:
[1] "lstat" "rm"    "dis"
Number of terminal nodes:  8
Residual mean deviance:  12.65 = 3099 / 245
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-14.1000  -2.0420  -0.0536   0.0000   1.9600  12.6000
```

Notice that the output of `summary()` indicates that only three of the variables have been used in constructing the tree. 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 variable `lstat` measures the percentage of individuals with lower socioeconomic status. The tree indicates that lower values of `lstat` correspond to more expensive houses. The tree predicts a median house price of $\$46,400$ 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.tree()`

```
> 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] 25.05
```

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.

8.3.3 Bagging and Random Forests

Here we apply bagging and random forests to the `Boston` data, using the `randomForest` package in `R`. 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:

`random Forest()`

```
> library(randomForest)
> set.seed(1)
> bag.boston=randomForest(medv~.,data=Boston,subset=train,
    mtry=13,importance=TRUE)
> bag.boston

Call:
 randomForest(formula = medv ~ ., data = Boston, mtry = 13,
     importance = TRUE,      subset = train)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 13

          Mean of squared residuals: 10.77
                    % Var explained: 86.96
```

The argument `mtry=13` indicates that all 13 predictors should be considered for each split of the tree—in other words, that bagging should be done. How well does this bagged model perform on the test set?

```
> yhat.bag = predict(bag.boston,newdata=Boston[-train,])
> plot(yhat.bag, boston.test)
> abline(0,1)
> mean((yhat.bag-boston.test)^2)
[1] 13.16
```

The test set MSE associated with the bagged regression tree is 13.16, almost half that obtained using an optimally-pruned single tree. We could change the number of trees grown by `randomForest()` using the `ntree` argument:

```
> bag.boston=randomForest(medv~.,data=Boston,subset=train,
    mtry=13,ntree=25)
> yhat.bag = predict(bag.boston,newdata=Boston[-train,])
> mean((yhat.bag-boston.test)^2)
[1] 13.31
```

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 = 6`.

```
> set.seed(1)
> rf.boston=randomForest(medv~.,data=Boston,subset=train,
    mtry=6,importance=TRUE)
> yhat.rf = predict(rf.boston,newdata=Boston[-train,])
> mean((yhat.rf-boston.test)^2)
[1] 11.31
```

The test set MSE is 11.31; this indicates that random forests yielded an improvement over bagging in this case.

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

```
> importance(rf.boston)
        %IncMSE  IncNodePurity
crim     12.384        1051.54
zn        2.103          50.31
indus     8.390        1017.64
chas      2.294          56.32
nox      12.791        1107.31
rm       30.754        5917.26
age      10.334         552.27
dis      14.641        1223.93
rad       3.583          84.30
tax       8.139         435.71
ptratio  11.274         817.33
black     8.097         367.00
lstat    30.962        7713.63
```

Two measures of variable importance are reported. The former is based upon the mean decrease of accuracy in predictions on the out of bag samples when a given variable is excluded from the model. The latter is a measure of the total decrease in node impurity that results from splits over that variable, averaged over all trees (this was plotted in Figure 8.9). In the case of regression trees, the node impurity is measured by the training RSS, and for classification trees by the deviance. Plots of these importance measures can be produced using the `varImpPlot()` function. `varImpPlot()`

```
> varImpPlot(rf.boston)
```

The results indicate that across all of the trees considered in the random forest, the wealth level of the community (`lstat`) and the house size (`rm`) are by far the two most important variables.

8.3.4 Boosting

Here we use the `gbm` package, and within it the `gbm()` function, to fit boosted regression trees to the `Boston` data set. We run `gbm()` with the option `distribution="gaussian"` since this is a regression problem; if it were a binary classification problem, we would use `distribution="bernoulli"`. The argument `n.trees=5000` indicates that we want 5000 trees, and the option `interaction.depth=4` limits the depth of each tree. `gbm()`

```
> library(gbm)
> set.seed(1)
> boost.boston=gbm(medv~.,data=Boston[train,],distribution=
        "gaussian",n.trees=5000,interaction.depth=4)
```

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

```
> summary(boost.boston)
       var   rel.inf
1    lstat   45.96
2       rm   31.22
3      dis    6.81
4     crim    4.07
5      nox    2.56
6  ptratio    2.27
7    black    1.80
8      age    1.64
9      tax    1.36
10   indus    1.27
11    chas    0.80
12     rad    0.20
13      zn    0.015
```

We see that `lstat` and `rm` are by far the most important variables. We can also produce *partial dependence plots* for these two variables. These plots illustrate the marginal effect of the selected variables on the response after *integrating* out the other variables. In this case, as we might expect, median house prices are increasing with `rm` and decreasing with `lstat`.

partial dependence plot

```
> par(mfrow=c(1,2))
> plot(boost.boston,i="rm")
> plot(boost.boston,i="lstat")
```

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

```
> yhat.boost=predict(boost.boston,newdata=Boston[-train,],
        n.trees=5000)
> mean((yhat.boost-boston.test)^2)
[1] 11.8
```

The test MSE obtained is 11.8; similar to the test MSE for random forests and superior to that for bagging. If we want to, we can perform boosting with a different value of the shrinkage parameter λ in (8.10). The default value is 0.001, but this is easily modified. Here we take $\lambda = 0.2$.

```
> boost.boston=gbm(medv~.,data=Boston[train,],distribution=
    "gaussian",n.trees=5000,interaction.depth=4,shrinkage=0.2,
    verbose=F)
> yhat.boost=predict(boost.boston,newdata=Boston[-train,],
        n.trees=5000)
> mean((yhat.boost-boston.test)^2)
[1] 11.5
```

In this case, using $\lambda = 0.2$ leads to a slightly lower test MSE than $\lambda = 0.001$.