Tree-Based Methods

November 1, 2017

Recursive partitioning is a fundamental tool in data mining. It helps us explore the stucture of a set of data, while developing easy to visualize decision rules for predicting a categorical (classification tree) or continuous (regression tree) outcome. we describe **tree-based methods** for regression and classification. These involve stratifying or segmenting the predictor space into a number of simple regions.
In order to make a prediction for a given observation, we typically use the mean or the mode of the training observations in the region to which it belongs. Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as decision tree methods.
Tree-based methods are simple and useful for interpretation. However, they typically are not competitive with the best supervised learning approaches, such as regression models, in terms of prediction accuracy. Hence in this chapter we also introduce **bagging, random forests**, and **boosting**.

The Basics of Decision Trees

Prediction Trees are used to predict a response or class Y from input $X1, X2, , Xn$. If it is a continuous response its called a regression tree, if it is categorical, its called a classification tree. At each node of the tree, we check the value of one the input Xi and depending of the (binary) answer we continue to the left or to the right subbranch. When we reach a leaf we will find the prediction (usually it is a simple statistic of the dataset the leaf represents, like the most common value from the available classes).

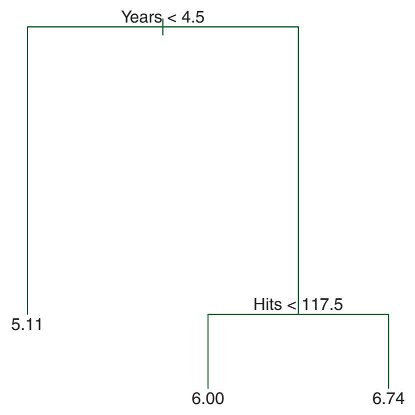
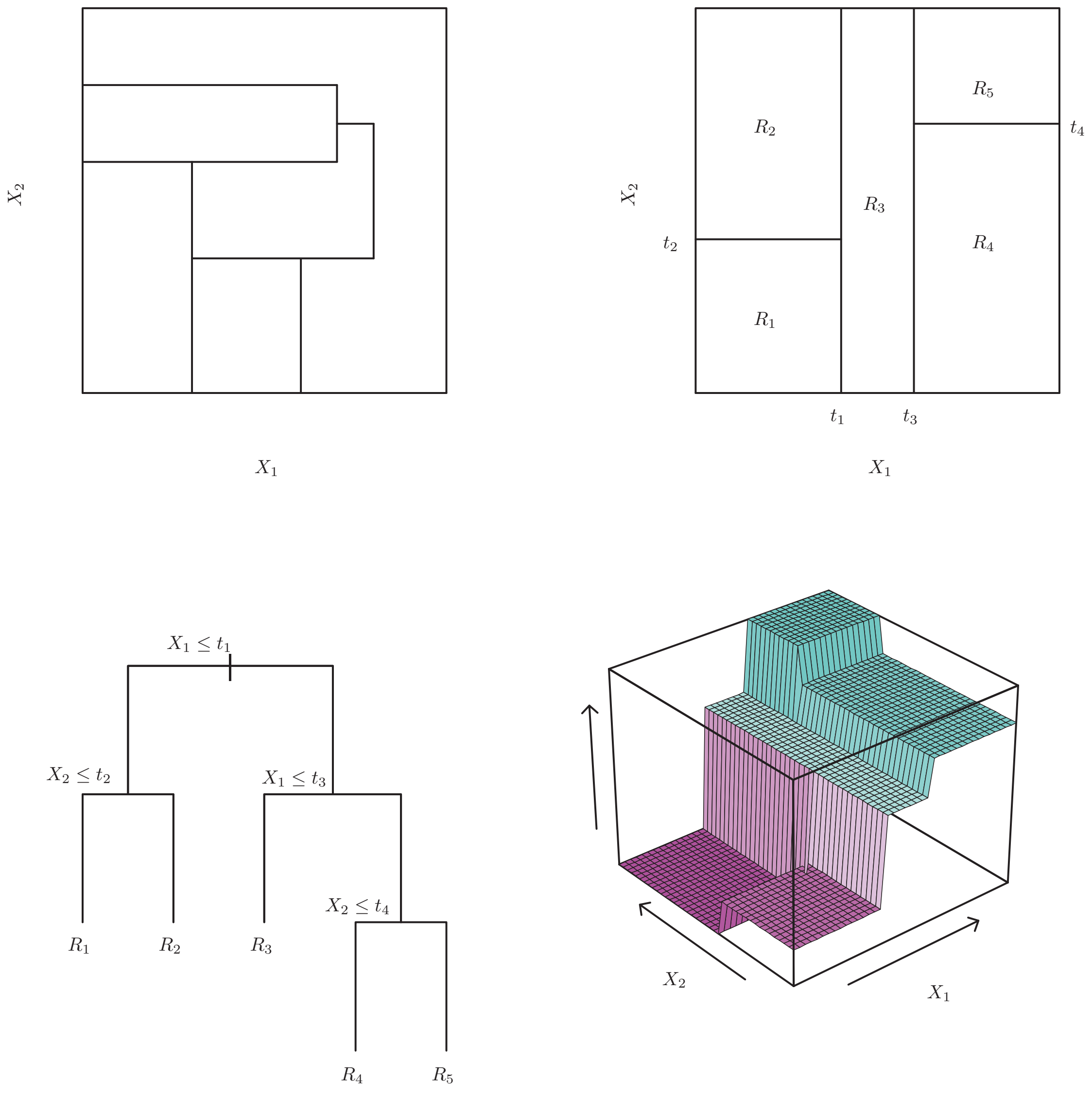


Figure 1: regression tree for predicting the log salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year.

Contrary to linear or polynomial regression which are global models (the predictive formula is supposed to hold in the entire data space), trees try to partition the data space into small enough parts where we can apply a simple different model on each part. The non-leaf part of the tree is just the procedure to determine for each data x what is the model (i.e, which leaf) we will use to classify it.
For example in figure 1, the left-hand branch corresponds to $Years < 4.5$, and the right-hand branch corresponds to $Years >= 4.5$. The tree has two internal nodes and three terminal nodes, or leaves. The number in each leaf is the mean of the response for the observations that fall there.


X_2
X_1
X_2
R_5
t_4
R_2
R_3
t_2
R_4
R_1
t_1
t_3
X_1
$X_1 \leq t_1$
$X_2 \leq t_2$
$X_1 \leq t_3$
$X_2 \leq t_4$
R_1
R_2
R_3
R_4
R_5
X_2
X_1

Tree Pruning

The process described above may produce good predictions on the training set, but is likely to overfit the data, leading to poor test set performance. This is because the resulting tree might be too complex. A smaller tree with fewer splits (that is, fewer regions $R_1, ..., R_J$) might lead to lower variance and better interpretation at the cost of a little bias.

Best Prune

Therefore, a better strategy is to grow a very large tree T_0, and then prune it back in order to obtain a subtree. How do we determine the way to prune the tree? Intuitively, our goal is to select a subtree that subtree leads to the lowest test error rate.

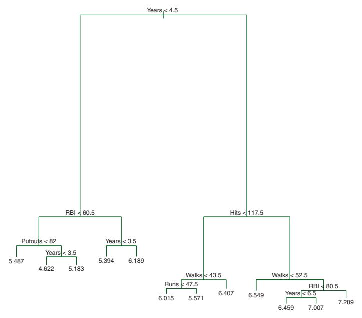


Figure 2: The unpruned tree

Advantages and Disadvantages of Trees

Decision trees for regression and classification have a number of advantages over the more classical approaches seen before:

- Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
- Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- Trees can easily handle qualitative predictors without the need to create dummy variables.

Disadvantages of Trees

- Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches seen in this course.

However, by aggregating many decision trees, using methods like bagging, random forests, and boosting, the predictive performance of trees can be substantially improved.

Fitting Regression Trees

Here we fit a regression tree to the Boston data set.

- The **tree library** is used to construct classification and regression trees.
- First, we create a training set, and
- fit the tree to the training data.
- we use the fitted tree to make predictions on the test set.

```
> library (MASS)
> 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] "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.
-14.10000  -2.04200  -0.05357   0.00000   1.96000  12.60000
```

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.

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

We use the **plot()** function to display the tree structure, and the text() function to display the node labels. The argument **pretty=0** instructs R to include the category names for any qualitative predictors, rather than simply displaying a letter for each category.
we move to the LEFT branch when the stated condition is true

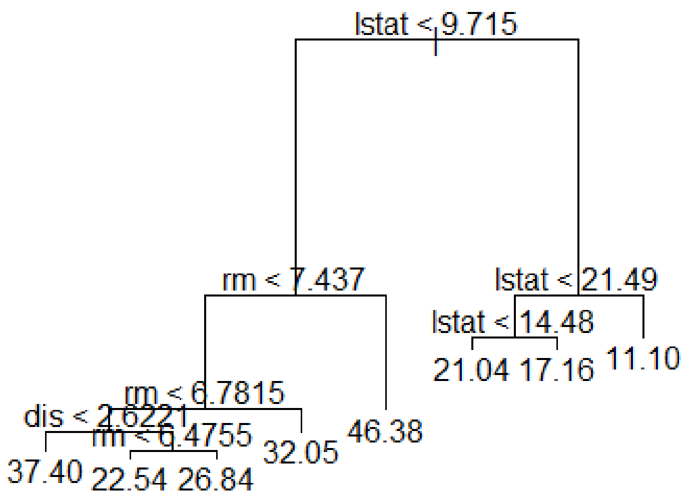


Figure 3: Regression Tree

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 $lstat < 9.715$).
We can compare the predictions with the dataset:

```
tree.boston2=tree(medv~lstat+dis,subset =train)
price.deciles <- quantile(Boston$medv, 0:10/10)
cut.prices    <- cut(Boston$medv, price.deciles, include.lowest=TRUE)
plot(Boston$lstat, Boston$rm, col=grey(10:2/11)[cut.prices], pch=20, xlab="Lstat",ylab="dis")
partition.tree(tree.boston2, ordvars=c("lstat","dis"), add=TRUE)
```

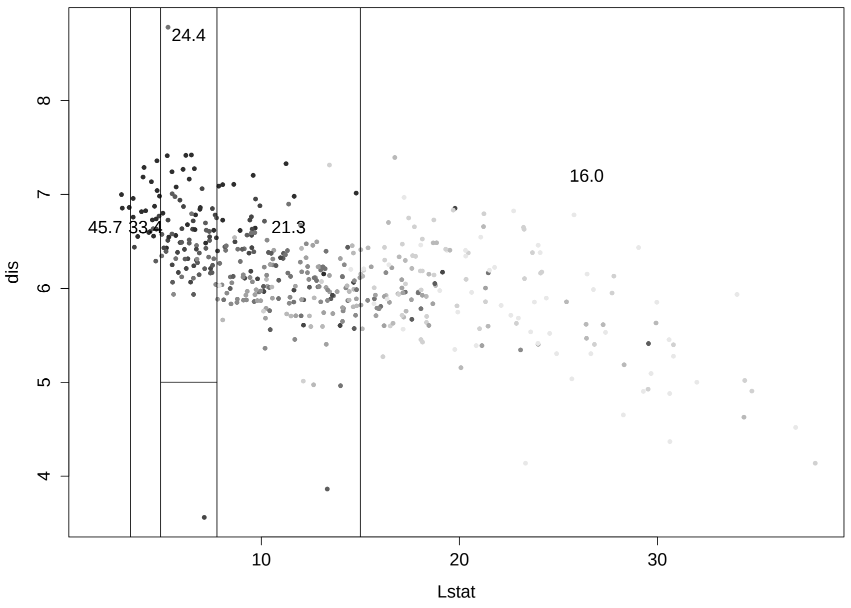


Figure 4: Partition-Tree

The flexibility of a tree is basically controlled by how many leaves they have, since thats how many cells they partition things into. The tree fitting function has a number of controls settings which limit how much it will grow — each node has to contain a certain number of points, and adding a node has to reduce the error by at least a certain amount. The default for the latter, min.dev, is 0:01; lets turn it down and see what happens:

```
tree.model2 <- tree(log(MedianHouseValue) ~ Longitude + Latitude, data=real.estate, mindev=0.0
plot(tree.model2)
text(tree.model2, cex=.75)
```

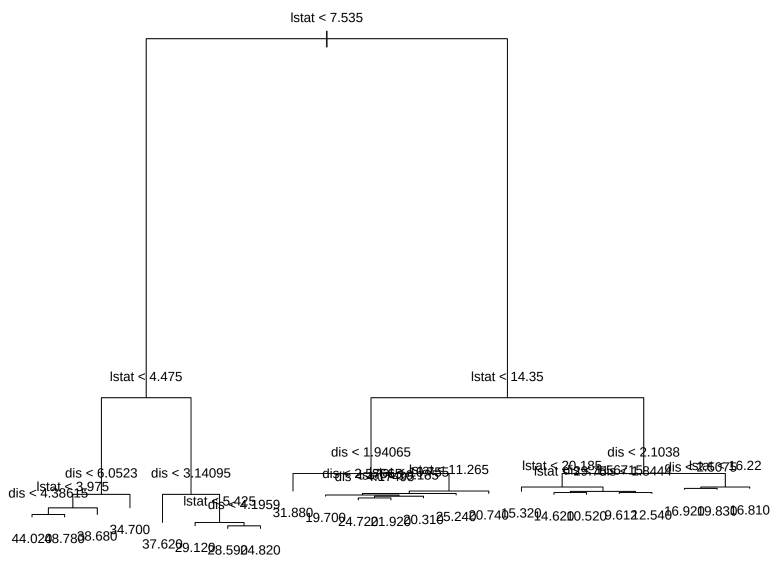


Figure 5: Controlled-Tree

```
> summary(tree.model2)

Regression tree:
tree(formula = medv ~ lstat + dis, subset = train, mindev = 0.001)
Number of terminal nodes:  23
Residual mean deviance:  16.72 = 3846 / 230
Distribution of residuals:
Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-14.3200  -2.0190  -0.2222   0.0000   1.7810  18.1200
```

Pruning the Tree

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

```
> cv.boston =cv.tree(tree.boston )
> cv.boston
> plot(cv.boston$size ,cv.boston$dev ,type=b)
$size
[1] 8 7 6 5 4 3 2 1

$dev
[1]  5226.322  5228.360  6462.626  6692.615  6397.438  7529.846 11958.691 21118.139

$k
[1]      -Inf  255.6581  451.9272  768.5087  818.8885 1559.1264 4276.5803 9665.3582

$method
```

```
[1] "deviance"

attr(,"class")
[1] "prune"         "tree.sequence"
```

The **cv.tree()** function reports the number of terminal nodes of each tree considered (size) as well as the corresponding error rate and the value of the cost-complexity parameter used.

Note that, despite the name, **dev** corresponds to the cross-validation error rate in this instance. The tree with 5 terminal nodes results in the lowest cross-validation error rate, with 6397.438 cross-validation errors

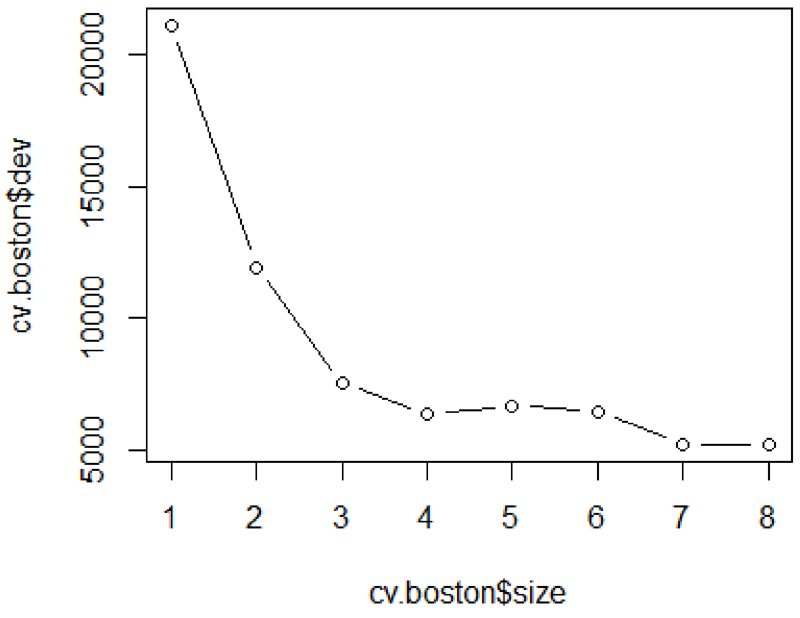


Figure 6: The most complex tree is selected by cross-validation.

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)
```

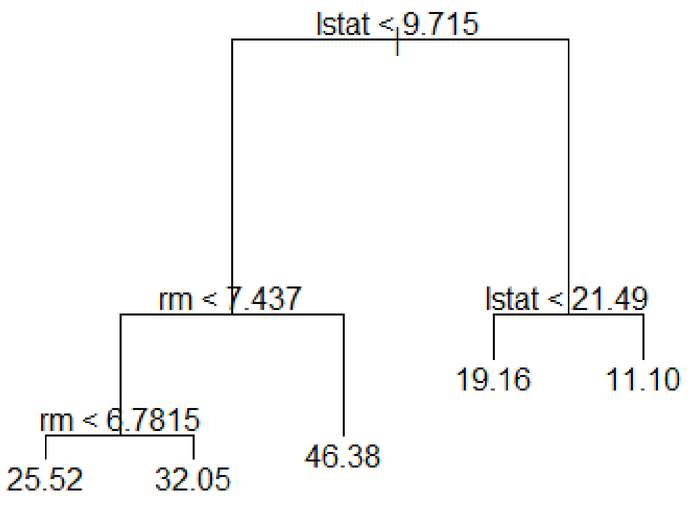


Figure 7: Pruning the Tree

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

```
> yhat=predict (tree.boston ,newdata =Boston [-train ,]) ######### Test data
> boston .test=Boston [-train ," medv"]
> plot(yhat ,boston .test)
> abline (0,1)
> mean((yhat -boston .test)^2)
[1] 25.05


-----------------------------
> yhat=predict (prune.boston ,newdata =Boston [-train ,]) ######### Test data
> boston.test=Boston [-train,"medv"]
> plot(yhat ,boston.test)
> abline (0,1)
> mean((yhat-boston.test)^2)
[1] 26.83413
```

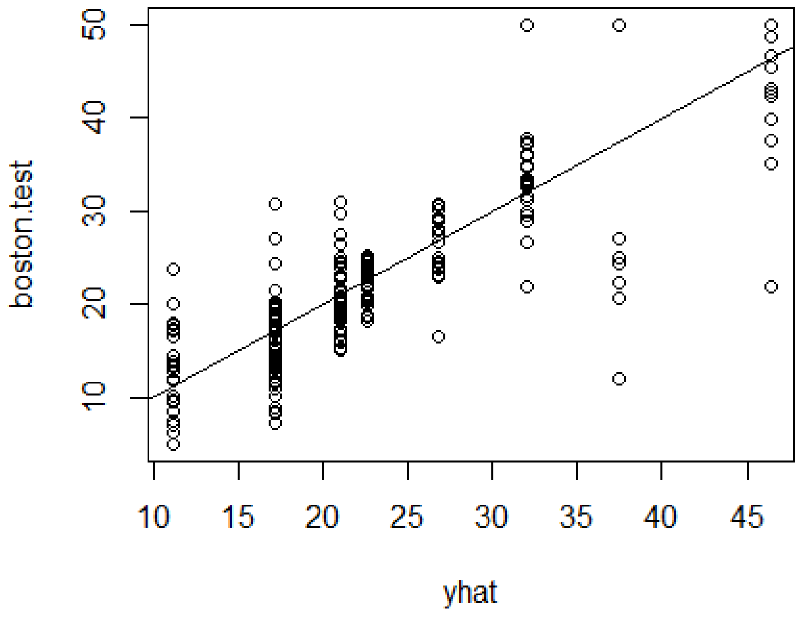


Figure 8

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.

Prune Tree using package rpart

Prune back the tree to avoid overfitting the data. Typically, you will want to select a tree size that minimizes the cross-validated error, the xerror column printed by printcp( ).
Prune the tree to the desired size using
prune(fit, cp=)

Specifically, use printcp( ) to examine the cross-validated error results, select the complexity parameter associated with minimum error, and place it into the prune( ) function.
Alternatively, you can use the code fragment

```
fit$cptable[which.min(fit$cptable[,"xerror"]),"CP"]
```

to automatically select the complexity parameter associated with the smallest cross-validated error.

Grow the Tree usign package rpart

To grow a tree, use
rpart(formula, data=, method=,control=) where

formula is in the format
outcome ∼ predictor1+predictor2+predictor3+ect.
data= specifies the data frame
method= "class" for a classification tree
"anova" for a regression tree
control= optional parameters for controlling tree growth. For example, control=rpart.control(minsplit=30, cp=0.001) requires that the minimum number of observations in a node be 30 before attempting a split and that a split must decrease the overall lack of fit by a factor of 0.001 (cost complexity factor) before being attempted.

Examine the results

The following functions help us to examine the results.

printcp(fit) display cp table
plotcp(fit) plot cross-validation results
rsq.rpart(fit) plot approximate R-squared and relative error for different splits (2 plots). labels are only appropriate for the "anova" method.
print(fit) print results
summary(fit) detailed results including surrogate splits
plot(fit) plot decision tree
text(fit) label the decision tree plot
post(fit, file=) create postscript plot of decision tree

Nonparametric Trees

The party package provides nonparametric regression trees for nominal, ordinal, numeric, censored, and multivariate responses. You can create a regression or classification tree via the function
ctree(formula, data=)
The type of tree created will depend on the outcome variable (nominal factor, ordered factor, numeric, etc.). Tree growth is based on statistical stopping rules, so pruning should not be required.

R code

```
library (MASS)
?Boston
attach(Boston)
#############################################################
### Grow Tree, Prune Tree, pridiction
library (tree)
?tree
set.seed (1)
?sample
train = sample (1: nrow(Boston ), nrow(Boston )/2)  ## Training data
tree.boston =tree(medv~.,Boston ,subset =train) ### Grow Tree
summary (tree.boston )
plot(tree.boston ) ##########################plot the  tree
```

```
text(tree.boston ,pretty =0)
tree.boston
####################We can compare the predictions with the dataset
tree.boston2=tree(medv~lstat+dis,subset =train)
price.deciles <- quantile(Boston$medv, 0:10/10)
cut.prices    <- cut(Boston$medv, price.deciles, include.lowest=TRUE)
plot(Boston$lstat, Boston$rm, col=grey(10:2/11)[cut.prices], pch=20, xlab="Lstat",ylab="dis")
partition.tree(tree.boston2, ordvars=c("lstat","dis"), add=TRUE)
#############The flexibility of a tree is basically controlled
tree.model2 <- tree(medv~lstat+dis,subset =train, mindev=0.001)
plot(tree.model2)
text(tree.model2, cex=.75)
summary(tree.model2)
################################ Prune the Tree
?cv.tree
?plot
cv.boston =cv.tree(tree.boston )
cv.boston
plot(cv.boston$size ,cv.boston$dev ,type="b")#### finding the best number of nodes
prune.boston =prune.tree(tree.boston ,best =5)## Prune the Tree
plot(prune.boston )
text(prune.boston ,pretty =0)
################################# Predict response using Unpruning Tree
yhat=predict (tree.boston ,newdata =Boston [-train ,]) ######### Test data
boston.test=Boston [-train,"medv"]
plot(yhat ,boston.test)
abline (0,1)
mean((yhat-boston.test)^2)
#################Predict response using prune Tree
yhat=predict (prune.boston ,newdata =Boston [-train ,]) ######### Test data
boston.test=Boston [-train,"medv"]
plot(yhat ,boston.test)
abline (0,1)
mean((yhat-boston.test)^2)
##############################################################################
####### For classification Tree
table(predict(fit2), Boston[-train,]$medv)
######################################################################
############################################Regression Tree: another library {rpart}
library(rpart).
#### Grow Tree
fit1 <- rpart(medv~.,Boston ,subset =train, method="anova")
plot(fit1)
text(fit1)
printcp(fit1) # display the results
plotcp(fit1) # visualize cross-validation results
summary(fit1) # detailed summary of splits
##### Creat Additional Plots
```

```
# create additional plots
par(mfrow=c(1,2)) # two plots on one page
rsq.rpart(fit1) # visualize cross-validation results
##### plot Tree using information
# plot tree
par(mfrow=c(1,1))
plot(fit1, uniform=TRUE,
main="Regression Tree for medv ")
text(fit1, use.n=TRUE, all=TRUE, cex=.75)
# prune the tree
cpfit=fit1$cptable[which.min(fit1$cptable[,"xerror"]),"CP"]
pfit<- prune(fit1, cp=cpfit) # from cptable

# plot the pruned tree
plot(pfit, uniform=TRUE,
main="Pruned Regression Tree for Medv")
text(pfit, use.n=TRUE, all=TRUE, cex=.8)

############## To make prediction
Prediction <- predict(pfit, Boston[-train,])
boston.test=Boston [-train,"medv"]
plot(Prediction ,boston.test)
abline (0,1)
mean((Prediction-boston.test)^2)

################################ Nonparametric Trees via {party} package
# Conditional Inference Tree for Mileage-----------> do more research on this topic
library(party)
fit2 <- ctree(medv~.,data=na.omit(Boston) ,subset =train)
plot(fit2, main="Conditional Inference Tree for medv")
tr.pred = predict(fit2, newdata=Boston[-train,])
plot(tr.pred ,boston.test)
abline (0,1)
mean((tr.pred-boston.test)^2)
```