**10 – Tree-based Regression Models**

**10.1 - Introduction and Motivation**Tree-based modelling began as a primarily exploratory technique for uncovering structure in data. Specifically, the technique is useful for classification and regression problems where one has a set of predictor variables and a single response . When is nominal or ordinal (i.e non-numeric) then classification rules are determined from the given set of predictors which can be any mixture of data types, e.g.,

When the response is numeric, then regression tree rules for prediction are of the form:

The first tree models were called CART models, (**C**lassification **A**nd **R**egression **T**rees) and were fit using an algorithm called ***recursive partitioning*** which is described in the case of regression trees in the next section. As computing power has increased there has been work on improving the performance of tree-based models. We will certainly examine these improvements as these tend to produce among the best, if not the best, off-the-shelf predictive models. Thus tree-based models have become quite popular.

Some additional reasons for their popularity:

**1.** In certain applications, especially where the set of predictors contains a mix of numeric variables and factors, tree-based models are sometimes easier to interpret and discuss than linear models.

**2.** Tree-based models are invariant to monotone transformations of predictor variables, thus the precise form in which these appear in the model is irrelevant. As we have seen in several earlier covered methods and associated examples this is a particularly appealing property!!

**3.** Tree-based models are more adept at capturing non-additive behavior; the standard MLR model does not allow interactions between variables unless they are pre-specified and of a particular multiplicative form. We saw that the MARS models could find interactions automatically by specifying a degree = 2 fit, but these interactions will only have a very specific functional form. Projection pursuit and neural networks are also non-additive and will have variable interactions though they will generally be quite complex. Tree-models because of the way they are constructed naturally have interactions, typically lots of them!

**10.2 - Regression Trees and Recursive Partitioning**

The form of the fitted surface or smooth obtained from a regression tree is

where the are constants and the are regions defined a series of binary splits. If all the predictors are numeric these regions form a set of disjoint hyper-rectangles with sides parallel to the axes such that

Regardless of how the neighborhoods are defined if we use the least squares criterion for each region

the best estimator of the response, , is just the average of the in the region , i.e.

.

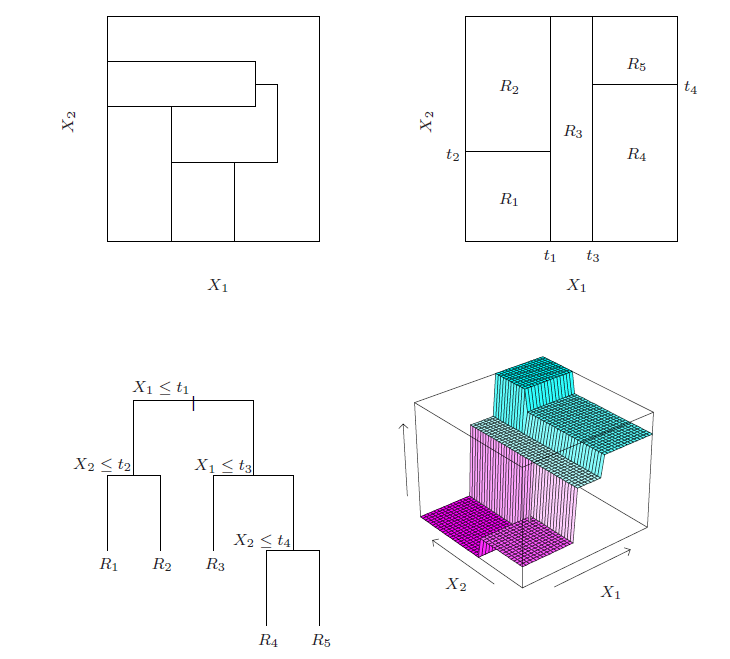
Thus to obtain a fitted regression tree model we need to somehow obtain the neighborhoods This is accomplished by an algorithm called *recursive partitioning*, see Breiman et al. (1984). We present the basic idea below though an example for the case where the number of neighborhoods and the number of predictor variables The task of determining neighborhoods is solved by determining a *split coordinate or variate* , i.e. which variable to split on, and *split point* . A split coordinate and split point define the rectangles as

The residual sum of squares (RSS) for a split determined by is

The goal at any given stage is to find the pair such that is minimal or the overall RSS is maximally reduced. This may seem overwhelming, however this only requires examining at most splits for each variable because the points in a neighborhood only change when the split point crosses an observed value. If we wish to split into three neighborhoods, i.e. split or after the first split, we have possibilities for the first split and possibilities for the second split, given the first split. In total we have operations to find the best splits for neighborhoods. In general for neighborhoods we have,

possibilities if all predictors are numeric! This gets too big for an exhaustive search, therefore we use the technique for recursively. This is the basic idea of recursive partitioning. One starts with the first split and obtains as explained above. This split stays fixed and the same splitting procedure is applied recursively to the two regions . Thus it is a greedy algorithm as previous choices are not reconsidered once they are made. This procedure is then repeated until we reach some stopping criterion such as the nodes become homogenous or contain very few observations. The rpart function uses two such stopping criteria. A node will not be split if it contains fewer minsplit observations (default =20). Additionally we can specify the minimum number of observations in terminal node by specifying a value for minbucket (default = ).

The figures below are from pg. 306 of *Elements of Statistical Learning* (Hastie, et al.) show a hypothetical tree fit based on two numeric predictors .



Let's examine these ideas using the ozone pollution data for the Los Angeles Basin discussed earlier in the course. For simplicity we consider the case where . Here we will develop a regression tree using rpart for predicting upper ozone concentration using the temperature at Sandburg Air Force Base (safb) and inversion base height (inbh).

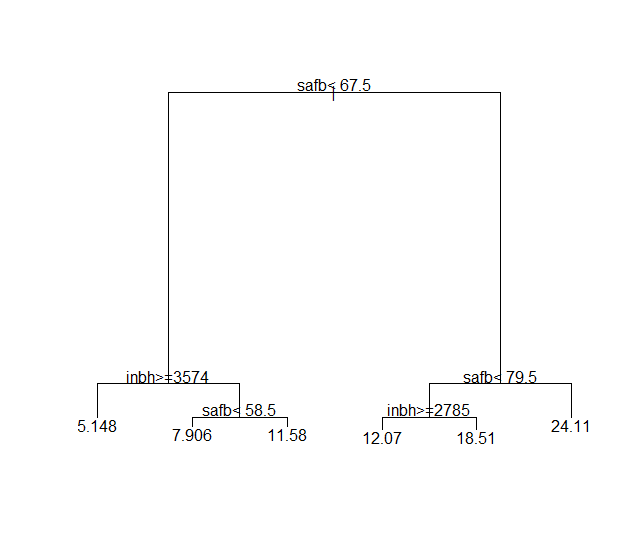
> library(rpart)

> oz.rpart <- rpart(upoz ~ inbh + safb,data=Ozdata)

> summary(oz.rpart)

> plot(oz.rpart)

> text(oz.rpart)



> post(oz.rpart,"Regression Tree for Upper Ozone Concentration")

**Plot the fitted surface**

> x1 = seq(min(Ozdata$inbh),max(Ozdata$inbh),length=100)

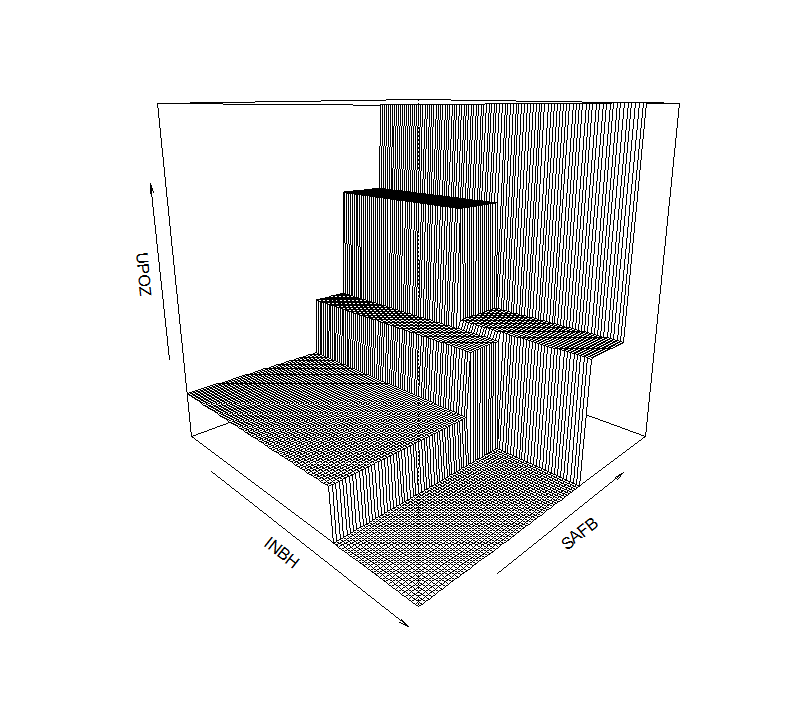
> x2 = seq(min(Ozdata$safb),max(Ozdata$safb),length=100)

> x = expand.grid(inbh=x1,safb=x2)

> ypred = predict(oz.rpart,newdata=x)

> persp(x1,x2,z=matrix(ypred,100,100),theta=45,xlab="INBH",

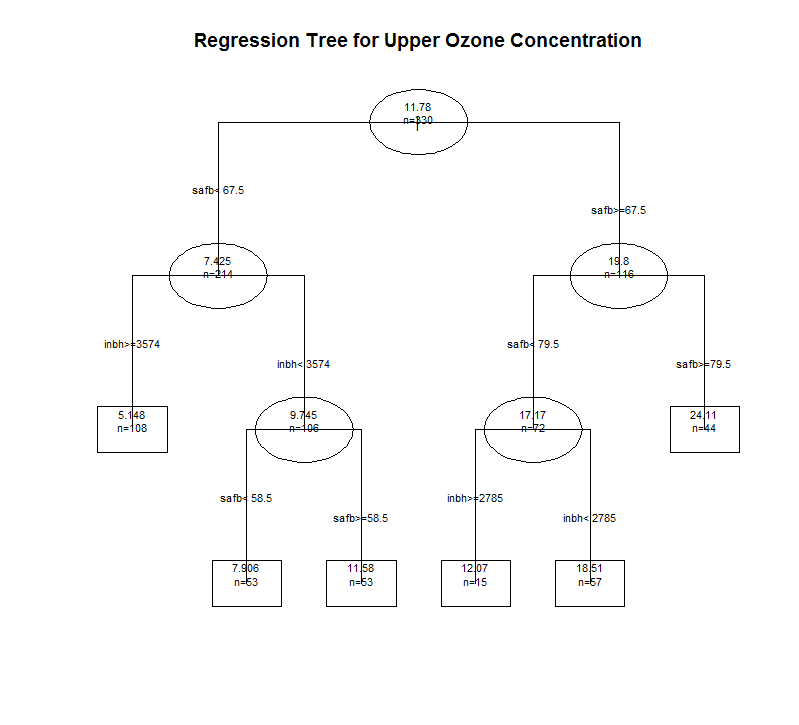
ylab="SAFB",zlab="UPOZ")



> plot(oz.rpart,uniform=T,branch=1,compress=T,margin=0.05,cex=.5)

> text(oz.rpart,all=T,use.n=T,fancy=T,cex=.7)

> title(main="Regression Tree for Upper Ozone Concentration")



**Example 10.1: Infant Mortality Rates for 77 Largest U.S. Cities in 2000**

In this example we will examine how to build regression trees using functions in the packages rpart and tree. We will also examine use of the maptree and rpart.plot packages to plot the results.

> infmort.rpart = rpart(infmort~.,data=City,minsplit=10)

> summary(infmort.rpart)

Call:

rpart(formula = infmort ~ ., data = City, minsplit = 10)

n= 77

CP nsplit rel error xerror xstd

1 0.53569704 0 1.00000000 1.0108944 0.18722505

2 0.10310955 1 0.46430296 0.5912858 0.08956209

3 0.08865804 2 0.36119341 0.6386809 0.09834998

4 0.03838630 3 0.27253537 0.5959633 0.09376897

5 0.03645758 4 0.23414907 0.6205958 0.11162033

6 0.02532618 5 0.19769149 0.6432091 0.11543351

7 0.02242248 6 0.17236531 0.6792245 0.11551694

8 0.01968056 7 0.14994283 0.7060502 0.11773100

9 0.01322338 8 0.13026228 0.6949660 0.11671223

10 0.01040108 9 0.11703890 0.6661967 0.11526389

11 0.01019740 10 0.10663782 0.6749224 0.11583334

12 0.01000000 11 0.09644043 0.6749224 0.11583334

Node number 1: 77 observations, complexity param=0.535697

mean=12.03896, MSE=12.31978

left son=2 (52 obs) right son=3 (25 obs)

Primary splits:

pct.black < 29.55 to the left, improve=0.5356970, (0 missing)

growth < -5.55 to the right, improve=0.4818361, (0 missing)

pct1par < 31.25 to the left, improve=0.4493385, (0 missing)

precip < 36.45 to the left, improve=0.3765841, (0 missing)

laborchg < 2.85 to the right, improve=0.3481261, (0 missing)

Surrogate splits:

growth < -2.6 to the right, agree=0.896, adj=0.68, (0 split)

pct1par < 31.25 to the left, agree=0.896, adj=0.68, (0 split)

laborchg < 2.85 to the right, agree=0.857, adj=0.56, (0 split)

poverty < 21.5 to the left, agree=0.844, adj=0.52, (0 split)

income < 24711 to the right, agree=0.818, adj=0.44, (0 split)

Node number 2: 52 observations, complexity param=0.08865804

mean=10.25769, MSE=4.433595

left son=4 (34 obs) right son=5 (18 obs)

Primary splits:

precip < 36.2 to the left, improve=0.3647980, (0 missing)

pct.black < 12.6 to the left, improve=0.3395304, (0 missing)

pct.hisp < 4.15 to the right, improve=0.3325635, (0 missing)

pct1hous < 29.05 to the left, improve=0.3058060, (0 missing)

hisp.pop < 14321 to the right, improve=0.2745090, (0 missing)

Surrogate splits:

pct.black < 22.3 to the left, agree=0.865, adj=0.611, (0 split)

pct.hisp < 3.8 to the right, agree=0.865, adj=0.611, (0 split)

hisp.pop < 7790.5 to the right, agree=0.808, adj=0.444, (0 split)

growth < 6.95 to the right, agree=0.769, adj=0.333, (0 split)

taxes < 427 to the left, agree=0.769, adj=0.333, (0 split)

Node number 3: 25 observations, complexity param=0.1031095

mean=15.744, MSE=8.396064

left son=6 (20 obs) right son=7 (5 obs)

Primary splits:

pct1par < 45.05 to the left, improve=0.4659903, (0 missing)

growth < -5.55 to the right, improve=0.4215004, (0 missing)

pct.black < 62.6 to the left, improve=0.4061168, (0 missing)

pop2 < 637364.5 to the left, improve=0.3398599, (0 missing)

black.pop < 321232.5 to the left, improve=0.3398599, (0 missing)

Surrogate splits:

pct.black < 56.85 to the left, agree=0.92, adj=0.6, (0 split)

growth < -15.6 to the right, agree=0.88, adj=0.4, (0 split)

welfare < 22.05 to the left, agree=0.88, adj=0.4, (0 split)

unemprt < 11.3 to the left, agree=0.88, adj=0.4, (0 split)

black.pop < 367170.5 to the left, agree=0.84, adj=0.2, (0 split)

Node number 4: 34 observations, complexity param=0.0383863

mean=9.332353, MSE=2.830424

left son=8 (5 obs) right son=9 (29 obs)

Primary splits:

medrent < 614 to the right, improve=0.3783900, (0 missing)

black.pop < 9584.5 to the left, improve=0.3326486, (0 missing)

pct.black < 2.8 to the left, improve=0.3326486, (0 missing)

income < 29782 to the right, improve=0.2974224, (0 missing)

medv < 160100 to the right, improve=0.2594415, (0 missing)

Surrogate splits:

area < 48.35 to the left, agree=0.941, adj=0.6, (0 split)

income < 35105 to the right, agree=0.941, adj=0.6, (0 split)

medv < 251800 to the right, agree=0.941, adj=0.6, (0 split)

popdens < 9701.5 to the right, agree=0.912, adj=0.4, (0 split)

black.pop < 9584.5 to the left, agree=0.912, adj=0.4, (0 split)

Node number 5: 18 observations, complexity param=0.02242248

mean=12.00556, MSE=2.789414

left son=10 (7 obs) right son=11 (11 obs)

Primary splits:

july < 78.95 to the right, improve=0.4236351, (0 missing)

pctmanu < 11.65 to the right, improve=0.3333122, (0 missing)

pct.AIP < 1.5 to the left, improve=0.3293758, (0 missing)

pctdeg < 18.25 to the left, improve=0.3078859, (0 missing)

precip < 49.7 to the right, improve=0.3078859, (0 missing)

Surrogate splits:

oldhous < 15.05 to the left, agree=0.833, adj=0.571, (0 split)

precip < 46.15 to the right, agree=0.833, adj=0.571, (0 split)

area < 417.5 to the right, agree=0.778, adj=0.429, (0 split)

pctdeg < 19.4 to the left, agree=0.778, adj=0.429, (0 split)

unemprt < 5.7 to the right, agree=0.778, adj=0.429, (0 split)

Node number 6: 20 observations, complexity param=0.03645758

mean=14.755, MSE=4.250475

left son=12 (10 obs) right son=13 (10 obs)

Primary splits:

growth < -5.55 to the right, improve=0.4068310, (0 missing)

medv < 50050 to the right, improve=0.4023256, (0 missing)

pct.AIP < 0.85 to the right, improve=0.4019953, (0 missing)

pctrent < 54.3 to the right, improve=0.3764815, (0 missing)

pctold < 13.95 to the left, improve=0.3670365, (0 missing)

Surrogate splits:

pctold < 13.95 to the left, agree=0.85, adj=0.7, (0 split)

laborchg < -1.8 to the right, agree=0.85, adj=0.7, (0 split)

black.pop < 165806 to the left, agree=0.80, adj=0.6, (0 split)

pct.black < 45.25 to the left, agree=0.80, adj=0.6, (0 split)

pct.AIP < 1.05 to the right, agree=0.80, adj=0.6, (0 split)

Node number 7: 5 observations

mean=19.7, MSE=5.416

Node number 8: 5 observations

mean=6.84, MSE=1.5784

Node number 9: 29 observations, complexity param=0.01968056

mean=9.762069, MSE=1.79063

left son=18 (3 obs) right son=19 (26 obs)

Primary splits:

laborchg < 55.9 to the right, improve=0.3595234, (0 missing)

growth < 61.7 to the right, improve=0.3439875, (0 missing)

taxes < 281.5 to the left, improve=0.3185654, (0 missing)

july < 82.15 to the right, improve=0.2644400, (0 missing)

pct.hisp < 5.8 to the right, improve=0.2537809, (0 missing)

Surrogate splits:

growth < 61.7 to the right, agree=0.966, adj=0.667, (0 split)

welfare < 3.85 to the left, agree=0.966, adj=0.667, (0 split)

pct1par < 17.95 to the left, agree=0.966, adj=0.667, (0 split)

ptrans < 1 to the left, agree=0.966, adj=0.667, (0 split)

pop2 < 167632.5 to the left, agree=0.931, adj=0.333, (0 split)

Node number 10: 7 observations

mean=10.64286, MSE=2.21102

Node number 11: 11 observations, complexity param=0.0101974

mean=12.87273, MSE=1.223802

left son=22 (6 obs) right son=23 (5 obs)

Primary splits:

pctmanu < 11.95 to the right, improve=0.7185868, (0 missing)

july < 72 to the left, improve=0.5226933, (0 missing)

black.pop < 54663.5 to the left, improve=0.5125632, (0 missing)

pop2 < 396053.5 to the left, improve=0.3858185, (0 missing)

pctenr < 88.65 to the right, improve=0.3858185, (0 missing)

Surrogate splits:

popdens < 6395.5 to the left, agree=0.818, adj=0.6, (0 split)

black.pop < 54663.5 to the left, agree=0.818, adj=0.6, (0 split)

taxes < 591 to the left, agree=0.818, adj=0.6, (0 split)

welfare < 8.15 to the left, agree=0.818, adj=0.6, (0 split)

poverty < 15.85 to the left, agree=0.818, adj=0.6, (0 split)

Node number 12: 10 observations, complexity param=0.01322338

mean=13.44, MSE=1.8084

left son=24 (5 obs) right son=25 (5 obs)

Primary splits:

pct1hous < 30.1 to the right, improve=0.6936518, (0 missing)

precip < 41.9 to the left, improve=0.6936518, (0 missing)

laborchg < 1.05 to the left, improve=0.6902813, (0 missing)

welfare < 13.2 to the right, improve=0.6619479, (0 missing)

ptrans < 5.25 to the right, improve=0.6087241, (0 missing)

Surrogate splits:

precip < 41.9 to the left, agree=1.0, adj=1.0, (0 split)

pop2 < 327405.5 to the right, agree=0.9, adj=0.8, (0 split)

black.pop < 127297.5 to the right, agree=0.9, adj=0.8, (0 split)

pctold < 11.75 to the right, agree=0.9, adj=0.8, (0 split)

welfare < 13.2 to the right, agree=0.9, adj=0.8, (0 split)

Node number 13: 10 observations, complexity param=0.02532618

mean=16.07, MSE=3.2341

left son=26 (5 obs) right son=27 (5 obs)

Primary splits:

pctrent < 52.9 to the right, improve=0.7428651, (0 missing)

pct1hous < 32 to the right, improve=0.5460870, (0 missing)

pct1par < 39.55 to the right, improve=0.4378652, (0 missing)

pct.hisp < 0.8 to the right, improve=0.4277646, (0 missing)

pct.AIP < 0.85 to the right, improve=0.4277646, (0 missing)

Surrogate splits:

area < 62 to the left, agree=0.8, adj=0.6, (0 split)

pct.hisp < 0.8 to the right, agree=0.8, adj=0.6, (0 split)

pct.AIP < 0.85 to the right, agree=0.8, adj=0.6, (0 split)

pctdeg < 18.5 to the right, agree=0.8, adj=0.6, (0 split)

taxes < 560 to the right, agree=0.8, adj=0.6, (0 split)

Node number 18: 3 observations

mean=7.4, MSE=1.886667

Node number 19: 26 observations, complexity param=0.01040108

mean=10.03462, MSE=1.061494

left son=38 (14 obs) right son=39 (12 obs)

Primary splits:

pct.hisp < 20.35 to the right, improve=0.3575042, (0 missing)

hisp.pop < 55739.5 to the right, improve=0.3013295, (0 missing)

pctold < 11.55 to the left, improve=0.3007143, (0 missing)

pctrent < 41 to the right, improve=0.2742615, (0 missing)

taxes < 375.5 to the left, improve=0.2577731, (0 missing)

Surrogate splits:

hisp.pop < 39157.5 to the right, agree=0.885, adj=0.75, (0 split)

pctold < 11.15 to the left, agree=0.769, adj=0.50, (0 split)

pct1par < 23 to the right, agree=0.769, adj=0.50, (0 split)

pctrent < 41.85 to the right, agree=0.769, adj=0.50, (0 split)

precip < 15.1 to the left, agree=0.769, adj=0.50, (0 split)

Node number 22: 6 observations

mean=12.01667, MSE=0.4013889

Node number 23: 5 observations

mean=13.9, MSE=0.276

Node number 24: 5 observations

mean=12.32, MSE=0.6376

Node number 25: 5 observations

mean=14.56, MSE=0.4704

Node number 26: 5 observations

mean=14.52, MSE=0.9496

Node number 27: 5 observations

mean=17.62, MSE=0.7136

Node number 38: 14 observations

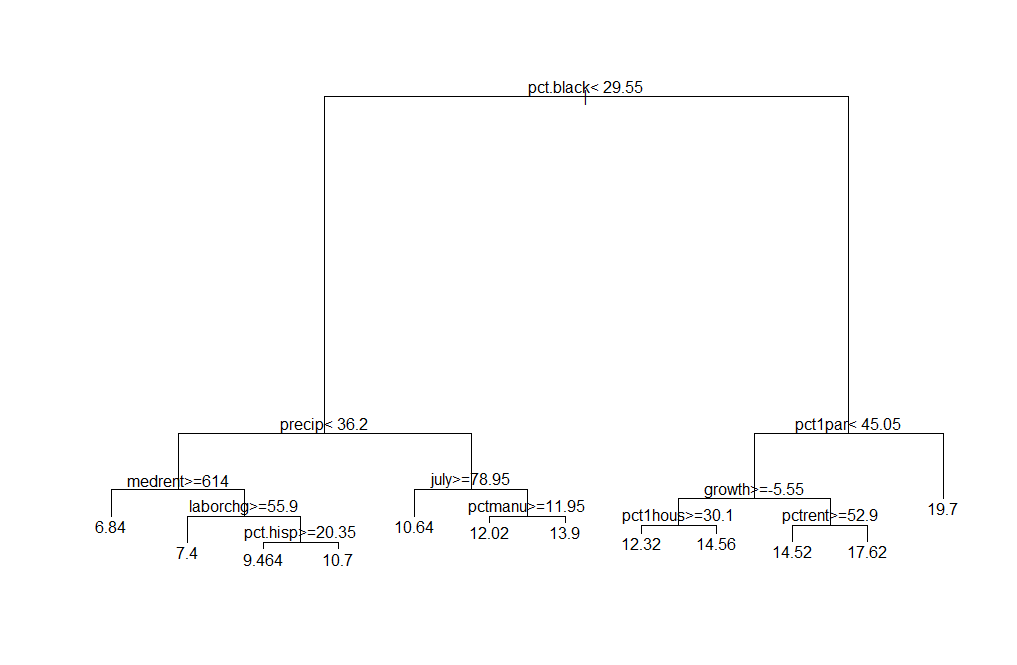
mean=9.464286, MSE=0.7994388

Node number 39: 12 observations

mean=10.7, MSE=0.545

> plot(infmort.rpart)

> text(infmort.rpart)



Typing the name of an rpart object at the prompt will display the tree non-graphically. Below is the tree infmort.rpart displayed non-graphical and graphically. You should be able to make the connections between the output and graphical representation below.

> infmort.rpart

n= 77

node), split, n, deviance, yval

\* denotes terminal node

1) root 77 948.623100 12.038960

2) pct.black< 29.55 52 230.546900 10.257690

4) precip< 36.2 34 96.234410 9.332353

8) medrent>=614 5 7.892000 6.840000 \*

9) medrent< 614 29 51.928280 9.762069

18) laborchg>=55.9 3 5.660000 7.400000 \*

19) laborchg< 55.9 26 27.598850 10.034620

38) pct.hisp>=20.35 14 11.192140 9.464286 \*

39) pct.hisp< 20.35 12 6.540000 10.700000 \*

5) precip>=36.2 18 50.209440 12.005560

10) july>=78.95 7 15.477140 10.642860 \*

11) july< 78.95 11 13.461820 12.872730

22) pctmanu>=11.95 6 2.408333 12.016670 \*

23) pctmanu< 11.95 5 1.380000 13.900000 \*

3) pct.black>=29.55 25 209.901600 15.744000

6) pct1par< 45.05 20 85.009500 14.755000

12) growth>=-5.55 10 18.084000 13.440000

24) precip< 41.9 5 3.188000 12.320000 \*

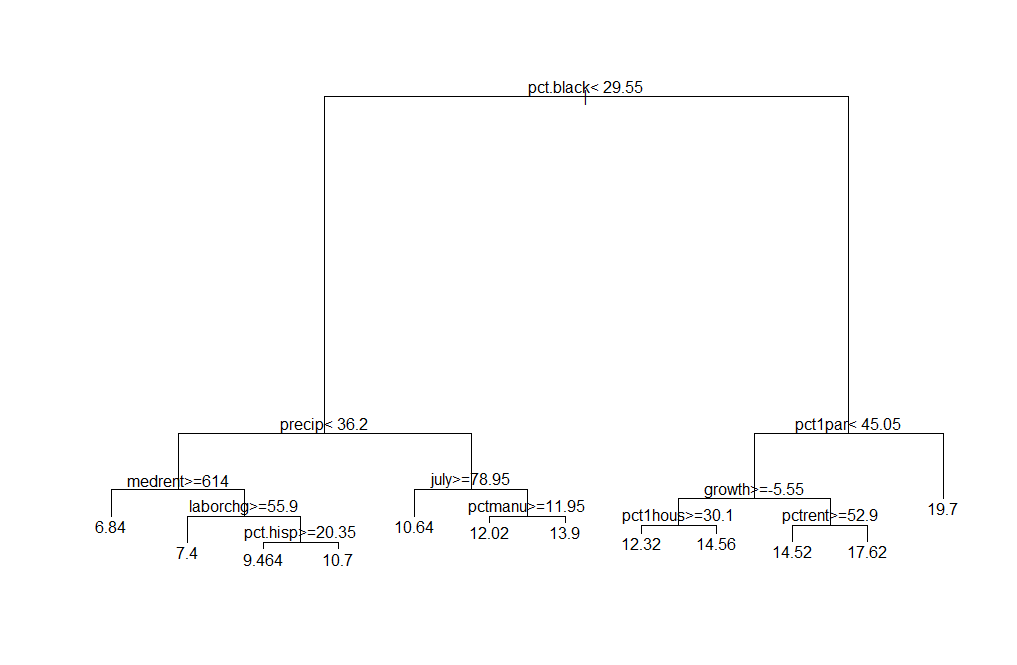
25) precip>=41.9 5 2.352000 14.560000 \*

13) growth< -5.55 10 32.341000 16.070000

26) pctrent>=52.9 5 4.748000 14.520000 \*

27) pctrent< 52.9 5 3.568000 17.620000 \*

7) pct1par>=45.05 5 27.080000 19.700000 \*



The command path.rpart allows you to interactively explore the paths to the terminal nodes in a fitted CART model.

> path.rpart(infmort.rpart) 🡨 clicking on 3 leftmost terminal nodes, right-click to stop

node number: 8

root

pct.black< 29.55

precip< 36.2

medrent>=614

node number: 18

root

pct.black< 29.55

precip< 36.2

medrent< 614

laborchg>=55.9

node number: 38

root

pct.black< 29.55

precip< 36.2

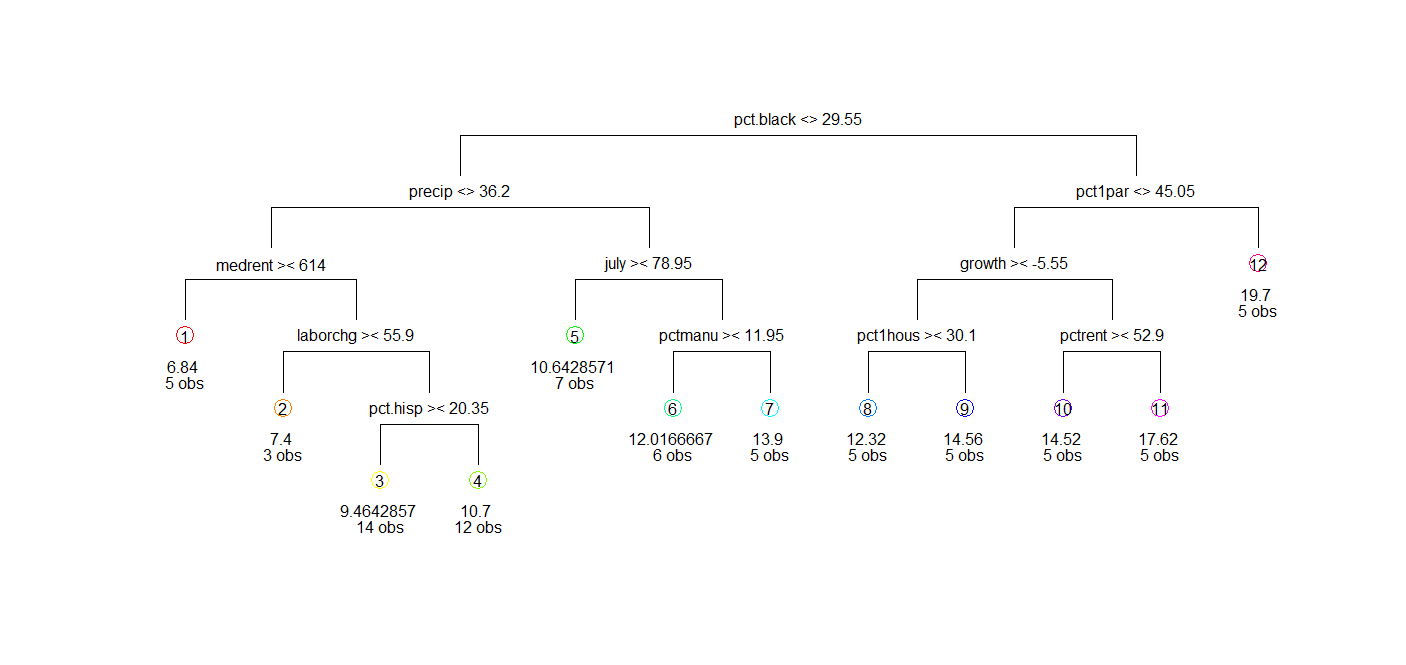
medrent< 614

laborchg< 55.9

pct.hisp>=20.35

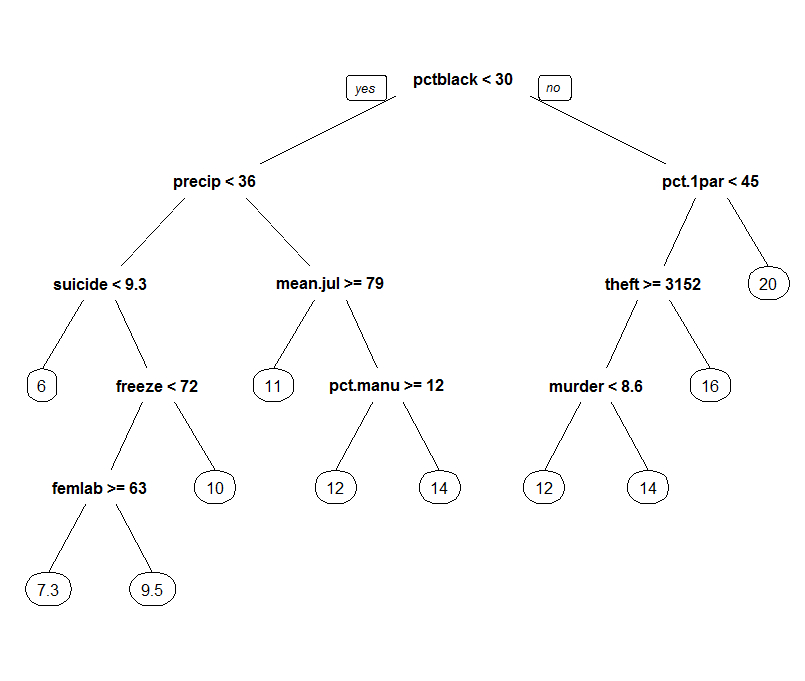
The package maptree has a function draw.tree() that plots trees slightly differently.

> draw.tree(infmort.rpart)

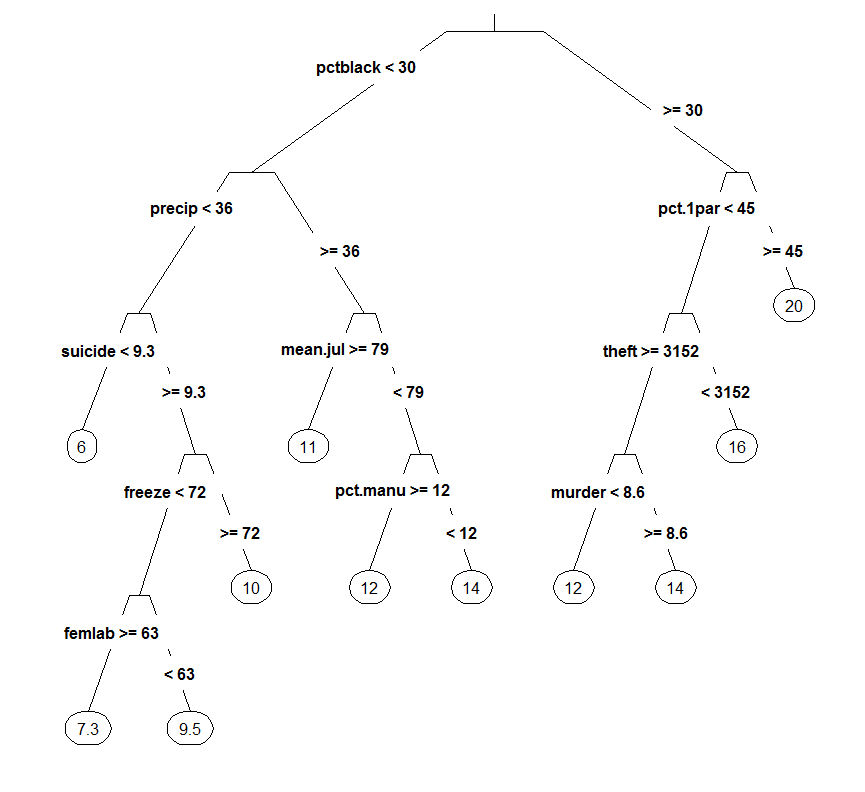


The package rpart.plot contains a function prp() that can be used to obtain pretty tree plots. There are **numerous** options that can set to customize the plot, see the help file.

> prp(infmort.rpart)



> prp(infmort.rpart,type=3)



**Examining cross-validation results**

> printcp(infmort.rpart)

Regression tree:

rpart(formula = infmort ~ ., data = City, control = rpart.control(minsplit = 10))

Variables actually used in tree construction:

[1] growth july laborchg medrent pct.black pct.hisp pct1par pctmanu

[9] pctrent precip

Root node error: 948.62/77 = 12.32

n= 77

CP nsplit rel error xerror xstd

1 0.535697 0 1.00000 1.01171 0.188827

2 0.103110 1 0.46430 0.62256 0.102400

3 0.088658 2 0.36119 0.54032 0.093558

4 0.038386 3 0.27254 0.48854 0.086377

5 0.036458 4 0.23415 0.62167 0.108241

6 0.025326 5 0.19769 0.63754 0.106466

7 0.022422 6 0.17237 0.62111 0.105139

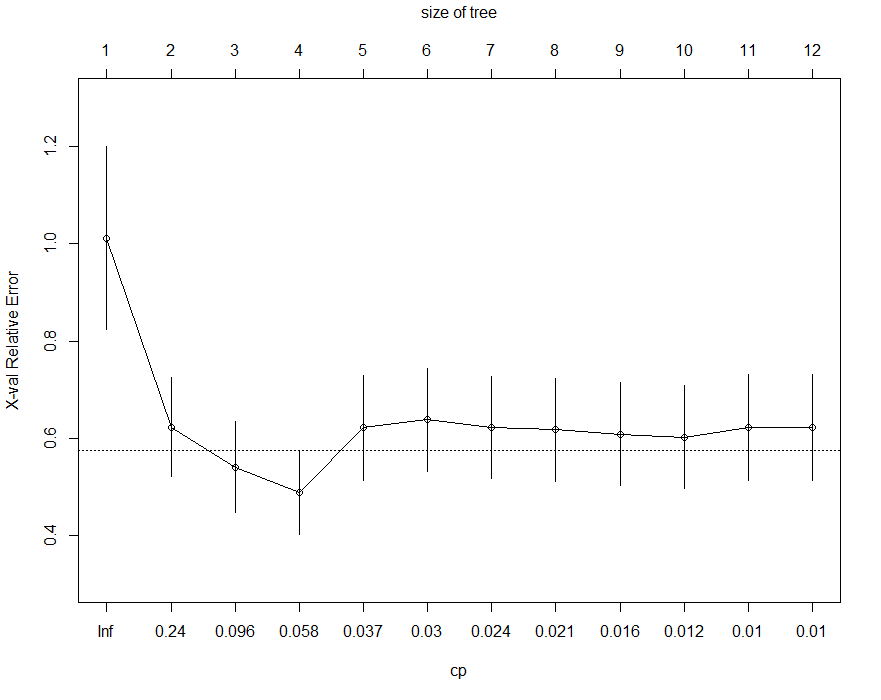
8 0.019681 7 0.14994 0.61694 0.106037

9 0.013223 8 0.13026 0.60831 0.105624

10 0.010401 9 0.11704 0.60209 0.105360

11 0.010197 10 0.10664 0.62146 0.108721

12 0.010000 11 0.09644 0.62146 0.108721  
> plotcp(infmort.rpart)



***The 1-SE rule for choosing a tree-size***

1. Find the smallest xerror and add the corresponding xstd to it.
2. Choose the first tree size that has a xerror smaller than the result from step 1.

A very small tree (2 splits or 3 terminal nodes) is suggested by cross-validation, but the larger trees cross-validate reasonably well so we might choose a larger tree just because it is more interesting from a practical standpoint.

> plot(City$infmort,predict(infmort.rpart))

> row.names(City)

[1] "New.York.NY" "Los.Angeles.CA" "Chicago.IL" "Houston.TX" "Philadelphia.PA"

[6] "San.Diego.CA" "Dallas.TX" "Phoenix.AZ" "Detroit.MI" "San.Antonio.TX"

[11] "San.Jose.CA" "Indianapolis.IN" "San.Francisco.CA" "Baltimore.MD" "Jacksonville.FL"

[16] "Columbus.OH" "Milwaukee.WI" "Memphis.TN" "Washington.DC" "Boston.MA"

[21] "El.Paso.TX" "Seattle.WA" "Cleveland.OH" "Nashville.Davidson.TN" "Austin.TX"

[26] "New.Orleans.LA" "Denver.CO" "Fort.Worth.TX" "Oklahoma.City.OK" "Portland.OR"

[31] "Long.Beach.CA" "Kansas.City.MO" "Virginia.Beach.VA" "Charlotte.NC" "Tucson.AZ"

[36] "Albuquerque.NM" "Atlanta.GA" "St.Louis.MO" "Sacramento.CA" "Fresno.CA"

[41] "Tulsa.OK" "Oakland.CA" "Honolulu.CDP.HI" "Miami.FL" "Pittsburgh.PA"

[46] "Cincinnati.OH" "Minneapolis.MN" "Omaha.NE" "Toledo.OH" "Buffalo.NY"

[51] "Wichita.KS" "Mesa.AZ" "Colorado.Springs.CO" "Las.Vegas.NV" "Santa.Ana.CA"

[56] "Tampa.FL" "Arlington.TX" "Anaheim.CA" "Louisville.KY" "St.Paul.MN"

[61] "Newark.NJ" "Corpus.Christi.TX" "Birmingham.AL" "Norfolk.VA" "Anchorage.AK"

[66] "Aurora.CO" "Riverside.CA" "St.Petersburg.FL" "Rochester.NY" "Lexington.Fayette.KY"

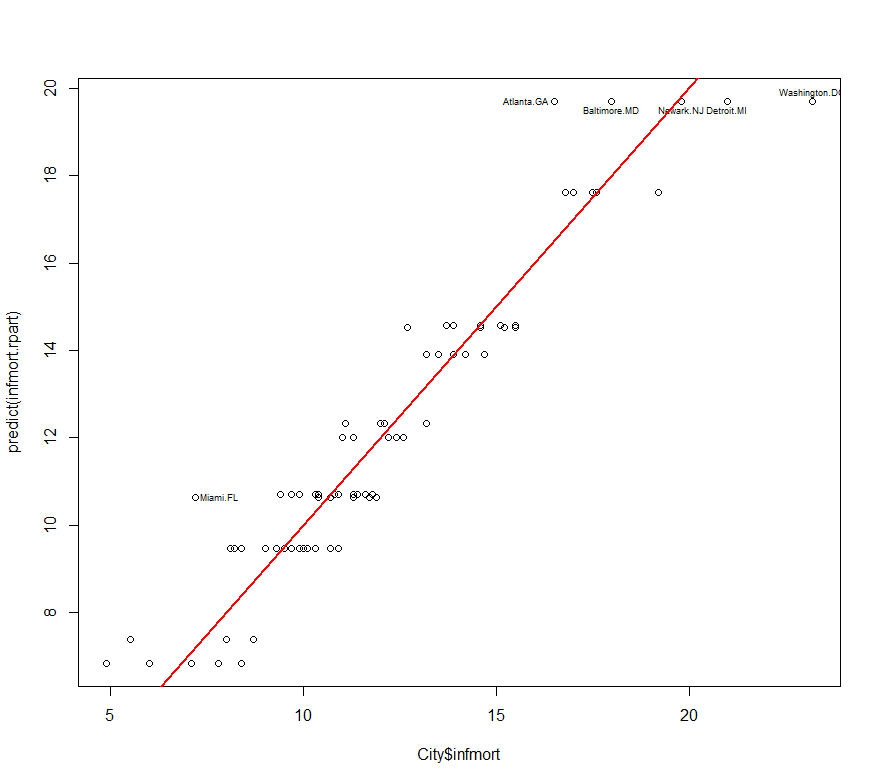
[71] "Jersey.City.NJ" "Baton.Rouge.LA" "Akron.OH" "Raleigh.NC" "Stockton.CA"

[76] "Richmond.VA" "Mobile.AL"

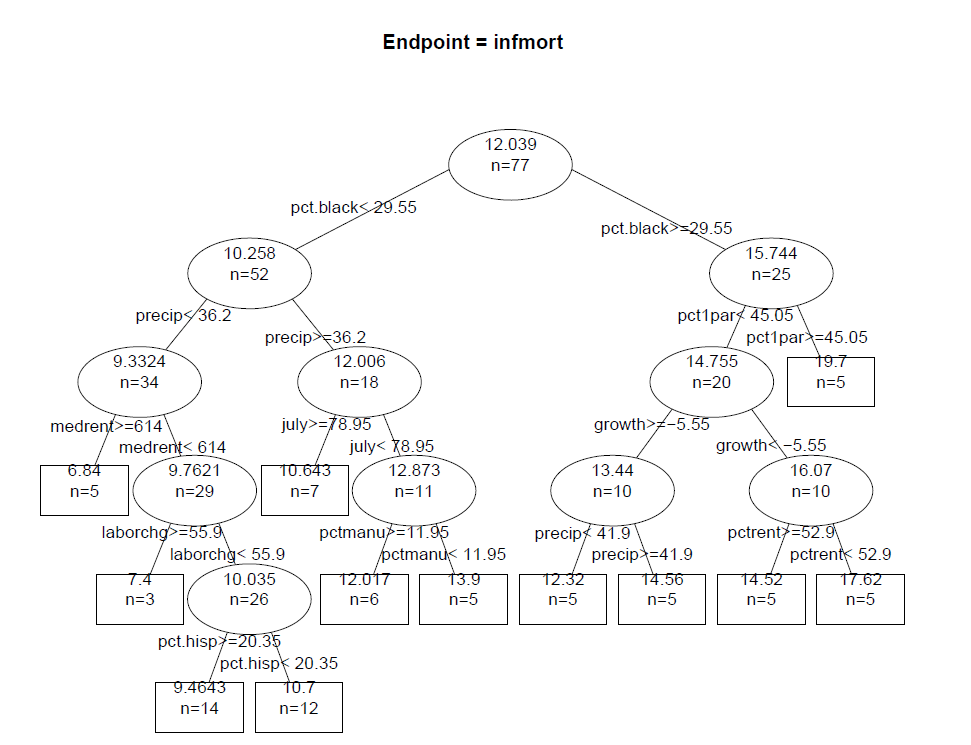
> identify(City$infmort,predict(infmort.rpart),labels=row.names(City),cex=.5)

[1] 14 19 37 44 61 63 🡨 identify some interesting points

> abline(0,1,col=”red”,lwd=2) 🡨 adds line to the plot



> post(infmort.rpart) 🡨 creates a postscript version of tree. You may need to download a postscript viewer add-on for Adobe Reader ® to open them, although the most recent versions of Adobe on your laptop may already be postscript compatible. I have ***Adobe*** ***Distiller ®*** on my desktop and laptop which converts postscript files to Adobe Reader format.  
  
If you don’t have one use Google to search for a “Postscript Viewer” and grab the one off of a site like ***cnet*** - (<http://download.cnet.com/Postscript-Viewer/3000-2094_4-10845650.html>) 🡨 old one, maybe not even around anymore!

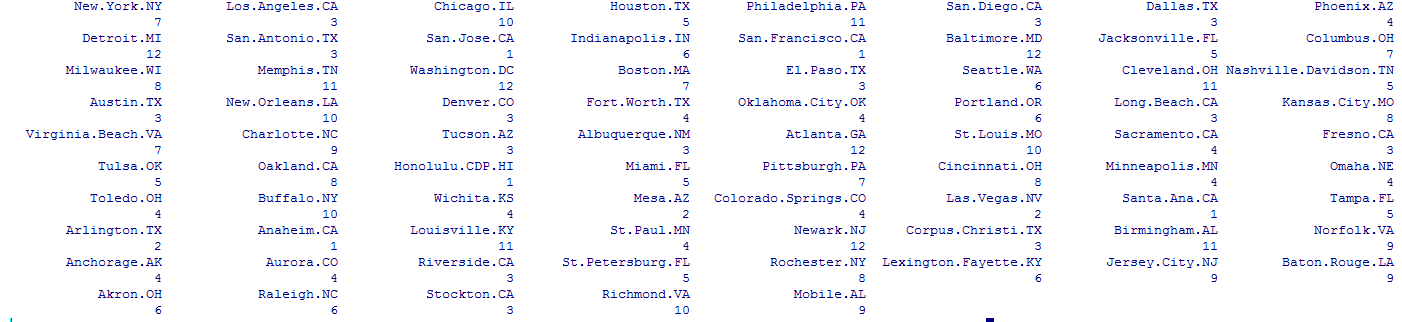


Postscript version of the infant mortality regression tree.

Another function in the maptree library is the group.tree command that will label the observations in according to the terminal nodes they are in. This can be particularly interesting when the observations have meaningful labels or are spatially distributed.

> infmort.groups = group.tree(infmort.rpart)

> infmort.groups



Here is a little function to display groups of observations in a data set given the group identifier.

groups = function(g,dframe) {

ng <- length(unique(g))

for(i in 1:ng) {

cat(paste("GROUP ", i))

cat("\n")   
 cat("=========================================================\n")

cat(row.names(dframe)[g == i])

cat("\n\n")

}

cat(" \n\n")

}

> groups(infmort.groups,City)

GROUP 1

====================================================================

San.Jose.CA San.Francisco.CA Honolulu.CDP.HI Santa.Ana.CA Anaheim.CA

GROUP 2

==================================

Mesa.AZ Las.Vegas.NV Arlington.TX

GROUP 3

===============================================================================

Los.Angeles.CA San.Diego.CA Dallas.TX San.Antonio.TX El.Paso.TX Austin.TX Denver.CO Long.Beach.CA Tucson.AZ Albuquerque.NM Fresno.CA Corpus.Christi.TX Riverside.CA Stockton.CA

GROUP 4

===============================================================================

Phoenix.AZ Fort.Worth.TX Oklahoma.City.OK Sacramento.CA Minneapolis.MN Omaha.NE Toledo.OH Wichita.KS Colorado.Springs.CO St.Paul.MN Anchorage.AK Aurora.CO

GROUP 5

===============================================================================

Houston.TX Jacksonville.FL Nashville.Davidson.TN Tulsa.OK Miami.FL Tampa.FL St.Petersburg.FL

GROUP 6

===============================================================================

Indianapolis.IN Seattle.WA Portland.OR Lexington.Fayette.KY Akron.OH Raleigh.NC

GROUP 7

=================================================================

New.York.NY Columbus.OH Boston.MA Virginia.Beach.VA Pittsburgh.PA

GROUP 8

=================================================================

Milwaukee.WI Kansas.City.MO Oakland.CA Cincinnati.OH Rochester.NY

GROUP 9

===============================================================

Charlotte.NC Norfolk.VA Jersey.City.NJ Baton.Rouge.LA Mobile.AL

GROUP 10

============================================================

Chicago.IL New.Orleans.LA St.Louis.MO Buffalo.NY Richmond.VA

GROUP 11

===================================================================

Philadelphia.PA Memphis.TN Cleveland.OH Louisville.KY Birmingham.AL

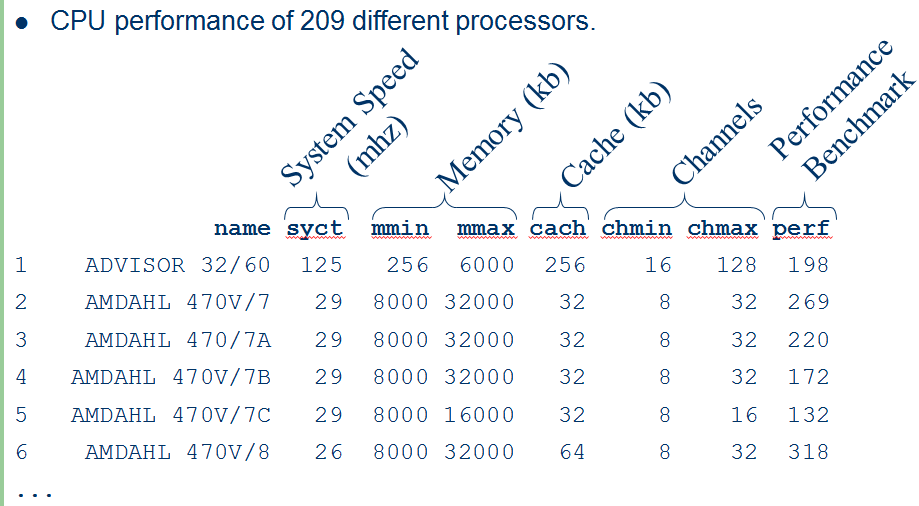
GROUP 12

==========================================================

Detroit.MI Baltimore.MD Washington.DC Atlanta.GA Newark.NJ

The groups of cities certainly make sense intuitively.

**Example 10.2: Predicting/Modeling CPU Performance**



> cpus = read.table(file.choose(),hReader=T,sep=”,”) 🡨 Open **CPUS.csv**

> head(cpus)

name syct mmin mmax cach chmin chmax perf perf2

1 ADVISOR 32/60 125 256 6000 256 16 128 198 199

2 AMDAHL 470V/7 29 8000 32000 32 8 32 269 253

3 AMDAHL 470/7A 29 8000 32000 32 8 32 220 253

4 AMDAHL 470V/7B 29 8000 32000 32 8 32 172 253

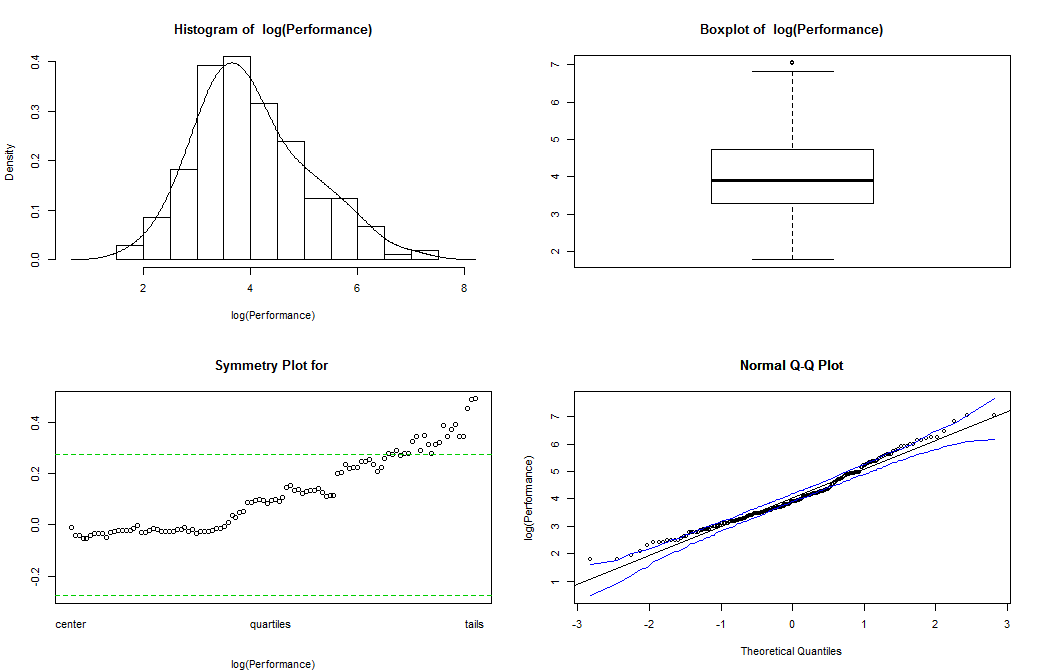
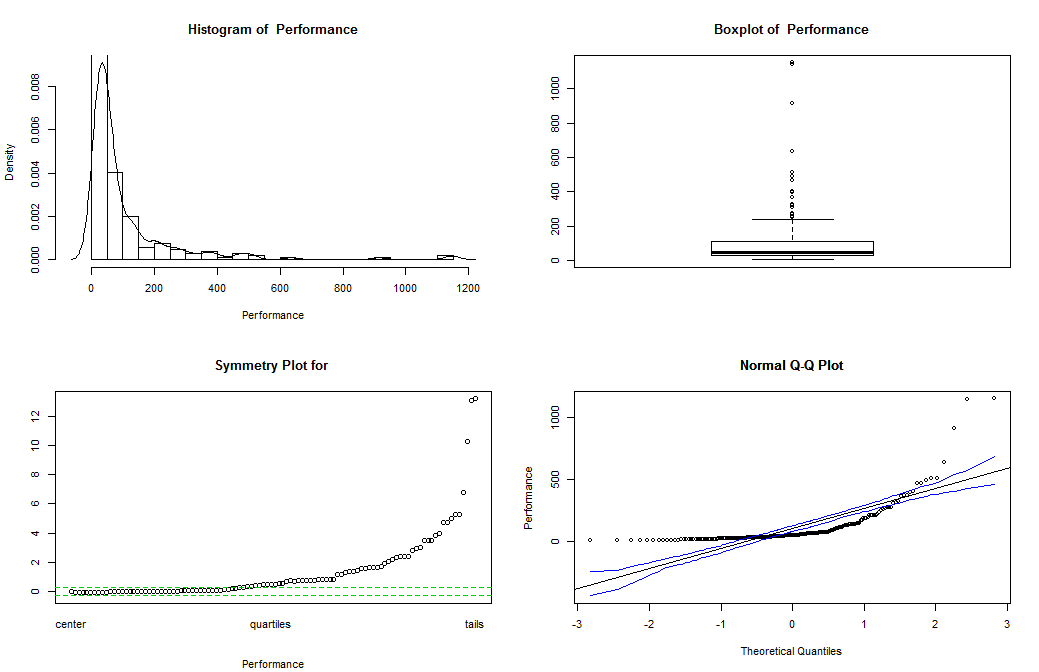
5 AMDAHL 470V/7C 29 8000 16000 32 8 16 132 132

6 AMDAHL 470V/8 26 8000 32000 64 8 32 318 290

> Performance = cpus$perf

> Statplot(Performance)

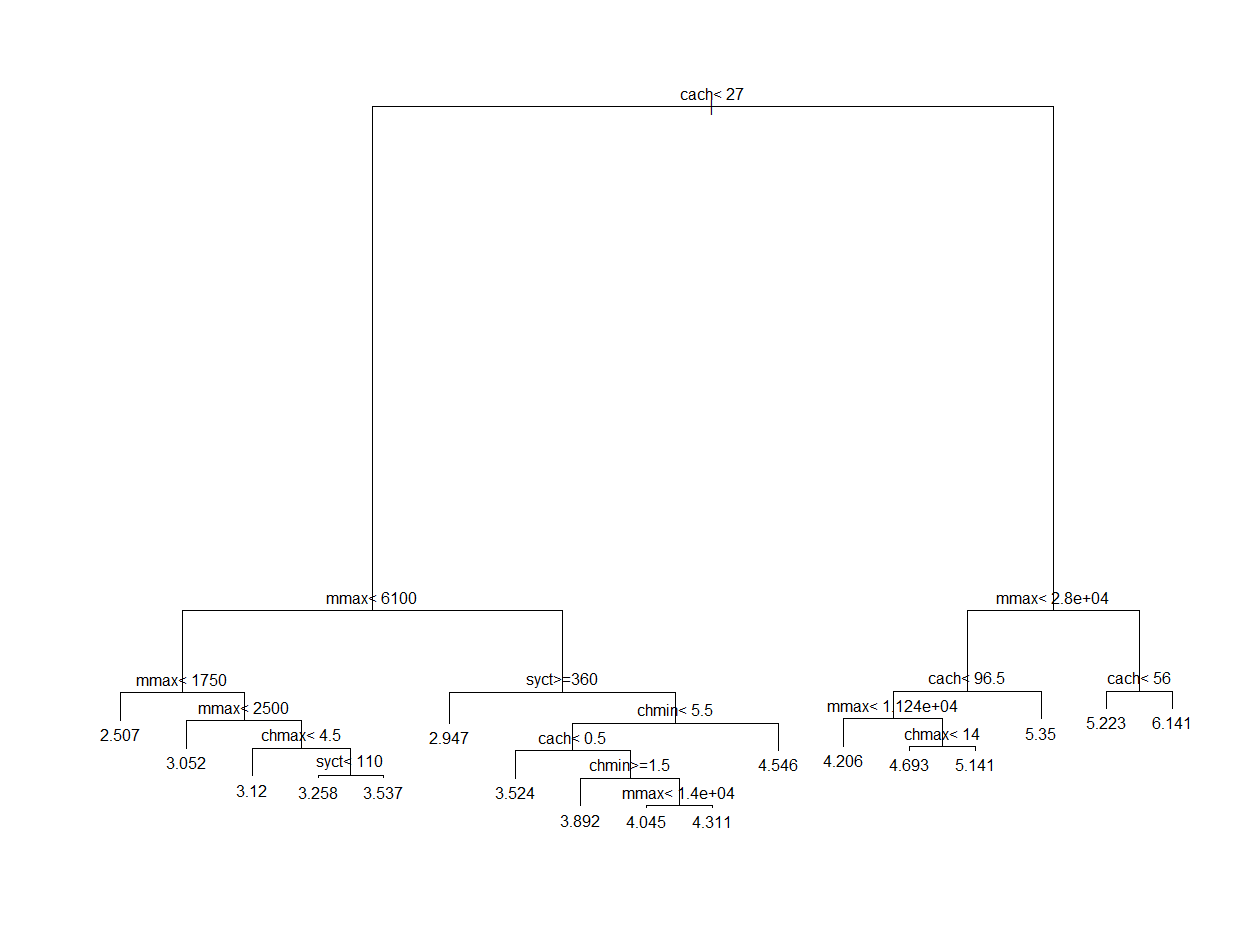
> Statplot(log(Performance))



> cpus.tree = rpart(log(Performance)~.,data=cpus[,2:7],cp=.001)

By default rpart() uses a complexity penalty of cp=.01 which will prune off more terminal nodes than we might want to consider initially. I will generally use a smaller value of cp (e.g. .001) to lead to a tree that is larger but will likely over fit the data.

> plot(cpus.tree)

> text(cpus.tree)  


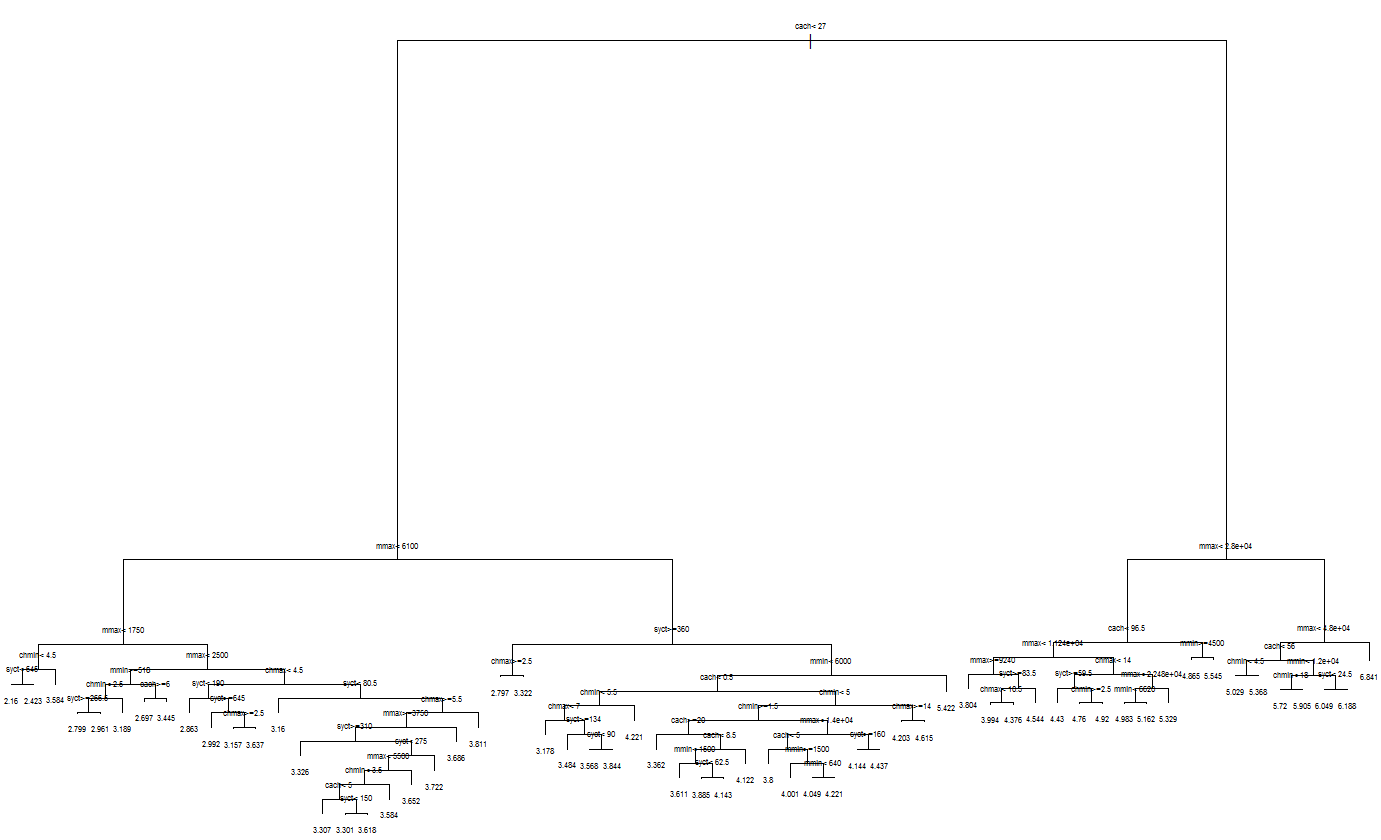
Also if you really want a large tree you can use the arguments highlighted below when calling rpart: rpart(y~.,data=mydata,minbucket=##,minsplit=##)

For example, if we fit the CART model using:

> cpus.tree = rpart(log(Performance)~.,data=cpus[,2:7],cp=.00001,minbucket=2)

> plot(cpus.tree)

> text(cpus.tree,cex=.5)



> printcp(cpus.tree)

Regression tree:

rpart(formula = log(Performance) ~ ., data = cpus[, 2:7], cp = 1e-05,

minbucket = 2)

Variables actually used in tree construction:

[1] cach chmax chmin mmax mmin syct

Root node error: 228.61/209 = 1.0938

n= 209

CP nsplit rel error xerror xstd

1 0.54933100 0 1.000000 1.00477 0.096716

2 0.08933429 1 0.450669 0.47894 0.048608

3 0.08759455 2 0.361335 0.45882 0.045428

4 0.03281415 3 0.273740 0.31828 0.032422

5 0.02692063 4 0.240926 0.31199 0.030132

6 0.01909116 5 0.214005 0.30362 0.029564

7 0.01724504 6 0.194914 0.27203 0.027633

8 0.01579669 7 0.177669 0.26658 0.027205

9 0.01216883 9 0.146076 0.25963 0.026683

10 0.01050223 10 0.133907 0.25344 0.026468

11 0.00969338 11 0.123405 0.24696 0.025870

12 0.00547627 12 0.113711 0.22135 0.026214

13 0.00523041 13 0.108235 0.21462 0.027003

14 0.00438051 14 0.103005 0.20772 0.026765

15 0.00402956 15 0.098624 0.20579 0.024780

16 0.00358978 16 0.094595 0.20069 0.024356

17 0.00288966 17 0.091005 0.20008 0.024299

18 0.00259355 18 0.088115 0.19368 0.023576 🡨 first w/ xerror below .199733 (cp = .00259)

19 0.00255414 20 0.082928 0.19329 0.023328

20 0.00238224 22 0.077820 0.19297 0.023235

21 0.00223249 23 0.075438 0.18969 0.022864

22 0.00186302 24 0.073205 0.19084 0.022878

23 0.00178910 25 0.071342 0.19608 0.023220

24 0.00172390 27 0.067764 0.19773 0.023248

25 0.00148472 28 0.066040 0.19666 0.023268

26 0.00147100 29 0.064555 0.19645 0.023351

27 0.00141305 31 0.061613 0.19609 0.023344

28 0.00129428 32 0.060200 0.19385 0.023266

29 0.00128293 33 0.058906 0.19065 0.022090

30 0.00126917 34 0.057623 0.18836 0.021997

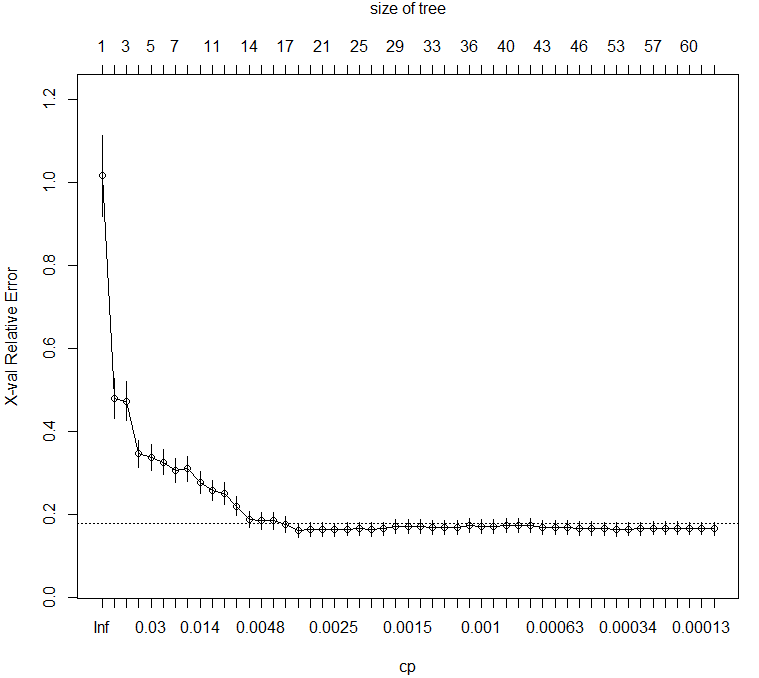
31 0.00120201 35 0.056354 0.19228 0.022338

32 0.00086124 36 0.055152 0.18265 0.020023

33 0.00081752 37 0.054291 0.17990 0.019833 🡪

34 0.00076013 39 0.052656 0.18245 0.019909

> plotcp(cpus.tree)



35 0.00075967 40 0.051895 0.18212 0.019864

36 0.00066606 41 0.051136 0.18145 0.019842

37 0.00064254 42 0.050470 0.18052 0.019771

38 0.00061811 43 0.049827 0.18052 0.019771

39 0.00060268 44 0.049209 0.18110 0.020011

40 0.00049825 45 0.048606 0.18370 0.021088

41 0.00041191 47 0.047610 0.18533 0.021070

42 0.00038741 51 0.045848 0.18541 0.021070

43 0.00035826 52 0.045460 0.18609 0.021063

44 0.00031498 54 0.044744 0.18719 0.021045

45 0.00028044 55 0.044429 0.18491 0.021067

46 0.00023848 56 0.044148 0.18483 0.021128

47 0.00021357 57 0.043910 0.18494 0.021124

48 0.00017221 58 0.043696 0.18477 0.021140

49 0.00015990 59 0.043524 0.18517 0.021128

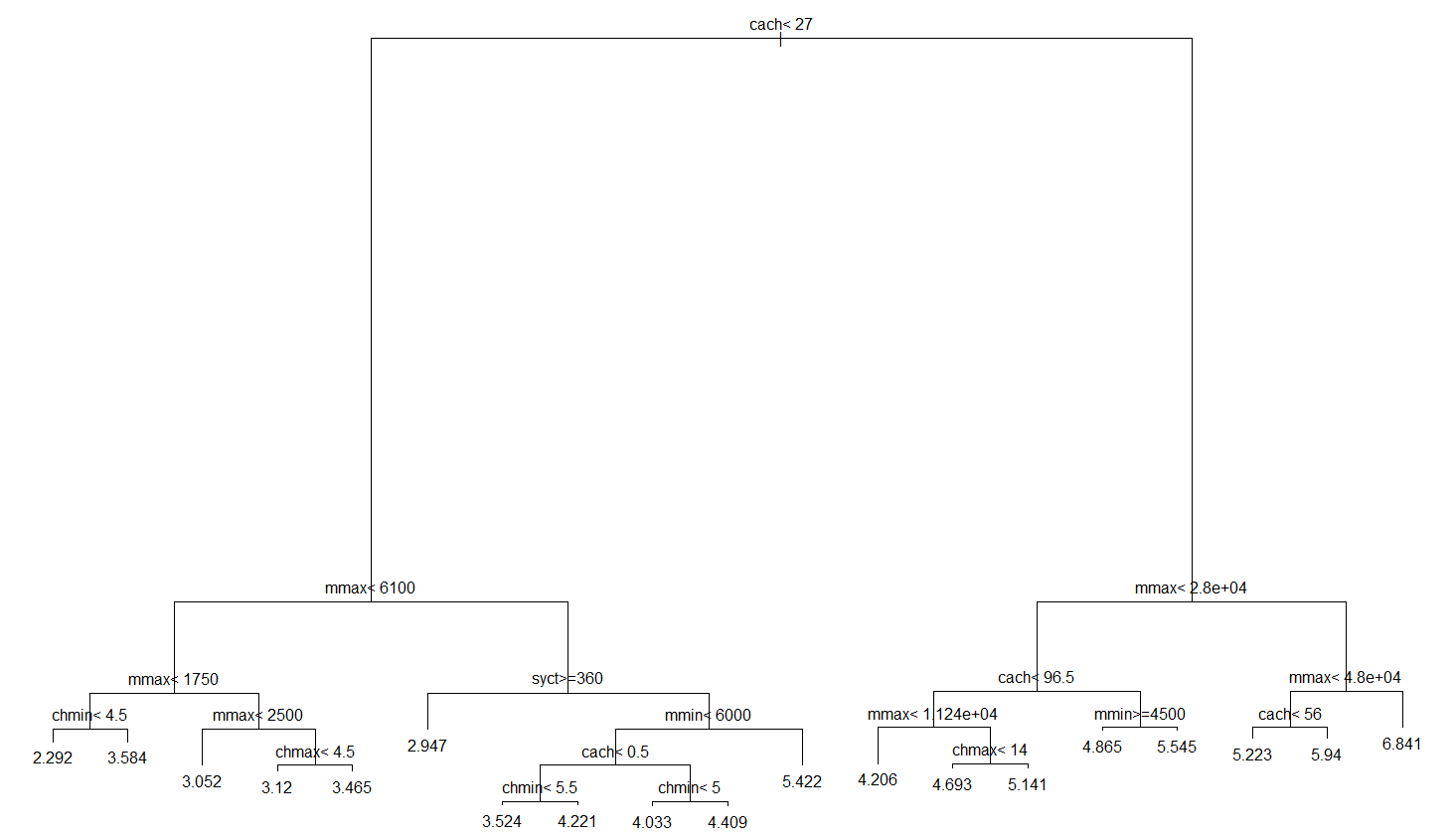
50 0.00011328 60 0.043364 0.18496 0.021139

51 0.00001000 61 0.043251 0.18501 0.021154

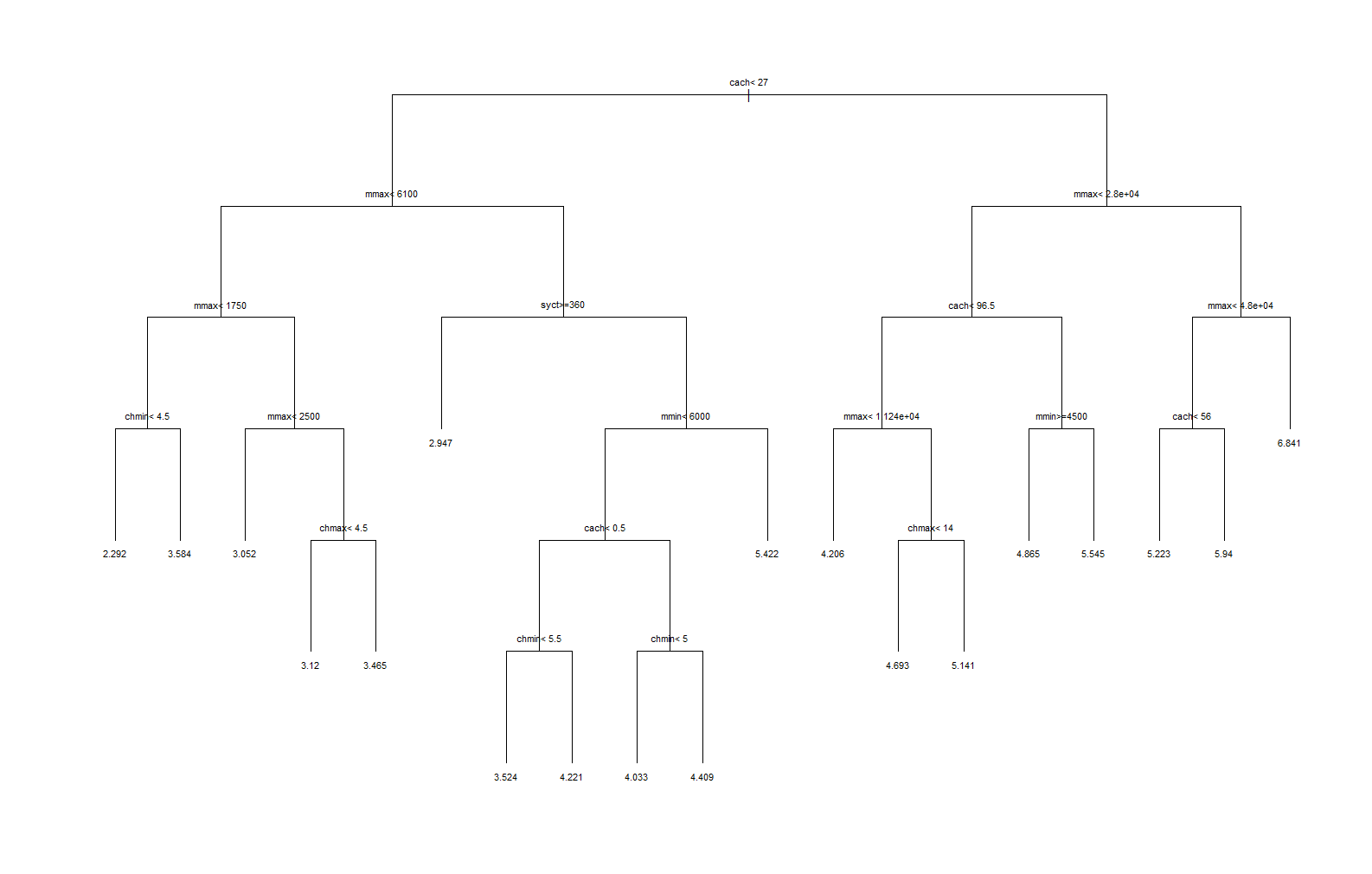
> cpus.tree = rpart(log(Performance)~.,data=cpus[,2:7],cp=.00259355,minbucket=2)

> plot(cpus.tree)

> text(cpus.tree)

> plot(cpus.tree,uniform=T)

> text(cpus.tree,cex=.7)



> printcp(cpus.tree)

Regression tree:

rpart(formula = log(Performance) ~ ., data = cpus[, 2:7], cp = 0.00259355,

minbucket = 2)

Variables actually used in tree construction:

[1] cach chmax chmin mmax mmin syct

Root node error: 228.61/209 = 1.0938

n= 209

CP nsplit rel error xerror xstd

1 0.5493310 0 1.000000 1.00863 0.097205

2 0.0893343 1 0.450669 0.47176 0.048632

3 0.0875946 2 0.361335 0.45115 0.047408

4 0.0328142 3 0.273740 0.32302 0.031693

5 0.0269206 4 0.240926 0.31939 0.030301

6 0.0190912 5 0.214005 0.27304 0.027673

7 0.0172450 6 0.194914 0.27966 0.027414

8 0.0157967 7 0.177669 0.26152 0.026094

9 0.0121688 9 0.146076 0.26614 0.026499

10 0.0105022 10 0.133907 0.24905 0.025531

11 0.0096934 11 0.123405 0.24883 0.026049

12 0.0054763 12 0.113711 0.21767 0.024199 🡨 within 1 xstd of minimum xerror

13 0.0052304 13 0.108235 0.22393 0.024543

14 0.0043805 14 0.103005 0.21819 0.023931

15 0.0040296 15 0.098624 0.21689 0.024622

16 0.0035898 16 0.094595 0.21665 0.024544 🡪 .21665 + .024544 = .241194

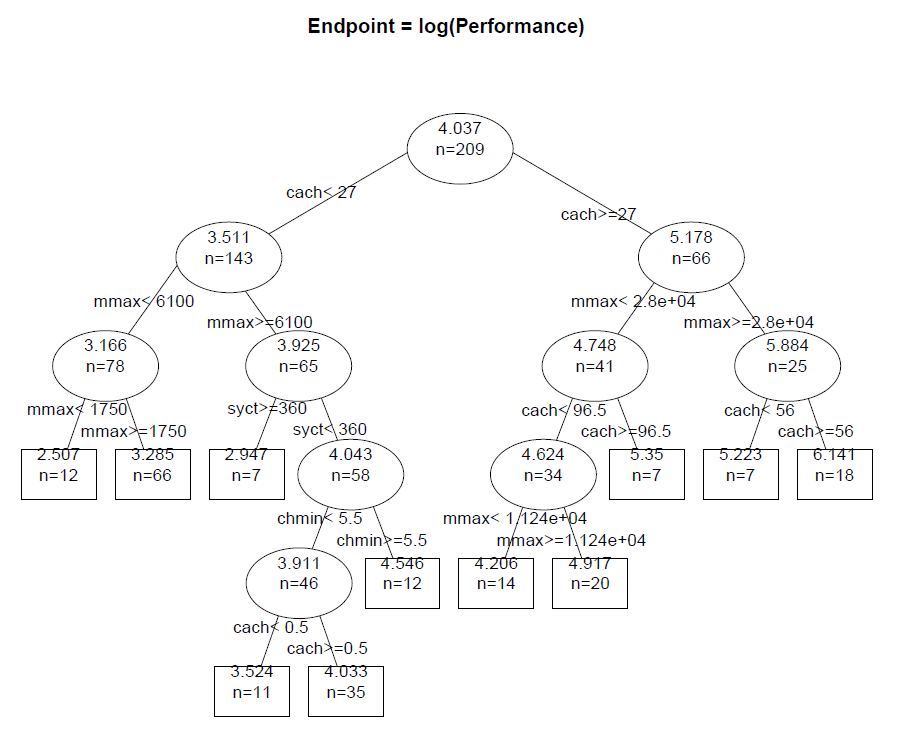
17 0.0028897 17 0.091005 0.21802 0.024536

18 0.0025936 18 0.088115 0.22252 0.024837

Prune the tree back to a 10 split, 11 terminal node tree using cp = .0055.

> cpus.tree = rpart(log(Performance)~.,data=cpus[,2:7],cp=.0055)

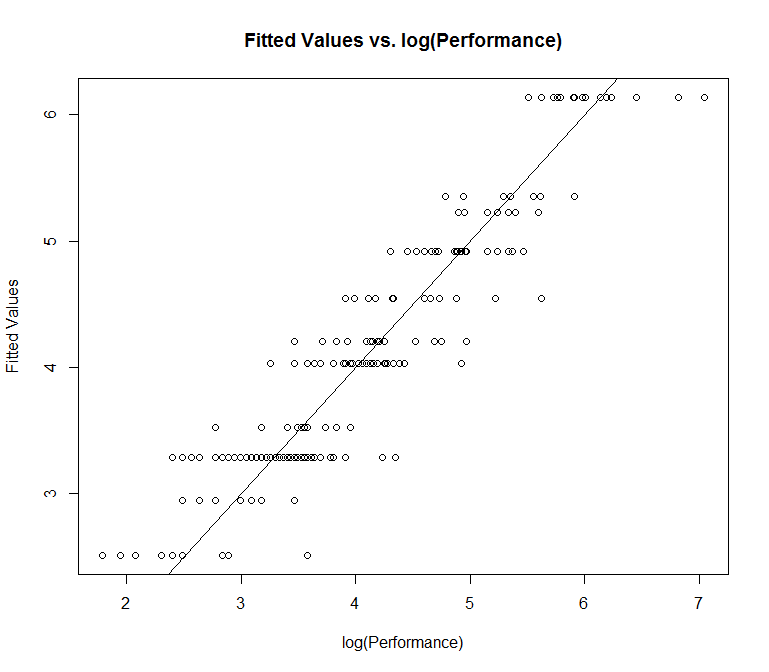
> post(cpus.tree,digits=3)



We can plot and the residuals () vs. the fitted values ().

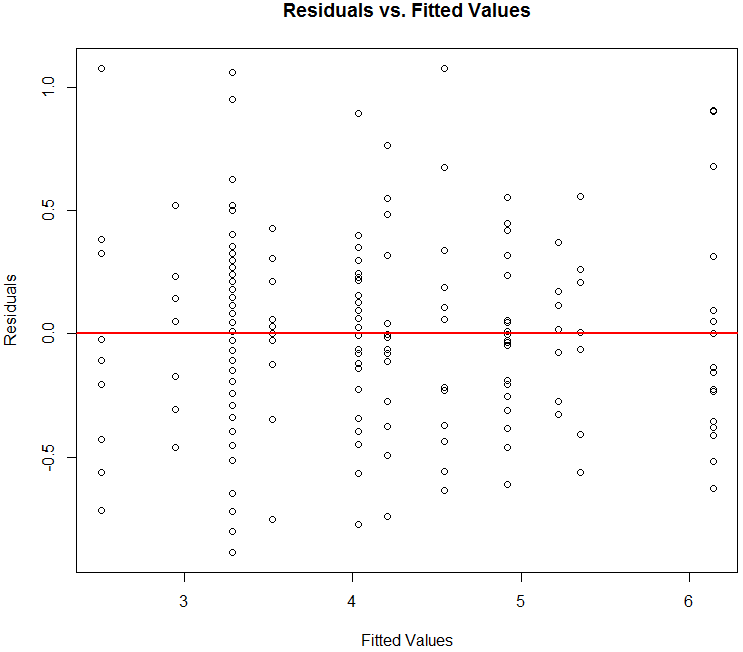
> plot(log(Performance),predict(cpus.tree),ylab="Fitted Values",  
main="Fitted Values vs. log(Performance)")

> abline(0,1)

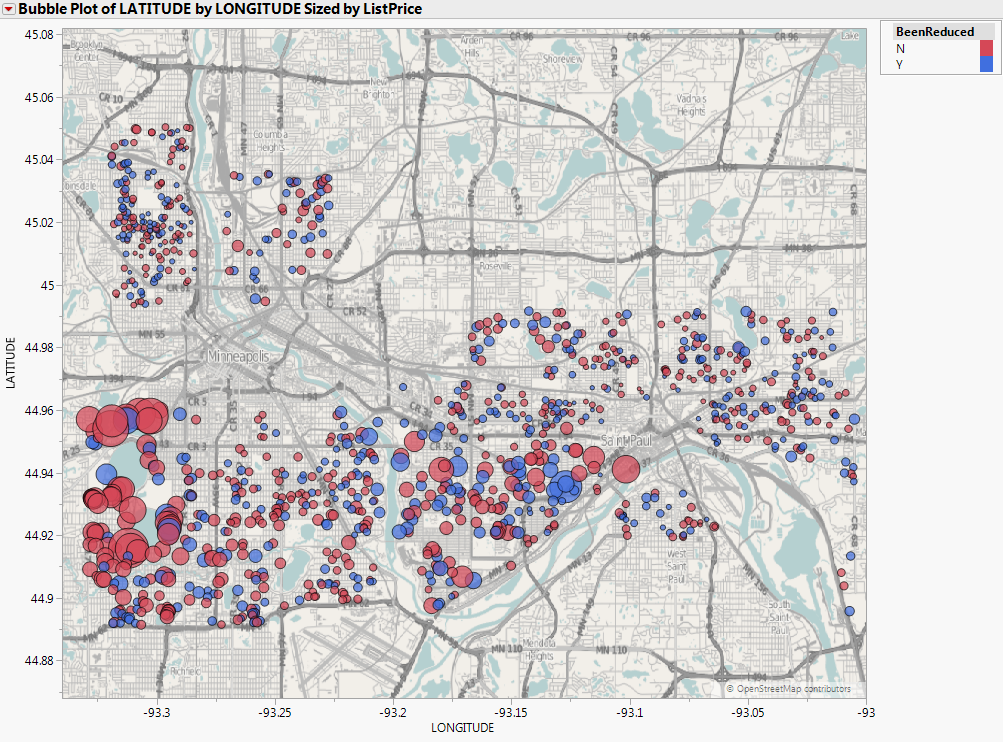


> plot(predict(cpus.tree),resid(cpus.tree),xlab="Fitted Values",ylab="Residuals",main="Residuals vs. Fitted Values")

> abline(h=0,col="red",lwd=2)



**Example 10.3 – Twin Cities Home Prices**



The Twin Cities home prices were used in Assignment 1 and we will now consider fitted a regression tree to these data. By examining the plot above can you see an advantage a tree model might have for these data?

Read in the data and remove the missing values

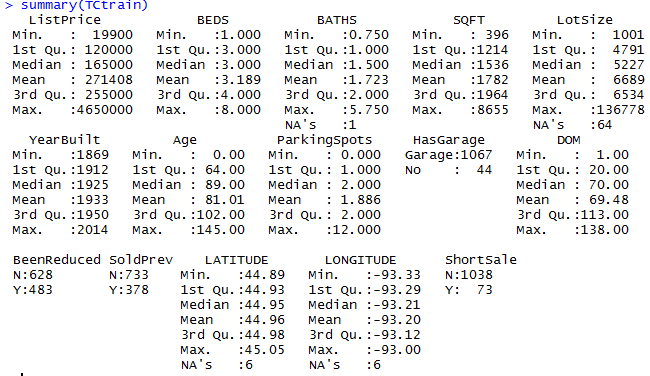
> TCtrain = read.table(file.choose(),header=T,sep=",")  
> names(TCtrain)

[1] "ListPrice" "BEDS" "BATHS" "SQFT" "LotSize"

[6] "YearBuilt" "Age" "ParkingSpots" "HasGarage" "DOM"

[11] "BeenReduced" "SoldPrev" "LATITUDE" "LONGITUDE" "ShortSale"

> summary(TCtrain)



> dim(TCtrain)

[1] 1111 15

> TCtrain = na.omit(TCtrain)

> dim(TCtrain)

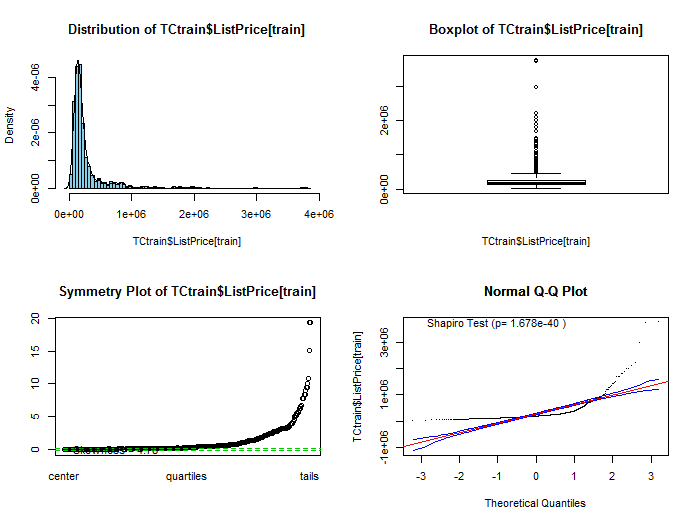
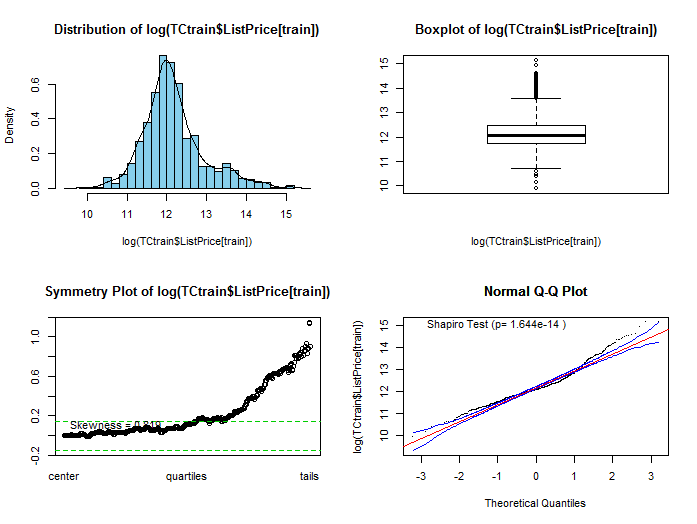
[1] 1040 15  
  
Create training and validation sets from the TCtrain observations.

> train = sample(1:1040,size=floor(.7\*1040),replace=F)

> valid = (-train)

> Statplot(TCtrain$ListPrice[train])

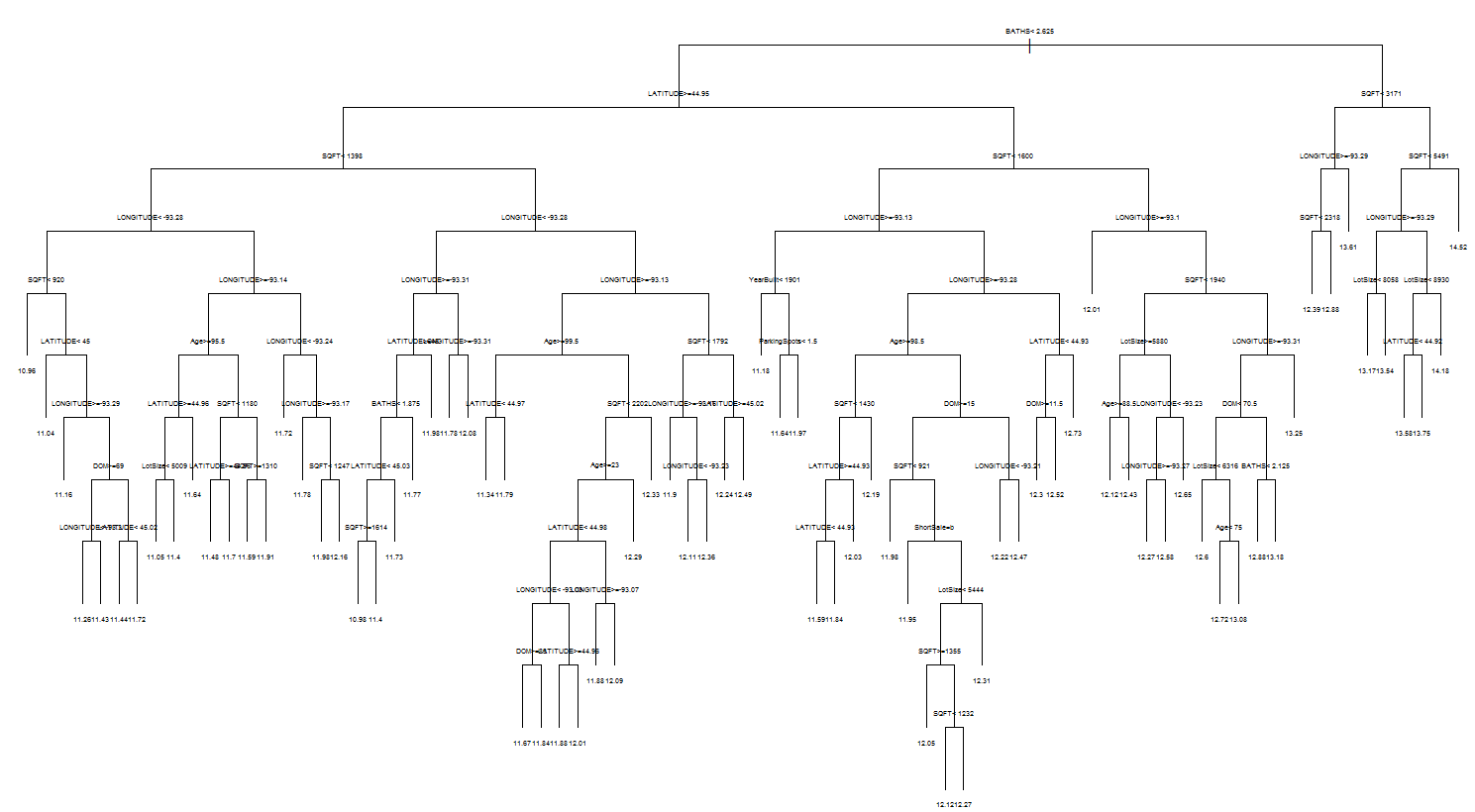
> Statplot(log(TCtrain$ListPrice[train]))

> home.rpart = rpart(log(ListPrice)~.,data=TCtrain[train,],cp=.0001,minbucket=5)

> plot(home.rpart,uniform=T)

> text(home.rpart,cex=.45,digits=4)



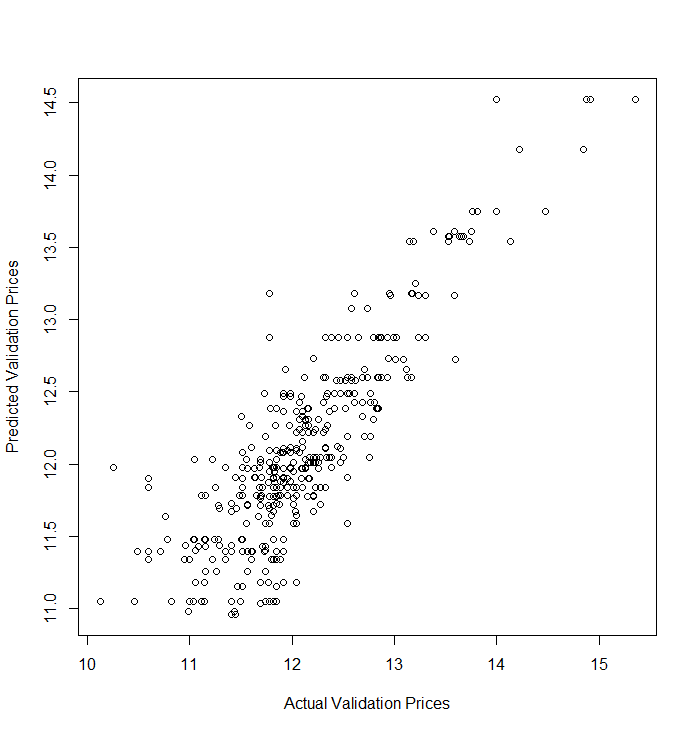
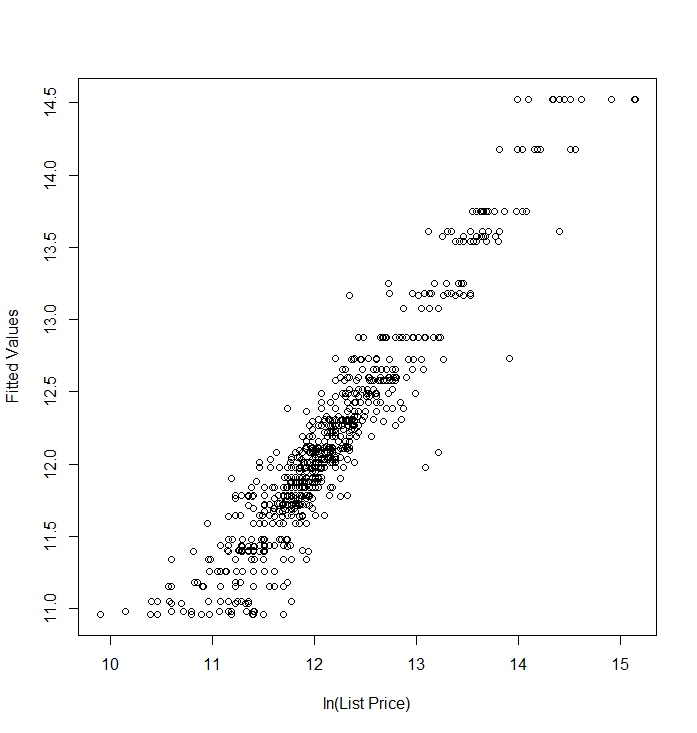
> plot(log(TCtrain$ListPrice)[train],predict(home.rpart))

> abline(0,1,col="red",lwd=2)

> cor(log(TCtrain$ListPrice)[train],predict(home.rpart))^2

[1] 0.9000216

Fitted vs. Actual for Training Cases Predicted vs. Actual for Validation Cases



> ypred = predict(home.rpart,newdata=TCtrain[valid,])

> ypred = exp(ypred)

> yact = TCtrain$ListPrice[valid]  
  
> PredAcc(yact,ypred)

RMSEP

===============

196184

MAE

===============

73772.14

MAPE

===============

32.6051

RMSEP MAE MAPE

1 196184 73772.14 32.6051

> cor(yact,ypred)^2

[1] 0.8006805

Our current model is probably overfitting the training data. We should consider the results from the internal cross-validation rpart to choose a cost-complexity value to prune our tree.

> printcp(home.rpart)

Regression tree:

rpart(formula = log(ListPrice) ~ ., data = TCtrain[train, ],

cp = 1e-04, minbucket = 5)

Variables actually used in tree construction:

[1] Age BATHS DOM LATITUDE LONGITUDE LotSize

[7] ParkingSpots ShortSale SQFT YearBuilt

Root node error: 431.36/728 = 0.59253

n= 728

CP nsplit rel error xerror xstd

1 0.45604708 0 1.000000 1.00446 0.067585

2 0.13009321 1 0.543953 0.59727 0.041906

3 0.04486790 2 0.413860 0.48111 0.037982

4 0.03462398 3 0.368992 0.44125 0.032194

5 0.02936454 4 0.334368 0.41252 0.030835

6 0.01585186 5 0.305003 0.35050 0.027437

7 0.01184748 6 0.289151 0.34324 0.026100

8 0.01162340 8 0.265456 0.33651 0.025815

9 0.01156554 9 0.253833 0.33651 0.025815

10 0.01072403 10 0.242268 0.33499 0.026343

11 0.01013585 11 0.231544 0.32874 0.026284

12 0.00946351 12 0.221408 0.31725 0.026034

13 0.00707933 13 0.211944 0.30946 0.026050

14 0.00675538 14 0.204865 0.29997 0.025838

15 0.00589732 15 0.198109 0.29462 0.025727

16 0.00586189 16 0.192212 0.29318 0.025779

17 0.00540258 17 0.186350 0.29047 0.025719

18 0.00466254 18 0.180948 0.28407 0.024877 🡨 within 1 xstd of xerror minimum (cp = .00466)

19 0.00347791 19 0.176285 0.28270 0.024701

20 0.00333295 20 0.172807 0.27163 0.023245

21 0.00331767 21 0.169474 0.27175 0.023259

22 0.00315954 23 0.162839 0.27194 0.022728

23 0.00276518 24 0.159679 0.27429 0.023385

24 0.00273884 25 0.156914 0.26958 0.023239

25 0.00256867 26 0.154175 0.26984 0.023289

26 0.00255701 27 0.151607 0.26826 0.023100

27 0.00218348 28 0.149050 0.26465 0.022979

28 0.00210663 29 0.146866 0.26432 0.022853

29 0.00194970 31 0.142653 0.26179 0.022008

30 0.00166057 32 0.140703 0.26255 0.021811

31 0.00164841 33 0.139043 0.26161 0.021274 🡪 .26151 + .021274 = .2872884

32 0.00163456 35 0.135746 0.26161 0.021274

33 0.00163161 36 0.134111 0.26228 0.021281

34 0.00150829 37 0.132480 0.26197 0.021275

35 0.00146389 38 0.130971 0.26501 0.021485

36 0.00129502 40 0.128044 0.26154 0.020830

37 0.00129193 41 0.126749 0.26640 0.021269

38 0.00125129 42 0.125457 0.26741 0.021318

39 0.00124502 43 0.124205 0.26749 0.021309

40 0.00120619 44 0.122960 0.26896 0.021324

41 0.00114328 45 0.121754 0.26973 0.021336

42 0.00107603 46 0.120611 0.27068 0.021351

43 0.00106330 47 0.119535 0.27126 0.021427

44 0.00104862 48 0.118472 0.27207 0.021452

45 0.00096608 51 0.115326 0.27286 0.021477

46 0.00095289 52 0.114360 0.27251 0.021485

47 0.00095150 54 0.112454 0.27251 0.021485

48 0.00088278 56 0.110551 0.27231 0.021501

49 0.00082785 57 0.109668 0.27343 0.021535

50 0.00077227 58 0.108840 0.27475 0.021708

51 0.00068758 59 0.108068 0.27597 0.021725

52 0.00061466 60 0.107380 0.27527 0.021709

53 0.00057843 61 0.106766 0.27550 0.021738

54 0.00057114 62 0.106187 0.27567 0.021767

55 0.00053599 64 0.105045 0.27572 0.021760

56 0.00053447 65 0.104509 0.27705 0.021757

57 0.00049694 66 0.103975 0.27827 0.021864

58 0.00044861 67 0.103478 0.27846 0.021863

59 0.00043784 68 0.103029 0.27873 0.021861

60 0.00043258 69 0.102591 0.27838 0.021864

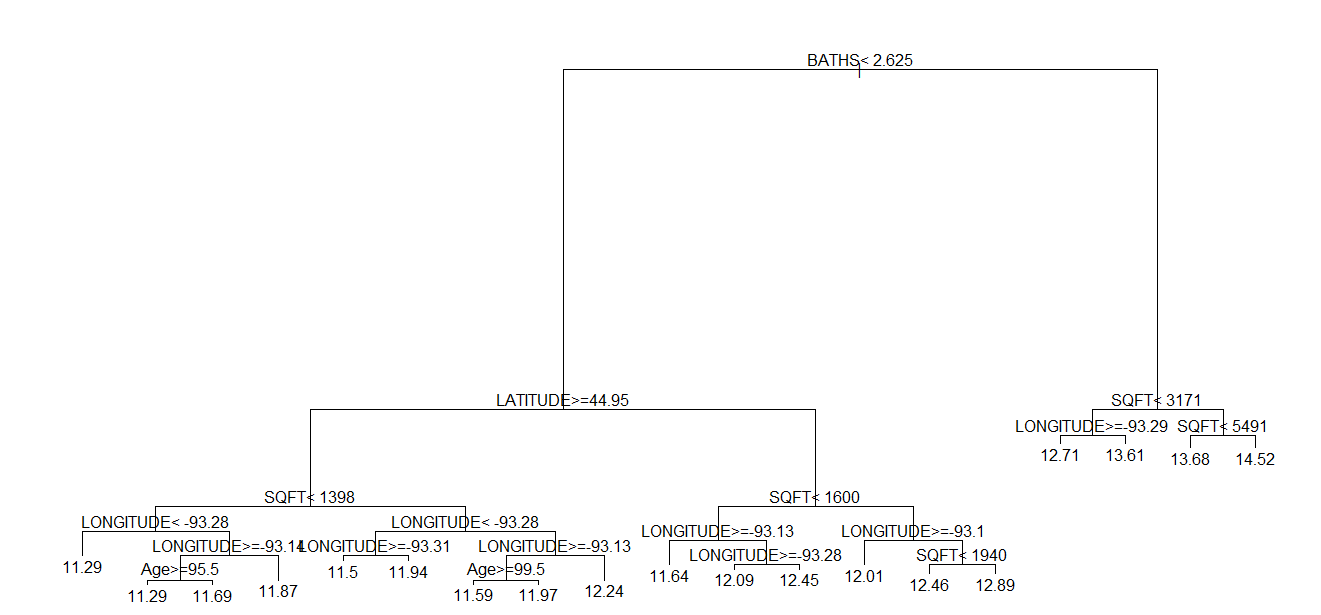
61 0.00038692 70 0.102159 0.27875 0.021848

62 0.00035964 71 0.101772 0.27820 0.021907

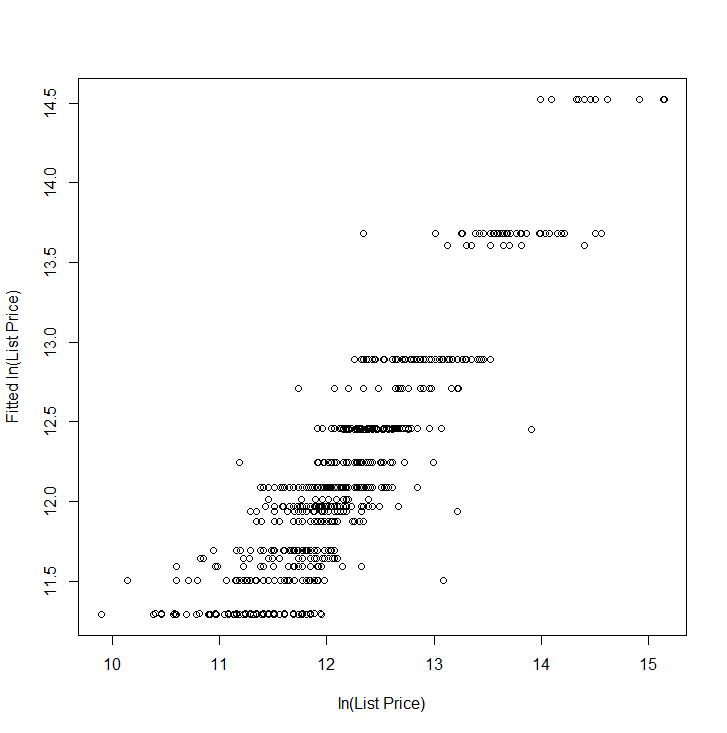
> home.rpart2 = rpart(log(ListPrice)~.,data=TCtrain[train,],cp=0.00466254,minbucket=5)

> plot(home.rpart2)

> text(home.rpart2)



> plot(log(TCtrain$ListPrice)[train],predict(home.rpart2),  
 xlab="ln(List Price)",ylab="Fitted ln(List Price)")



> ypred = predict(home.rpart2,newdata=TCtrain[valid,])

> ypred = exp(ypred)

> yact = TCtrain$ListPrice[valid]

> PredAcc(yact,ypred)

RMSEP

===============

213455.5

MAE

===============

80839.5

MAPE

===============

34.98121

RMSEP MAE MAPE

1 213455.5 80839.5 34.98121

> home.rpart3 = rpart(log(ListPrice)~.,data=TCtrain[train,],cp=0.0003,minbucket=5)

> ypred = predict(home.rpart3,newdata=TCtrain[valid,])

> ypred = exp(ypred)

> PredAcc(yact,ypred)

RMSEP

===============

196031.5

MAE

===============

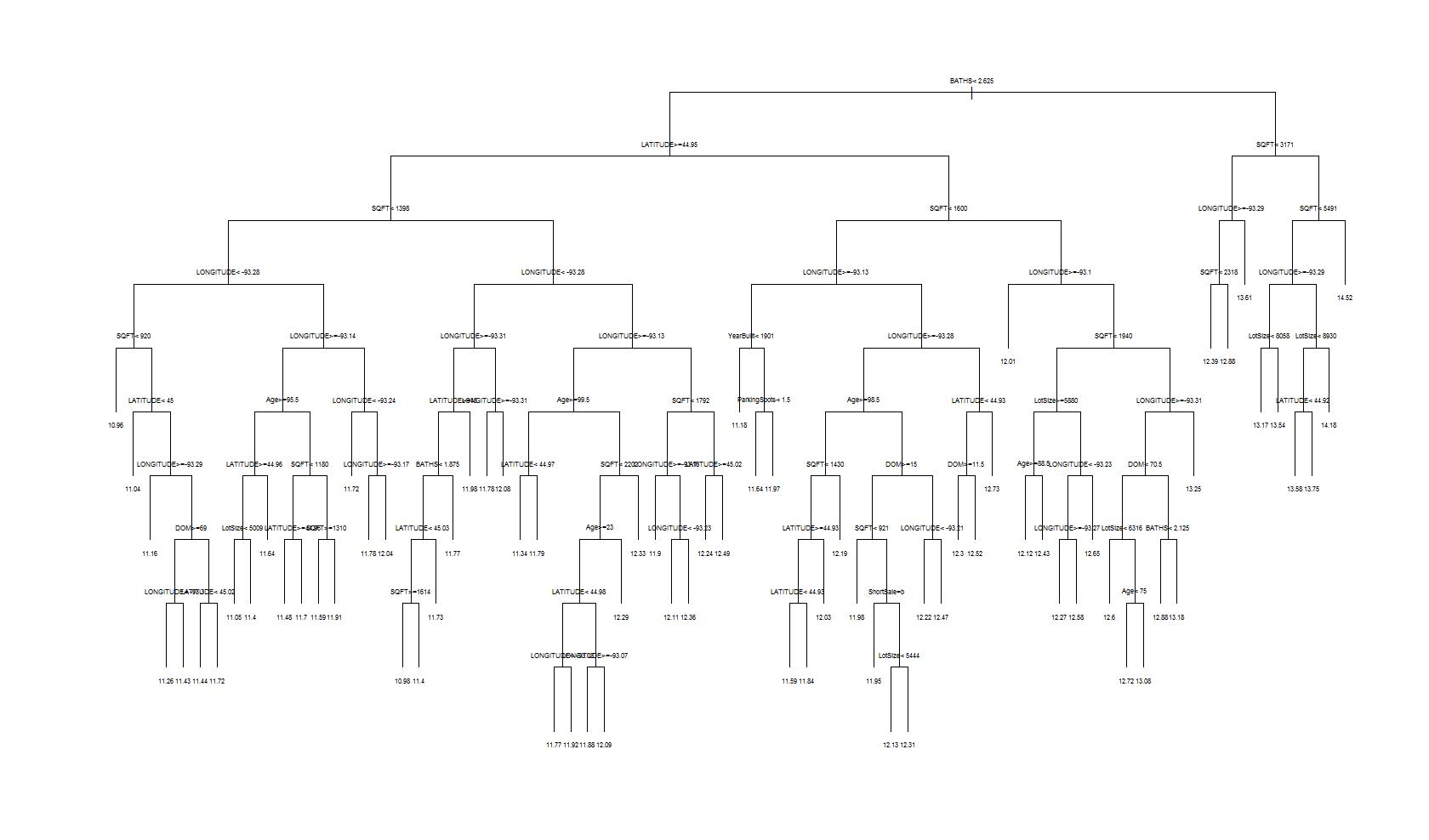
73347.87

MAPE

===============

32.48529

RMSEP MAE MAPE  
 196031.5 73347.87 32.3625



> ypred = predict(home.rpart3,newdata=TChome.test)

> ypred = exp(ypred)

> PredAcc(yact,ypred)

RMSEP

===============

333245.1

MAE

===============

235095.2

MAPE

===============

38.92

**Split-Sample Monte Carlo Cross-Validation**

rpart.sscv = function(fit,data,p=.667,B=100,  
cp=fit$control$cp,minbucket=fit$control$minbucket) {

MSE = rep(0,B)

MAE = rep(0,B)

MAPE = rep(0,B)

y = fit$y

n = nrow(data)

ss <- floor(n\*p)

for (i in 1:B) {

sam = sample(1:n,ss,replace=F)

fit2 = rpart(formula(fit),data=data[sam,],cp=cp,minbucket=minbucket)

ynew = predict(fit2,newdata=data[-sam,])

MSE[i] = mean((y[-sam]-ynew)^2)

MAE[i] = mean(abs(y[-sam]-ynew))

MAPE[i] = mean((abs(y[-sam]-ynew)/y[-sam]))

}

RMSEP = sqrt(mean(MSE))

MAEP = mean(MAE)

MAPEP = mean(MAPE)

cat("RMSEP\n")

cat("===============\n")

cat(RMSEP,"\n\n")

cat("MAEP\n")

cat("===============\n")

cat(MAEP,"\n\n")

cat("MAPEP\n")

cat("===============\n")

cat(MAPEP,"\n\n")

temp = data.frame(MSEP=MSE,MAEP=MAE,MAPEP=MAPE)

return(temp)

}

If a response transformation is used you would need to replace the code highlighted above. If is the response as is the case for this example we would use the code below instead, which I did the function rpart.sslog used in the examples below.  
  
 ynew = exp(ynew)

yact = exp(y[-sam])

MSE[i] = mean((yact-ynew)^2)

MAE[i] = mean(abs(yact-ynew))

MAPE[i]= mean((abs(yact-ynew)/yact))

Below we are performing split-sample Monte Carlo cross-validation for CART models fit to the Twin Cities home price data using the log list price as the response. The cost-complexity parameter (cp) is being varied from a small value to larger values. We could also experiment with different settings of minbucket as well.

> home.rpart = rpart(log(ListPrice)~.,data=TCtrain,cp=.00001,minbucket=5)

> results = rpart.sslog(home.rpart,TCtrain,B=1000)

RMSEP

===============

203818.5

MAEP

===============

82454.1

MAPEP

===============

0.327224

> home.rpart = rpart(log(ListPrice)~.,data=TCtrain,cp=.0001,minbucket=5)

> results = rpart.sslog(home.rpart,TCtrain,B=1000)

RMSEP

===============

202191

MAEP

===============

81535.32

MAPEP

===============

0.3253579

> home.rpart = rpart(log(ListPrice)~.,data=TCtrain,cp=.001,minbucket=5)

> results = rpart.sslog(home.rpart,TCtrain,B=1000)

RMSEP

===============

202559.3

MAEP

===============

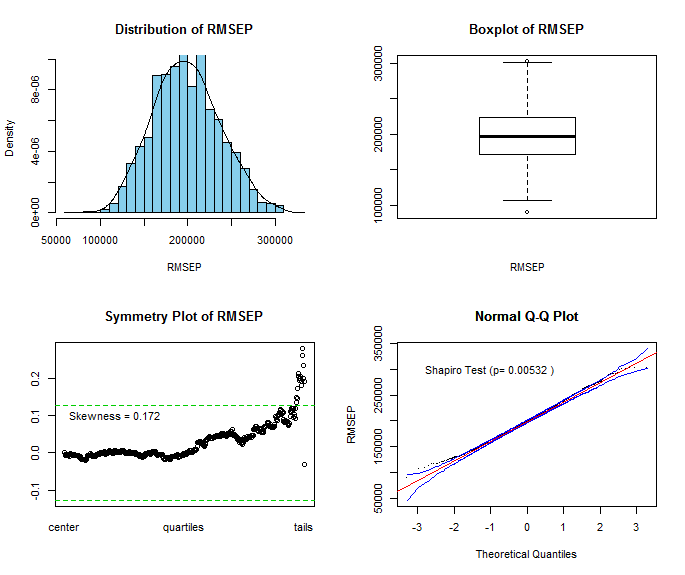
81481.49

MAPEP

===============

0.324541

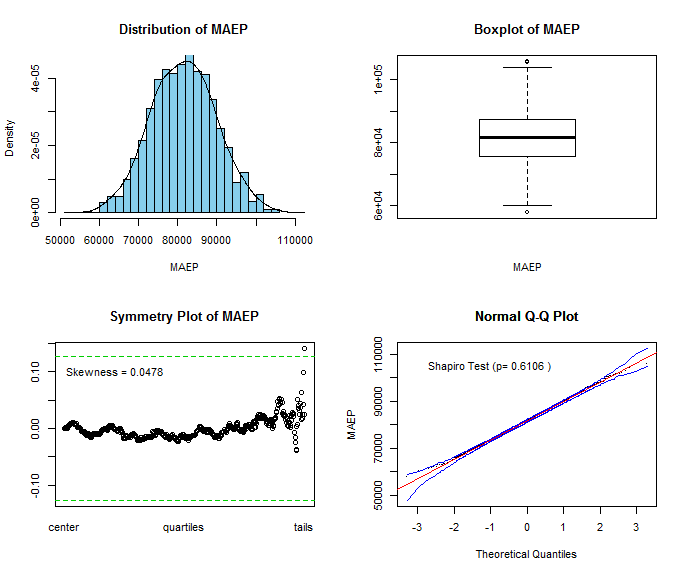
It is interesting to consider the variability in the MSEP, MAEP, and MAPEP for the B = 1000 split-sample fits.



> RMSEP = sqrt(results$MSEP)

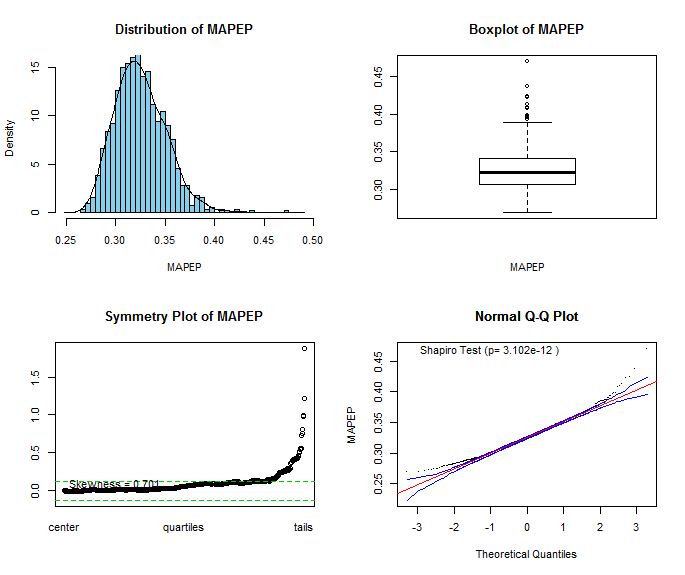
> Statplot(RMSEP)

> MAEP = results$MAEP

> Statplot(MAEP)  


> MAPEP = results$MAPEP

> Statplot(MAPEP)



**10.3 - Bagging for Regression Trees**

Suppose we are interested in predicting a numeric response variable

and

For example, , might come from an OLS model selected using Mallow’s with all potential predictors or from a CART model using ***x*** with a complexity parameter cp = .005. Letting denote , where the expectation is with respect to the distribution underlying the training sample (since, viewed as a random variable, is a function of training sample, which can be viewed as a high-dimensional random variable) and not (which is considered fixed), we have that:

Thus in theory, if our prediction could be based on instead of then we would have a smaller mean squared error for prediction (and a smaller RMSEP as well). How can we approximate ? We could take a large number of samples of size *n* from the population and fit the specified model to each. Then average across these samples to get an average model , more specifically we could get the average prediction from the different models for a given set of predictors *.* Of course, this is silly as we only take one sample of size *n* in general when conducting any study. However, we can approximate this via the bootstrap. The bootstrap remember involves taking *B* random samples of size *n* drawn with replacement from our original sample . For each bootstrap sample, *b*, we will obtain an estimated model and average those to obtain a final estimate of , i.e.

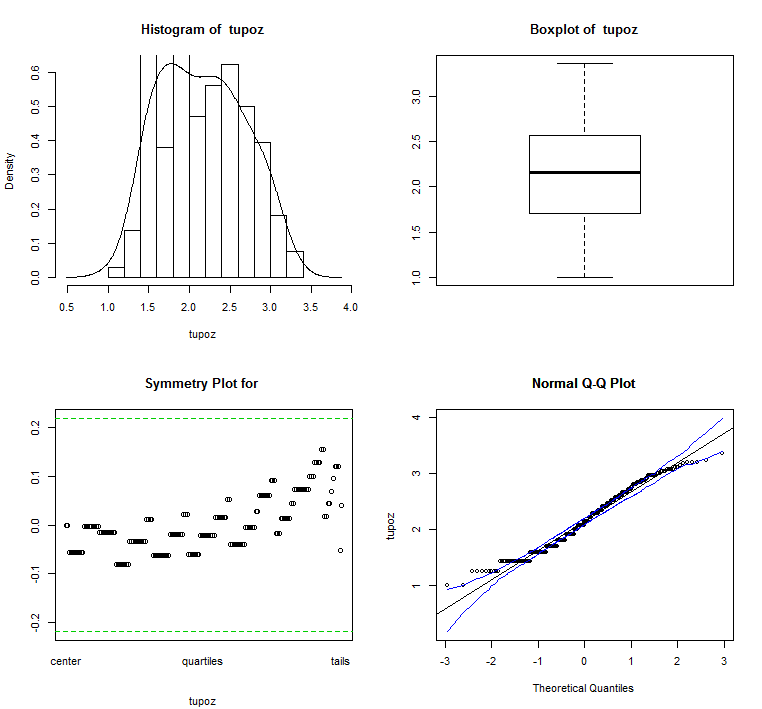
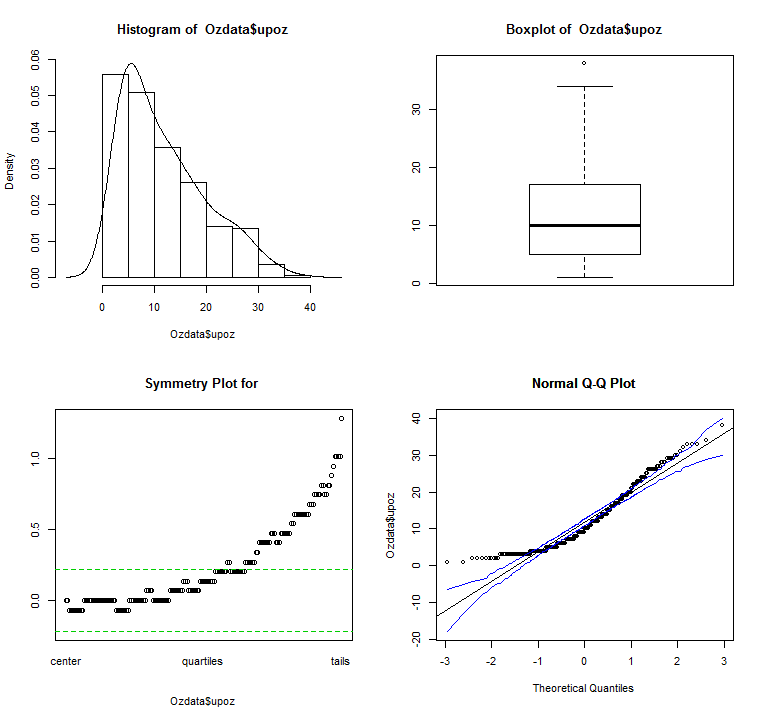
This estimator for should in theory be better than the one obtained from fitting one model to the training data. This process of averaging the predicted values from a given is called ***bagging***. Bagging works best when the fitted models vary substantially from one bootstrap sample to the next. Modeling schemes that are complicated and involve the effective estimation of a large number parameters will benefit from bagging most. Projection Pursuit, neural networks, CART, and MARS are examples of algorithms where this is likely to be the case. Bagging (or model averaging) is an example of what is referred to as an ***ensemble model*** in statistical learning, where results from different models are combined, in the case of bagging through averaging. We will examine other types of ensemble models later in this section.

**Example 10.4: Bagging with the LA Upper Ozone Concentration**

> Ozdata = read.table(file.choose(),header=T,sep=”,”) 🡨 read in the file **Ozone.csv**  
> Statplot(Ozdata$upoz)

> tupoz = Ozdata$upoz^.333

> Statplot(tupoz)



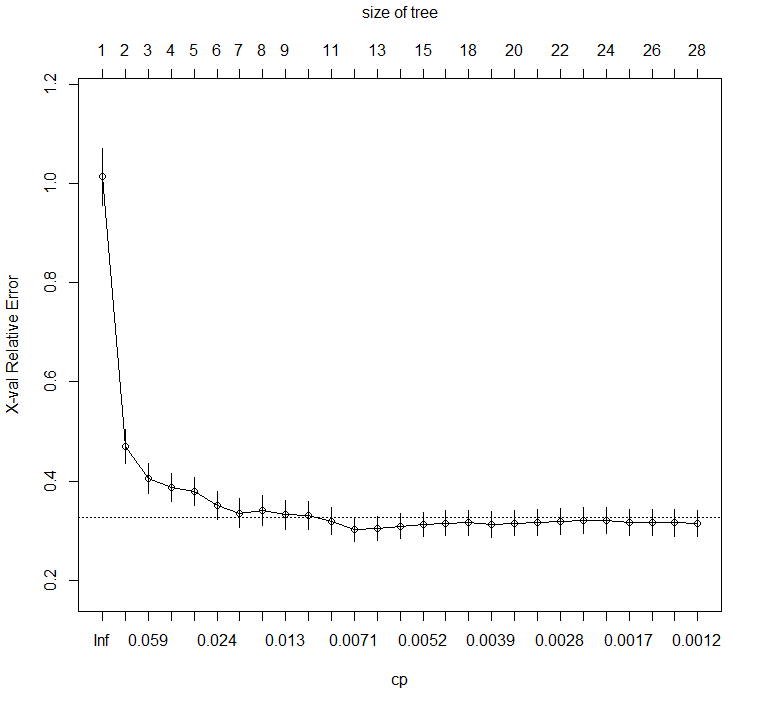
> names(Ozdata)

[1] "upoz" "day" "v500" "wind" "hum" "safb" "inbh" "dagg" "inbt" "vis"   
  
> Ozdata2 = data.frame(tupoz,Ozdata[,-1]) 🡨 form new data frame with transformed response.

> names(Ozdata2)

[1] "tupoz" "day" "v500" "wind" "hum" "safb" "inbh" "dagg" "inbt" "vis"  
> oz.rpart = rpart(tupoz~.,data=Ozdata2,cp=.001)

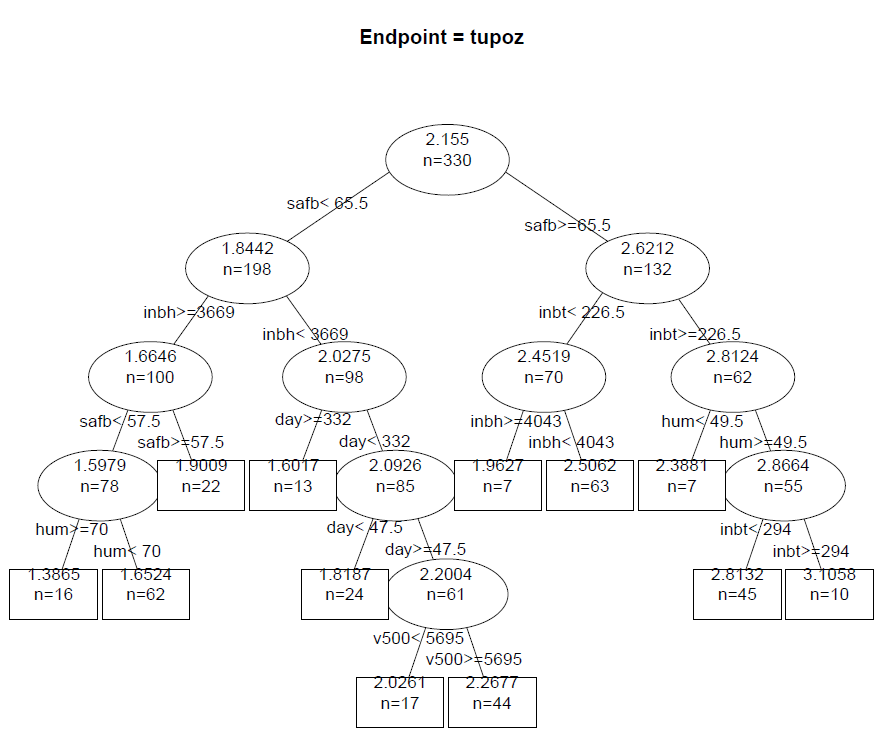
> plotcp(oz.rpart)



The complexity parameter cp =.007 looks like a good choice.

> oz.rpart2 = rpart(tupoz~.,data=Ozdata2,cp=.007)

> post(oz.rpart2)



> badRMSE = sqrt(mean(resid(oz.rpart2)^2))

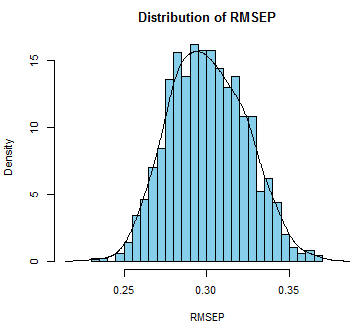
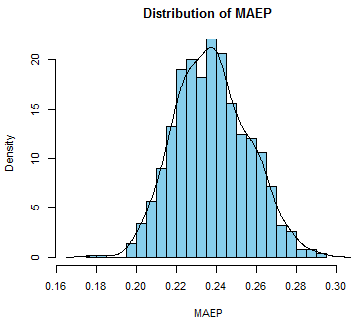
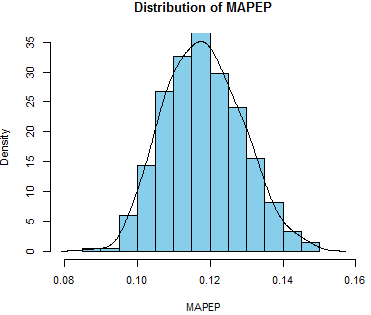
> badRMSE

[1] 0.2336983

Using the split-sample Monte Carlo function above we obtain the following. The response was not back-transformed to the original scale at this point.

> results = rpart.sscv(oz.rpart2,Ozdata2,B=1000)

RMSEP

===============

0.301543

MAE

===============

0.2377987

MAPE

===============

0.1181766

We now consider using bagging to improve the predictions from CART applied only to one model fit to our training sample. The package ipred from CRAN contains a function bagging which will perform bagging on regression trees obtain using CART.

Below we use bagging to fit 10-bootstrap regression trees. The predictions for the cases not appearing in each of the 10 bootstrap samples are used to estimate the RMSEP in the usual way.

> oz.bag10 = bagging(tupoz~.,data=Ozdata2,cp=.007,nbagg=10,coob=T)

> print(oz.bag10)

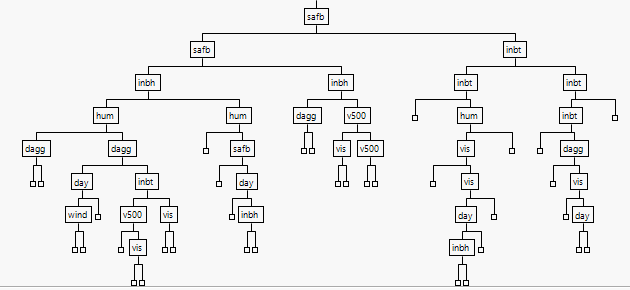
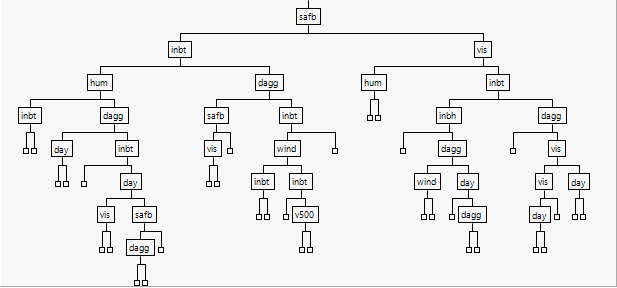
**Bagging regression trees with 10 bootstrap replications**

Call: bagging.data.frame(formula = tupoz ~ ., data = Ozdata2, cp = 0.007,

nbagg = 10, coob = T)

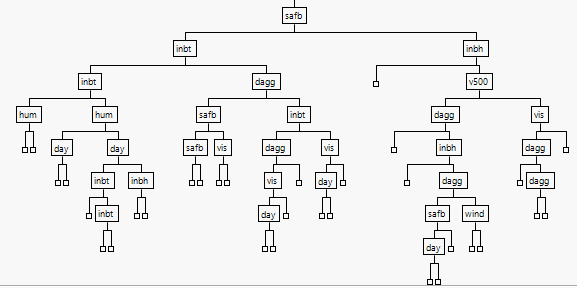
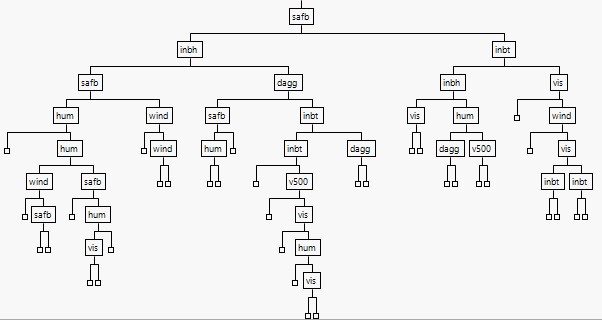
Out-of-bag estimate of root mean squared error: 0.2751 🡨 RMSEP estimate  
  
Recall that the final estimated bagged estimate () is the average of the predictions from the 10-bootstrap regression trees, i.e.

Graphically,

+

+

+…+

10

As we can see the regression trees fit to the 10 bootstrap samples differ to some degree.

Increasing the number of bootstrap samples used () should improve the performance of the bagged estimate of the regression tree.

> oz.bag25 = bagging(tupoz~.,data=Ozdata2,cp=.007,nbagg=25,coob=T)

> print(oz.bag25)

Bagging regression trees with 25 bootstrap replications

Call: bagging.data.frame(formula = tupoz ~ ., data = Ozdata2, cp = 0.007,

nbagg = 25, coob = T)

Out-of-bag estimate of root mean squared error: 0.2691

The RMSEP estimate is lower for the bagged estimate when .

> oz.bag100 = bagging(tupoz~.,data=Ozdata2,cp=.007,nbagg=100,coob=T)

> print(oz.bag100)

Bagging regression trees with 100 bootstrap replications

Call: bagging.data.frame(formula = tupoz ~ ., data = Ozdata2, cp = 0.007,

nbagg = 100, coob = T)

Out-of-bag estimate of root mean squared error: 0.2625

> oz.bag1000 = bagging(tupoz~.,data=Ozdata2,cp=.007,nbagg=1000,coob=T)

> print(oz.bag1000)

Bagging regression trees with 1000 bootstrap replications

Call: bagging.data.frame(formula = tupoz ~ ., data = Ozdata2, cp = 0.007,

nbagg = 1000, coob = T)

Out-of-bag estimate of root mean squared error: 0.2604

🡨 warning this is slow! (1 minute or so)

> oz.bag10000 = bagging(tupoz~.,data=Ozdata2,cp=.007,nbagg=10000,coob=T)

> print(oz.bag100000)

Bagging regression trees with 10000 bootstrap replications

Call: bagging.data.frame(formula = tupoz ~ ., data = Ozdata2, cp = 0.007,

nbagg = 10000, coob = T)

Out-of-bag estimate of root mean squared error: 0.2602

Fortunately, we see diminishing returns on the number of bootstrap samples used in the bagging process. It is also important to note that for larger more complex data using bootstrap samples could be very expensive computationally!

Below is code for a split-sample Monte Carlo cross-validation function that takes a model from specified using **rpart**, (not bagging) as an argument. The function bag.sscv will automatically grab any tuning parameters that have been set in the rpart fit, namely cp, minbucket, and minsplit. If you did not specify these yourselves it will still grab their default settings. It will compute the usual prediction measures (RMSEP, MAEP, and MAPEP) but also will save the out-of-bag estimates of the RMSEP from the bootstrap samples used in the bagging process.

We need can also specify the number the fraction of data to using the Monte Carlo split-sample training sets ( by default), the number of Monte Carlo simulations to run ( by default), and the number of bootstrap samples based upon the training data to use in the bagging process ( by default).

**Monte Carlo Split-Sample CV code for Bagging Regression Trees**

bag.sscv = function(fit,data,p=.667,M=100,B=25,  
cp=fit$control$cp,minbucket=fit$control$minbucket,minsplit=fit$control$minsplit) {

OOBMSE = rep(0,M)

MSE = rep(0,M)

MAE = rep(0,M)

MAPE = rep(0,M)

y = fit$y

n = nrow(data)

ss <- floor(n\*p)

for (i in 1:M) {

sam = sample(1:n,ss,replace=F)

fit2 = bagging(formula(fit),data=data[sam,],nbagg=B,cp=cp,  
 minbucket=minbucket,minsplit=minsplit,coob=T)

ynew = predict(fit2,newdata=data[-sam,])

OOBMSE[i] = fit2$err  
 MSE[i] = mean((y[-sam]-ynew)^2)

MAE[i] = mean(abs(y[-sam]-ynew))

MAPE[i] = mean((abs(y[-sam]-ynew)/y[-sam]))

}

OOB.RMSEP = mean(OOBMSE)

RMSEP = sqrt(mean(MSE))

MAEP = mean(MAE)

MAPEP = mean(MAPE)

cat("OOB RMSEP\n")

cat("===============\n")

cat(OOB.RMSEP,"\n\n")

cat("RMSEP\n")

cat("===============\n")

cat(RMSEP,"\n\n")

cat("MAE\n")

cat("===============\n")

cat(MAEP,"\n\n")

cat("MAPE\n")

cat("===============\n")

cat(MAPEP,"\n\n")

temp = data.frame(OOB.RMSEP=OOBMSE,MSEP=MSE,MAEP=MAE,MAPEP=MAPE)

return(temp)

}

We will consider some examples of using this function with the LA ozone data. We first fit a model using

> oz.rpart = rpart(tupoz~.,data=Ozdata2,cp=.007)  
> results = bag.sscv(oz.rpart,Ozdata2,M=100,B=25)

OOB RMSEP

===============

0.2728544

RMSEP

===============

0.2686007

MAE

===============

0.2125346

MAPE

===============

0.1066078

> results = bag.sscv(oz.rpart,Ozdata2,M=100,B=100) 🡨 takes about a minute to run

OOB RMSEP

===============

0.2686108

RMSEP

===============

0.262795

MAE

===============

0.2082436

MAPE

===============

0.1042062

> oz.rpart2 = rpart(tupoz~.,data=Ozdata2,cp=.0001,minbucket=5) 🡨 overfitting?  
> results = bag.sscv(oz.rpart2,Ozdata2,M=100,B=25)

OOB RMSEP

===============

0.2737925

RMSEP

===============

0.268216

MAE

===============

0.2139821

MAPE

===============

0.1071728

> results = bag.sscv(oz.rpart2,Ozdata2,M=100,B=50)

OOB RMSEP

===============

0.2712698

RMSEP

===============

0.2627633

MAE

===============

0.2082096

MAPE

===============

0.1037805

> results = bag.sscv(oz.rpart2,Ozdata2,M=100,B=75)

OOB RMSEP

===============

0.2691056

RMSEP

===============

0.2646359

MAE

===============

0.2100651

MAPE

===============

0.1052538

Surprisingly the bagged CART model with is worse than the model.   
Using also produced worse results.

**Example 10.3 (cont’d) – Twin Cities Home Prices**

We will now consider developing a bagged regression tree for these data.

> TCtrain = read.table(file.choose(),header=T,sep=",") <- read **TChomes (train).csv**

> names(TCtrain)

[1] "ListPrice" "BEDS" "BATHS" "SQFT" "LotSize"

[6] "YearBuilt" "Age" "ParkingSpots" "HasGarage" "DOM"

[11] "BeenReduced" "SoldPrev" "LATITUDE" "LONGITUDE" "ShortSale"

> TCtrain = na.omit(TCtrain)

> dim(TCtrain)

[1] 1040 15

> TCtrain = TCtrain[,-6] 🡨 remove YearBuilt (redundant with Age)

> TCtrain$ListPrice = log(TCtrain$ListPrice)

> bag.sslog = function(fit,data,p=.667,M=100,B=25,

cp=fit$control$cp,  
 minbucket=fit$control$minbucket,  
 minsplit=fit$control$minsplit) {

OOBMSE = rep(0,M)

MSE = rep(0,M)

MAE = rep(0,M)

MAPE = rep(0,M)

y = fit$y

yact = exp(y)

n = nrow(data)

ss <- floor(n\*p)

for (i in 1:M) {

sam = sample(1:n,ss,replace=F)

fit2 = bagging(formula(fit),data=data[sam,],nbagg=B,cp=cp,

minbucket=minbucket,minsplit=minsplit,coob=T)

ynew = predict(fit2,newdata=data[-sam,])

ynew = exp(ynew)

MSE[i] = mean((yact[-sam]-ynew)^2)

MAE[i] = mean(abs(yact[-sam]-ynew))

MAPE[i] = mean((abs(yact[-sam]-ynew)/yact[-sam]))

}

RMSEP = sqrt(mean(MSE))

MAEP = mean(MAE)

MAPEP = mean(MAPE)

cat("RMSEP\n")

cat("===============\n")

cat(RMSEP,"\n\n")

cat("MAE\n")

cat("===============\n")

cat(MAEP,"\n\n")

cat("MAPE\n")

cat("===============\n")

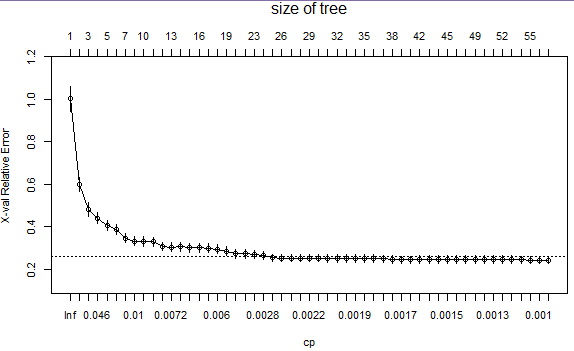
cat(MAPEP,"\n\n")

temp = data.frame(MSEP=MSE,MAEP=MAE,MAPEP=MAPE)

return(temp)

}

> tc.rpart = rpart(ListPrice~.,data=TCtrain,cp=.001,minbucket=5)

> plotcp(tc.rpart)  


> printcp(tc.rpart)

Regression tree:

rpart(formula = ListPrice ~ ., data = TCtrain, cp = 0.001, minbucket = 5)

Variables actually used in tree construction:

[1] Age BATHS BEDS BeenReduced DOM LATITUDE

[7] LONGITUDE SoldPrev SQFT

Root node error: 599.38/1040 = 0.57633

n= 1040

CP nsplit rel error xerror xstd

1 0.4449269 0 1.00000 1.00149 0.058912

2 0.1152848 1 0.55507 0.59743 0.034260

3 0.0539480 2 0.43979 0.48037 0.030800

4 0.0387817 3 0.38584 0.43925 0.026548

5 0.0274396 4 0.34706 0.40499 0.025649

6 0.0246713 5 0.31962 0.38497 0.024104

7 0.0105418 6 0.29495 0.34541 0.021516

8 0.0101895 8 0.27386 0.33143 0.021630

9 0.0092555 9 0.26367 0.32951 0.021554

10 0.0088995 10 0.25442 0.32769 0.021562

11 0.0073737 11 0.24552 0.30701 0.020480

12 0.0070159 12 0.23815 0.30388 0.021562

13 0.0067514 13 0.23113 0.30431 0.021563

14 0.0063446 14 0.22438 0.30060 0.021506

15 0.0062845 15 0.21803 0.30121 0.021489

16 0.0060403 16 0.21175 0.29865 0.021417

17 0.0059035 17 0.20571 0.29377 0.021383

18 0.0047908 18 0.19981 0.28380 0.021026

19 0.0038045 20 0.19022 0.27202 0.019794

20 0.0032352 21 0.18642 0.27119 0.019599

21 0.0031566 22 0.18318 0.26827 0.019419

22 0.0025252 23 0.18003 0.26374 0.019132

23 0.0023320 24 0.17750 0.25318 0.017901

24 0.0023071 25 0.17517 0.25209 0.017891

25 0.0023046 26 0.17286 0.25193 0.017882

26 0.0021779 27 0.17056 0.25145 0.017824

27 0.0021320 28 0.16838 0.25181 0.017794

28 0.0020703 29 0.16625 0.25215 0.017820

29 0.0020228 30 0.16418 0.25098 0.017815

30 0.0018637 31 0.16216 0.24976 0.017860

31 0.0018580 32 0.16029 0.24887 0.017856

32 0.0018503 33 0.15843 0.24907 0.017867

33 0.0018152 34 0.15658 0.24914 0.017866

34 0.0018039 35 0.15477 0.24875 0.017846

35 0.0017553 36 0.15296 0.24767 0.017810

36 0.0016969 37 0.15121 0.24482 0.017742

37 0.0016679 38 0.14951 0.24301 0.017698 🡨 .24301+.017698 = 0.260708

38 0.0016162 40 0.14618 0.24318 0.017716

39 0.0015581 41 0.14456 0.24434 0.017734

40 0.0015024 42 0.14300 0.24561 0.017797

41 0.0014633 43 0.14150 0.24613 0.017778

42 0.0014375 44 0.14004 0.24660 0.017837

43 0.0014197 45 0.13860 0.24596 0.017819

44 0.0013300 47 0.13576 0.24606 0.017835

45 0.0013244 48 0.13443 0.24487 0.017793

46 0.0013158 49 0.13311 0.24504 0.017785

47 0.0013051 50 0.13179 0.24437 0.017790

48 0.0012571 51 0.13048 0.24403 0.017793

49 0.0011888 52 0.12923 0.24586 0.017815

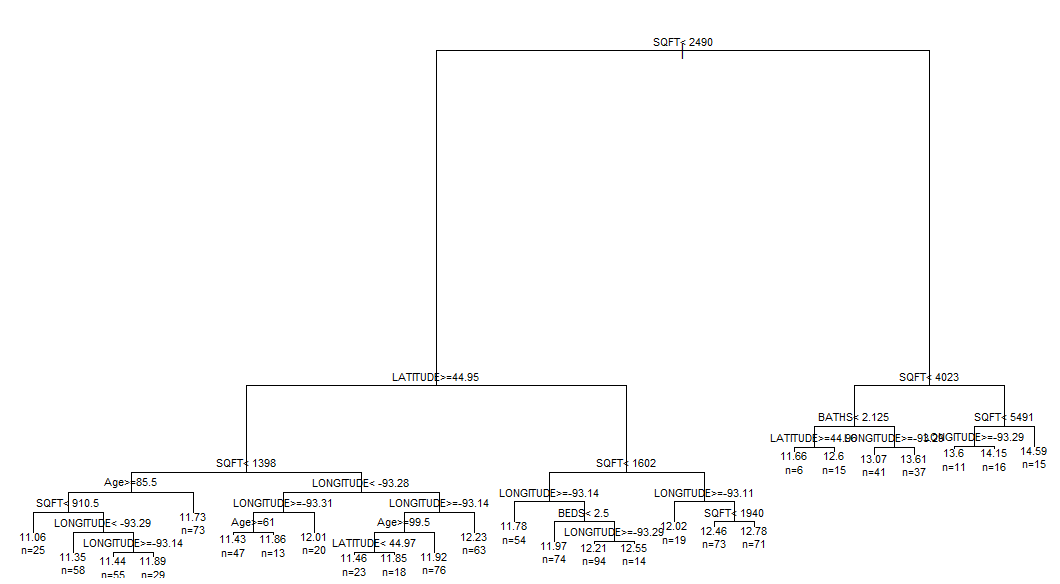
50 0.0011253 53 0.12804 0.24381 0.017063

51 0.0010101 54 0.12691 0.24110 0.016490

52 0.0010058 56 0.12489 0.24238 0.017093

53 0.0010000 57 0.12389 0.24254 0.017094

> tc.rpart = rpart(ListPrice~.,data=TCtrain,cp=.002332,minbucket=5)



> results = bag.sslog(tc.rpart,data=TCtrain) 🡨 takes a while to run!

RMSEP

===============

209823.3

MAE

===============

81119.58

MAPE

===============

0.3166725

> tc.rpart = rpart(ListPrice~.,data=TCtrain,cp=.002332,minbucket=5)

> results = bag.sslog(tc.rpart,data=TCtrain,B=50) 🡨 takes about 5 minutes to run!  
RMSEP

===============

These results seem pretty respectable, for these data. We could consider experimenting further with the tuning parameters (cp, minbucket, minsplit, and the number trees averaged to produce the final model (nbagg or B in MCSS-CV function).

204034

MAE

===============

78749.09

MAPE

===============

0.3175266