Classification 03

CART - Classification and regression trees

Josep Fortiana 2019-11-05

The `rpart` function (recursive partitioning) in the `rpart` R package implements the *Classification And Regresion Trees (CART)* algorithm by Breiman and Friedman. Its main features are: (a) It can predict either a numerical response (regression) or a qualitative response (classification), and (b) Ability to deal with both qualitative and numerical predictors. Furthermore, (c) In each CART partition step the optimized quantity is, by default, Gini's impurity index (see Victor Zhou's blog, instead of the information gain used in the predecessor algorithm ID3, or the χ^2 statistic used in CHAID. The `rpart` implementation includes information gain as an optional parameter, which allows us to simulate ID3, as we see below.

The CART algorithm works by recursively partitioning a `train` data set in successive steps, in each of which according to values of one of the predictor variables, chosen as the one optimizing the impurity index.

Each continuous numerical predictor is treated by sweeping its range and locating an interior point where the impurity index increment resulting from a partition attains an extreme value. The partition is then performed according to the variable for which this extreme value is the best. It is possible that two or more partitions in the resulting hierarchy is done according to the same predictor.

1. Classification with *CART*

1.1. `synth.tr` and `synth.te` data sets

Synthetic Classification Problem

Description

The `synth.tr` data frame has 250 rows and 3 columns. The `synth.te` data frame has 100 rows and 3 columns. It is intended that `synth.tr` be used from training and `synth.te` for testing.

Format

These data frames contains the following columns:

`xs` : x-coordinate

`ys` : y-coordinate

`yc` : class, coded as 0 or 1.

Source

Ripley, B.D. (1994), "Neural networks and related methods for classification (with discussion)". Journal of the Royal Statistical Society series B 56, 409–456.

```
require(MASS)
```

```
## Loading required package: MASS
```

```
data(synth.tr)
data(synth.te)
str(synth.tr)
```

```
## 'data.frame':    250 obs. of  3 variables:
##  $ xs: num  0.051 -0.748 -0.773 0.218 0.373 ...
##  $ ys: num  0.161 0.089 0.263 0.127 0.497 ...
##  $ yc: int  0 0 0 0 0 0 0 0 0 0 ...
```

```r
str(synth.te)
```

```
## 'data.frame':    1000 obs. of  3 variables:
##  $ xs: num  -0.971 -0.632 -0.774 -0.606 -0.539 ...
##  $ ys: num  0.429 0.252 0.691 0.176 0.377 ...
##  $ yc: int  0 0 0 0 0 0 0 0 0 0 ...
```

Cast `yc` as a factor (it is numerically coded 0/1). This is important for `rpart` as it deals with both classification and regression; if not explicitly stated the program has no way to know this response is a label in a classification problem.

```r
synth.tr$yc<-factor(synth.tr$yc)
synth.te$yc<-factor(synth.te$yc)
```

Tree construction

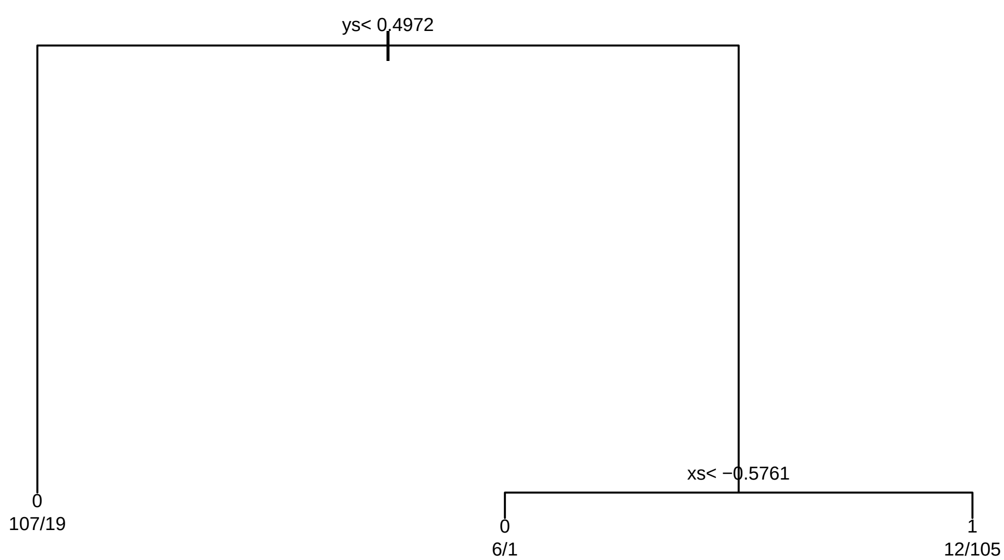
```r
#install.packages("rpart",dependencies=TRUE,repos="https://cloud.r-project.org")
require(rpart)
```

```
## Loading required package: rpart
```

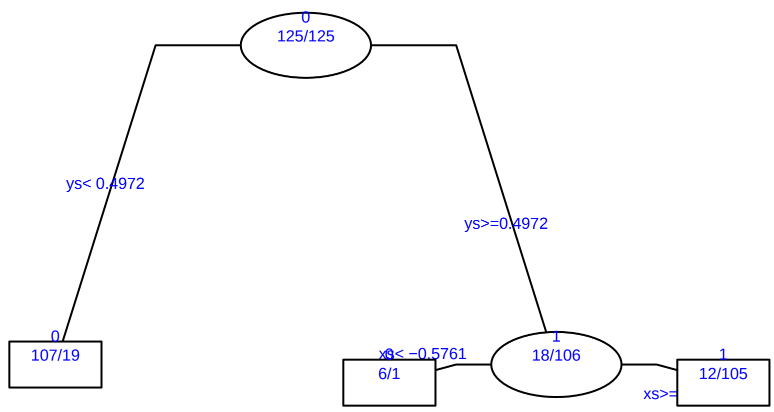
```r
synth.rpart<-rpart(yc~xs+ys,data=synth.tr)
```

Show the tree

```r
options(repr.plot.width=4,repr.plot.height=4)
plot(synth.rpart)
text(synth.rpart,use.n=TRUE,xpd=2,cex=0.6)
```



```r
#
# See the help in 'plot.rpart' and 'test.rpart'
#
# 'margin' space around tree
# 'branch' to decie whether to plot a 'square' tree or with not orthogonal branch angles
options(repr.plot.width=5.5,repr.plot.height=5.5)
plot(synth.rpart,branch=0.6,margin=0.2,lwd=2)
# 'splits' to label each partition
# 'all' to label all nodel (instead of only terminal nodes)
# 'xpd' to allow extending labels out of the plot rectangle
# 'cex' scale of labels
# 'fancy' to put frames around nodes
# 'fwidth' ,'fheight', scale for frames around nodes
text(synth.rpart,use.n=TRUE,fancy=TRUE,fwidth=0.4,fheight=0.4,splits=TRUE,cex=0.5,all=TRUE,xpd=2,col="bl
```



Confusion matrix

```r
synth.rpart.pred<- predict(synth.rpart,synth.te, type = "class")
synth.rpart.conf<-table(True = synth.te$yc, Pred = synth.rpart.pred)
synth.rpart.conf
```

```
##      Pred
## True   0   1
##    0 469  31
##    1  72 428
```

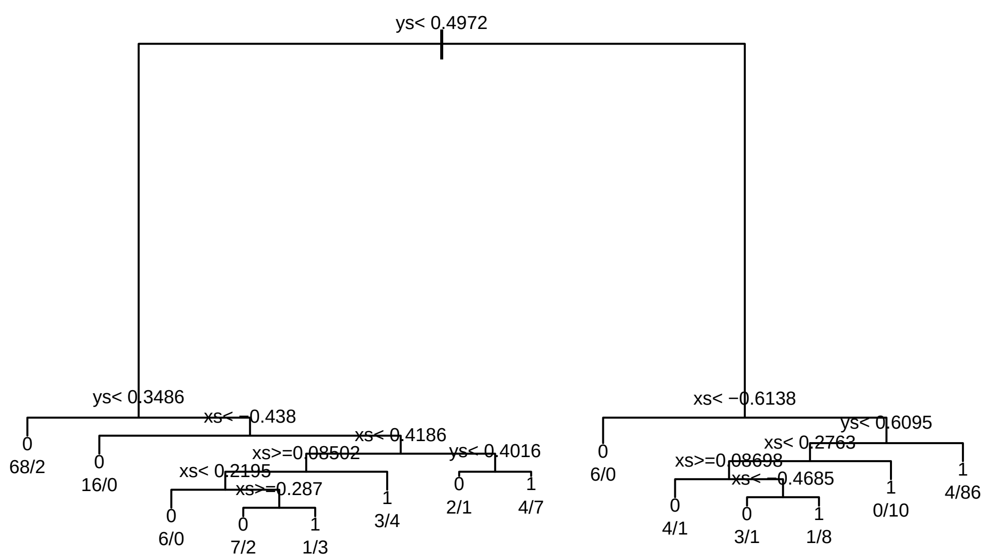
Misclassification error estimate

```
n<-sum(synth.rpart.conf)
n1<-sum(diag(synth.rpart.conf))
P.err<-100*(n-n1)/n
round(P.err,3)
```

```
## [1] 10.3
```

The resulting tree can be modulated by modifying control parameters in `rpart`. This can be done through the `rpart.control()` function. See the help to find about possible parameters to be modified and their default values. For instance:

```
options(repr.plot.width=5.5,repr.plot.height=5.5)
control.parms<-rpart.control(minsplit = 10,cp=0.005)
synth.rpart.2<-rpart(yc~xs+ys,data=synth.tr,control=control.parms)
plot(synth.rpart.2)
text(synth.rpart.2,use.n=TRUE,xpd=2,cex=0.6)
```



Confusion matrix

```
synth.rpart.2.pred<-predict(synth.rpart.2,synth.te,type="class")
synth.rpart.2.conf<-table(True = synth.te$yc,Pred=synth.rpart.2.pred)
synth.rpart.2.conf
```

```
##     Pred
## True   0   1
##    0 449  51
```

```
##    1  50 450
```

Misclassification error estimate

```
n.2<-sum(synth.rpart.2.conf)
n1.2<-sum(diag(synth.rpart.2.conf))
P.err.2<-100*(n.2-n1.2)/n
round(P.err.2,3)
```

```
## [1] 10.1
```

`minsplit` and `cp` are the most relevant parameters. Try with several values of these, and other, parameters, seeing their influence on the missclassification error estimate.

1.2. `Titanic` data

Survival of passengers on the Titanic

Description

This data set provides information on the fate of passengers on the fatal maiden voyage of the ocean liner 'Titanic', summarized according to economic status (class), sex, age and survival.

Format

In the `datasets` R package, under the `Titanic`, these data appear in an aggregated form, as a table of frequencies. A 4-dimensional array resulting from cross-tabulating 2201 observations on 4 variables. The variables and their levels are as follows:

```
No  Name      Levels
1.  Class:    "1st", "2nd", "3rd", "Crew",
2.  Age:      "Child", "Adult"
3.  Sex:      "Male", "Female",
4.  Survived: "No", "Yes".
```

For the purpose of the present laboratory you may find it unfolded as a `data.frame` with 2201 rows, one for each of the 2201 passengers and crew, in the file `Titanic.data.txt`.

Details

The sinking of the Titanic is a famous event, and new books are still being published about it. Many well-known facts—from the proportions of first-class passengers to the 'women and children first' policy, and the fact that that policy was not entirely successful in saving the women and children in the third class—are reflected in the survival rates for various classes of passenger.

These data were originally collected by the British Board of Trade in their investigation of the sinking. Note that there is not complete agreement among primary sources as to the exact numbers on board, rescued, or lost.

Due in particular to the very successful film 'Titanic', the last years saw a rise in public interest in the Titanic. Very detailed data about the passengers is now available on the Internet, at sites such as Encyclopedia Titanica.

Source Dawson, Robert J. MacG. (1995), The 'Unusual Episode' Data Revisited. Journal of Statistics Education, 3. doi: 10.1080/10691898.1995.11910499.

The source provides a data set recording class, sex, age, and survival status for each person on board of the Titanic, and is based on data originally collected by the British Board of Trade and reprinted in:

British Board of Trade (1990), Report on the Loss of the 'Titanic' (S.S.). British Board of Trade Inquiry Report (reprint). Gloucester, UK: Allan Sutton Publishing.

```r
Titanic1<-read.table("Titanic.data.txt",header=TRUE)
str(Titanic1)
```

```
## 'data.frame':    2201 obs. of  4 variables:
##  $ Class   : Factor w/ 4 levels "1st","2nd","3rd",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Age     : Factor w/ 2 levels "adult","child": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Sex     : Factor w/ 2 levels "female","male": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Survived: Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 2 2 2 ...
```

```r
#install.packages("rpart",dependencies=TRUE,repos="https://cloud.r-project.org")
#install.packages("rpart.plot",dependencies=TRUE,repos="https://cloud.r-project.org")
require(rpart)
require(rpart.plot)
```

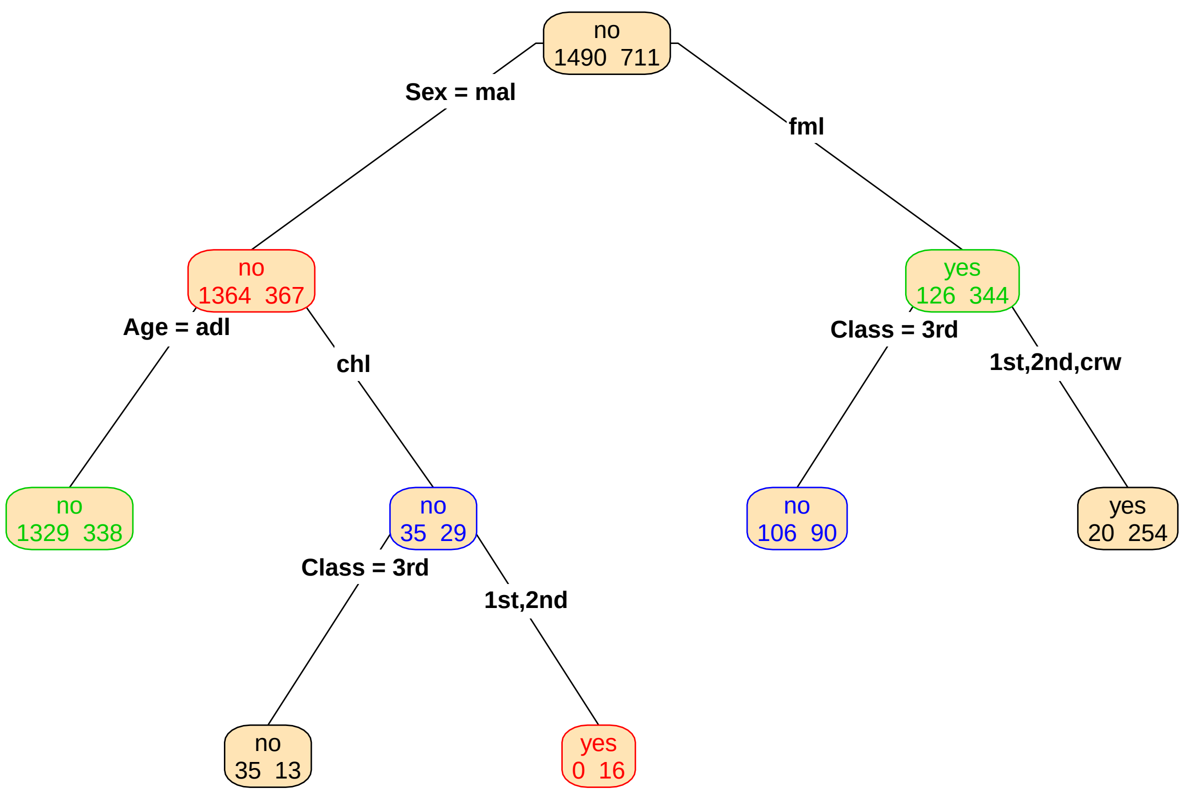
```
## Loading required package: rpart.plot
```

The following classification uses the `rpart()` function to reproduce the behavior of the `ID3` algorithm.

Then we try the `prp()`, in the `rpart.plot` package for plots. Here we show only one of the many possible graphics presentation options. RTFM to know them.

```r
options(repr.plot.width=5.5,repr.plot.height=5.5)
Titanic.ID3<-rpart(Survived~.,data=Titanic1,minsplit=10,parms=list(split="information"))
prp(Titanic.ID3,type=4,extra=1,col=1:4,tweak=1.2,box.col="Moccasin",cex=0.6)
```

```
## Warning: cex and tweak both specified, applying both
```



1.3. SAheart data

From the ElemStatLearn package, SAheart is a data frame with 462 observations on the following 10 variables.

1. sbp: systolic blood pressure.
2. tobacco: cumulative tobacco (kg).
3. ldl: low density lipoprotein cholesterol.
4. adiposity: a numeric vector.
5. famhist: family history of heart disease, a factor with levels Absent, Present.
6. typea: type-A behavior.
7. obesity: a numeric vector.
8. alcohol: current alcohol consumption.
9. age: age at onset
10. chd: response, coronary heart disease

Details

A retrospective sample of males in a heart-disease high-risk region of the Western Cape, South Africa. There are roughly two controls per case of CHD. Many of the CHD positive men have undergone blood pressure reduction treatment and other programs to reduce their risk factors after their CHD event. In some cases the measurements were made after these treatments. These data are taken from a larger dataset, described in Rousseauw et al, 1983, South African Medical Journal.

```
#install.packages("ElemStatLearn",dependencies=TRUE,repos="https://cloud.r-project.org")
require(ElemStatLearn)
```

```
## Loading required package: ElemStatLearn
```

```
data(SAheart)
str(SAheart)
```

```
## 'data.frame':    462 obs. of  10 variables:
##  $ sbp      : int  160 144 118 170 134 132 142 114 114 132 ...
##  $ tobacco  : num  12 0.01 0.08 7.5 13.6 6.2 4.05 4.08 0 0 ...
##  $ ldl      : num  5.73 4.41 3.48 6.41 3.5 6.47 3.38 4.59 3.83 5.8 ...
##  $ adiposity: num  23.1 28.6 32.3 38 27.8 ...
##  $ famhist  : Factor w/ 2 levels "Absent","Present": 2 1 2 2 2 2 1 2 2 2 ...
##  $ typea    : int  49 55 52 51 60 62 59 62 49 69 ...
##  $ obesity  : num  25.3 28.9 29.1 32 26 ...
##  $ alcohol  : num  97.2 2.06 3.81 24.26 57.34 ...
##  $ age      : int  52 63 46 58 49 45 38 58 29 53 ...
##  $ chd      : int  1 1 0 1 1 0 0 1 0 1 ...
```

Cast the response chd as a factor (it is numerically coded 0/1). This is important for rpart() as it deals with both classification and regression; if not explicitly stated the program has no way to know this response is a label in a classification problem.

```
SAheart$chd<-factor(SAheart$chd)
```

Optionally, famhist can be cast as a numeric predictor (in the given data.frame is a factor). This is not required for prediction trees, as the CART algorithm can directly process factor predictors, but it might be useful to compare results with other prediction methods.

```r
#SAheart$famhist<-as.numeric(SAheart$famhist)
```

As usual, split the dataset in `train` and `test` subsets.

```r
n<-nrow(SAheart)
ntrain<-ceiling(0.60*n)
Itrain<-sample(1:n,ntrain,replace=FALSE)
n<-nrow(SAheart)
ntrain<-ceiling(0.60*n)
Itrain<-sample(1:n,ntrain,replace=FALSE)
SAheart.train<-SAheart[Itrain,]
SAheart.test<-SAheart[-Itrain,]
```

Build a classification tree, compute the predictions for the `test` subset, obtain the confusion matrix and a misclassification error estimate.

```r
#
#  Insert your code here
#
```

1.4. `Carseats` data with the `tree` package (from ISLR, lab 8.3, pag. 329)

Code from the ISLR book

Sales of Child Car Seats

Description

A simulated data set containing sales of child car seats at 400 different stores.

Format

A data frame with 400 observations on the following 11 variables.

1. `Sales`: Unit sales (in thousands) at each location
2. `CompPrice`: Price charged by competitor at each location
3. `Income`: Community income level (in thousands of dollars)
4. `Advertising`: Local advertising budget for company at each location (in thousands of dollars)
5. `Population`: Population size in region (in thousands)
6. `Price`: Price company charges for car seats at each site
7. `ShelveLoc`: A factor with levels "Bad", "Good" and "Medium" indicating the quality of the shelving location for the car seats at each site
8. `Age`: Average age of the local population
9. `Education`: Education level at each location
10. `Urban`: A factor with levels "No" and "Yes" to indicate whether the store is in an urban or rural location
11. `US`: A factor with levels "No" and "Yes" to indicate whether the store is in the US or not

Source

Simulated data

References

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013) An Introduction to Statistical Learning with applications in R, Springer-Verlag, New York

```r
#install.packages("ISLR",dependencies=TRUE,repos="https://cloud.r-project.org")
require(ISLR)
```

```
## Loading required package: ISLR
```

```r
data(Carseats)
str(Carseats)
```

```
## 'data.frame':    400 obs. of  11 variables:
##  $ Sales      : num  9.5 11.22 10.06 7.4 4.15 ...
##  $ CompPrice  : num  138 111 113 117 141 124 115 136 132 132 ...
##  $ Income     : num  73 48 35 100 64 113 105 81 110 113 ...
##  $ Advertising: num  11 16 10 4 3 13 0 15 0 0 ...
##  $ Population : num  276 260 269 466 340 501 45 425 108 131 ...
##  $ Price      : num  120 83 80 97 128 72 108 120 124 124 ...
##  $ ShelveLoc  : Factor w/ 3 levels "Bad","Good","Medium": 1 2 3 3 1 1 3 2 3 3 ...
##  $ Age        : num  42 65 59 55 38 78 71 67 76 76 ...
##  $ Education  : num  17 10 12 14 13 16 15 10 10 17 ...
##  $ Urban      : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 2 2 1 1 ...
##  $ US         : Factor w/ 2 levels "No","Yes": 2 2 2 2 1 2 1 2 1 2 ...
```

```r
summary(Carseats$Sales)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   5.390   7.490   7.496   9.320  16.270
```

Discretize `Sales` into a factor, `High`, with two levels. Truncation value is slightly greater than the median, hence more than half of data are in the “No” class.

```r
High<-ifelse(Carseats$Sales<=8,"No","Yes")
table(High)
```

```
## High
##  No Yes
## 236 164
```

Append the new variable to the `data.frame` (avoid using the original `Sales` variable as a predictor!).

```r
Carseats.D<-data.frame(Carseats[,-1],High)
str(Carseats.D)
```

```
## 'data.frame':    400 obs. of  11 variables:
##  $ CompPrice  : num  138 111 113 117 141 124 115 136 132 132 ...
##  $ Income     : num  73 48 35 100 64 113 105 81 110 113 ...
##  $ Advertising: num  11 16 10 4 3 13 0 15 0 0 ...
##  $ Population : num  276 260 269 466 340 501 45 425 108 131 ...
##  $ Price      : num  120 83 80 97 128 72 108 120 124 124 ...
##  $ ShelveLoc  : Factor w/ 3 levels "Bad","Good","Medium": 1 2 3 3 1 1 3 2 3 3 ...
##  $ Age        : num  42 65 59 55 38 78 71 67 76 76 ...
##  $ Education  : num  17 10 12 14 13 16 15 10 10 17 ...
##  $ Urban      : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 2 2 1 1 ...
##  $ US         : Factor w/ 2 levels "No","Yes": 2 2 2 2 1 2 1 2 1 2 ...
##  $ High       : Factor w/ 2 levels "No","Yes": 2 2 2 1 1 2 1 2 1 1 ...
```

```r
#install.packages("tree",dependencies=TRUE,repos="https://cloud.r-project.org")
require(tree)
```

```
## Loading required package: tree
```

```r
# attach(Carseats) (not necessary to attach())
# Classification. Response is the new binary variable 'High'
#
Carseats.D.tree<-tree(High~.,Carseats.D)
# Alternative method, as in the book.
# If we did not remove 'Sales' from the dataset then we must discard it explicitly from the set of pred
#Carseats.D.tree<-tree(High~.-Sales,Carseats.D)
```

The `print()` method for the `tree` class generates a verbose description of nodes and splitting criteria.

```r
print(Carseats.D.tree)
```

```
## 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 ) *
```

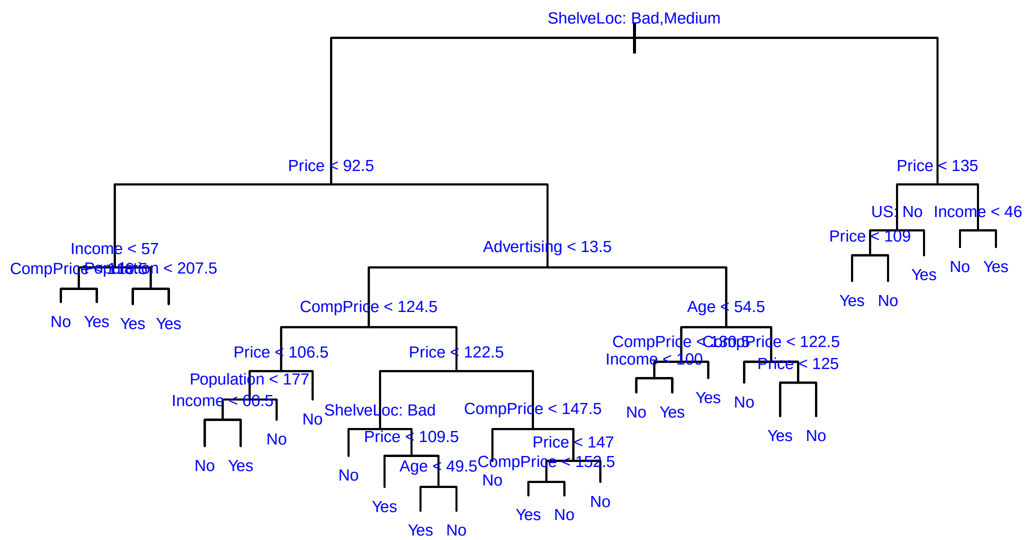
The `summary()` method just a short description.

```r
summary(Carseats.D.tree)
```

```
##
## Classification tree:
## tree(formula = High ~ ., data = Carseats.D)
## 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
```

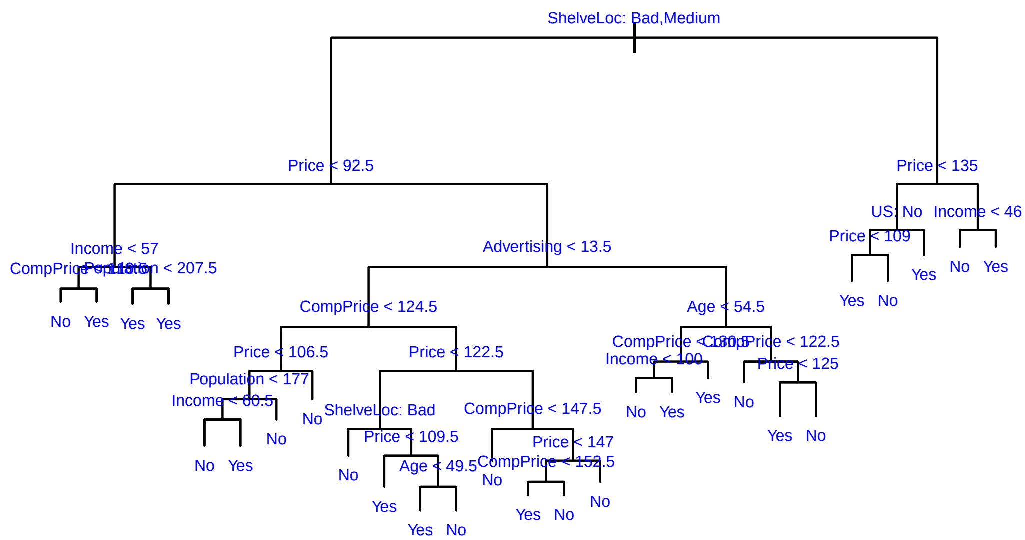
The default `plot()` method. Actually it calls `plot.tree()`

```r
options(repr.plot.width=6.5,repr.plot.height=6.5)
plot(Carseats.D.tree)
text(Carseats.D.tree,pretty=0,cex=0.5,col="blue")
```



See the `proportional` key word in the help for `plot.tree()`.

```
options(repr.plot.width=6.5,repr.plot.height=6.5)
plot(Carseats.D.tree,type='proportional')
text(Carseats.D.tree,pretty=0,cex=0.5,col="blue")
```



Cross-validation to assess misclassification errors

```r
# 'Hold-out' cross-validation
set.seed(2)
n<-nrow(Carseats.D)
n.train<-floor(0.5*n)
n.test<-n-n.train
I.train<-sample(1:n, n.train)
Carseats.D.test<-Carseats.D[-I.train,]
High.test<-High[-I.train]
```

Confusion matrix

```r
#Carseats.D.tree=tree(High~.-Sales,Carseats.D,subset=I.train)
Carseats.D.tree=tree(High~.,Carseats.D,subset=I.train)
Carseats.D.pred=predict(Carseats.D.tree,Carseats.D.test,type="class")
Carseats.D.conf<-table(Carseats.D.pred,High.test)
Carseats.D.conf
```

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

```r
P.err<-(n.test-sum(diag(Carseats.D.conf)))/n.test
paste("Misclassification error estimate = ", round(P.err,3))
```

```
## [1] "Misclassification error estimate =  0.23"
```

Pruning to avoid overfitting

Using the hardwired `cv.tree()` function to prune the obtained tree.

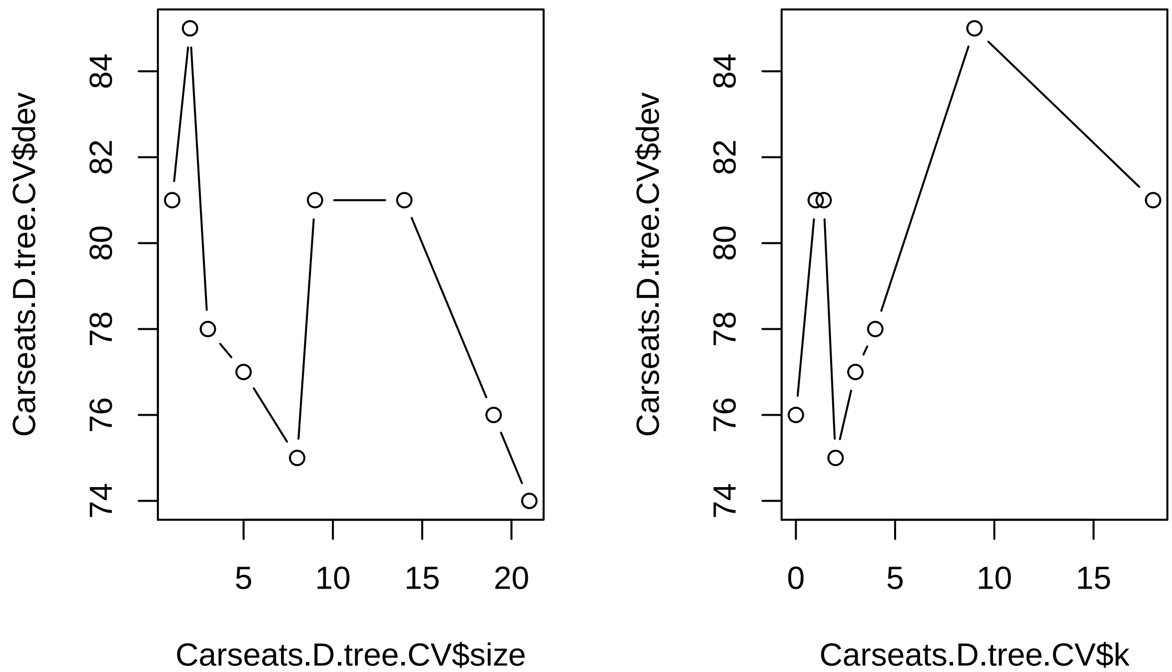
```r
set.seed(3)
Carseats.D.tree.CV=cv.tree(Carseats.D.tree,FUN=prune.misclass)
names(Carseats.D.tree.CV)
```

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

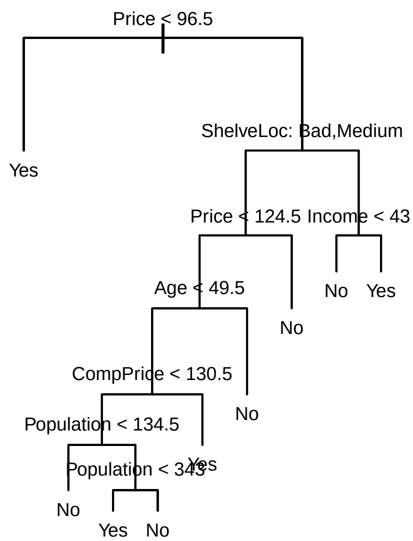
```r
Carseats.D.tree.CV
```

```
## $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"
```

```r
par(mfrow=c(1,2))
plot(Carseats.D.tree.CV$size,Carseats.D.tree.CV$dev,type="b")
plot(Carseats.D.tree.CV$k,Carseats.D.tree.CV$dev,type="b")
```




```
prune.carseats=prune.misclass(Carseats.D.tree,best=9)
plot(prune.carseats)
text(prune.carseats,pretty=0,cex=0.6)
```

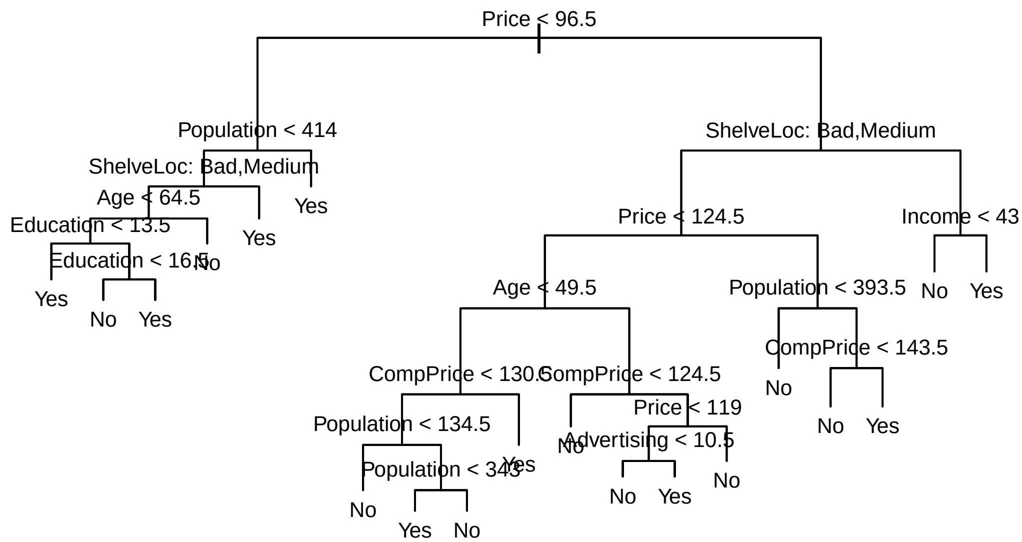



```
Carseats.D.pred=predict(prune.carseats,Carseats.D.test,type="class")
table(Carseats.D.pred,High.test)
```

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

```
#(94+60)/200
```

```
prune.carseats=prune.misclass(Carseats.D.tree,best=15)
plot(prune.carseats)
text(prune.carseats,pretty=0,cex=0.7)
```




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

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

```
#(86+62)/200
```

2. Regression with CART

2.0. Visualizing regression trees

Observe that in regression problems prediction functions are locally constant functions (step functions).

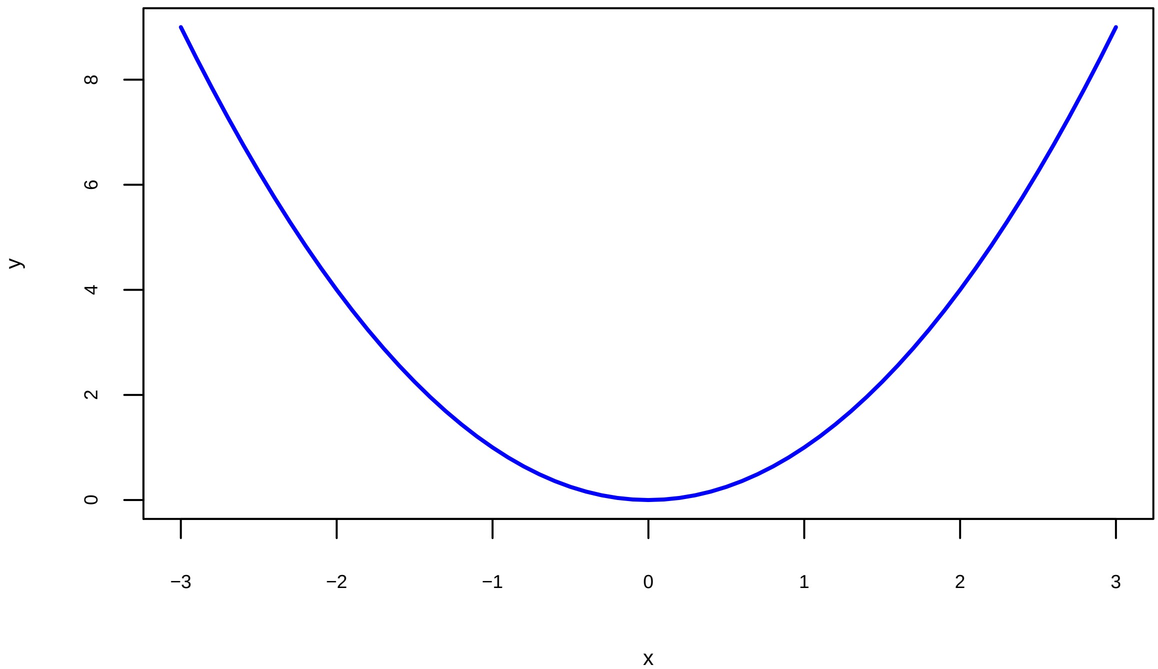
Regression with a single continuous predictor

```
#install.packages("rpart",dependencies=TRUE,repos="https://cloud.r-project.org")
require(rpart)
```

Training set: a collection of (x, y) pairs of values, coordinates of points on a parabola

```
x<-seq(-3,3,by=0.1)
n<-length(x)
y<-x^2
```

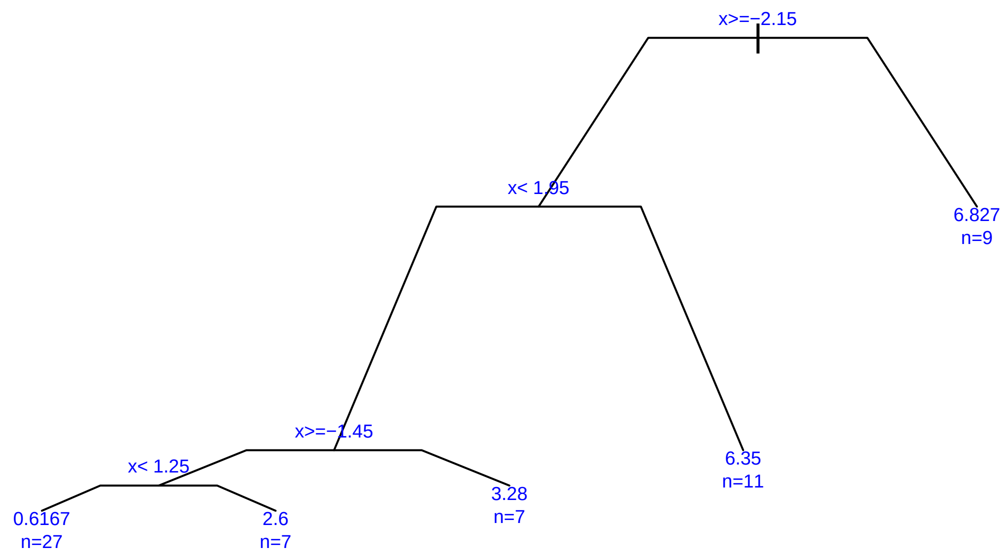
```r
options(repr.plot.width=4,repr.plot.height=4)
old.par<-par(cex.axis=0.6,cex.lab=0.7)
plot(x,y, "l",lwd=2,col="blue")
```




```r
par(old.par)
```

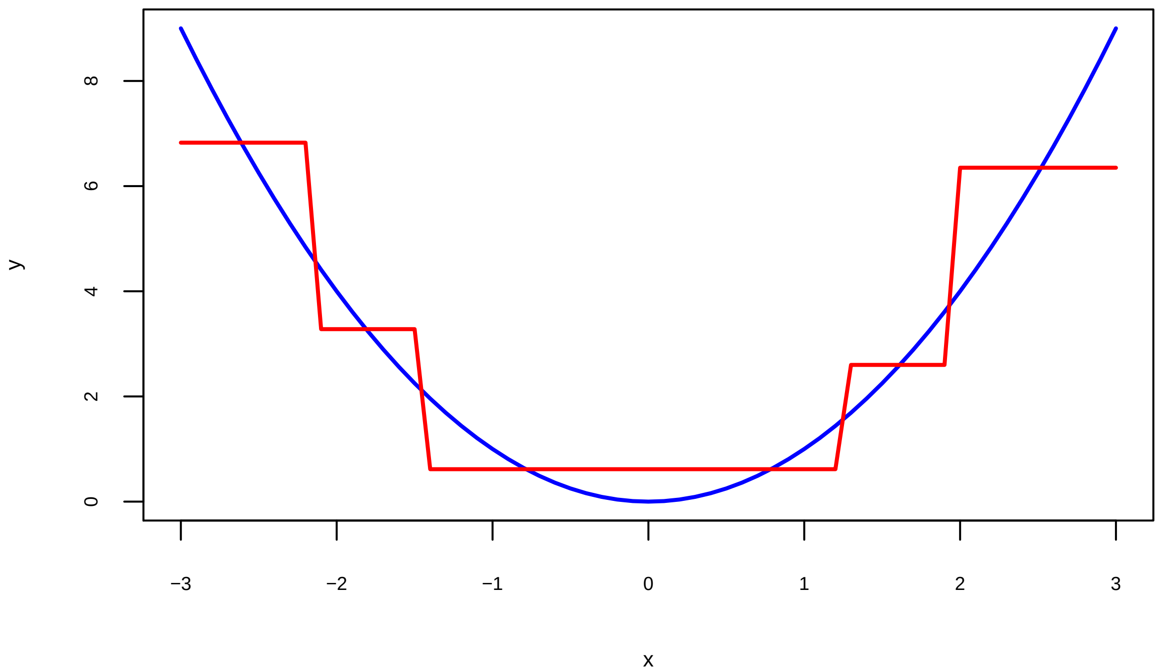
Build and show the tree

```r
r<-rpart(y~x)
options(repr.plot.width=4,repr.plot.height=4)
plot(r,branch=0.5)
text(r,use.n=TRUE,xpd=2,cex=0.6, col="blue")
```



Stepwise prediction function superimposed to training set

```
yp<-predict(r,newdata=data.frame(x))
options(repr.plot.width=4,repr.plot.height=4)
old.par<-par(cex.axis=0.6,cex.lab=0.7)
plot(x,y, "l",lwd=2,col="blue")
lines(x,yp,lwd=2,col="red")
```




```r
par(old.par)
```

Regression with two continuous numerical predictors

```r
## Run this code chunk directly in RStudio. Dynamical graphics do not work well within a Jupyter notebo
#install.packages("tcltk",dependencies=TRUE,repos="https://cloud.r-project.org")
#install.packages("tkrgl",dependencies=TRUE,repos="https://cloud.r-project.org")
#install.packages("tkrplot",dependencies=TRUE,repos="https://cloud.r-project.org")
#install.packages("TeachingDemos",dependencies=TRUE,repos="https://cloud.r-project.org")
require(tcltk)
```

```
## Loading required package: tcltk
```

```r
require(tkrgl)
```

```
## Loading required package: tkrgl
```

```
## Error: package or namespace load failed for 'tkrgl' in loadNamespace(i, c(lib.loc, .libPaths()), vers
##  namespace 'htmltools' 0.3.6 is already loaded, but >= 0.4.0 is required
```

```r
require(tkrplot)
```

```
## Loading required package: tkrplot
```

```r
require(TeachingDemos)
```

```
## Loading required package: TeachingDemos
```

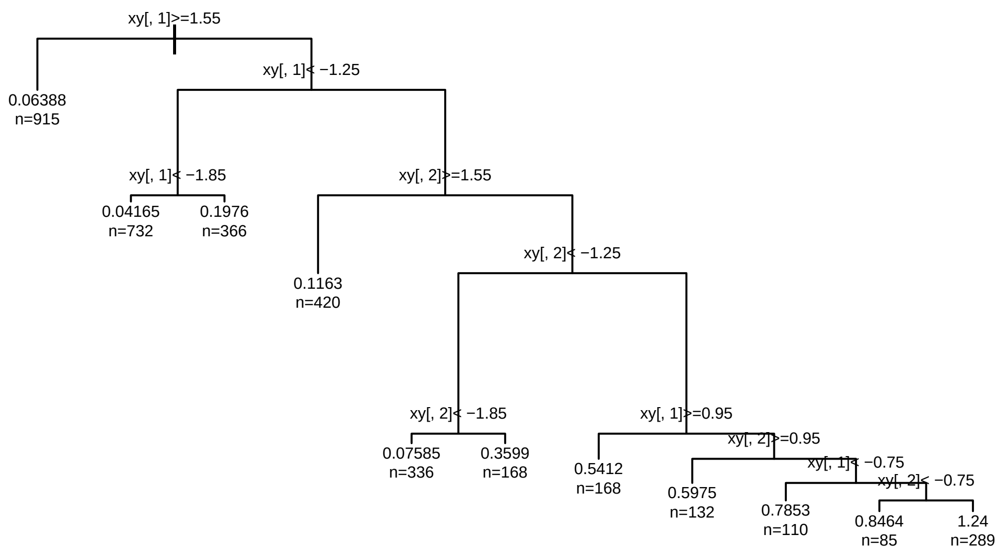
A training set, a collection of pairs (2-dim predictor: (x,y), response: fxy.m)

```
xy<-expand.grid(x,x)
fxy.v<-(1/2*pi)*exp(-0.5*(xy[,1]^2+xy[,2]^2))
fxy.m<-matrix(fxy.v,nrow=length(x))
```

```
## Run this code chunk directly in RStudio. Dynamical graphics do not work well within a Jupyter notebo
#rotate.persp(x,x,fxy.m)
#z<-fxy.v
```

Build and show the tree

```
xy.rpart<-rpart(fxy.v~xy[,1]+xy[,2])
options(repr.plot.width=5,repr.plot.height=5)
plot(xy.rpart)
text(xy.rpart,use.n=TRUE,xpd=2,cex=0.5)
```



Show the prediction function

```
## Run this code chunk directly in RStudio. Dynamical graphics do not work well within a Jupyter notebo
#xy.pred<-predict(xy.rpart,newdata=data.frame(xy))
#xy.pred.m<-matrix(xy.pred,nrow=length(x))
#rotate.persp(x,y,xy.pred.m)
```

2.1. `Boston` data (from `MASS`) with the `tree` package (from ISLR, lab 8.3.2, pag. 332)

Code from the ISLR book

Housing Values in Suburbs of Boston

Description

The Boston data frame has 506 rows and 14 columns.

Format

The Boston data frame has 506 rows and 14 columns (predictors). We have descriptions and summaries of predictors as follows:

1. `crim`: per capita crime rate by town.
2. `zn`: proportion of residential land zoned for lots over 25,000 sq.ft.
3. `indus`: proportion of non-retail business acres per town.
4. `chas`: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).
5. `nox`: nitrogen oxides concentration (parts per 10 million).
6. `rm`: average number of rooms per dwelling.
7. `age`: proportion of owner-occupied units built prior to 1940.
8. `dis`: weighted mean of distances to five Boston employment centres.
9. `rad`: index of accessibility to radial highways.
10. `tax`: full-value property-tax rate per $10,000.
11. `ptratio`: pupil-teacher ratio by town.
12. `black`: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town.
13. `lstat`: lower status of the population (percent).
14. `medv`: median value of owner-occupied homes in $1000s.

```r
require(MASS)
data(Boston)
str(Boston)
```

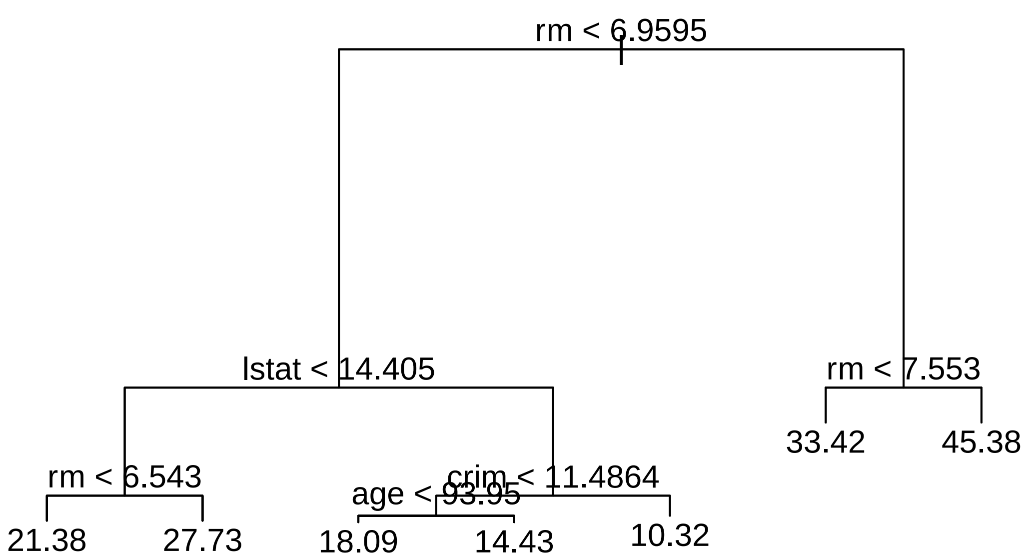
```
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ black  : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

```r
set.seed(1)
train = sample(1:nrow(Boston), nrow(Boston)/2)
```

```r
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
```

```r
options(repr.plot.width=5,repr.plot.height=5)
plot(tree.boston)
text(tree.boston,pretty=0)
```



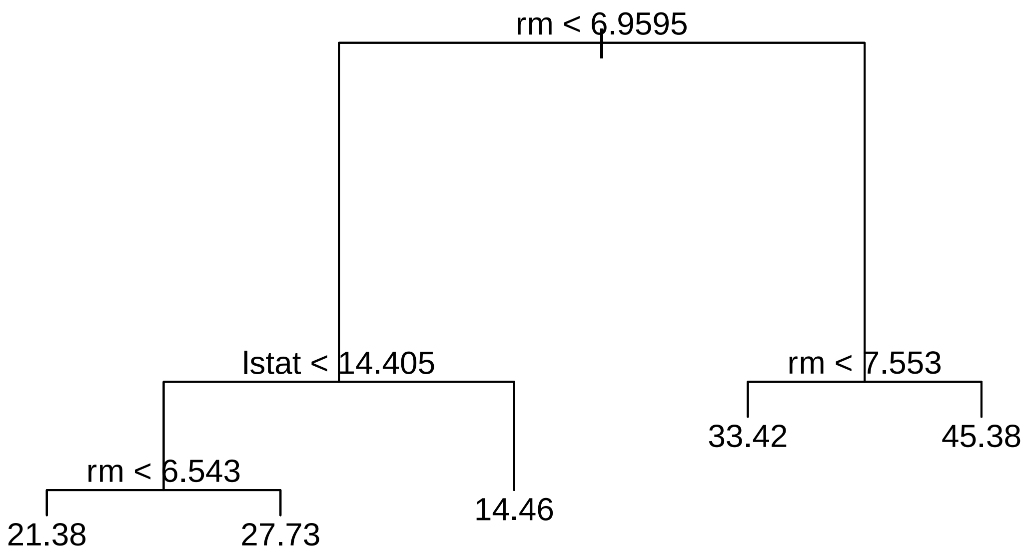

```r
cv.boston=cv.tree(tree.boston)
```

```r
plot(cv.boston$size,cv.boston$dev,type='b')
```

Pruning to avoid overfitting

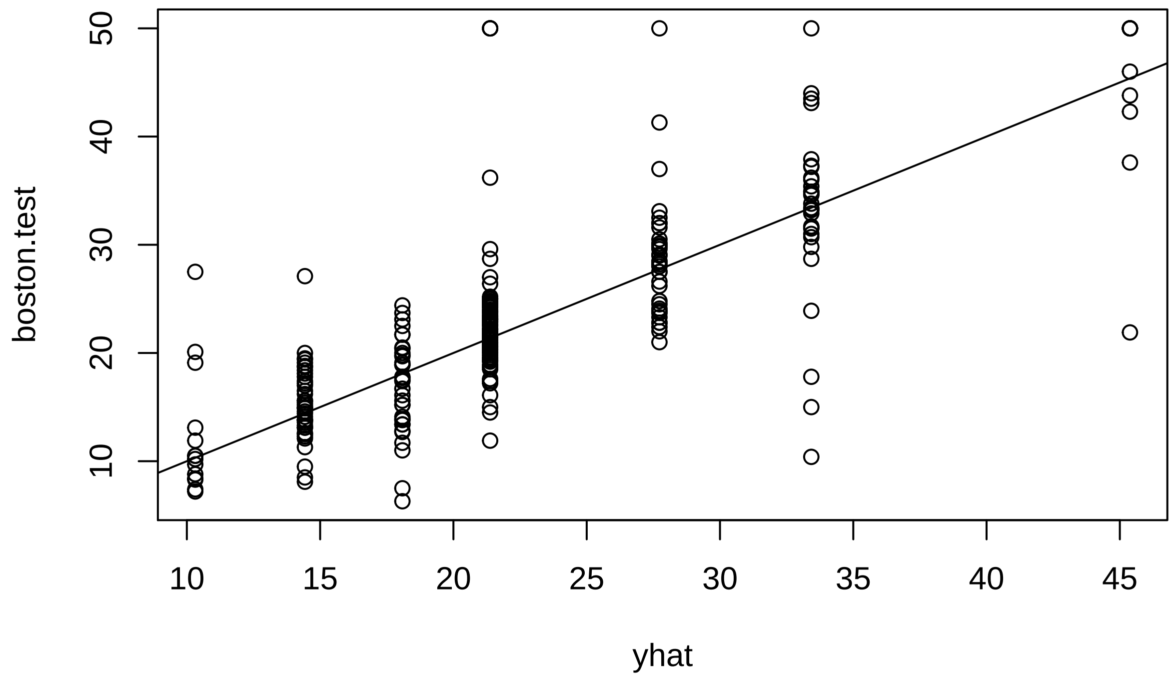
```
prune.boston=prune.tree(tree.boston,best=5)
```

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




```
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
```

2.2. cpus data (from MASS)

Performance of Computer CPUs

Description

A relative performance measure and characteristics of 209 CPUs.

Format

The components are:

1. `name`: manufacturer and model.
2. `syct`: cycle time in nanoseconds.
3. `mmin`: minimum main memory in kilobytes.
4. `mmax`: maximum main memory in kilobytes.
5. `cach`: cache size in kilobytes.
6. `chmin`: minimum number of channels.
7. `chmax`: maximum number of channels.
8. `perf`: published performance on a benchmark mix relative to an IBM 370/158-3.

9. `estperf`: estimated performance (by Ein-Dor & Feldmesser).

Source

P. Ein-Dor and J. Feldmesser (1987), *Attributes of the performance of central processing units: a relative performance prediction model.* Comm. ACM. 30, 308–317.

References

Venables, W. N. and Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth edition. Springer.

```
require(MASS)
data(cpus)
str(cpus)
```

```
## 'data.frame':    209 obs. of  9 variables:
##  $ name   : Factor w/ 209 levels "ADVISOR 32/60",..: 1 3 2 4 5 6 8 9 10 7 ...
##  $ syct   : int  125 29 29 29 29 26 23 23 23 23 ...
##  $ mmin   : int  256 8000 8000 8000 8000 8000 16000 16000 16000 32000 ...
##  $ mmax   : int  6000 32000 32000 32000 16000 32000 32000 32000 64000 64000 ...
##  $ cach   : int  256 32 32 32 32 64 64 64 64 128 ...
##  $ chmin  : int  16 8 8 8 8 8 16 16 16 32 ...
##  $ chmax  : int  128 32 32 32 16 32 32 32 32 64 ...
##  $ perf   : int  198 269 220 172 132 318 367 489 636 1144 ...
##  $ estperf: int  199 253 253 253 132 290 381 381 749 1238 ...
```

Discard the first variable, the brand label.

If desired this list of brand labels can be saved as an auxiliary vector, to be used in plots.

We also discard the last (ninth) variable, an estimate of the response variable by the authors of the original paper.l:

```
labels<-cpus[,1]
cpus<-cpus[,2:8]
rownames(cpus)<-as.character(labels)
str(cpus)
```

```
## 'data.frame':    209 obs. of  7 variables:
##  $ syct : int  125 29 29 29 29 26 23 23 23 23 ...
##  $ mmin : int  256 8000 8000 8000 8000 8000 16000 16000 16000 32000 ...
##  $ mmax : int  6000 32000 32000 32000 16000 32000 32000 32000 64000 64000 ...
##  $ cach : int  256 32 32 32 32 64 64 64 64 128 ...
##  $ chmin: int  16 8 8 8 8 8 16 16 16 32 ...
##  $ chmax: int  128 32 32 32 16 32 32 32 32 64 ...
##  $ perf : int  198 269 220 172 132 318 367 489 636 1144 ...
```

Prepare a new `data.frame` with the logarithmic transform of the response:

```
logcpus<-cpus[,1:6]
logcpus$logperf<-log10(cpus$perf)
str(logcpus)
```

```
## 'data.frame':    209 obs. of  7 variables:
##  $ syct   : int  125 29 29 29 29 26 23 23 23 23 ...
##  $ mmin   : int  256 8000 8000 8000 8000 8000 16000 16000 16000 32000 ...
##  $ mmax   : int  6000 32000 32000 32000 16000 32000 32000 32000 64000 64000 ...
##  $ cach   : int  256 32 32 32 32 64 64 64 64 128 ...
##  $ chmin  : int  16 8 8 8 8 8 16 16 16 32 ...
```

```
##  $ chmax  : int  128 32 32 32 16 32 32 32 32 64 ...
##  $ logperf: num  2.3 2.43 2.34 2.24 2.12 ...
```

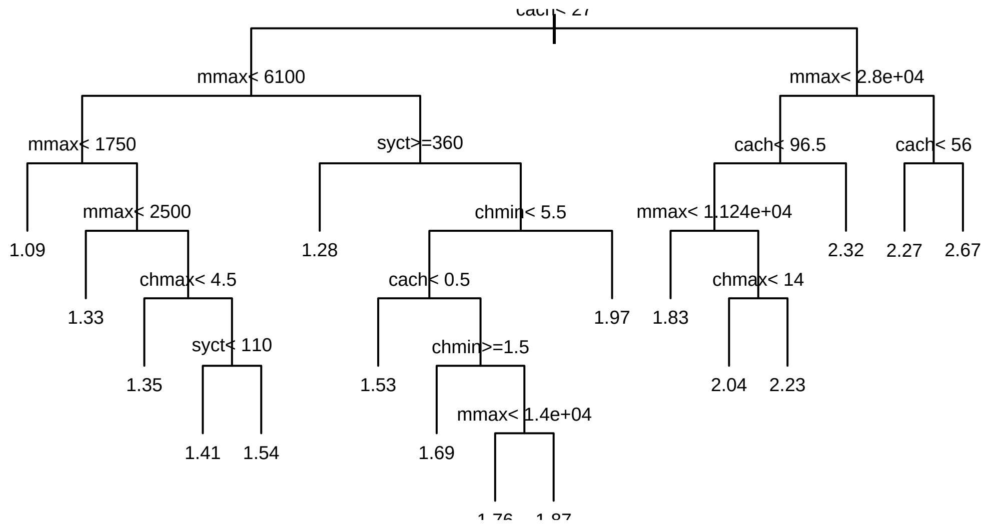
```r
#install.packages("rpart",dependencies=TRUE,repos="https://cloud.r-project.org")
require(rpart)
```

```r
logcpus.rpart.01<-rpart(logperf~.,data=logcpus,cp=1.0e-3)
```

```r
print(logcpus.rpart.01)
```

```
## n= 209
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##   1) root 209 43.11554000 1.753333
##     2) cach< 27 143 11.79085000 1.524647
##       4) mmax< 6100 78  3.89374400 1.374824
##         8) mmax< 1750 12  0.78425160 1.088732 *
##         9) mmax>=1750 66  1.94873300 1.426840
##          18) mmax< 2500 17  0.56676380 1.325292 *
##          19) mmax>=2500 49  1.14584500 1.462071
##            38) chmax< 4.5 14  0.35284960 1.354804 *
##            39) chmax>=4.5 35  0.56747110 1.504978
##              78) syct< 110 9  0.07741772 1.414738 *
##              79) syct>=110 26  0.39139370 1.536215 *
##       5) mmax>=6100 65  4.04520300 1.704434
##        10) syct>=360 7  0.12908090 1.279749 *
##        11) syct< 360 58  2.50124700 1.755690
##          22) chmin< 5.5 46  1.22622900 1.698613
##            44) cach< 0.5 11  0.20206270 1.530643 *
##            45) cach>=0.5 35  0.61627500 1.751403
##              90) chmin>=1.5 15  0.26087250 1.690337 *
##              91) chmin< 1.5 20  0.25751430 1.797203
##               182) mmax< 14000 13  0.08761793 1.756702 *
##               183) mmax>=14000 7  0.10896860 1.872420 *
##          23) chmin>=5.5 12  0.55071310 1.974483 *
##     3) cach>=27 66  7.64263500 2.248821
##       6) mmax< 28000 41  2.34141700 2.061986
##        12) cach< 96.5 34  1.59195100 2.008124
##          24) mmax< 11240 14  0.42462370 1.826635 *
##          25) mmax>=11240 20  0.38340130 2.135166
##            50) chmax< 14 10  0.07835528 2.037790 *
##            51) chmax>=14 10  0.11540350 2.232542 *
##        13) cach>=96.5 7  0.17173020 2.323601 *
##       7) mmax>=28000 25  1.52286300 2.555230
##        14) cach< 56 7  0.06929430 2.268365 *
##        15) cach>=56 18  0.65351270 2.666788 *
```

```r
options(repr.plot.width=5,repr.plot.height=5)
plot(logcpus.rpart.01,uniform=TRUE)
text(logcpus.rpart.01,digits=3,cex=0.6)
```



Pruning to avoid overfitting

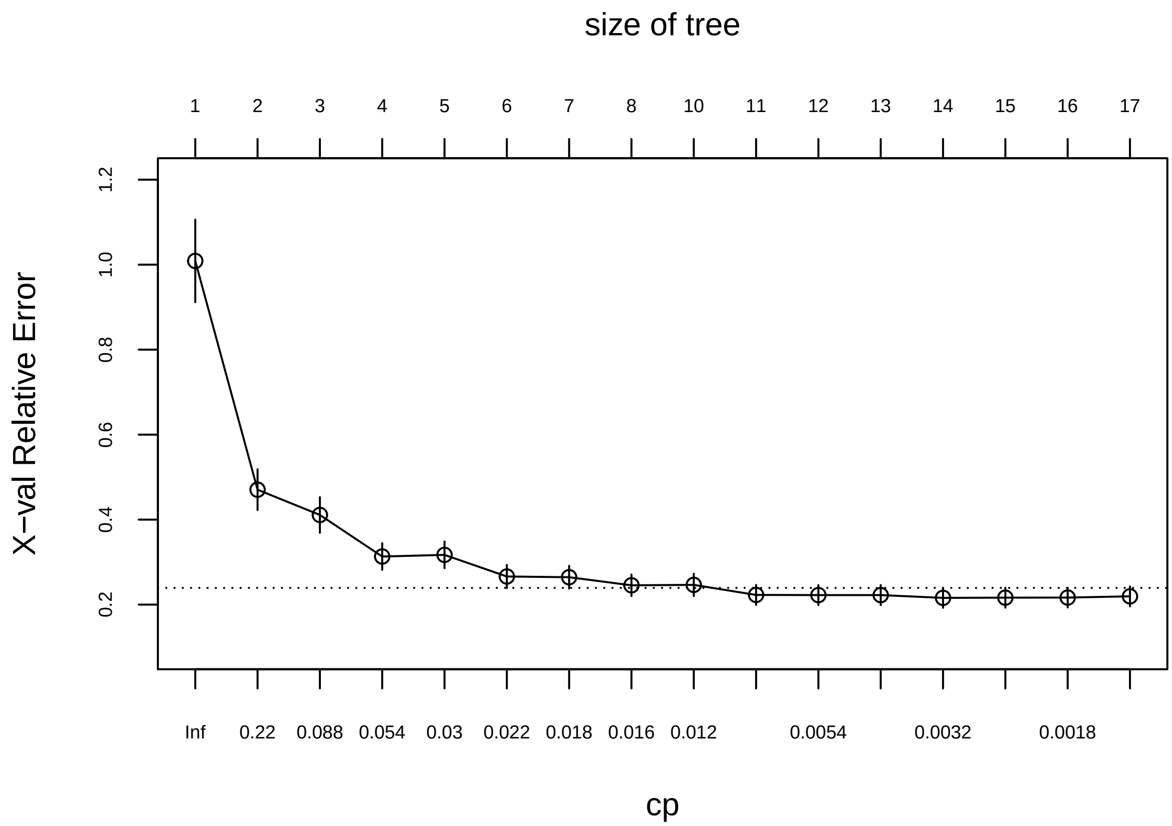
Print values of the complexity parameter cp

```
printcp(logcpus.rpart.01)
```

```
##
## Regression tree:
## rpart(formula = logperf ~ ., data = logcpus, cp = 0.001)
##
## Variables actually used in tree construction:
## [1] cach  chmax chmin mmax  syct
##
## Root node error: 43.116/209 = 0.20629
##
## n= 209
##
##           CP nsplit rel error  xerror     xstd
## 1  0.5492697      0   1.00000 1.00876 0.097175
## 2  0.0893390      1   0.45073 0.47045 0.047958
## 3  0.0876332      2   0.36139 0.41098 0.041916
## 4  0.0328159      3   0.27376 0.31317 0.031408
## 5  0.0269220      4   0.24094 0.31708 0.031502
## 6  0.0185561      5   0.21402 0.26620 0.027217
## 7  0.0167992      6   0.19546 0.26451 0.026823
## 8  0.0157908      7   0.17866 0.24550 0.025310
## 9  0.0094604      9   0.14708 0.24648 0.026248
```

```
## 10 0.0054766     10   0.13762 0.22284 0.023495
## 11 0.0052307     11   0.13215 0.22244 0.023620
## 12 0.0043985     12   0.12692 0.22244 0.023620
## 13 0.0022883     13   0.12252 0.21594 0.023442
## 14 0.0022704     14   0.12023 0.21639 0.023448
## 15 0.0014131     15   0.11796 0.21658 0.023365
## 16 0.0010000     16   0.11655 0.21947 0.023529
```

```r
plotcp(logcpus.rpart.01,cex.axis=0.6)
```



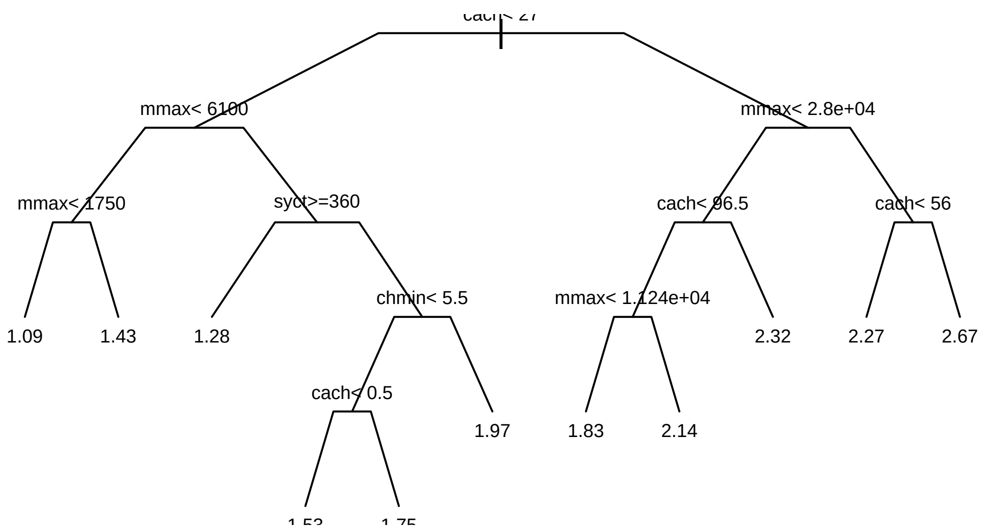

```r
print(logcpus.rpart.01,cp=0.006,digits=3)
```

```
## n= 209
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 209 43.1000 1.75
##    2) cach< 27 143 11.8000 1.52
##      4) mmax< 6.1e+03 78  3.8900 1.37
##        8) mmax< 1.75e+03 12  0.7840 1.09 *
##        9) mmax>=1.75e+03 66  1.9500 1.43 *
##      5) mmax>=6.1e+03 65  4.0500 1.70
##       10) syct>=360 7  0.1290 1.28 *
##       11) syct< 360 58  2.5000 1.76
##         22) chmin< 5.5 46  1.2300 1.70
##           44) cach< 0.5 11  0.2020 1.53 *
```

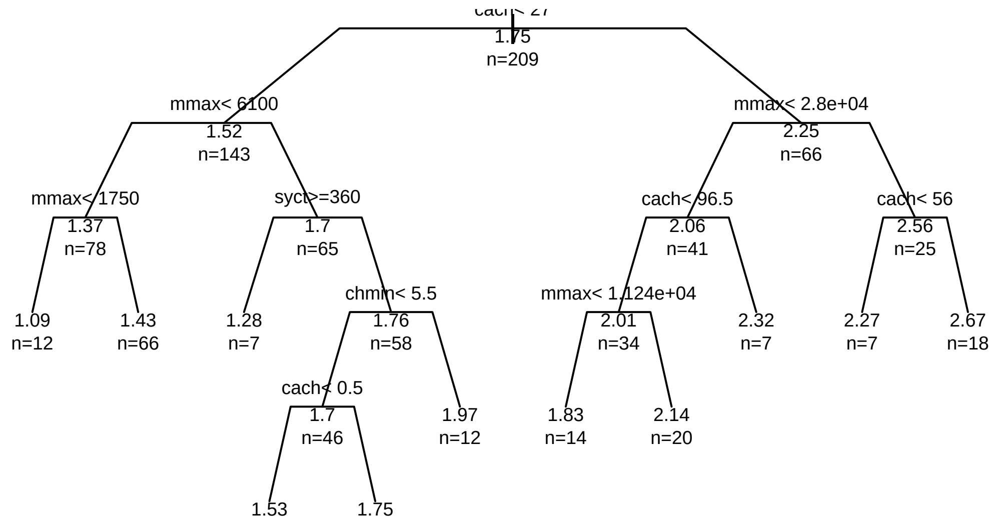
```
##           45) cach>=0.5 35  0.6160 1.75 *
##         23) chmin>=5.5 12  0.5510 1.97 *
##    3) cach>=27 66  7.6400 2.25
##      6) mmax< 2.8e+04 41  2.3400 2.06
##       12) cach< 96.5 34  1.5900 2.01
##         24) mmax< 1.12e+04 14  0.4250 1.83 *
##         25) mmax>=1.12e+04 20  0.3830 2.14 *
##       13) cach>=96.5 7  0.1720 2.32 *
##      7) mmax>=2.8e+04 25  1.5200 2.56
##       14) cach< 56 7  0.0693 2.27 *
##       15) cach>=56 18  0.6540 2.67 *
```

```r
logcpus.rpart.02<-prune(logcpus.rpart.01,cp=0.006)
```

```r
plot(logcpus.rpart.02,branch=0.4,uniform=TRUE)
text(logcpus.rpart.02,digits=3,cex=0.6)
```




```r
plot(logcpus.rpart.02,branch=0.6,compress=TRUE,uniform=TRUE)
text(logcpus.rpart.02,digits=3,all=TRUE,use.n=TRUE,cex=0.6)
```


cach< 27
1.75
n=209
mmax< 6100
1.52
n=143
mmax< 2.8e+04
2.25
n=66
mmax< 1750
1.37
n=78
syct>=360
1.7
n=65
cach< 96.5
2.06
n=41
cach< 56
2.56
n=25
1.09
n=12
1.43
n=66
1.28
n=7
chmin< 5.5
1.76
n=58
mmax< 1.124e+04
2.01
n=34
2.32
n=7
2.27
n=7
2.67
n=18
cach< 0.5
1.7
n=46
1.97
n=12
1.83
n=14
2.14
n=20
1.53
1.75