CART

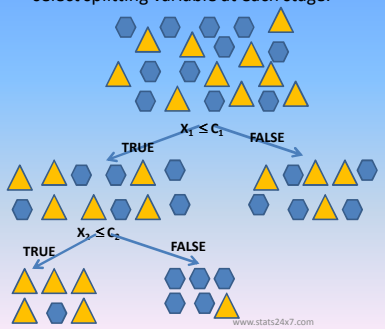
- Classification and Regression Trees

Idea behind CART

When data clustered and messy, fitting a smooth function may not be the thing to do.

We can instead split the data into homogeneous clusters, and then just compute the mean of each cluster (i.e., fit a constant).

Recursive Partitioning is used to split data into homogeneous strata. Parent node is split into 2 children nodes – process repeated for each child node. An ***impurity measure*** is used to select splitting variable at each stage.



Possible Stopping Criteria
- Max #(nodes)
- Min # cases in a node reached

Impurity Measures

Continuous Variable – sample variance
Qualitative Variable : several approaches

Binary Variable:

Variance $= p(1-p)$

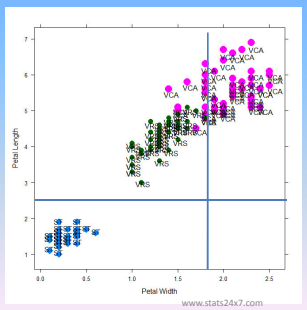
log likelihood $= p\ln(p)(1-p)\ln(1-p)$

Several categories:

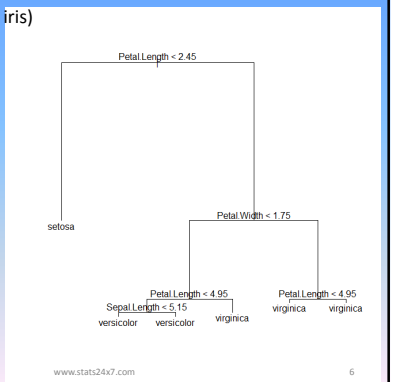
max entropy $= -\sum p_k \ln(1-p_k)$

Gini index $= \sum_{i \neq j} p_i p_j = 1 - \sum_j p_j^2$


```
iris <- read.csv("M:/DataMining/Data/iris.csv", header=TRUE)
attach(iris)
xyplot(Petal.Length~Petal.Width, data=iris, groups=Species)
```




```
# install and load R-package tree

ir.tr <- tree(Species ~., iris)
plot(ir.tr); text(ir.tr)
```



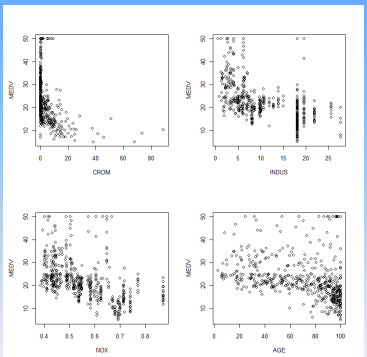
Boston Housing data set is a famous data set in data mining applications.

Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol. 5, 81-102, 1978.

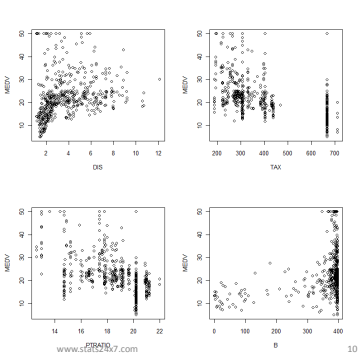
BOSTON HOUSING DATA SET

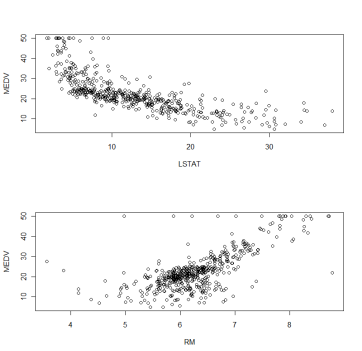
- The data set bostonhousing.csv has data on the following 14 variables:
- 1. CRIM per capita crime rate by town
- 2. ZN proportion of residential land zoned for lots over 25,000 sq.ft.
- 3. INDUS proportion of non-retail business acres per town
- 4. CHAS Charles River **dummy variable** (= 1 if tract bounds river; 0 otherwise)
- 5. NOX nitric 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 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. B 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- 13. LSTAT % lower status of the population
- 14. MEDV Median value of owner-occupied homes in $1000's
-
- Develop a model for MEDV as a function of the other variables.


```
nf <- layout(matrix(c(1,2,3,4),2,2,byrow=TRUE), c(2,2), c(2,2), TRUE)
layout.show(nf)
plot(MEDV~CROM)
plot(MEDV~INDUS)
plot(MEDV~NOX)
plot(MEDV~AGE)
```




```
plot(MEDV~DIS)
plot(MEDV~TAX)
plot(MEDV~PTRATIO)
plot(MEDV~B)
```






```
boston <-
  read.csv("K:/DataMining/Data/bostonhousing.csv")
attach(boston)

library(fields)
stats(boston)
```

yields the descriptives of all variables in the dataset.

	CROM	ZN	INDUS	CHAS	NOX	RM	AGE
N	506	506	506	506	506	506	506
mean	3.613528	11.364	11.137	0.069	0.555	6.285	68.5749
Std.Dev.	8.601544	23.322	6.86	0.254	0.116	0.703	28.14886
min	0.0063	0	0.46	0	0.385	3.561	2.9
Q1	0.08205	0	5.19	0	0.449	5.886	45.025
median	0.25655	0	9.69	0	0.538	6.209	77.5
Q3	3.677075	12.5	18.1	0	0.624	6.624	94.075
max	88.9762	100	27.74	1	0.871	8.78	100

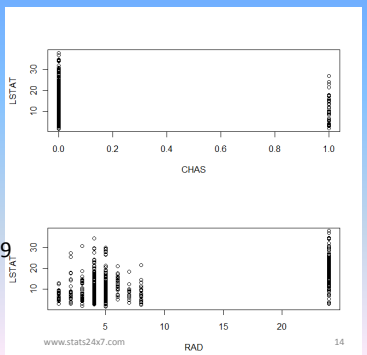
	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
N	506	506	506	506	506	506	506
mean	3.795	9.549	408.237	18.456	356.674	12.653	22.533
Std.Dev.	2.106	8.707	168.537	2.165	91.295	7.141	9.197
min	1.13	1	187	12.6	0.32	1.73	5
Q1	2.1	4	279	17.4	375.378	6.95	17.025
median	3.207	5	330	19.05	391.44	11.36	21.2
Q3	5.188	24	666	20.2	396.225	16.955	25
max	12.127	24	711	22	396.9	37.97	50


```
layout(1:2)
plot(LSTAT~CHAS)
plot(LSTAT~RAD)
```

Notice that

CHAS is an indicator (dummy) variable, and

RAD is a categorical (index) variable with 9 distinct values – so need 8 dummy variables.




```
dim(boston)
# [1] 506  14

bos <- matrix(NA, nrow=506, ncol=14)
bos <- as.matrix(boston)

# RAD has 9 levels, so need 8 dummy variables
DRAD1 <- matrix(0,nrow=506,ncol=1)
DRAD2 <- matrix(0,nrow=506,ncol=1)
DRAD3 <- matrix(0,nrow=506,ncol=1)
DRAD4 <- matrix(0,nrow=506,ncol=1)
DRAD5 <- matrix(0,nrow=506,ncol=1)
DRAD6 <- matrix(0,nrow=506,ncol=1)
DRAD7 <- matrix(0,nrow=506,ncol=1)
DRAD8 <- matrix(0,nrow=506,ncol=1)
```



```
for (i in 1:506)
{
if (RAD[i]==1) DRAD1[i] <- 1
if (RAD[i]==2) DRAD2[i] <- 1
if (RAD[i]==3) DRAD3[i] <- 1
if (RAD[i]==4) DRAD4[i] <- 1
if (RAD[i]==5) DRAD5[i] <- 1
if (RAD[i]==6) DRAD6[i] <- 1
if (RAD[i]==7) DRAD7[i] <- 1
if (RAD[i]==8) DRAD8[i] <- 1
}

lm1 <-
lm(MEDV~CROM+INDUS+CHAS+NOX+RM+AGE+DIS+TAX+PTRATIO+B
+LSTAT+DRAD1+DRAD2+DRAD3+DRAD4+DRAD5+DRAD6+DRAD7+DR
AD8)
summary(lm1)
```

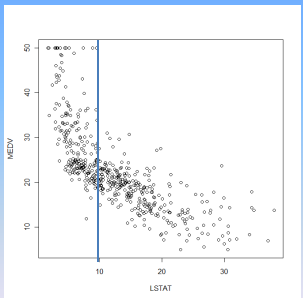


```
Coefficients:
             Estimate Std. Error t value Pr(|t|)
(Intercept)  43.748990   5.972154   7.325 9.97e-13 ***
CROM         -0.100591   0.033085  -3.040 0.002490 **
INDUS        -0.012843   0.063848  -0.201 0.840663
CHAS          2.598506   0.875356   2.969 0.003140 **
NOX         -17.819389   3.951770  -4.509 8.17e-06 ***
RM            3.914301   0.422262   9.270  < 2e-16 ***
AGE          -0.004622   0.013351  -0.346 0.729325
DIS          -1.244827   0.188120  -6.617 9.69e-11 ***
TAX          -0.006146   0.003892  -1.579 0.114921
PTRATIO      -1.165676   0.137567  -8.473 2.86e-16 ***
B             0.009524   0.002699   3.529 0.000457 ***
LSTAT        -0.521490   0.051327 -10.160  < 2e-16 ***
DRAD1        -6.744298   1.804636  -3.737 0.000208 ***
DRAD2        -5.467031   1.890413  -2.892 0.004000 **
DRAD3        -2.844654   1.749869  -1.626 0.104675
DRAD4        -4.505391   1.405118  -3.206 0.001433 **
DRAD5        -4.656084   1.437148  -3.240 0.001278 **
DRAD6        -6.298036   1.538710  -4.093 4.98e-05 ***
DRAD7        -2.661927   1.826728  -1.457 0.145704
DRAD8        -3.161463   1.701878  -1.858 0.063826 .
Multiple R-squared: 0.7421,    Adjusted R-squared: 0.732
F-statistic:  73.6 on 19 and 486 DF,  p-value: < 2.2e-16
```



```
Coefficients:
             Estimate Std. Error t value Pr(|t|)
(Intercept)  39.335339   5.234947   7.514 2.73e-13 ***
CROM         -0.088983   0.030914  -2.878 0.004171 **
CHAS          2.629026   0.861666   3.051 0.002403 **
NOX         -18.727568   3.307407  -5.662 2.54e-08 ***
RM            3.916544   0.408371   9.591  < 2e-16 ***
DIS          -1.204978   0.165035  -7.301 1.15e-12 ***
PTRATIO      -1.159077   0.129557  -8.946  < 2e-16 ***
B             0.009336   0.002656   3.515 0.000480 ***
LSTAT        -0.531909   0.048022 -11.076  < 2e-16 ***
DRAD1        -4.197494   1.170845  -3.585 0.000371 ***
DRAD2        -2.754293   1.094040  -2.518 0.012134 *
DRAD4        -2.242361   0.623589  -3.596 0.000356 ***
DRAD5        -2.310129   0.683590  -3.379 0.000784 ***
DRAD6        -4.184201   1.052826  -3.974 8.12e-05 ***

Residual standard error: 4.754 on 492 degrees of freedom
Multiple R-squared: 0.7397,    Adjusted R-squared: 0.7328
F-statistic: 107.5 on 13 and 492 DF,  p-value: < 2.2e-16
```

plot(LSTAT, MEDV)

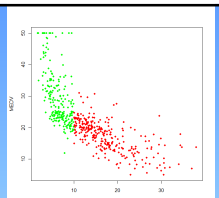


Split data at LSTAT = 10.

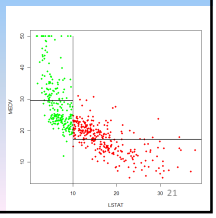
```
a <- 10
b.left <- boston[,"LSTAT"] < a
b.right <- boston[,"LSTAT"] >= a
```

Plot the two split on the same graph:

```
plot(boston[,"LSTAT"], boston[,"MEDV"], xlab="LSTAT",
ylab="MEDV",col=c("red","green")[b.left+1], pch=16, cex=.8)
```

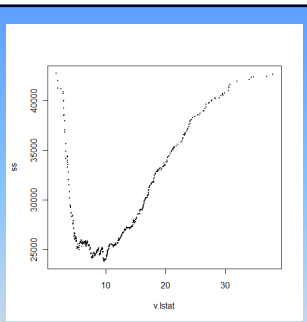



```
# add lines at mean response values for the two halves
lines(c(a,a), c(0,50))
lines(c(0,a),  rep(mean(boston[b.left,"MEDV"]), 2),  lwd=2)
lines(c(a,40), rep(mean(boston[b.right,"MEDV"]), 2), lwd=2)
```



How do we know LSTAT = 10 gives the best 1st split?

```
# sort LSTAT values for plotting
v.lstat <- sort(unique(boston[,"LSTAT"]))

ss <- rep(NA, length(v.lstat))
for(i in 1:(length(v.lstat)))
{
      spl <- v.lstat[i]
      b.left  <- boston[,"LSTAT"] < spl;     b.right <- !b.left
      ss[i] <-
       sum((boston[b.left,"MEDV"]  - mean(boston[b.left,"MEDV"]))^2) +
       sum((boston[b.right,"MEDV"] - mean(boston[b.right,"MEDV"]))^2)
    }
plot(v.lstat, ss, pch=16, cex=.2)
```

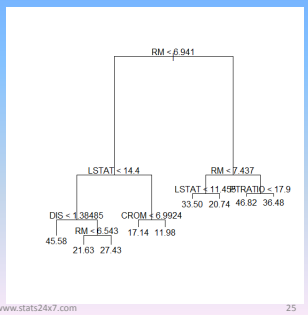



It is easy to see that LSTAT = 10 gives the minimum SUM OF SQUARES (SS) over *ALL POSSIBLE SPLITS.*


```
a <- 10
b.left <- boston[,"LSTAT"] < a
b.right <- boston[,"LSTAT"] = a

ss.10 <- sum((boston[b.left,"MEDV"]  -
mean(boston[b.left,"MEDV"]))^2) +
+      sum((boston[b.right,"MEDV"] -
mean(boston[b.right,"MEDV"]))^2)

 ss.10
[1] 24111.08

 min(ss)
[1] 23820.1
```

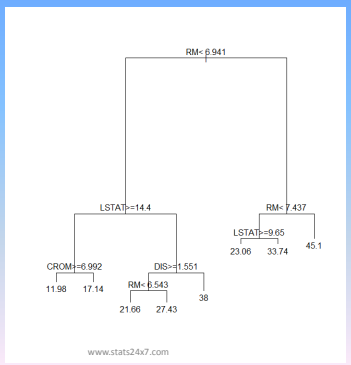
install and load R-package tree

```
boston.tr <- tree(MEDV ~., boston)
plot(boston.tr); text(boston.tr)
```



#Rpart is another R-package for CART, install and load rpart

```
fit <- rpart(MEDV~CROM+ZN+INDUS+CHAS+NOX+RM+AGE+DIS+RAD+TAX+PTRATIO+B+LSTAT,
data=boston)
plot(fit)
text(fit)
```



The party package of R uses a parametric model in recursive partitioning:

1) A parametric model is fitted to the full data.
2) Parameter instability is tested over a set of partitioning variables.
3) The variable with highest instability is used to split data.
4) Procedure is repeated for each child node.

Boston housing data will be used to illustrate the procedure.

```
# Convert CHAS and RAD to factors
boston$CHAS <- factor(boston$CHAS, levels=0:1, labels = c("no", "yes"))
boston$RAD <- factor( boston$RAD, ordered = TRUE)
```



```
# OLS is used for the model fitting, and
# instability of parameter is assesses using Bonferroni
# adjustment
control.par <- mob_control(alpha=.05, bonferroni=TRUE,
minsplit=40, objfun=deviance, verbose=TRUE)

# model MEDV as a function of LSTAT and RM plus all
partitioning variables
mob1 <-
mob(MEDV~LSTAT+RM|CROM+CHAS+NOX+DIS+PTRATIO
+B+RAD, control = control.par, model = linearModel)
```



```
Fluctuation tests of splitting variables:
             CROM         CHAS          NOX          DIS      PTRATIO        B
statistic 7.399234e+01 4.908879e+01 6.321138e+01 1.013319e+02 7.60051e+01 3.339619e+01
p.value   1.006287e-13 3.258859e-08 2.555821e-11 6.655466e-20 3.55881e-14 7.302362e-05
             RAD
statistic 7.685939e+01
p.value   2.288228e-14

Best splitting variable: DIS
Perform split? yes

Node properties:
DIS <= 1.6582; criterion = 1, statistic = 101.332
----------------------------------------
Fluctuation tests of splitting variables:
             CROM         CHAS          NOX          DIS      PTRATIO        B
statistic 8.187771e+01 4.110237e+01 9.560163e+01 3.497670e+01 7.997064e+01
7.067411e+01
p.value   1.685016e-15 1.708528e-06 1.323506e-18 3.388114e-05 4.530194e-15 5.515470e-
13
             RAD
statistic 8.343392e+01
p.value   7.511532e-16
Best splitting variable: NOX
Perform split? yes
```

Node properties:
NOX <= 0.655; criterion = 1, statistic = 95.602

Fluctuation tests of splitting variables:

	CROM	CHAS	NOX	DIS	PTRATIO	B
statistic	3.457337e+01	12.8541418	22.21193255	19.97970227	8.014125e+01	12.5715672
p.value	4.080716e-05	0.4898428	0.01306554	0.03494108	4.104767e-15	0.5283952

	RAD
statistic	21.61332177
p.value	0.01705065

Best splitting variable: PTRATIO
Perform split? yes

Node properties:
PTRATIO <= 19.2; criterion = 1, statistic = 80.141

Fluctuation tests of splitting variables:

	CROM	CHAS	NOX	DIS	PTRATIO	B
statistic	19.9675392	10.8026857	16.1420955	22.06771089	4.134665e+01	11.8472459

	RAD
17.23960984	

	CROM	CHAS	NOX	DIS	PTRATIO	B	RAD
p.value	0.0265156	0.7113743	0.1323357	0.01021371	8.179616e-07	0.5622984	0.08478723

Best splitting variable: PTRATIO
Perform split? yes

Node properties:
PTRATIO <= 14.9; criterion = 1, statistic = 41.347

Fluctuation tests of splitting variables:

	CROM	CHAS	NOX	DIS	PTRATIO	B	RAD
statistic	9.9762064	12.4129558	8.3077281	11.2807251	8.980504	10.1135121	6.6781977
p.value	0.7880782	0.4519443	0.9460855	0.6086947	0.896195	0.7705713	0.9955196

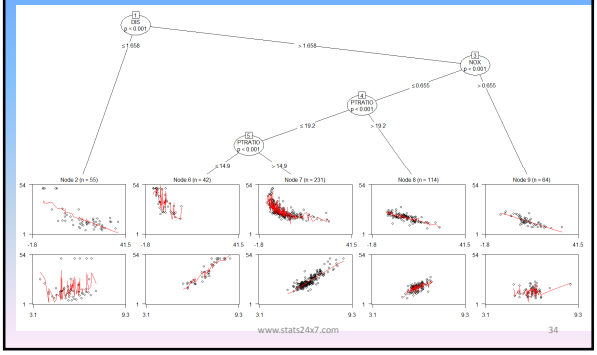
Best splitting variable: CHAS
Perform split? no

Fluctuation tests of splitting variables:

	CROM	CHAS	NOX	DIS	PTRATIO	B	RAD
statistic	5.0424258	1.575127	4.2758081	10.1220794	8.649540	3.922884	9.342154
p.value	0.9951466	1.000000	0.9993386	0.5162606	0.725515	0.999801	0.627116

Best splitting variable: DIS
Perform split? no

# get visual output
plot(mob1)



coef(mob1)

	(Intercept)	LSTAT	RM
2	47.805558	-1.02831030	-1.4659481
6	-32.765950	-0.76153809	10.3638032
7	-33.907963	-0.07031379	9.3655556
8	3.372720	-0.46940157	3.5698254
9	28.379282	-0.72025904	-0.1238229

CART can be used for logistic regression as well.

The R-package mlbench has a benchmark data set – Pima Indians Diabetes data.

```
# read data
data("PimaIndiansDiabetes2", package = "mlbench")
pid <- PimaIndiansDiabetes2
library(fields)
stats(PimaIndiansDiabetes2)
```



```
# prints the 1-st 6 lines of a dataframe
head(pid)
```

	pregnant	glucose	pressure	triceps	insulin	mass	pedigree	age	diabetes
1	6	148	72	35	NA	33.6	0.627	50	pos
2	1	85	66	29	NA	26.6	0.351	31	neg
3	8	183	64	NA	NA	23.3	0.672	32	pos
4	1	89	66	23	94	28.1	0.167	21	neg
5	0	137	40	35	168	43.1	2.288	33	pos
6	5	116	74	NA	NA	25.6	0.201	30	neg


```
library(fields)
stats(pid)
```

	pregnant	glucose	pressure	triceps	insulin	mass	pedigree	age	diabetes
N	768	763	733	541	394	757	768	768	NA
mean	3.845052	121.687	72.405	29.153	155.548	32.457	0.472	33.241	NA
Std.Dev.	3.369578	30.536	12.382	10.477	118.776	6.925	0.331	11.76	NA
min	0	44	24	7	14	18.2	0.078	21	NA
Q1	1	99	64	22	76.25	27.5	0.244	24	NA
median	3	117	72	29	125	32.3	0.373	29	NA
Q3	6	141	80	36	190	36.6	0.626	41	NA
max	17	199	122	99	846	67.1	2.42	81	NA
missing	0	5	35	227	374	11	0	0	NA

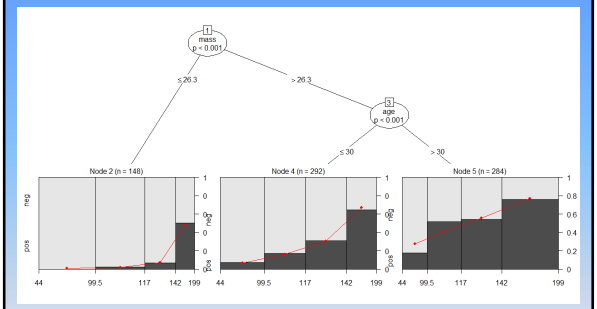
Variables triceps and insulin have a large number of missing (NA) values, and hence should be dropped before predicting P(diabetes = POS).

If the NA values are removed from the data first, and then the variables triceps and insulin removed, we lose too may observations:

```
pid1 <- na.omit(pid) # remove NA
pid2 <- pid1[,-c(4,5)] # then remove triceps and insulin

dim(pid2)
[1] 392   7

pid3 <- pid[,-c(4,5)] # first remove triceps and insulin
pid4 <- na.omit(pid3) # then remove NA

dim(pid3)
[1] 768   7
```



```
log1 <-
mob(diabetes~glucose|pregnant+glucose+pressure+mass+pedigree
+age, data = pid3, model=glinearModel,family=binomial())

plot(log1)  # plots the regression tree (see next slide)
```

Node 2 shows: women with low BMI have lower risk of diabetes, risk goes up with glucose.

Node 4 shows: women with average and high bmi , <= 30 years have higher average risk which increases with glucose.

Node 5 shows: women with average and high bmi , > 30 years have higher average risk which increases slowly with glucose.


```
> coef(log1)
  (Intercept)    glucose
2  -10.999447 0.06456780
4   -6.573067 0.04504490
5   -3.318569 0.02748038

> exp(coef(log1)[,2])
     2      4      5
1.066698 1.046075 1.027861
```

Odds go up by 6.7%, 4.6%, and 2.78% with glucose in the three partitions.

CART - continued

In this lecture, we will go over:

Pruning a tree, Prediction using a tree

Classification trees

In the last lecture, Classification and Regression Tree methodology was introduced. Decision trees have following features:

- Nonparametric, do not require normality
- Can be used for various data types – continuous, categorical (ordinal/nominal/binary)
- Can identify important variables, interactions, outliers.

Three main steps are involved:

Splitting

- select a variable and a split using Gini index(classification) or SS (regression)
- Split data into 2 subsets; process repeated on the two subsets (reason why this is called the *greedy* algorithm)
- Continue until a large tree with nodes containing a small number of similar observations are obtained

Pruning

Uses a cost complexity measure (CP) to trim the tree

Tree Selection

Use X-validation / test data to select a tree. Tree with small X-validation error are preferred, and so are smaller tree with comparable accuracy.

•Pruning is used to avoid the problem of overfitting.

•In backward pruning, a tree is grown until all possible leaf nodes have been reached, and then the tree is pruned.

•Post-pruning has been shown to increase accuracy up to 25%.

Pruning is done using a cost complexity measure:

$R_{CP} = R + CP \times T$

where

R = tree risk (SS_residuals for regression,
% mis-classification for classification)

T = # (terminal nodes)

CP = complexity parameter

Tree Selection is done using 10-fold X-validation:

Split data into 10 random subsets – D_j , j = 1, 2, …, 10

Leave out D_1, compute tree on D_j , j = 2, …, 10, evaluate using D_1

Repeat above by leaving out D_2, D_3, …, D_{10}

Add error across 10 subsets to get X-validation error rate.

R-library rpart

- rpart – fits the tree model
- rpart.control – used to specify parameters
- summary.rpart– outputs summary of fitted model
- snip.rpart – interactive pruning
- prune.rpart – pruning
- plot.rpart and text.rpart – plotting tree

Example 1:

```
# we will use the data cu.summary of library rpart
# to illustrate tree-pruning.
# load library rpart
library(rpart)
car <- cu.summary # data is in this library
```


head(car)

			Price	Country	Reliability	Mileage	Type
Acura	Integra	4	11950	Japan	Muchbetter	NA	Small
Dodge	Colt	4	6851	Japan	<NA>	NA	Small
Dodge	Omni	4	6995	USA	Muchworse	NA	Small
Eagle	Summit	4	8895	USA	better	33	Small
Ford	Escort	4	7402	USA	worse	33	Small
Ford	Festiva	4	6319	Korea	better	37	Small


```
# grow(fit) regression tree
tree <- rpart(Mileage~Price + Country +
Reliability + Type, method="anova",
data=cu.summary)

print(tree)
```


Regression tree:

```
rpart(formula = Mileage ~ Price + Country + Reliability + Type,
    data = cu.summary, method = "anova")
```

Variables actually used in tree construction:

[1] Price Type

Root node error: 1354.6/60 = 22.576

n=60 (57 observations deleted due to missingness)

	CP	nsplit	rel error	xerror	xstd
1	0.622885	0	1.00000	1.03846	0.178102
2	0.132061	1	0.37711	0.59928	0.117844
3	0.025441	2	0.24505	0.41922	0.091326
4	0.011604	3	0.21961	0.39572	0.082687
5	0.010000	4	0.20801	0.42330	0.086660

Explanation of output:
•X-validation splits data into k subsets (rpart default k = 10)

•Each of k subsets is left out 1-by-1, model fitted to remaining data and used to predict the 'left-out' data.

•At the k-th *fold,* subset k = test data, remaining = training
(test data is also referred to as OUT-OF-BAG data)

Data analysis and graphics using R: an example-based approach
By John Hilary Maindonald, John Braun

• In regression modeling,
prediction error = SS_Residuals

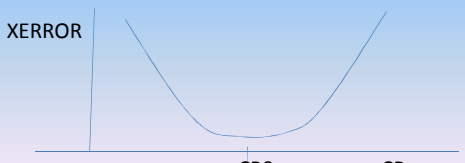
• In classification schemes, we look at %(mis-classification) or % (correct classification).

average error = total error/#(observations)

The Cost-Complexity parameter (CP)

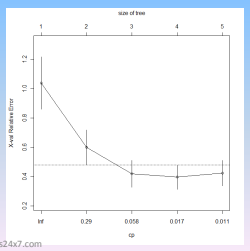
•The # of splits is controlled in by CP that puts a cost on each splits. Splitting is stopped when increase in cost is not worth the reduction in lack-of-fit.

•High CP leads to a small tree; low CP results in a complex tree.

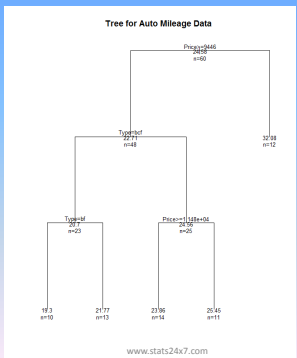


From CP-table on slide 5, CP0 ≈ .012

```
# plot x-validation results
plotcp(tree)

# summary(tree) will give detailed information.
```

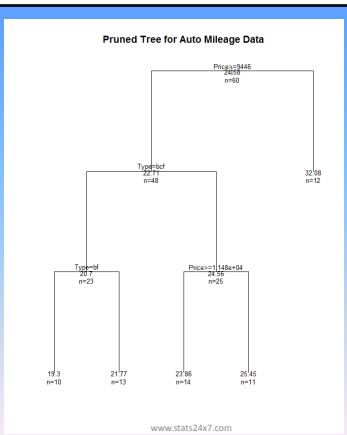



```
# plot the tree
plot(tree, uniform=TRUE, main = "Tree for Auto Mileage Data")
text(tree, use.n=TRUE, all=TRUE, cex=.8)
```



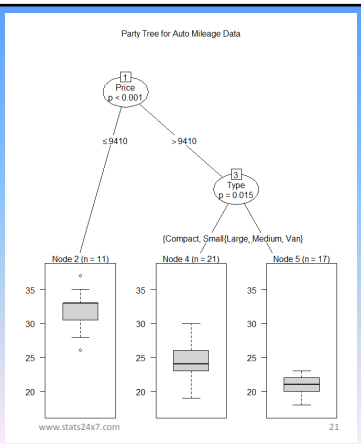

```
# prune the tree using CP = CP0 = 0.01160389)
Pruned_tree <- prune(tree, cp = 0.01160389)
plot(Pruned_tree, uniform=TRUE, main = "Pruned Tree
for Auto Mileage Data")
text(Pruned_tree, use.n=TRUE, all=TRUE, cex=.8)
```

Above produces the pruned tree on the next slide (turns out to be the same tree as the original tree on slide 10)




```
# the library party can also be used to get a
# classification tree

library(party)
tree2 <- ctree(Mileage~Price + Country + Reliability +
Type, data=na.omit(cu.summary))
plot(tree2, main = "Party Tree for Auto Mileage Data")
```


Regression tree for the auto mileage data from package rparty



Example 2: Boston housing data

```
library(MASS)
library(rpart)

boston <-
read.csv("K:/TEACH/DataMining_Fall2009/Data/bostonhousing.csv",
header=TRUE)

boston.tree1 <- rpart(boston$MEDV ~ ., method="anova",
data=boston, control=rpart.control(cp=.0001))

summary(boston.tree1) # part of output shown on next 2 slided
```


	CP	nsplit	relerror	xerror	xstd
1	0.452744	0	1	1.00376	0.082951
2	0.171172	1	0.54726	0.62267	0.056205
3	0.071658	2	0.37608	0.43498	0.046558
4	0.036164	3	0.30443	0.34667	0.041835
5	0.033369	4	0.26826	0.34455	0.042497
6	0.026613	5	0.23489	0.32982	0.042491
7	0.015851	6	0.20828	0.2987	0.040039
8	0.008245	7	0.19243	0.27099	0.035942
9	0.007265	8	0.18418	0.2448	0.031118
10	0.006931	9	0.17692	0.2464	0.031142
11	0.006126	10	0.16999	0.24681	0.031149
12	0.004805	11	0.16386	0.23788	0.030804
13	0.004561	12	0.15905	0.23904	0.029659
14	0.003941	13	0.15449	0.24033	0.02974
15	0.003316	14	0.15055	0.23825	0.029921
16	0.003121	15	0.14724	0.23335	0.02992
17	0.002246	16	0.14412	0.22916	0.029966
18	0.002235	18	0.13962	0.2317	0.029981
19	0.002172	19	0.13739	0.23241	0.029986

	CP	nsplit	relerror	xerror	xstd
20	0.001934	20	0.13522	0.2325	0.029975
21	0.001717	21	0.13328	0.23127	0.029911
22	0.001444	22	0.13157	0.22863	0.028511
23	0.00141	23	0.13012	0.22964	0.028523
24	0.001364	24	0.12871	0.23008	0.028522
25	0.001278	25	0.12735	0.22908	0.028587
26	0.001247	26	0.12607	0.22882	0.028591
27	0.001137	28	0.12358	0.22833	0.028607
28	0.000963	29	0.12244	0.229	0.02866
29	0.000849	30	0.12148	0.22741	0.028659
30	0.00071	31	0.12063	0.22612	0.028719
31	0.000588	32	0.11992	0.22623	0.028709
32	0.000512	33	0.11933	0.22477	0.028705
33	0.000379	34	0.11882	0.22433	0.028709
34	0.000372	35	0.11844	0.22398	0.028709
35	0.000344	36	0.11807	0.22389	0.028713
36	0.000331	37	0.11772	0.22386	0.028713
37	0.000227	39	0.11706	0.22429	0.02871
38	0.000196	40	0.11683	0.22458	0.028778

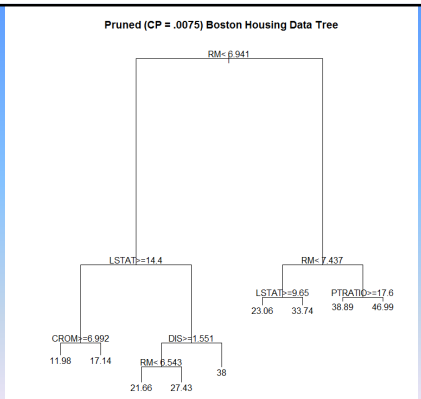
The complexity table above shows that:

Tree with27 nodes gives min xerror

Trees with 9, 10, 11, or 12 nodes are within 1 sd of minimum (.2283 + .0286 = .2569).

Using 1 sd rule, we will use CP inside (.007, .008).


```
pruned.boston 1<- prune(boston.tree1, cp = .0075)

plot(pruned.boston1, main = "Pruned (CP = .0075) Boston Housing Data Tree")

text(pruned.boston1)
```

From the above tree, we see that:

Important split is by RM (# rooms/dwelling)

2nd split on LSTAT (% lower status of population) is also quite important.

Houses with high RM have more average value .

Houses with small RM and low status in population with high crime rate are cheaper.

Prediction Using the Fitted Tree

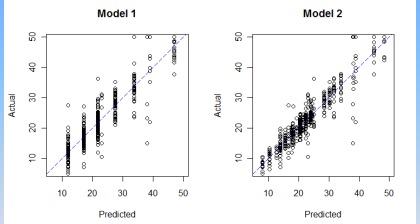
```
# min xerror rule
pruned.boston2 <- prune(boston.tree1, cp = .0011)

# predict using both models
boston.predict1 <- predict(pruned.boston1)
boston.predict2 <- predict(pruned.boston2)

boston.pred.mat <- cbind(boston$MEDV,
boston.predict1, boston.predict2)
boston.pred.mat <- data.frame(boston.pred.mat)
names(boston.pred.mat) <- c("MEDV", "pred1", "pred2")
cor(boston.pred.mat)
```



```
          MEDV      pred1      pred2
MEDV  1.0000000 0.9032262  0.9367825
pred1 0.9032262 1.0000000  0.9641792
pred2 0.9367825 0.9641792  1.0000000
```

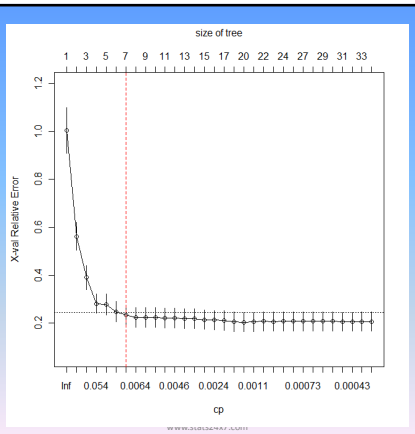
Model 2 (min xerror) has a higher correlation with observed MEDV, but has 27 nodes.

```
# plot observed vs predicted by the two models

par(mfrow=c(1,2), pty="s")
with(boston.pred.mat,{
eqscplot(boston.predict1,MEDV,
xlim=range(boston.predict1,boston.predict2),ylab="Actual",
xlab="Predicted",main="Model 1")
abline(0,1,col="blue",lty=5)
eqscplot(boston.predict2,MEDV,
xlim=range(boston.predict1,boston.predict2),ylab="Actual",
xlab="Predicted",main="Model 2")
abline(0,1,col="blue",lty=5)
par(mfrow=c(1,1))
})
```

Evaluate Tree Model Using Training/Test Data

```
n <- nrow(boston)
# sample 75% of data
boston.samp <- sample(n, round(n*.75))
btrain <- boston[boston.samp,]
btest <- boston[-boston.samp,]

btrain.rp <- rpart(btrain$MEDV ~ ., method="anova",
data=btrain, control=rpart.control(cp=.0001))
summary(btrain.rp)

plotcp(btrain.rp)
abline(v=7,lty=2,col="red")
```


```
prune.btrain <- prune(btrain.rp, cp=.01)
plot(prune.btrain)
text(prune.btrain)
```