

Additive Models, Trees, etc.

Based in part on Chapter 9 of Hastie, Tibshirani, and Friedman

David Madigan

Predictive Modeling

Goal: learn a mapping: $y = f(\mathbf{x}; \theta)$

- Need:
1. A model structure
 2. A score function
 3. An optimization strategy

Categorical $y \in \{c_1, \dots, c_m\}$: classification

Real-valued y : regression

Note: usually assume $\{c_1, \dots, c_m\}$ are mutually exclusive and exhaustive

Generalized Additive Models

- Highly flexible form of predictive modeling for regression and classification:

$$g(E[Y|X]) = \alpha + f_1(X_1) + \dots + f_p(X_p)$$

- g (“link function”) could be the identity or logit or log or whatever
- The f s are smooth functions often fit using natural cubic splines

Basic Backfitting Algorithm

1. Initialize: $\hat{\alpha} = \frac{1}{N} \sum_1^N y_i$, $\hat{f}_j \equiv 0 \quad \forall i, j$.

2. Cycle: $j = 1, 2, \dots, p, \dots, 1, 2, \dots, p, \dots$,

$$\hat{f}_j \leftarrow \mathcal{S}_j \left[\left\{ y_i - \hat{\alpha} - \sum_{k \neq j} \hat{f}_k(x_{ik}) \right\}_1^N \right]$$

until the functions \hat{f}_j change less than some specified threshold

arbitrary smoother - could be natural cubic splines

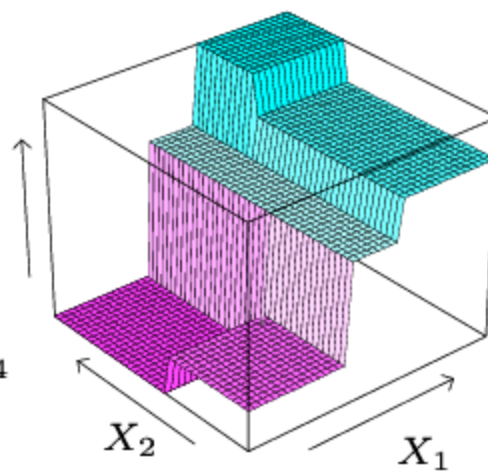
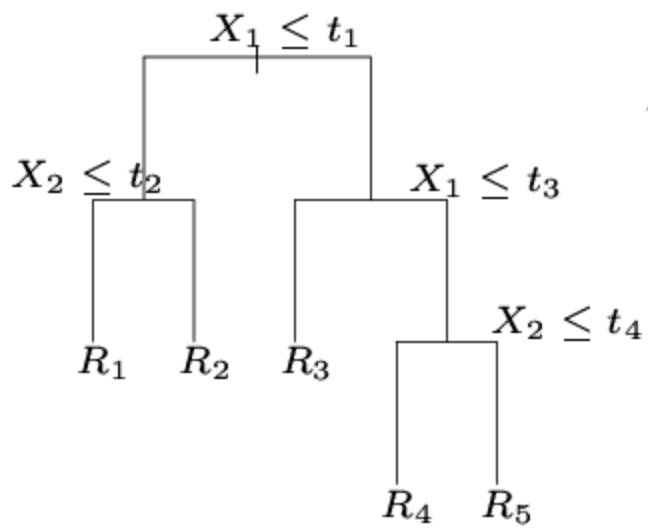
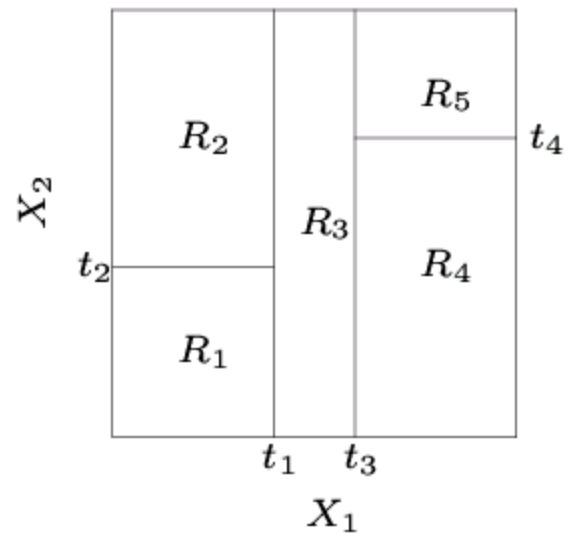
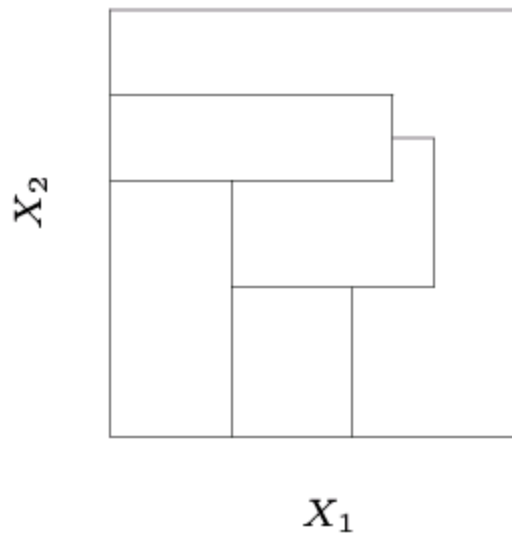
Example using R's gam function

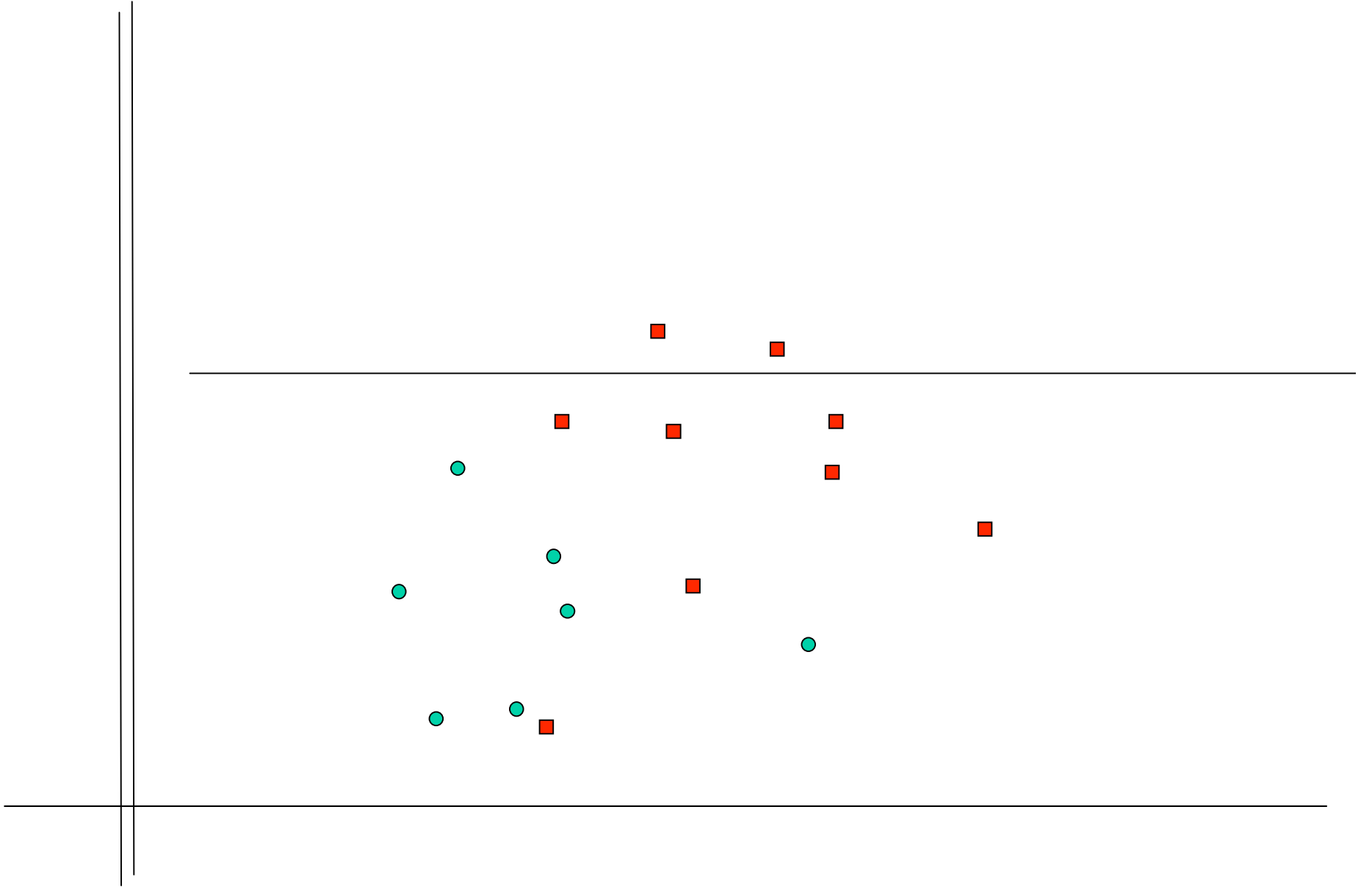
```
library(mgcv)
set.seed(0)
n<-400
x0 <- runif(n, 0, 1)
x1 <- runif(n, 0, 1)
x2 <- runif(n, 0, 1)
x3 <- runif(n, 0, 1)
pi <- asin(1) * 2
f <- 2 * sin(pi * x0)
f <- f + exp(2 * x1) - 3.75887
f <- f + 0.2 * x2^11 * (10 * (1 - x2))^6 + 10 * (10 * x2)^3 * (1 - x2)^10 - 1.396
e <- rnorm(n, 0, 2)
y <- f + e
b<-gam(y~s(x0)+s(x1)+s(x2)+s(x3))
summary(b)
plot(b,pages=1)
```

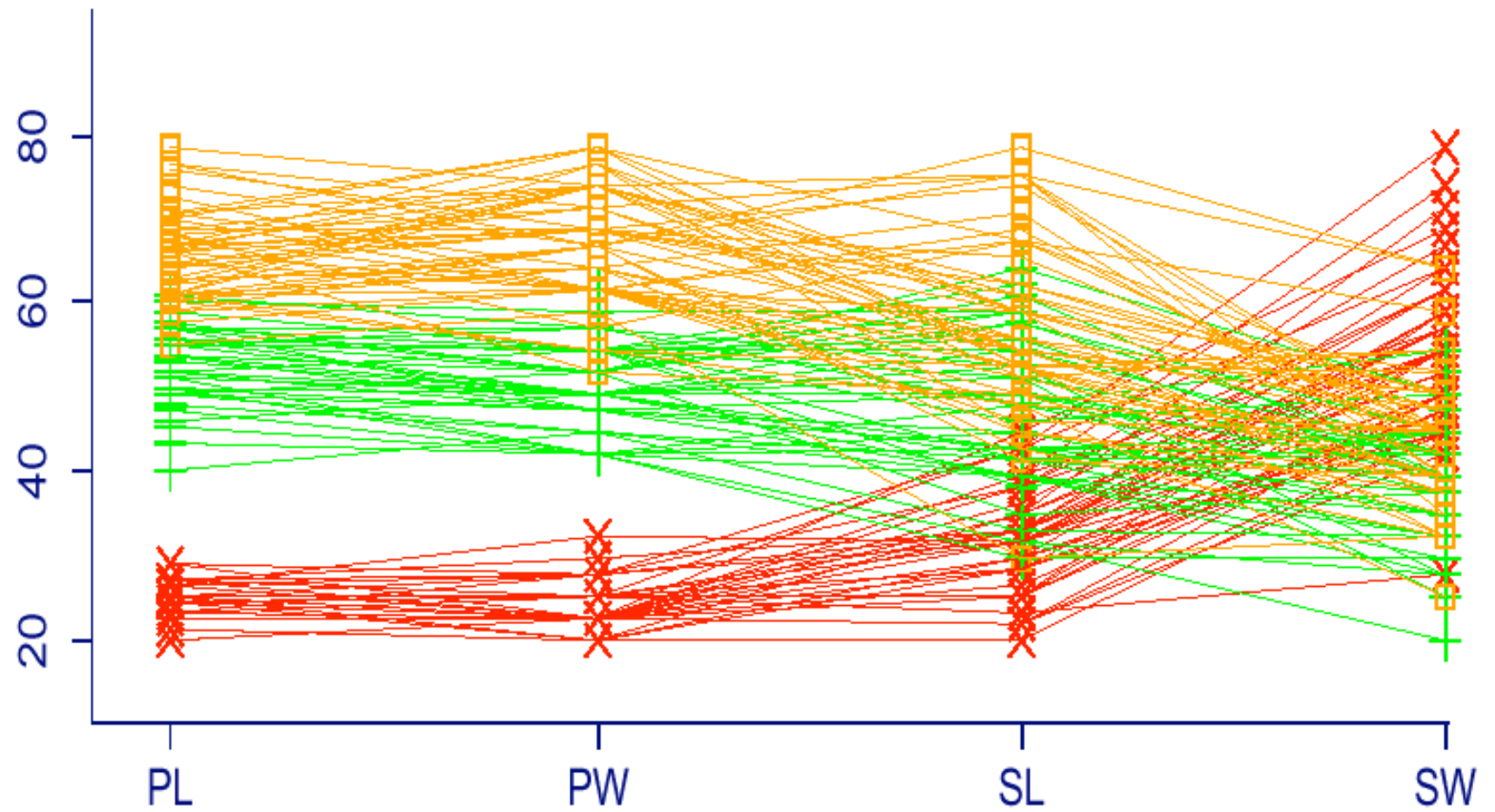
<http://www.math.mcgill.ca/sysdocs/R/library/mgcv/html/gam.html>

Tree Models

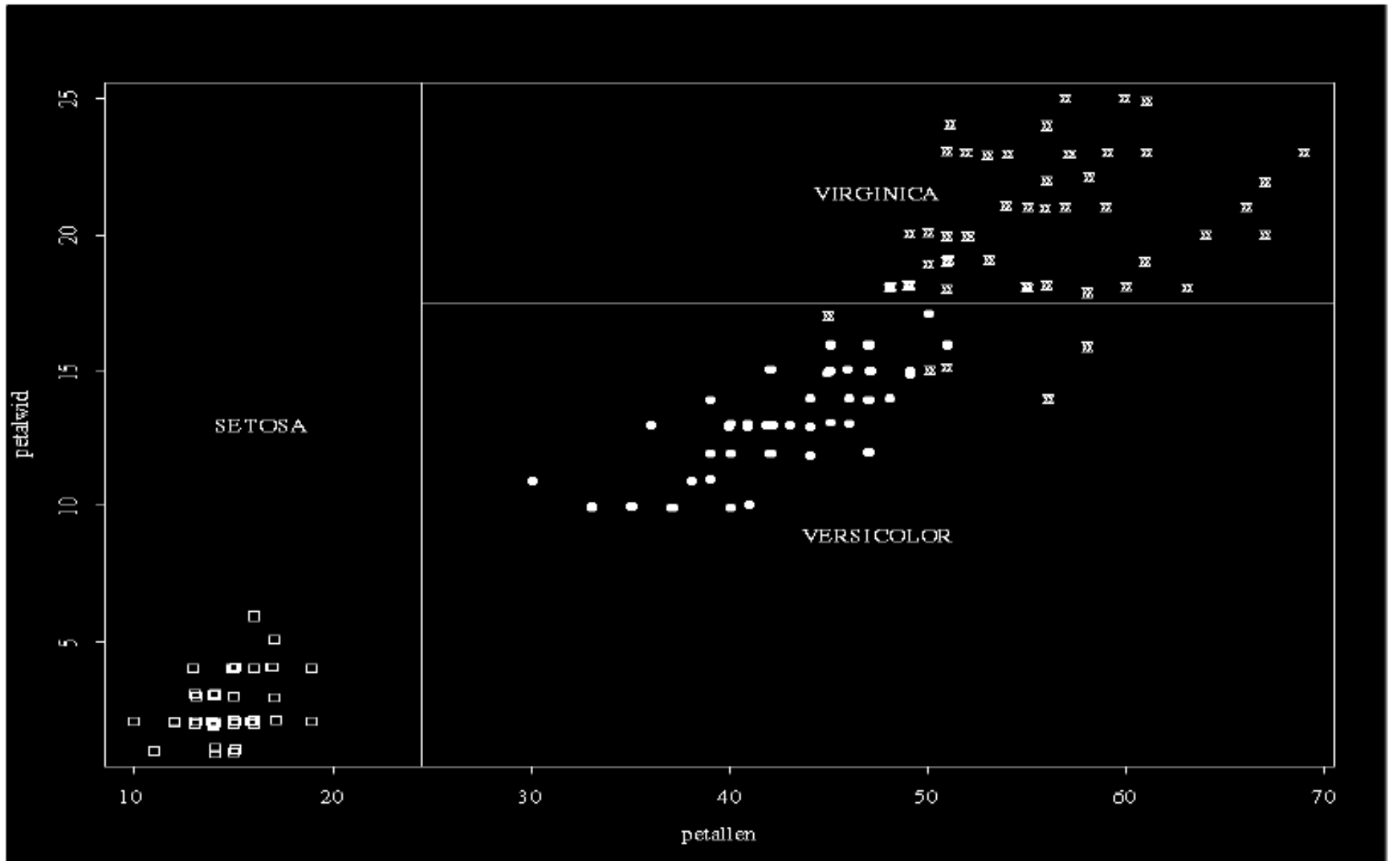
- Easy to understand: recursively divide predictor space into regions where response variable has small variance
- Predicted value is majority class (classification) or average value (regression)
- Can handle mixed data, missing values, etc.
- Usually grow a large tree and prune it back rather than attempt to optimally stop the growing process

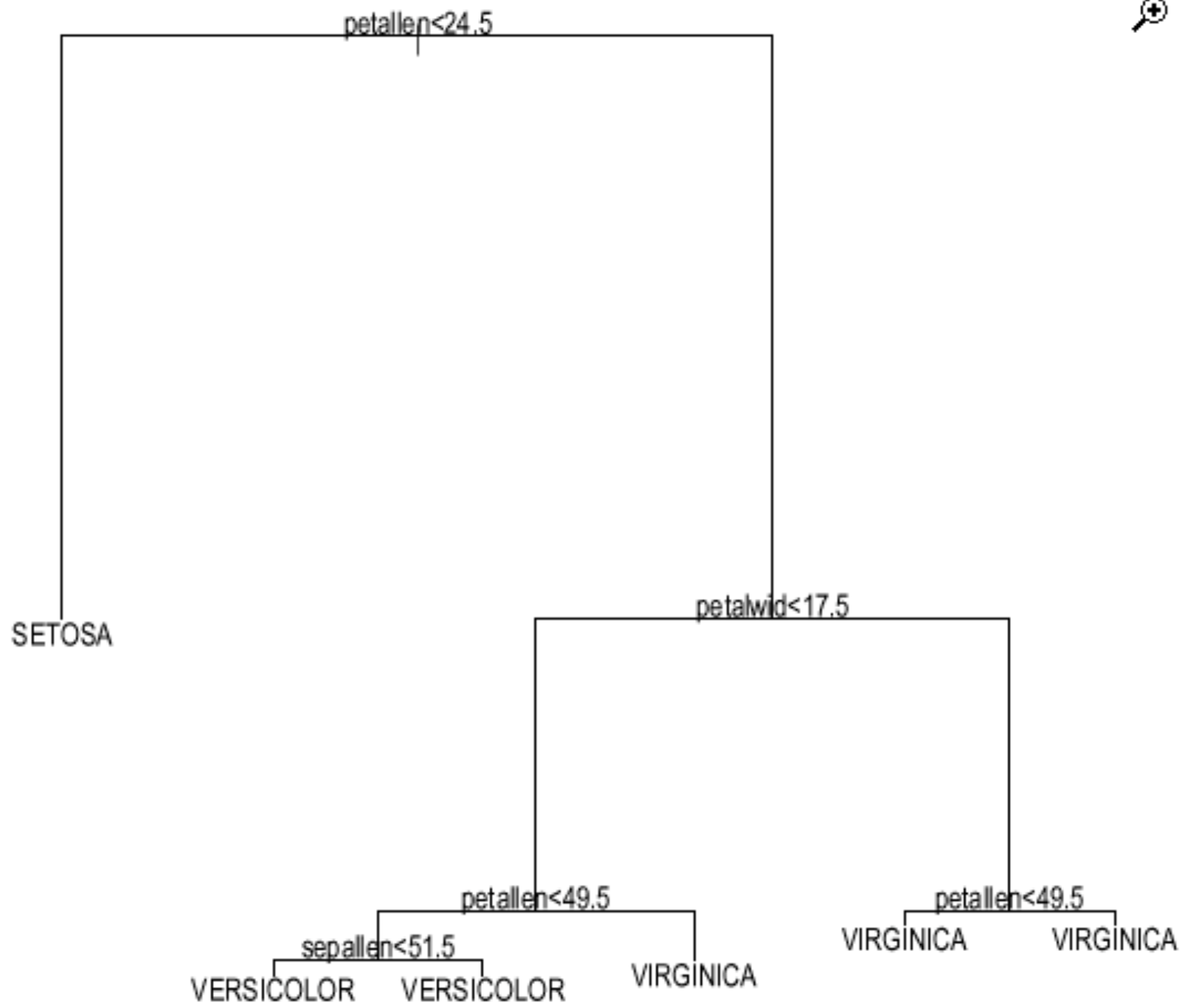


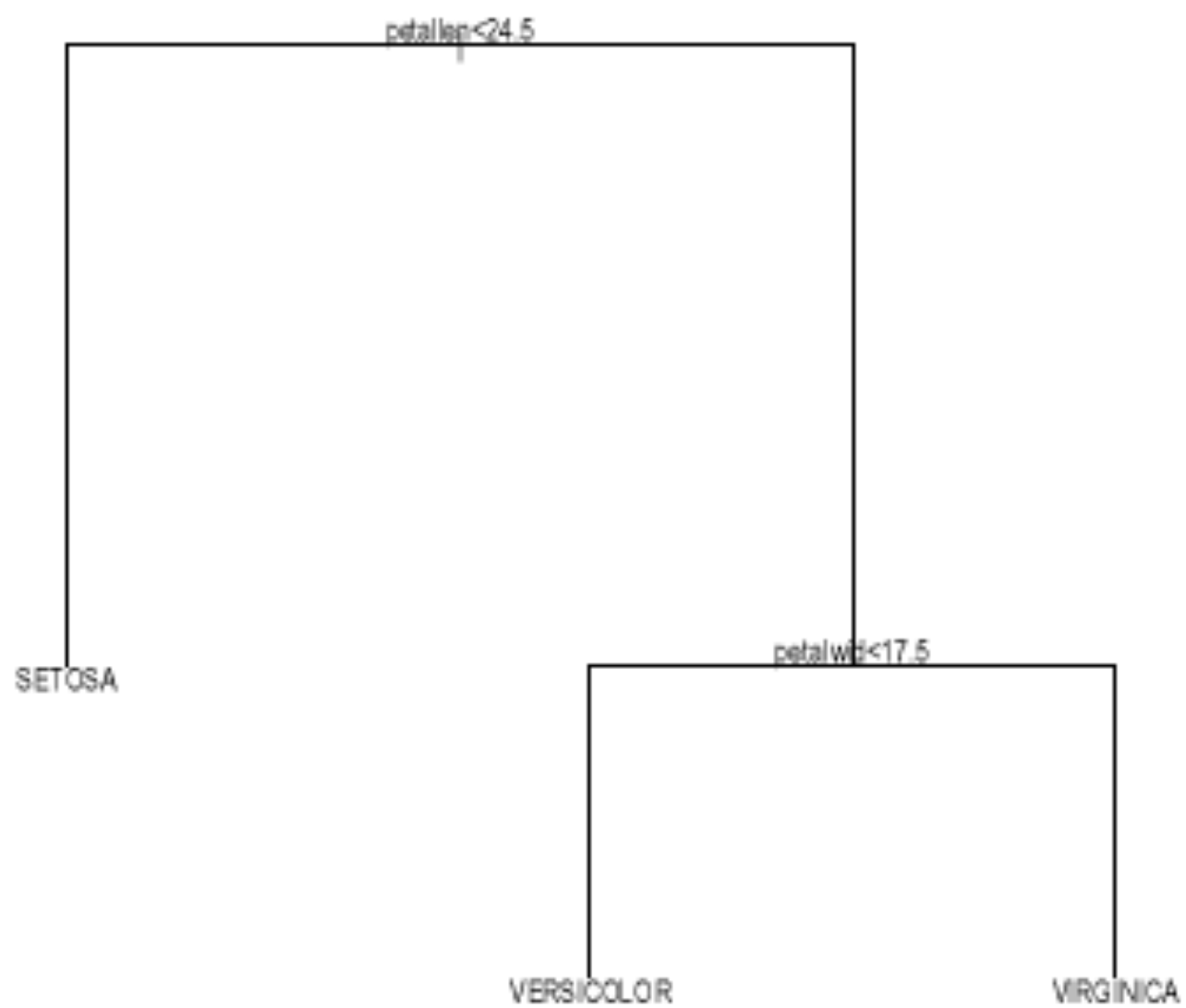




Red x: Setosa, Green +: Versicolor, Orange box: Virginica







Algorithms for Decision Tree Induction

- Basic algorithm (a greedy algorithm)
 - Tree is constructed in a **top-down recursive divide-and-conquer manner**
 - At start, all the training examples are at the root
 - Attributes are categorical (if continuous-valued, they are discretized in advance)
 - Examples are partitioned recursively based on selected attributes
 - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., **information gain**)
- Conditions for stopping partitioning
 - All samples for a given node belong to the same class
 - There are no remaining attributes for further partitioning – **majority voting** is employed for classifying the leaf
 - There are no samples left

Information Gain (ID3/C4.5)

- Select the attribute with the highest information gain
- Assume there are two classes, P and N
 - Let the set of examples S contain p elements of class P and n elements of class N
 - The amount of information, needed to decide if an arbitrary example in S belongs to P or N is defined as

$$I(p, n) = -\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n}$$

e.g. $I(0.5, 0.5) = 1$; $I(0.9, 0.1) = 0.47$; $I(0.99, 0.01) = 0.08$;

Shannon Entropy

- Minimum average message length, in bits, that must be sent to communicate the true value of the random variable to a recipient.
- Intuitively, use fewer bits to encode the more common symbols

Information Gain in Decision Tree Induction

- Assume that using attribute A a set S will be partitioned into sets $\{S_1, S_2, \dots, S_v\}$
 - If S_i contains p_i examples of P and n_i examples of N , the **entropy**, or the expected information needed to classify objects in all subtrees S_i is

$$E(A) = \sum_{i=1}^v \frac{p_i + n_i}{p + n} I(p_i, n_i)$$

- The encoding information that would be gained by branching on A $Gain(A) = I(p, n) - E(A)$

Training Dataset

This follows an example from Quinlan's ID3

age	income	student	credit rating	buys computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
30...40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31...40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31...40	medium	no	excellent	yes
31...40	high	yes	fair	yes
>40	medium	no	excellent	no

Attribute Selection by Information Gain Computation

γ Class P: buys_computer = "yes"

$$E(age) = \frac{5}{14} I(2,3) + \frac{4}{14} I(4,0) + \frac{5}{14} I(3,2) = 0.694$$

γ Class N: buys_computer = "no"

γ $I(p, n) = I(9, 5) = 0.940$

γ Compute the entropy for *age*:

age	p_i	n_i	$I(p_i, n_i)$
≤ 30	2	3	0.971
30...40	4	0	0
> 40	3	2	0.971

Hence

$$Gain(age) = I(p, n) - E(age) = 0.246$$

Similarly

$$Gain(income) = 0.029$$

$$Gain(student) = 0.151$$

$$Gain(credit_rating) = 0.048$$

Gini Index

- If a data set T contains examples from n classes, *gini* index, $gini(T)$ is defined as

$$gini(T) = 1 - \sum_{j=1}^n p_j^2$$

where p_j is the relative frequency of class j in T .

- If a data set T is split into two subsets T_1 and T_2 with sizes N_1 and N_2 respectively, the *gini* index of the split data contains examples from n classes, the *gini* index $gini(T)$ is defined as

$$gini_{split}(T) = \frac{N_1}{N} gini(T_1) + \frac{N_2}{N} gini(T_2)$$

- The attribute provides the smallest $gini_{split}(T)$ is chosen to split the node

Regression Trees

- Typically choose split to minimize within node sum-of-squares

Avoid Overfitting in Classification

- The generated tree may overfit the training data
 - Too many branches, some may reflect anomalies due to noise or outliers
 - Result is in poor accuracy for unseen samples
- Two approaches to avoid overfitting
 - Prepruning: Halt tree construction early—do not split a node if this would result in the goodness measure falling below a threshold
 - Difficult to choose an appropriate threshold
 - Postpruning: Remove branches from a “fully grown” tree—get a sequence of progressively pruned trees
 - Use a set of data different from the training data to decide which is the “best pruned tree”

Approaches to Determine the Final Tree Size

- Separate training (2/3) and testing (1/3) sets
- Use cross validation, e.g., 10-fold cross validation
- Use minimum description length (MDL) principle:
 - halting growth of the tree when the encoding is minimized

Missing Predictor Values

- For categorical predictors, simply create a value “missing”
- For continuous predictors, evaluate split using the complete cases; once a split is chosen find a first “surrogate predictor” that gives the most similar split
- Then find the second best surrogate, etc.
- At prediction time, use the surrogates in order

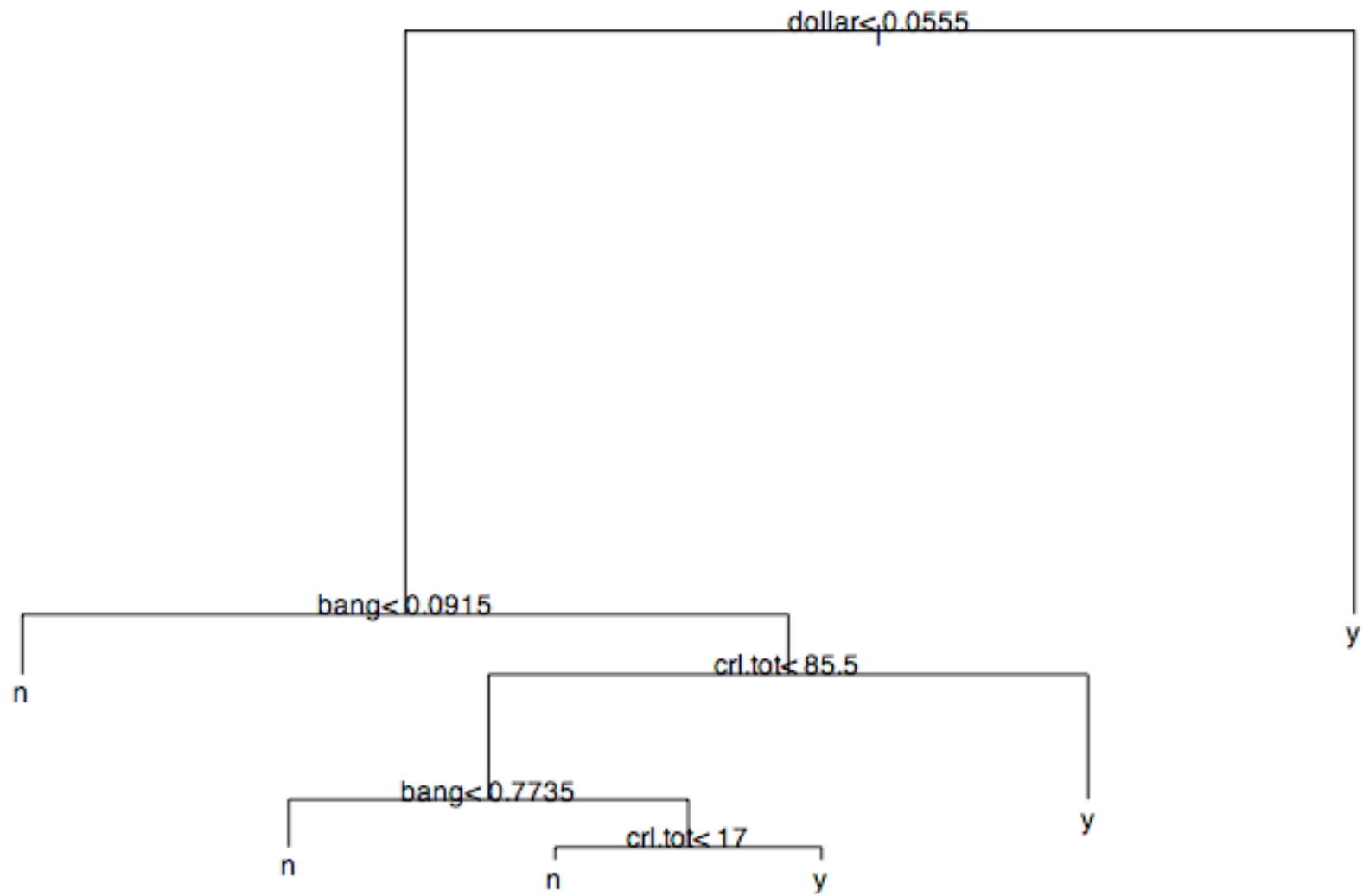
Bagging and Random Forests

- Big trees tend to have high variance and low bias
- Small trees tend to have low variance and high bias
- Is there some way to drive the variance down without increasing bias?
- Bagging can do this to some extent

```
library(DAAG)
spam.sample <- spam7[sample(seq(1,4601), 500, replace=FALSE), ]
boxplot(split(spam.sample$crl.tot, spam.sample$yesno))
```

```
> names(spam.sample)
[1] "crl.tot" "dollar"  "bang"    "money"   "n000"    "make"
[7] "yesno"
```

```
library(rpart)
spam.rpart <- rpart(formula = yesno ~
  crl.tot + dollar + bang + money + n000
  + make, method="class", data=spam7)
plot(spam.rpart)
text(spam.rpart)
printcp(spam.rpart)
```



```
> printcp(spam.rpart)
```

Classification tree:

```
rpart(formula = yesno ~ crl.tot + dollar + bang + money + n000 +  
      make, data = spam7, method = "class")
```

Variables actually used in tree construction:

```
[1] bang    crl.tot  dollar
```

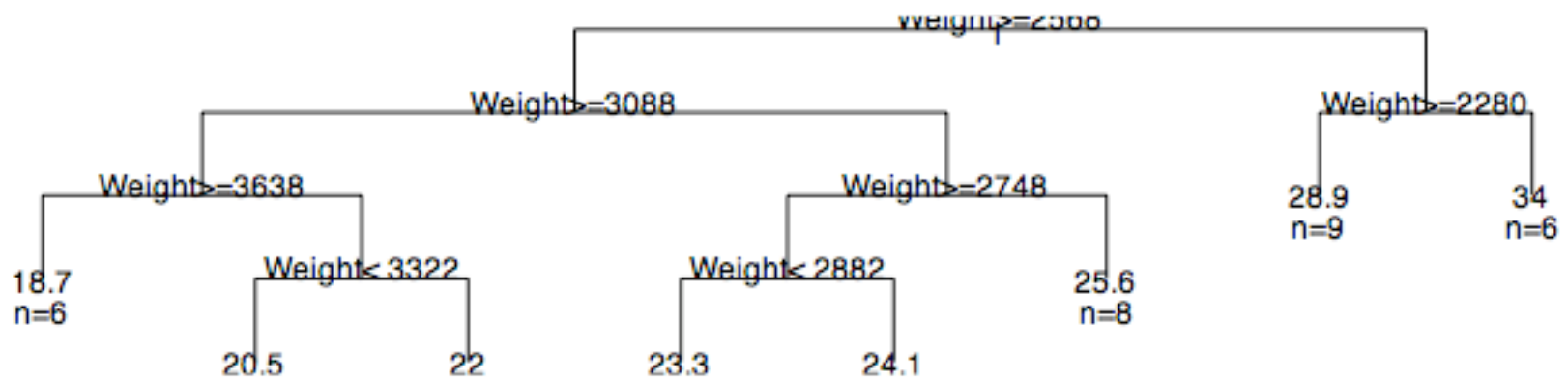
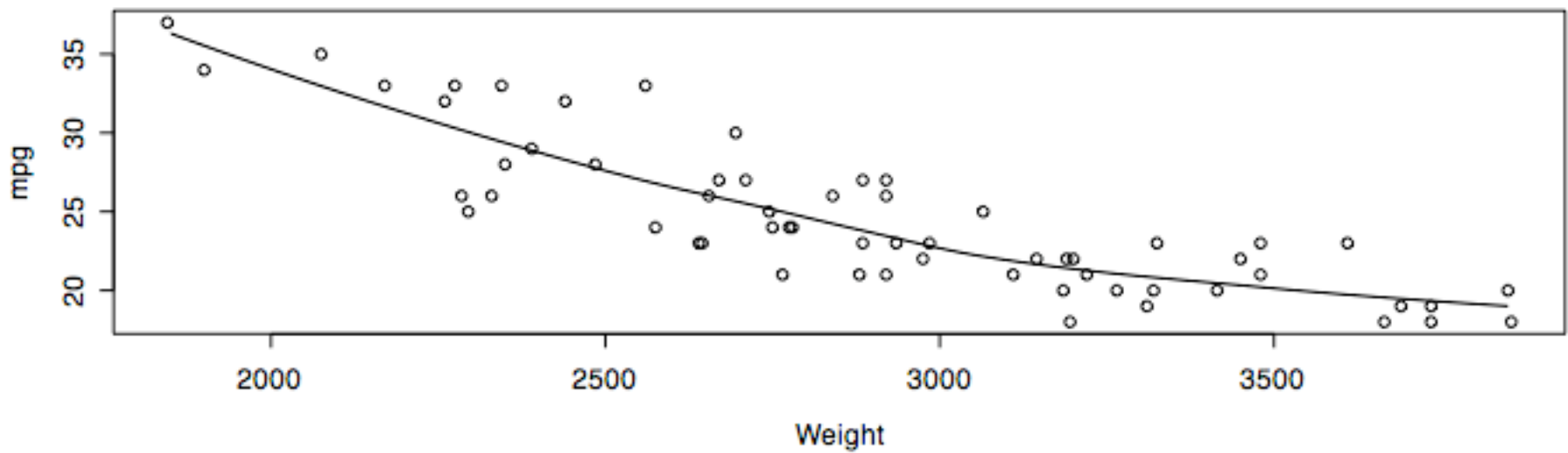
Root node error: 1813/4601 = 0.39404

n= 4601

	CP	nsplit	rel error	xerror	xstd
1	0.476558	0	1.00000	1.00000	0.018282
2	0.075565	1	0.52344	0.55764	0.015492
3	0.011583	3	0.37231	0.38224	0.013382
4	0.010480	4	0.36073	0.38279	0.013390
5	0.010000	5	0.35025	0.38003	0.013350

```
car.lo <- loess(Mileage ~ Weight, car.test.frame)
plot(car.lo, xlab="Weight", ylab="mpg")
lines(seq(1850,3850), predict(car.lo,
data.frame(Weight=seq(1850,3850))))
```

```
car.tree <- rpart(Mileage ~ Weight, data=car.test.frame,
control = list(minsplit = 10, minbucket = 5, cp = 0.0001),
method = "anova")
plot(car.tree, uniform=TRUE)
text(car.tree, digits=3, use.n=TRUE)
```



```
> print(car.tree)
```

```
n= 60
```

```
node), split, n, deviance, yval
```

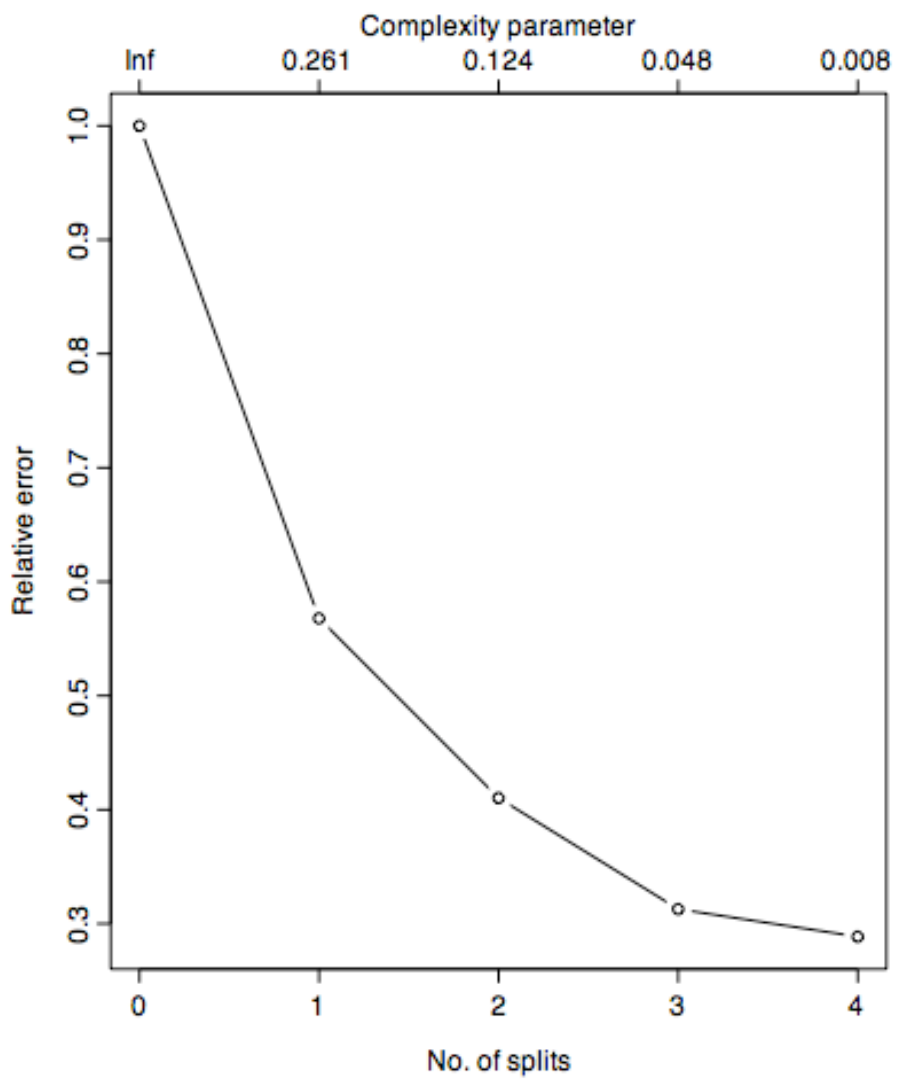
```
* denotes terminal node
```

```
1) root 60 1354.583000 24.58333
  2) Weight>=2567.5 45 361.200000 22.46667
    4) Weight>=3087.5 22 61.318180 20.40909
      8) Weight>=3637.5 6 3.333333 18.66667 *
      9) Weight< 3637.5 16 32.937500 21.06250
        18) Weight< 3322.5 10 16.500000 20.50000 *
        19) Weight>=3322.5 6 8.000000 22.00000 *
    5) Weight< 3087.5 23 117.652200 24.43478
      10) Weight>=2747.5 15 60.400000 23.80000
        20) Weight< 2882.5 6 19.333330 23.33333 *
        21) Weight>=2882.5 9 38.888890 24.11111 *
      11) Weight< 2747.5 8 39.875000 25.62500 *
  3) Weight< 2567.5 15 186.933300 30.93333
    6) Weight>=2280 9 76.888890 28.88889 *
    7) Weight< 2280 6 16.000000 34.00000 *
```

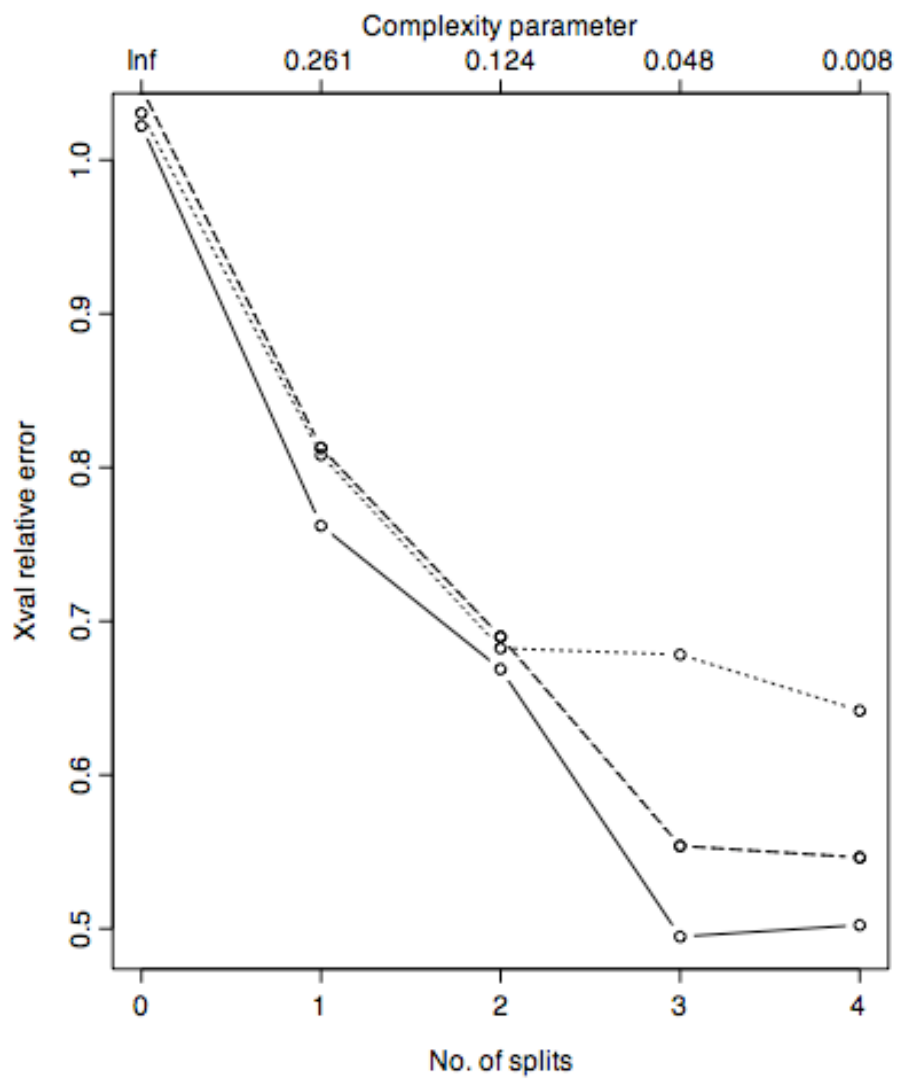
cp is a proxy for the number of splits

cp should be set small enough that the cross-validated error rate achieves a minimum

```
"g11.8" <-  
function(device="")  
{  
  if(device!="")gfile(idth=4.5, height=2.5, device=device)  
  titl <- paste("R plot designed to assist in choosing tree size for the",  
               "\ncar weight & mileage data.")  
  oldpar <- par(mar = c(4.1, 4.1, 3.1, 2.6), mgp = c(2.75, 0.75, 0))  
  on.exit(par(oldpar))  
  car.rpart1 <- rpart(Price~., data=car.test.frame, cp=0.0025)  
  car.rpart2 <- rpart(Price~., data=car.test.frame, cp=0.0025)  
  car.rpart3 <- rpart(Price~., data=car.test.frame, cp=0.0025)  
  pr1 <- printcp(car.rpart1)  
  pr2 <- printcp(car.rpart2)  
  pr3 <- printcp(car.rpart3)  
  cp <- pr1[,"CP"]  
  cp0 <- sqrt(cp*c(Inf,cp[-length(cp)]))  
  nsplit <- pr1[,"nsplit"]  
  plot(nsplit,pr1[,"rel error"], xlab="No. of splits",  
       ylab="Relative error", type="b")  
  axis(3,at=nsplit, labels=paste(round(cp0,3)))  
  mtext(side=3, line=2, "Complexity parameter")  
  mtext(side =3, line=2.5,"A", adj=-0.2)  
  plot(nsplit,pr1[,"xerror"], xlab="No. of splits",  
       ylab="Xval relative error", type="b")  
  lines(nsplit,pr2[,"xerror"],lty=2, type="b")  
  lines(nsplit,pr2[,"xerror"],lty=3, type="b")  
  lines(nsplit,pr3[,"xerror"],lty=3, type="b")  
  axis(3,at=nsplit,labels=paste(round(cp0, 3)))  
  mtext(side=3,line=2, "Complexity parameter")  
  mtext(side =3, line=2.5,"B", adj=-0.2)  
  cat(titl, "\n")  
  if(device!="")dev.off()  
  invisible()  
}
```



D

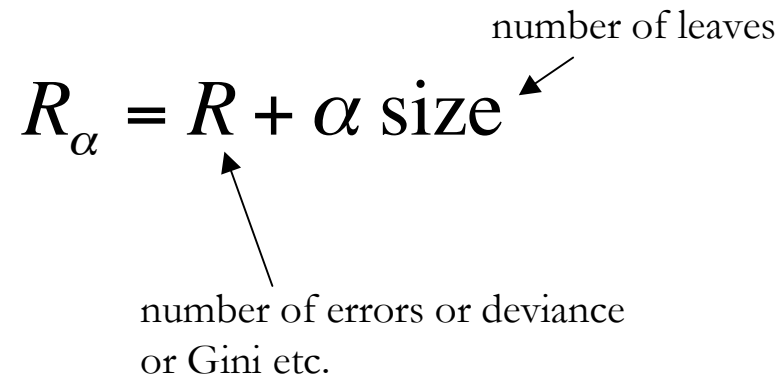


rpart grows a large tree and then prunes back to minimize the cost-complexity measure:

$$R_{\alpha} = R + \alpha \text{ size}$$

number of leaves

number of errors or deviance
or Gini etc.

The diagram shows the equation $R_{\alpha} = R + \alpha \text{ size}$. An arrow points from the text "number of leaves" to the term "size". Another arrow points from the text "number of errors or deviance or Gini etc." to the term "R".

$\alpha=0$ grows the largest tree possible!

cp is the parameter α divided by R for the root tree

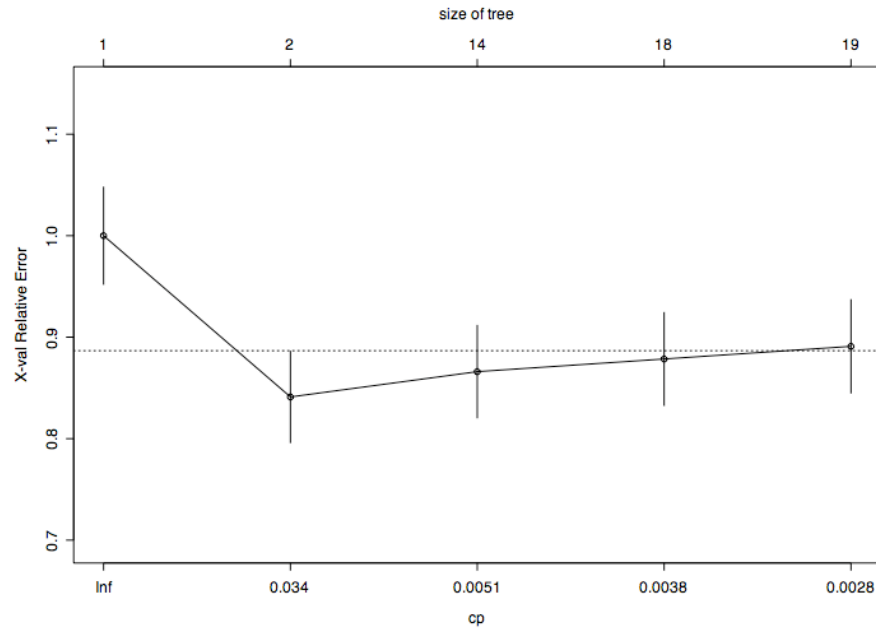
```
> library(DAAG)
```

```
> summary(mifem)
```

outcome	age	yronset	premi	smstat
live:974	Min. :35.00	Min. :85.00	y :311	c :390
dead:321	1st Qu.:57.00	1st Qu.:87.00	n :928	x :280
	Median :63.00	Median :89.00	nk: 56	n :522
	Mean :60.92	Mean :88.79		nk:103
	3rd Qu.:66.00	3rd Qu.:91.00		
	Max. :69.00	Max. :93.00		
diabetes	highbp	hichol	angina	stroke
y :248	y :813	y :452	y :472	y : 153
n :978	n :406	n :655	n :724	n :1063
nk: 69	nk: 76	nk:188	nk: 99	nk: 79

```
mifem.rpart <- rpart(outcome ~ ., method="class",
data=mifem, cp=0.0025)
```

```
plotcp(mifem.rpart)
printcp(mifem.rpart)
```



Root node error: 321/1295 = 0.24788

n= 1295

	CP	nsplit	rel error	xerror	xstd
1	0.2024922	0	1.00000	1.00000	0.048405
2	0.0056075	1	0.79751	0.84112	0.045541
3	0.0046729	13	0.71651	0.86604	0.046030
4	0.0031153	17	0.69782	0.87850	0.046269
5	0.0025000	18	0.69470	0.89097	0.046504

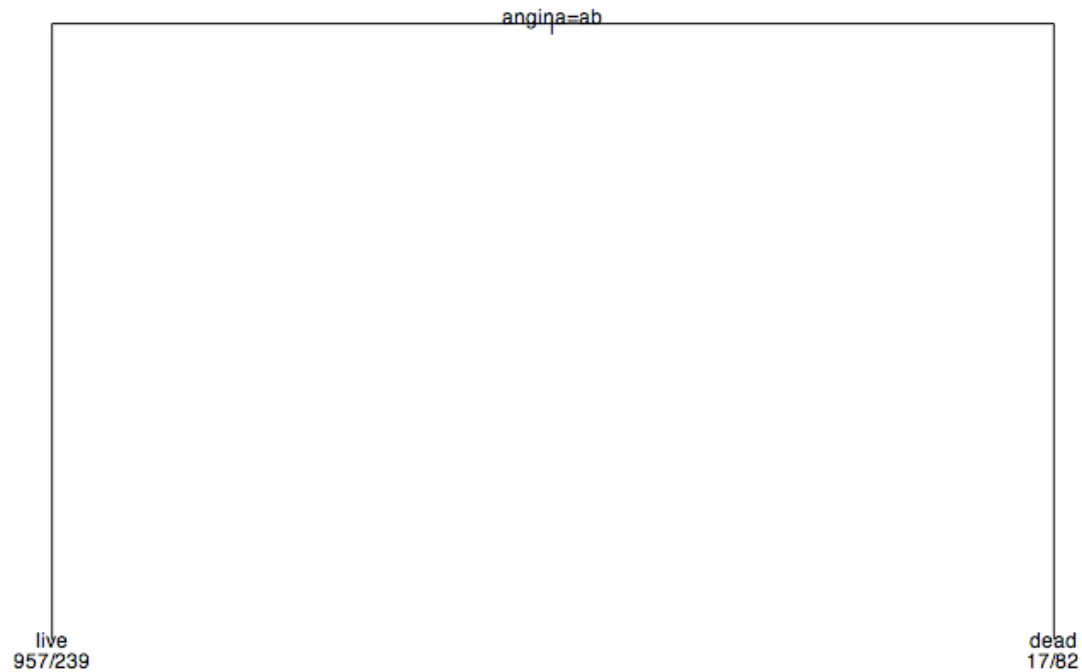
```
mifemb.rpart <- prune(mifem.rpart, cp=0.03)
```

```
plot(mifemb.rpart)
```

```
par(xpd=TRUE)
```

```
text(mifemb.rpart, use.n=T, digits=3)
```

```
par(xpd=TRUE)
```



```
spam7a.rpart <- rpart(formula=yesno ~ crl.tot + dollar + bang + money + n000 + make, method="class", data=spam7, cp=0.001)
```

```
> printcp(spam7a.rpart)
```

```
Classification tree:
```

```
rpart(formula = yesno ~ crl.tot + dollar + bang + money + n000 + make, data = spam7, method = "class", cp = 0.001)
```

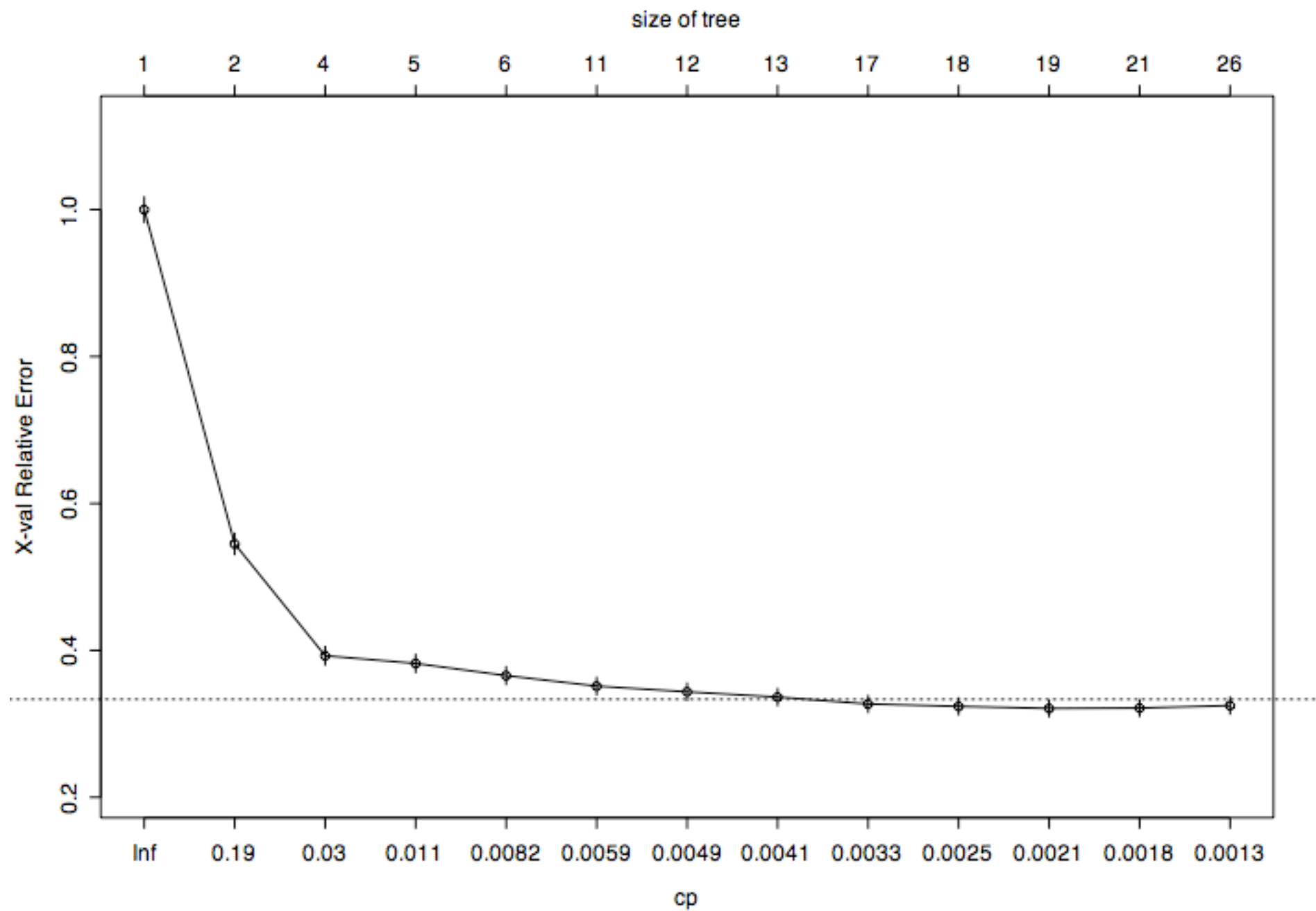
```
Variables actually used in tree construction:
```

```
[1] bang    crl.tot  dollar  money   n000
```

```
Root node error: 1813/4601 = 0.39404
```

```
n= 4601
```

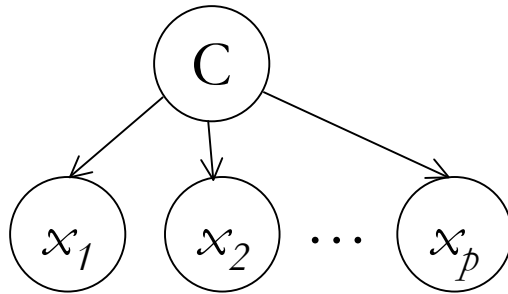
	CP	nsplit	rel error	xerror	xstd
1	0.4765582	0	1.00000	1.00000	0.018282
2	0.0755654	1	0.52344	0.54495	0.015363
3	0.0115830	3	0.37231	0.39272	0.013531
4	0.0104799	4	0.36073	0.38224	0.013382
5	0.0063431	5	0.35025	0.36569	0.013139
6	0.0055157	10	0.31660	0.35135	0.012921
7	0.0044126	11	0.31109	0.34363	0.012801
8	0.0038610	12	0.30667	0.33646	0.012688
9	0.0027579	16	0.29123	0.32708	0.012536
10	0.0022063	17	0.28847	0.32377	0.012482
11	0.0019305	18	0.28627	0.32101	0.012436
12	0.0016547	20	0.28240	0.32157	0.012446
13	0.0010000	25	0.27413	0.32488	0.012500



Naïve Bayes Classification

Recall: $p(c_k | \mathbf{x}) \propto p(\mathbf{x} | c_k)p(c_k)$

Now suppose:



Then: $p(c_k | \mathbf{x}) \propto p(c_k) \prod_{j=1}^p p(x_j | c_k)$

Equivalently:

$$\log \frac{p(c_k | \mathbf{x})}{p(c_{-k} | \mathbf{x})} = \log \frac{p(c_k)}{p(c_{-k})} + \sum \frac{p(x_j | c_k)}{p(x_j | c_{-k})}$$

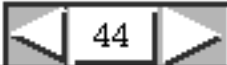









“weights of evidence”



Evidence Balance Sheet

Evidence Balance Sheet [Healthy? = YES]



Indicant	State	WOE 	Target Probability
Initial			 0.54
Age	50--60		 0.51
Rest-Bp	High		 0.51
Sex	Female		 0.66
Chest-Pain	Asymptoma		 0.41

Naïve Bayes (cont.)

- Despite the crude conditional independence assumption, works well in practice (see Friedman, 1997 for a partial explanation)
- Can be further enhanced with boosting, bagging, model averaging, etc.
- Can relax the conditional independence assumptions in myriad ways (“Bayesian networks”)

Patient Rule Induction (PRIM)

- Looks for regions of predictor space where the response variable has a high average value
- Iterative procedure. Starts with a region including all points. At each step, PRIM removes a slice on one dimension
- If the slice size α is small, this produces a very patient rule induction algorithm

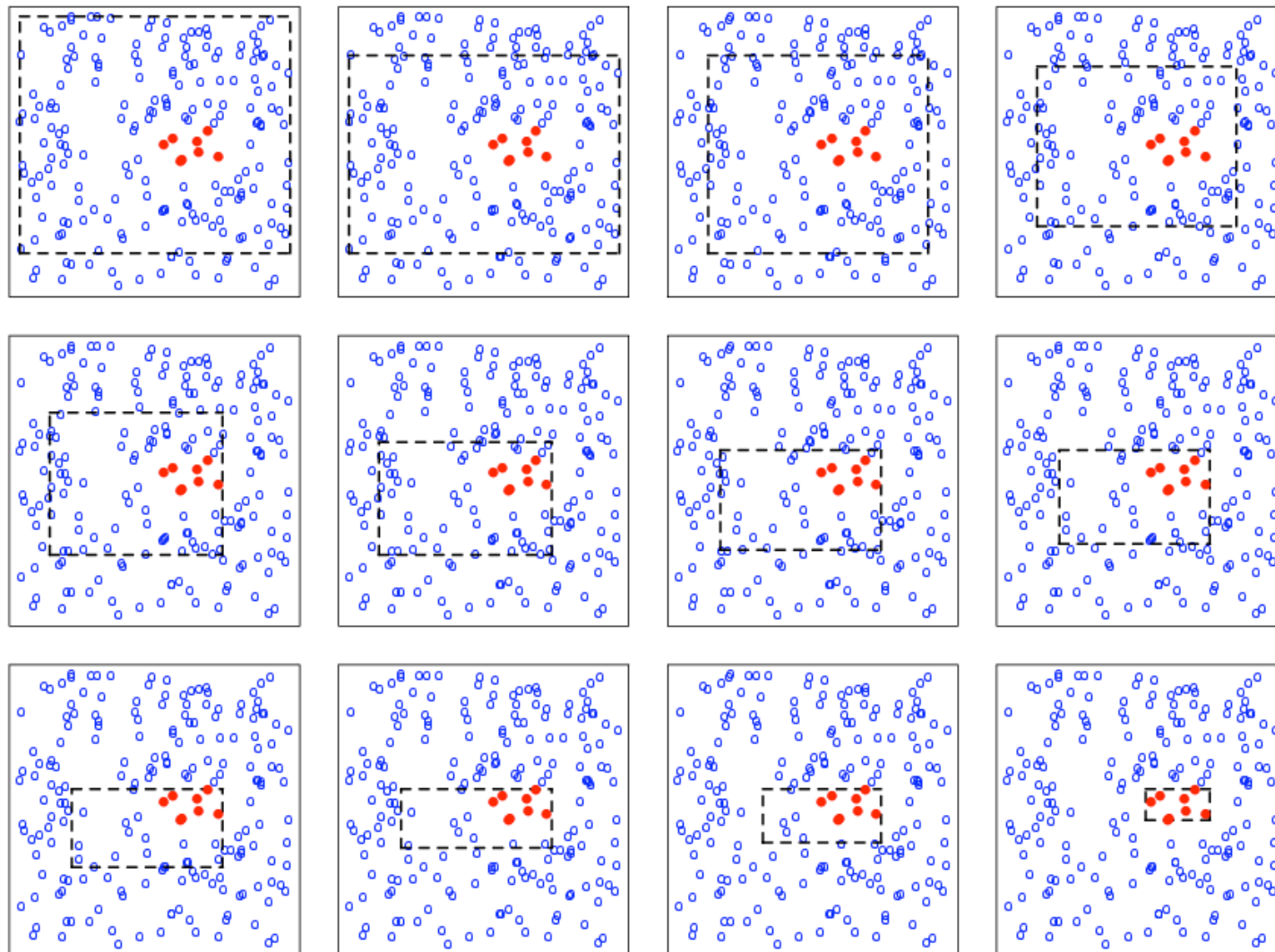


Figure 9.7: *Illustration of PRIM algorithm. There are two classes, indicated by the blue (class 0) and red (class 1) points. The procedure starts with a rectan-*

PRIM Algorithm

1. Start with all of the training data, and a maximal box containing all of the data
2. Consider shrinking the box by compressing along one face, so as to peel off the proportion α of observations having either the highest values of a predictor X_j or the lowest. Choose the peeling that produces the highest response mean in the remaining box
3. Repeat step 2 until some minimal number of observations remain in the box
4. Expand the box along any face so long as the resulting box mean increases
5. Use cross-validation to choose a box from the sequence of boxes constructed above. Call the box B1
6. Remove the data in B1 from the dataset and repeat steps 2-5.