Session 13

Tree models, continued

Classification: more on cars

```
cars93 <- subset(Cars93, select = -c(Manufacturer,
  Model, Rear.seat.room, Luggage.room, Make))
print(names(cars93), quote = FALSE)
```

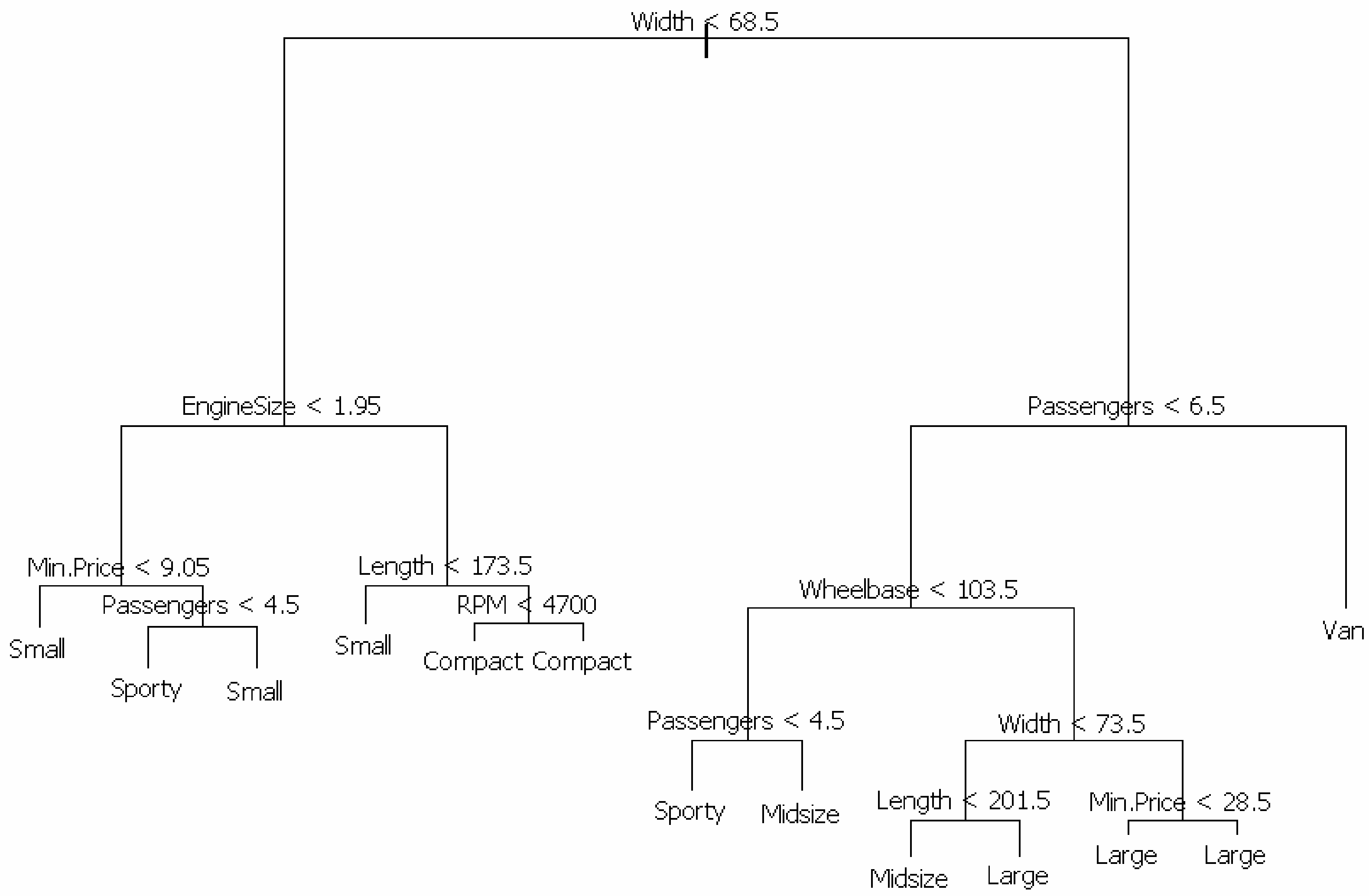
```
 [1] Type               Min.Price
 [3] Price              Max.Price
 [5] MPG.city           MPG.highway
 [7] AirBags            DriveTrain
 [9] Cylinders          EngineSize
[11] Horsepower         RPM
[13] Rev.per.mile       Man.trans.avail
[15] Fuel.tank.capacity Passengers
[17] Length             Wheelbase
[19] Width              Turn.circle
[21] Weight             Origin
```

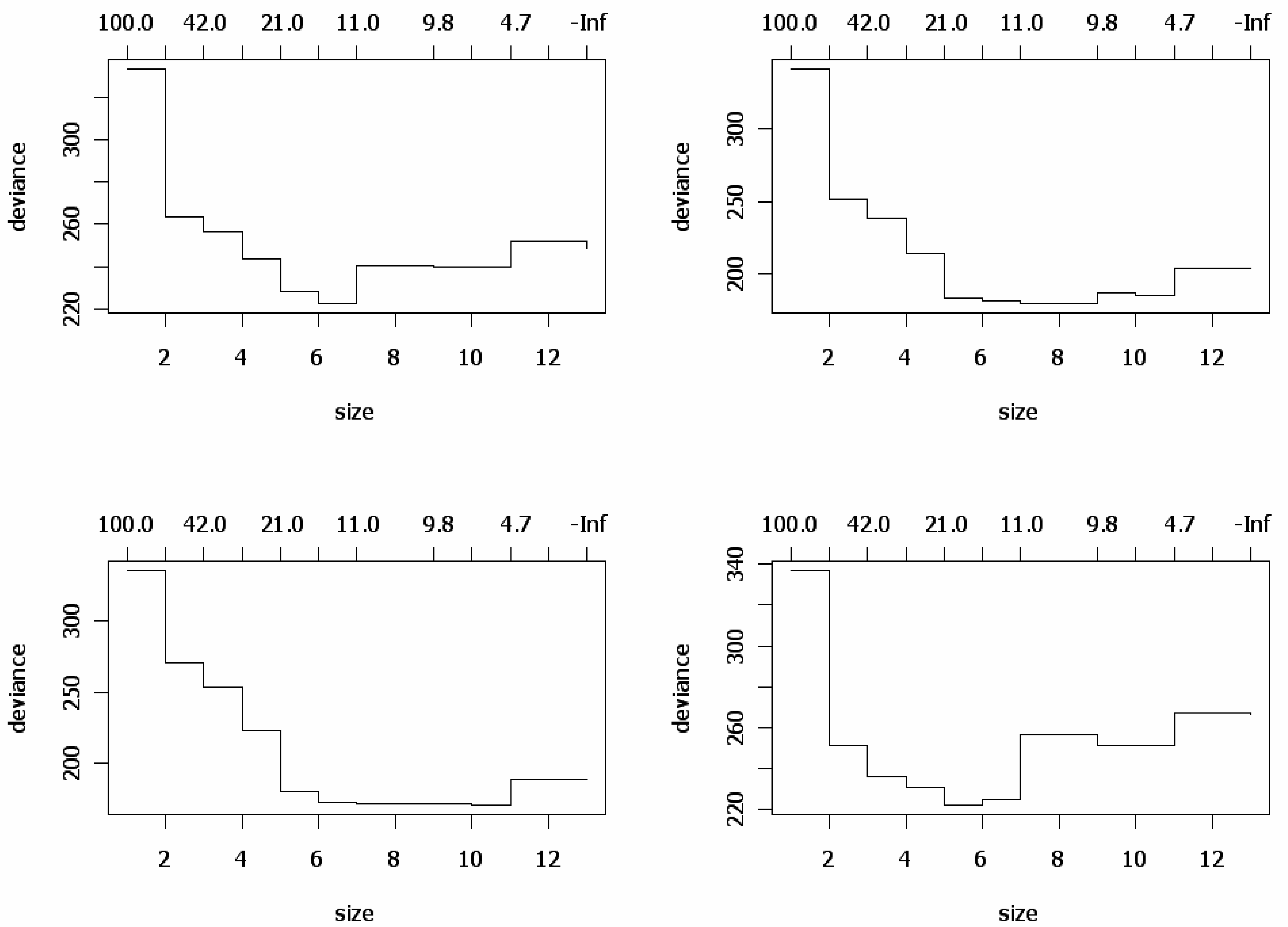
Project

- We want to build a tree classifier for the Type of car from the other variables (no good reason!)
- We omit variables that have missing values and factors with large numbers of levels
- We use the R **`tree`** package for illustrative purposes
- The full cycle is
  - build an initial tree
  - check size by cross-validation
  - prune to something sensible

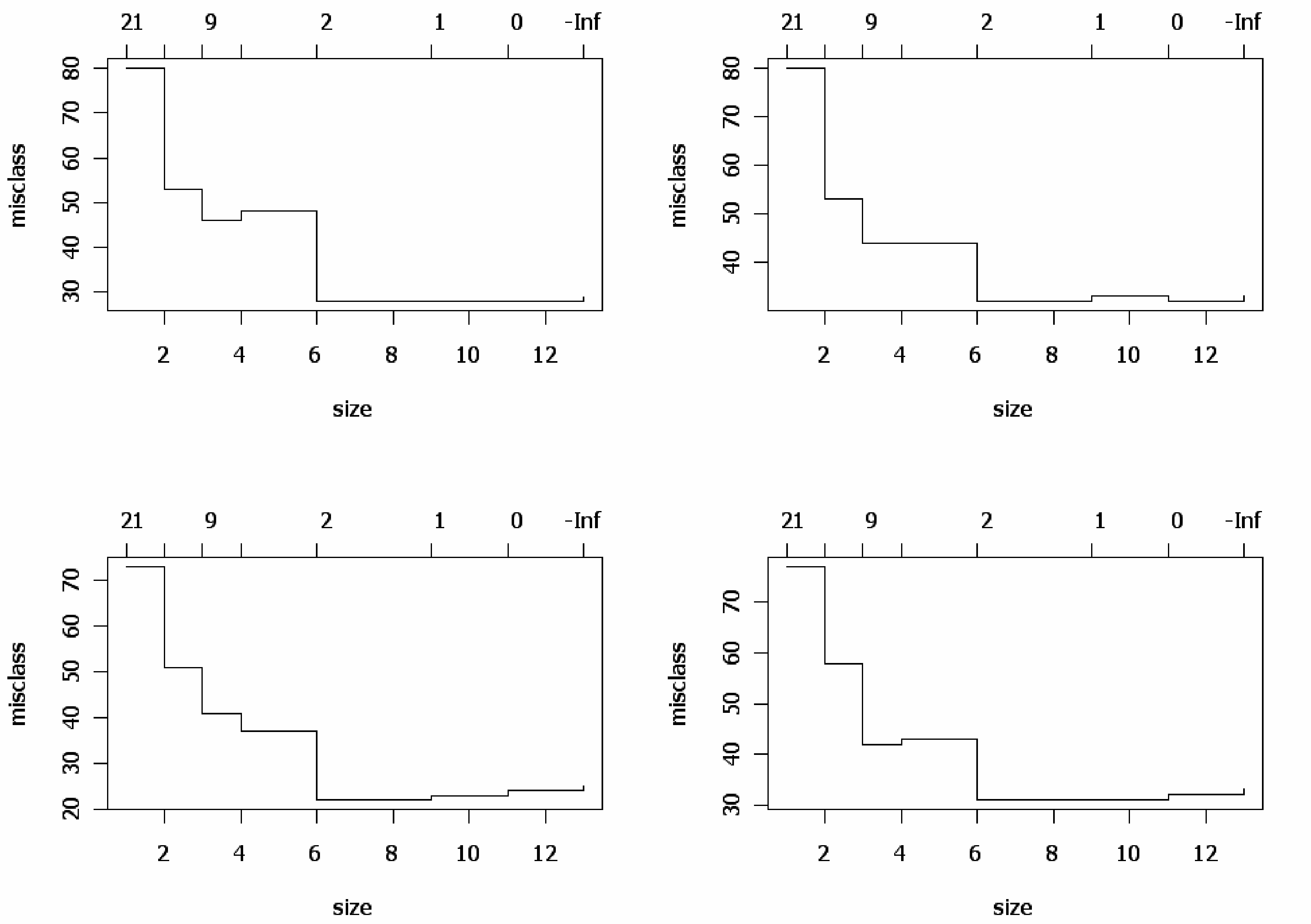
Construction and cross-validation

```
library(tree)

cars93.t1 <- tree(Type ~ ., cars93, minsize = 5)
x11(width = 8, height = 6)
plot(cars93.t1); text(cars93.t1, cex = 0.75)

par(mfrow = c(2,2))
for(j in 1:4)
  plot(cv.tree(cars93.t1, FUN=prune.tree))


# an alternative criterion
for(j in 1:4)
  plot(cv.tree(cars93.t1, FUN=prune.misclass))
```


Width < 68.5
EngineSize < 1.95
Passengers < 6.5
Min.Price < 9.05
Length < 173.5
Wheelbase < 103.5
Passengers < 4.5
RPM < 4700
Van
Small
Sporty
Small
Small
Compact Compact
Passengers < 4.5
Width < 73.5
Sporty
Midsize
Length < 201.5
Min.Price < 28.5
Large
Large
Midsize
Large


100.0 42.0 21.0 11.0 9.8 4.7 -Inf
deviance
220 260 300
2 4 6 8 10 12
size
100.0 42.0 21.0 11.0 9.8 4.7 -Inf
deviance
200 250 300
2 4 6 8 10 12
size
100.0 42.0 21.0 11.0 9.8 4.7 -Inf
deviance
200 250 300
2 4 6 8 10 12
size
100.0 42.0 21.0 11.0 9.8 4.7 -Inf
deviance
220 260 300 340
2 4 6 8 10 12
size

```
for(j in 1:4)
  plot(cv.tree(cars93.t1, FUN = prune.misclass))
```

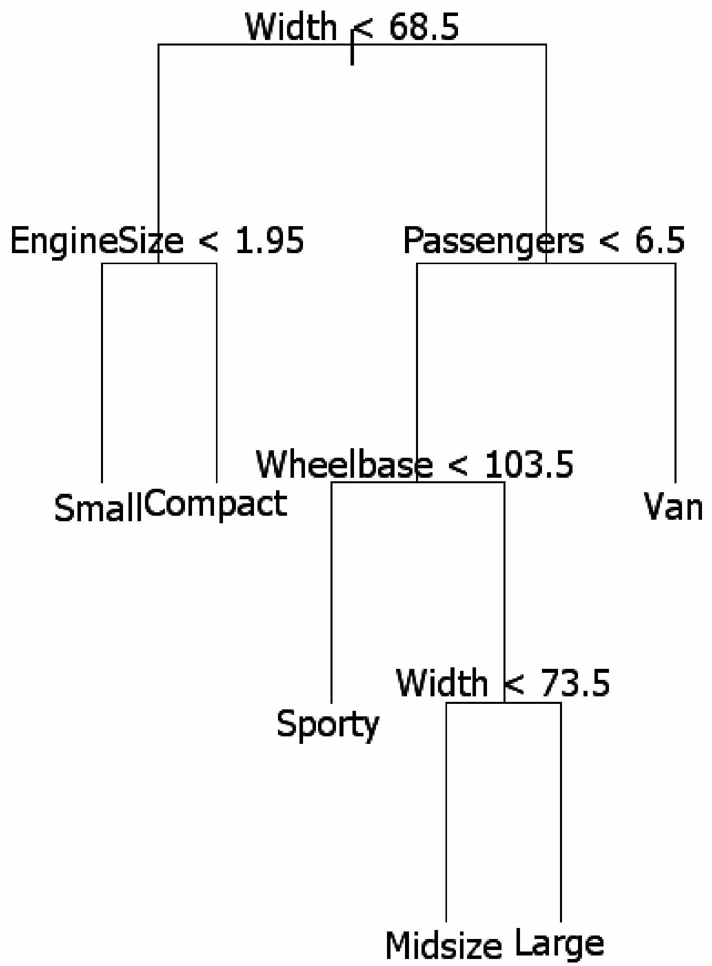


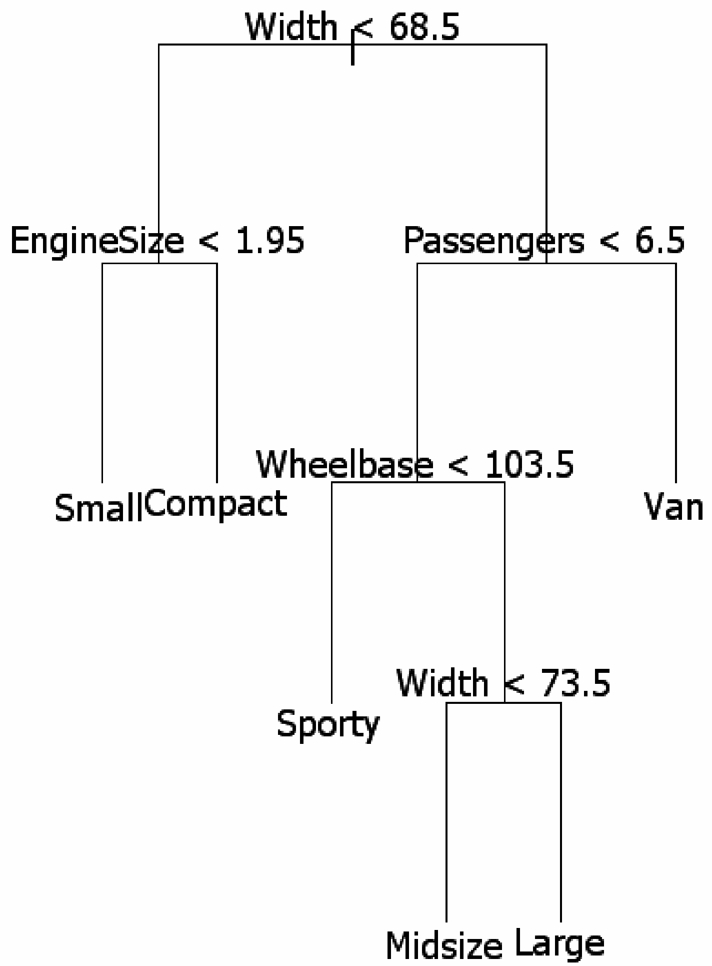
Pruning

- Can use **`snip.tree()`** to prune manually
- The function **`prune.tree()`** enacts optimal deviance pruning
- The function **`prune.misclass()`** enacts optimal misclassification rate pruning

```
par(mfrow = c(1,2))
cars93.t2 <- prune.misclass(cars93.t1, best = 6)
plot(cars93.t2, type = "u"); text(cars93.t2)

cars93.t3 <- prune.tree(cars93.t1, best = 6)
plot(cars93.t3, type = "u"); text(cars93.t3)
```

- Both methods lead to the same tree, here.


Width < 68.5
EngineSize < 1.95
Passengers < 6.5
Small
Compact
Wheelbase < 103.5
Van
Sporty
Width < 73.5
Midsize
Large


Width < 68.5
EngineSize < 1.95
Passengers < 6.5
Small
Compact
Wheelbase < 103.5
Van
Sporty
Width < 73.5
Midsize
Large

The confusion matrix is not very confused

```
pred <- predict(cars93.t3, Cars93, type =
  "class")
with(cars93, table(pred, Type))
 Type
pred       Compact Large Midsize Small Sporty Van
  Compact       15     0       0     1      2   0
  Large          0     9       1     0      0   0
  Midsize        0     2      19     0      0   0
  Small          0     0       0    20      4   0
  Sporty         1     0       2     0      8   0
  Van            0     0       0     0      0   9
```

- Real test comes from a train/test sample: exercise!

Comparison with multinomial

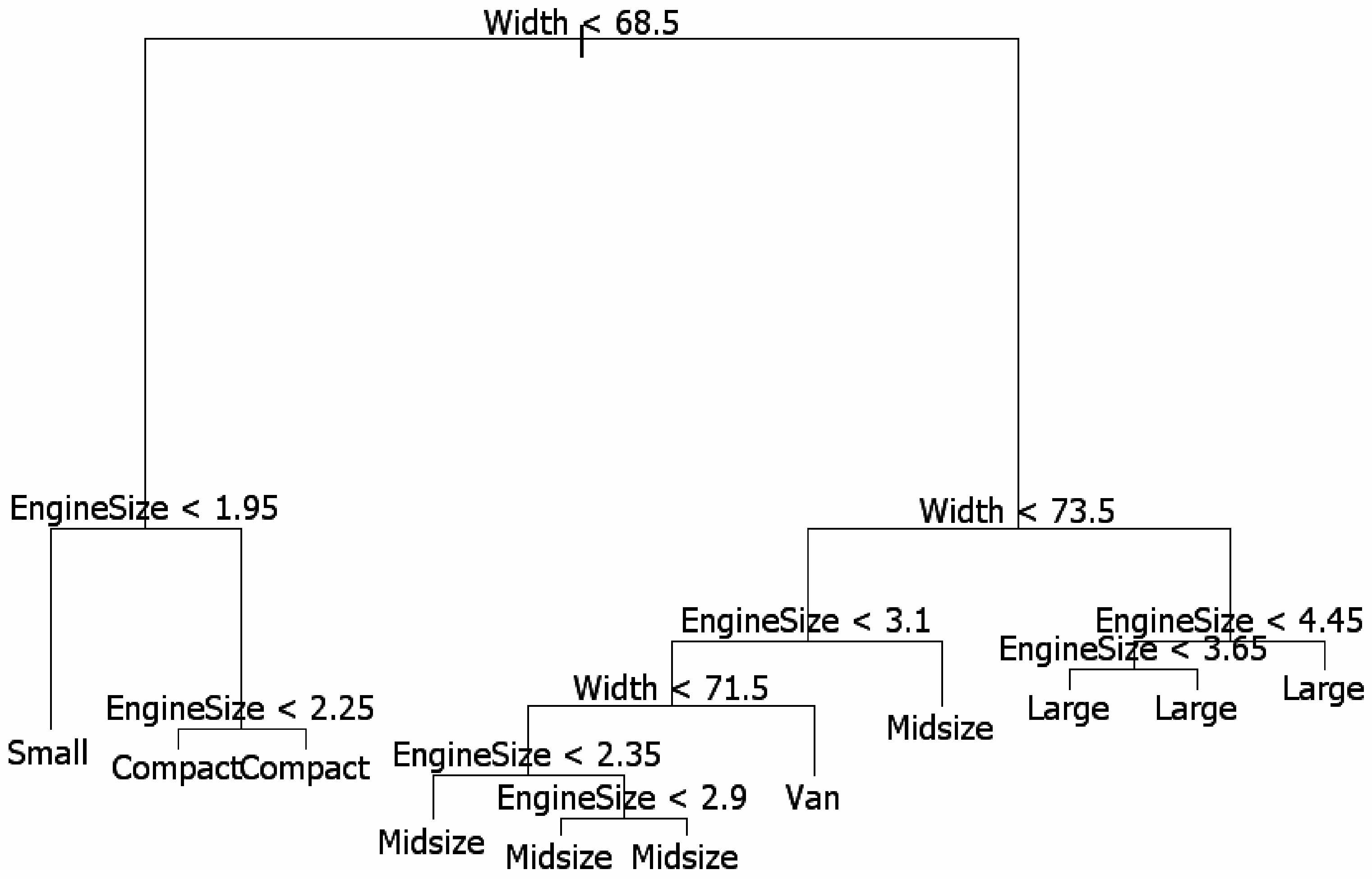
```
library(nnet) # multinomial is a neural network model
m <- multinom(Type ~ Width + EngineSize +
  Passengers + Origin, cars93, maxit = 1000)
pfm <- predict(m, type = "class")
with(cars93, table(Type, pfm))
```

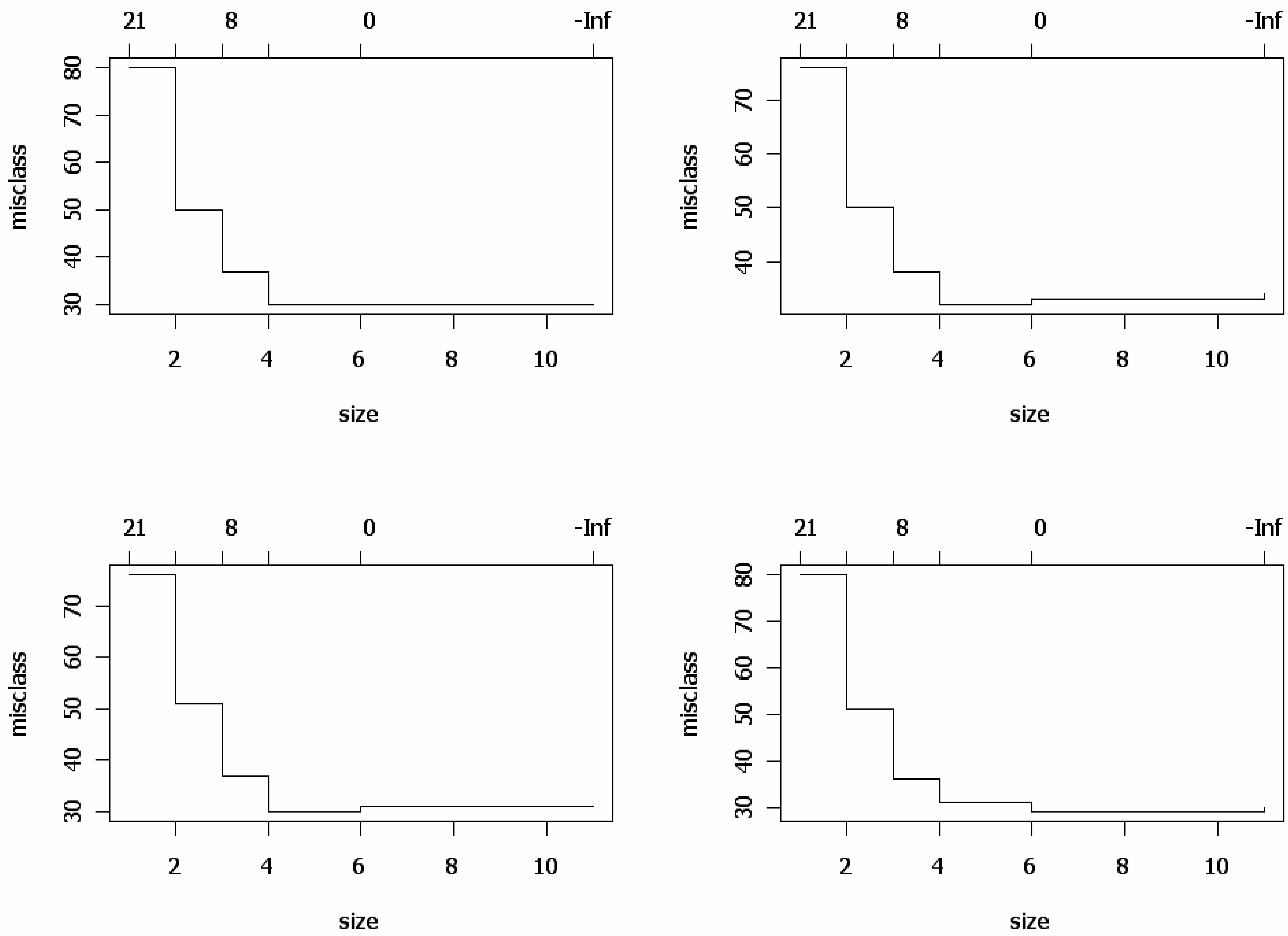
```
 pfm
Type      Compact Large Midsize Small Sporty Van
  Compact      13     0       1     1      1   0
  Large         0    11       0     0      0   0
  Midsize       0     1      21     0      0   0
  Small         1     0       0    19      1   0
  Sporty        1     0       0     2     11   0
  Van           0     0       0     0      0   9
```

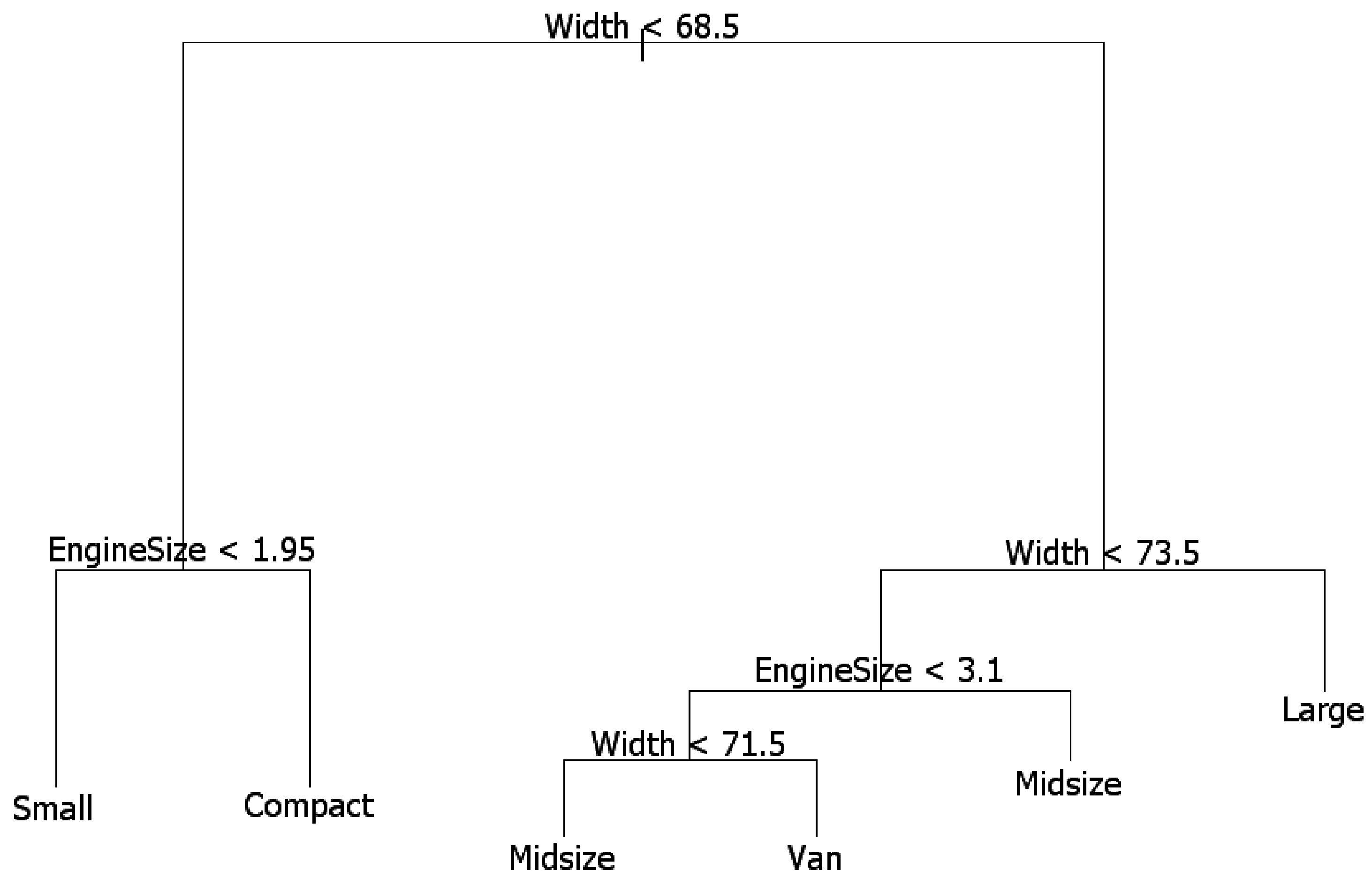
The special case of one or two predictors

- Choose two likely useful predictors:

```
cars.2t <- tree(Type ~ Width + EngineSize,
  Cars93)
par(mfrow = c(1,1))
plot(cars.2t); text(cars.2t)

par(mfrow = c(2,2))
for(j in 1:4)
  plot(cv.tree(cars.2t, FUN=prune.misclass))
par(mfrow = c(1,1))
cars.2t1 <- prune.misclass(cars.2t, best = 6)
plot(cars.2t1); text(cars.2t1)

partition.tree(cars.2t1)
```


Width < 68.5
EngineSize < 1.95
Width < 73.5
EngineSize < 3.1
EngineSize < 4.45
EngineSize < 3.65
Width < 71.5
EngineSize < 2.25
Small
CompactCompact
EngineSize < 2.35
EngineSize < 2.9
Van
Midsize
Large
Large
Large
Midsize
Midsize Midsize

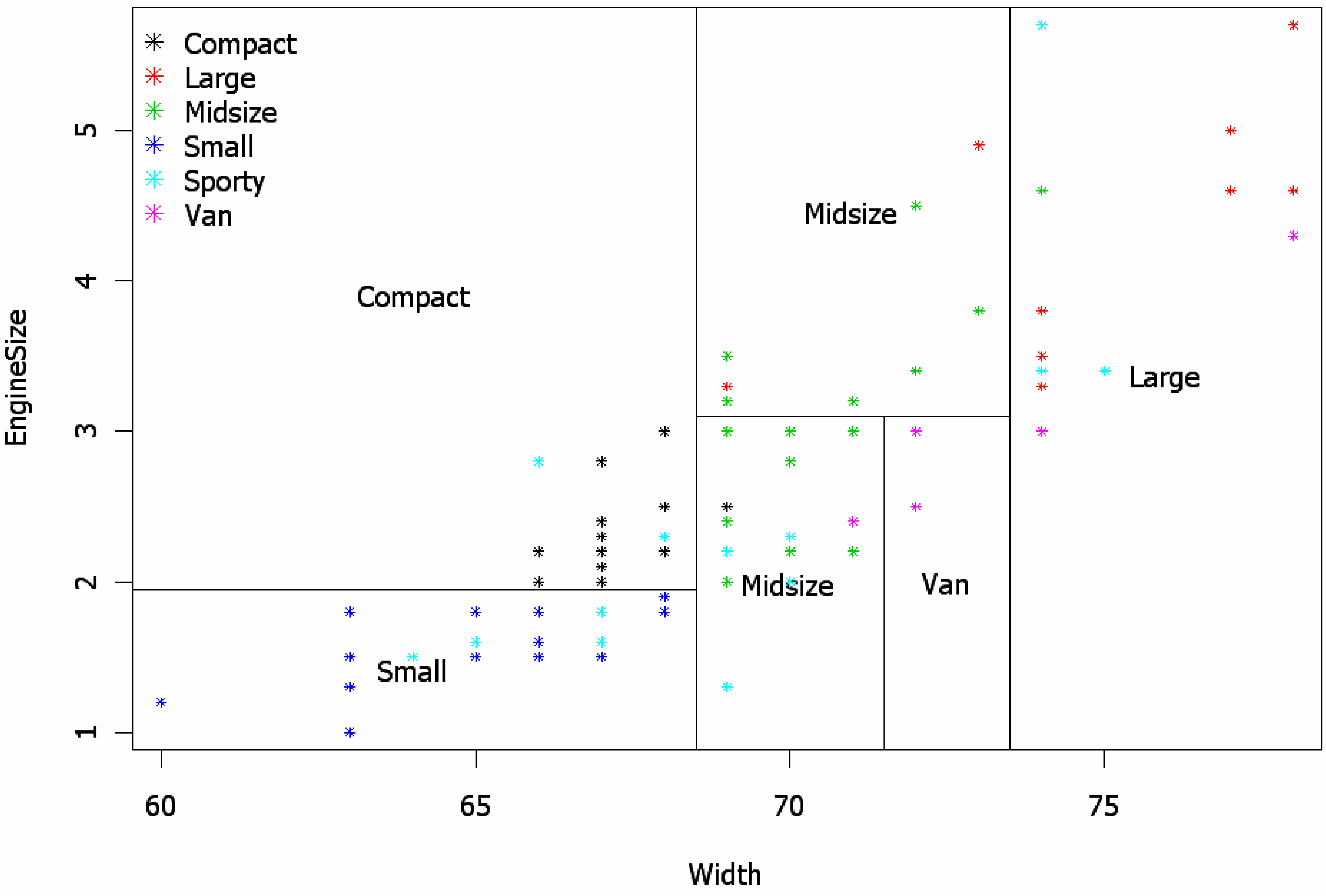

21
8
0
-Inf
misclass
30
40
50
60
70
80
2
4
6
8
10
size
21
8
0
-Inf
misclass
40
50
60
70
2
4
6
8
10
size
21
8
0
-Inf
misclass
30
40
50
60
70
2
4
6
8
10
size
21
8
0
-Inf
misclass
30
40
50
60
70
80
2
4
6
8
10
size


Width < 68.5
EngineSize < 1.95
Width < 73.5
EngineSize < 3.1
Large
Width < 71.5
Midsize
Small
Compact
Midsize
Van

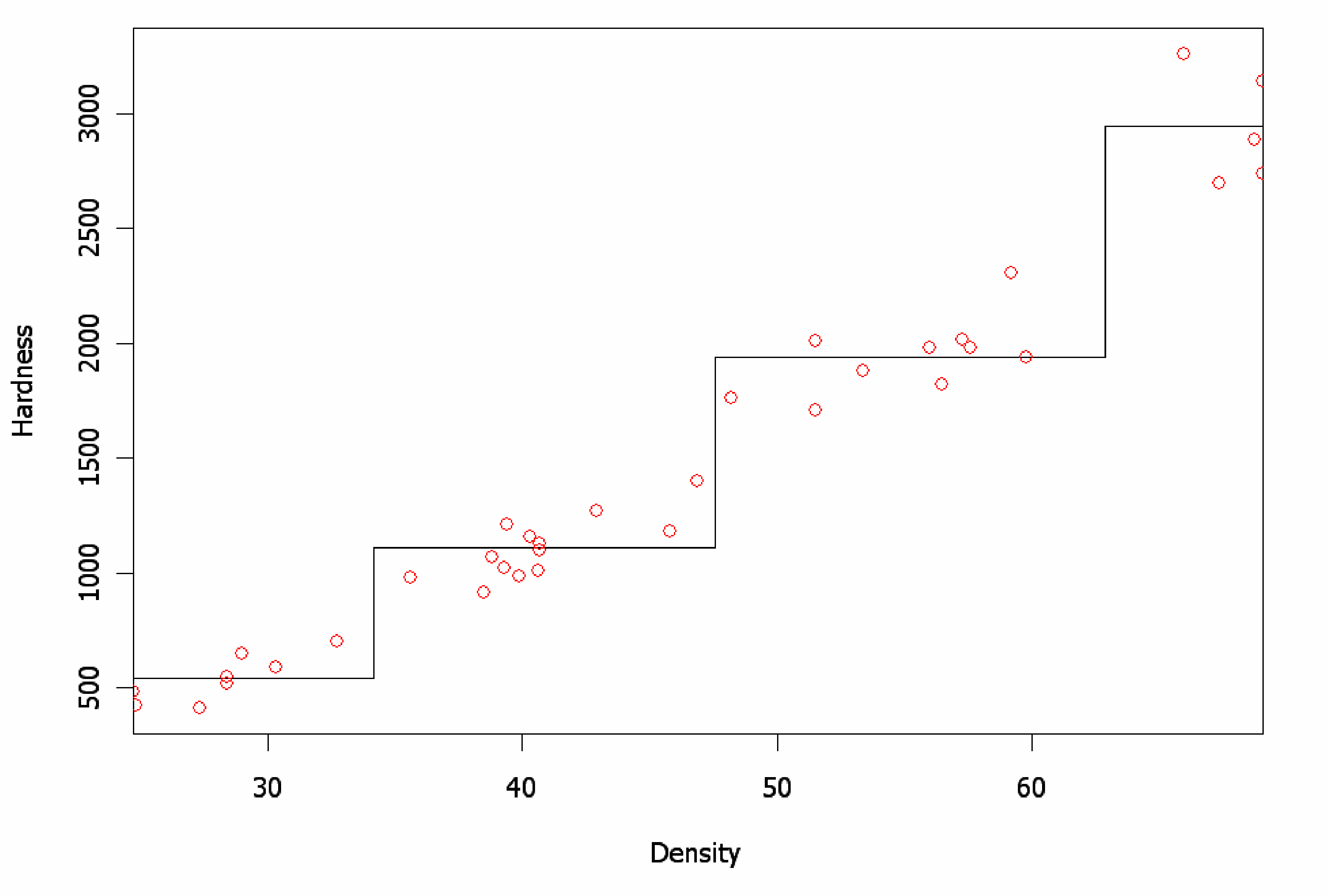
An alternative display

- If there are only two variables, the tree may be given as a two-dimensional diagram.

```
partition.tree(cars.2t1)

with(cars93, {
  points(Width, EngineSize, pch=8,
    col = as.numeric(Type), cex = 0.5)
  legend("topleft", levels(Type), pch = 8,
    col = 1:length(levels(Type)), bty = "n")
})
```


Compact
Large
Midsize
Small
Sporty
Van
Midsize
Compact
Large
Midsize
Van
Small
EngineSize
1
2
3
4
5
60
65
70
75
Width

```
janka.t1 <- tree(Hardness ~ Density, janka)
partition.tree(janka.t1)
with(janka, points(Density, Hardness, col="red"))
```

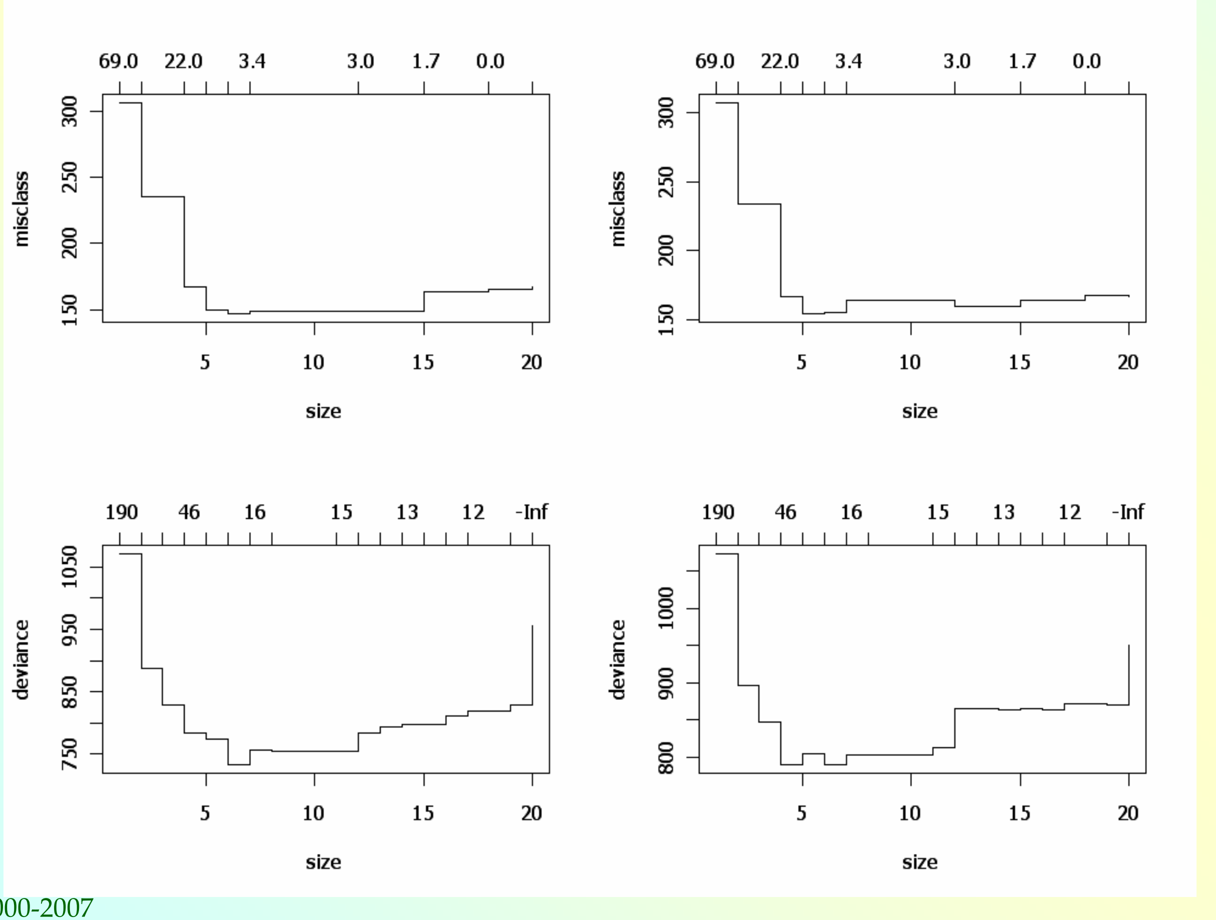


Binary data examples

- The credit card data provides a more realistic example.
- A previous exercise used this data with **`rpart`**
- Consider now using the tree package instead.

```
set.seed(32867700) # my home phone number
ind <- sample(nrow(CC), 810)
CCTrain <- CC[ind, ]
CCTest <- CC[-ind, ]
Store(CCTrain, CCTest)
```

```
CC.t1 <- tree(credit.card.owner ~ ., CCTrain)
par(mfrow = c(2,2))
for(j in 1:2)
  plot(cv.tree(CC.t1, FUN = prune.misclass))
for(j in 1:2)
  plot(cv.tree(CC.t1, FUN = prune.tree))
```



Testing

```
CC.t2 <- prune.misclass(CC.t1, best = 6)

testPred2 <- function(fit, data = CCTest) {
  pred <- predict(fit, data, type = "class")
  Y <- formula(fit)[[2]]
  Cmatrix <- with(data, table(eval(Y), pred))
  tot <- sum(Cmatrix)
  err <- tot - sum(diag(Cmatrix))
  100*err/tot
}

testPred2(CC.t1)  #  [1] 18.39506
testPred2(CC.t2)  #  [1] 18.27160
```

Simple bagging

```
### simple bagging

baggedTree <- local({
  bsample <- function(data)
    data[sample(nrow(data), rep = TRUE), ]

  function (object, data = eval(object$call$data),
    nBags = 200, ...) {
    bagsFull <- list()
    for (j in 1:nBags)
      bagsFull[[j]] <- update(object, data = bsample(data))
    attr(bagsFull, "formula") <- formula(object)
    class(bagsFull) <- "bagTree"
    bagsFull
  }
})
```

Methods and tests

```
formula.bagTree <- function(x, ...) attr(x, "formula")

predict.bagTree <- function(object, newdata, ...) {
  vals <- sapply(object, predict, newdata, type =
   "class")
  svals <- sort(unique(vals))
  mVote <- apply(vals, 1, function(x)
       which.max(table(factor(x, levels = svals))))
  svals[mVote]
}

CC.bag <- baggedTree(CC.t1)

testPred2(CC.bag) # [1] 13.70370
```

Random Forests

```
library(randomForest)
CC.rf <- randomForest(credit.card.owner ~ .,
  CCTrain)

testPred2(CC.rf)  # [1] 13.20988
```

- Random forests wins, but by a very slight margin
- Bagged methods are (in this case) far superior to trees, pruned or otherwise
- Also better (again, in this case) than the parametric models.