Real Estate Data-Set

Vignesh Hariharan

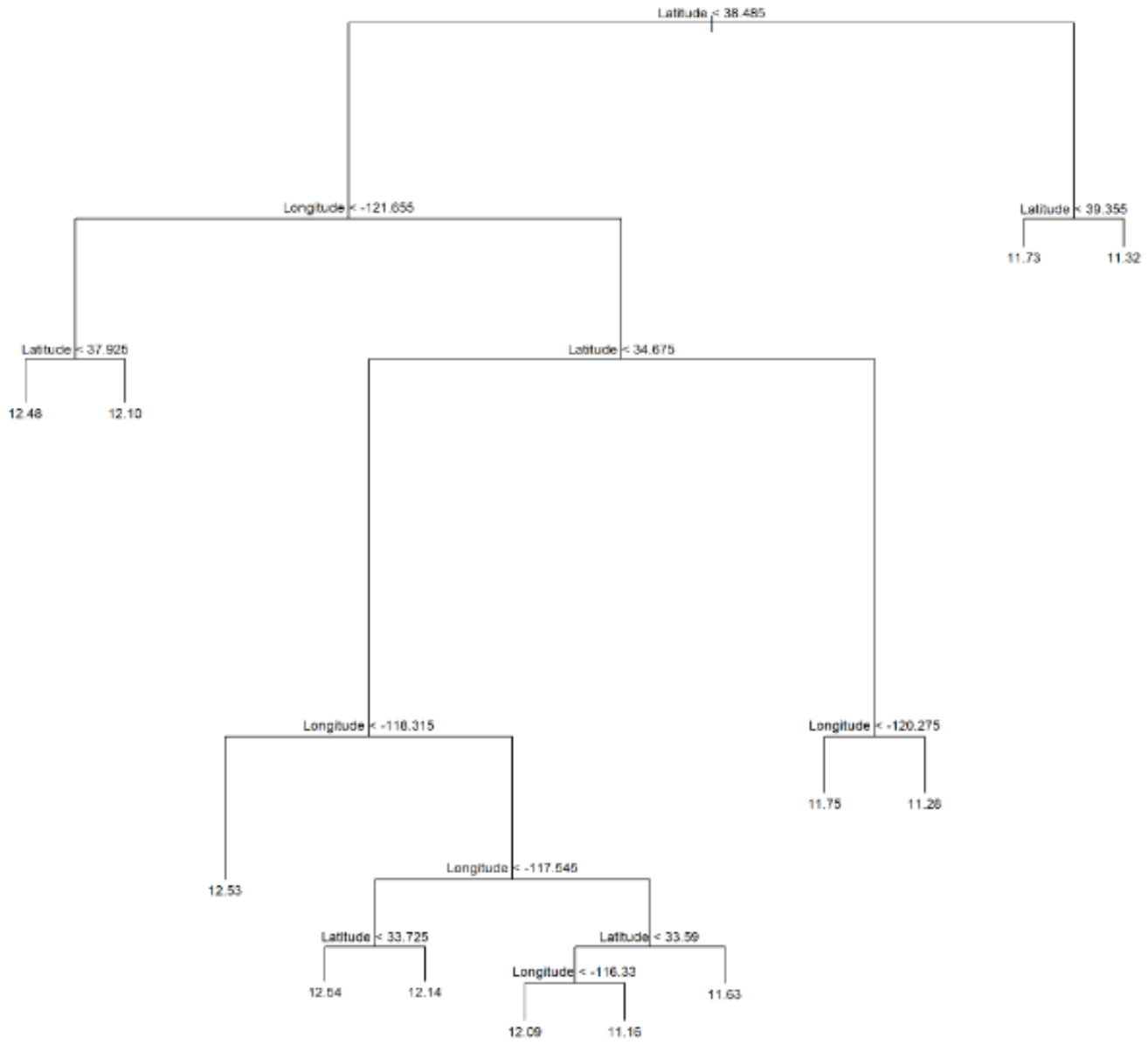
Classification & Regression Trees

Prediction Trees are used to predict a response or class YY from input X1,X2,…,XnX1,X2,…,Xn. If it is a continuous response it's called a regression tree, if it is categorical, it's called a classification tree. At each node of the tree, I check the value of one the input XiXi and depending of the (binary) answer I continue to the left or to the right subbranch. When I reach a leaf I will find the prediction (usually it is a simple statistic of the dataset the leaf represents, like the most common value from the available classes).
Contrary to linear or polynomial regression which are global models (the predictive formula is supposed to hold in the entire data space), trees try to partition the data space into small enough parts where I can apply a simple different model on each part. The non-leaf part of the tree is just the procedure to determine for each data xx what is the model (i.e, which leaf) I will use to classify it.

Regression Trees

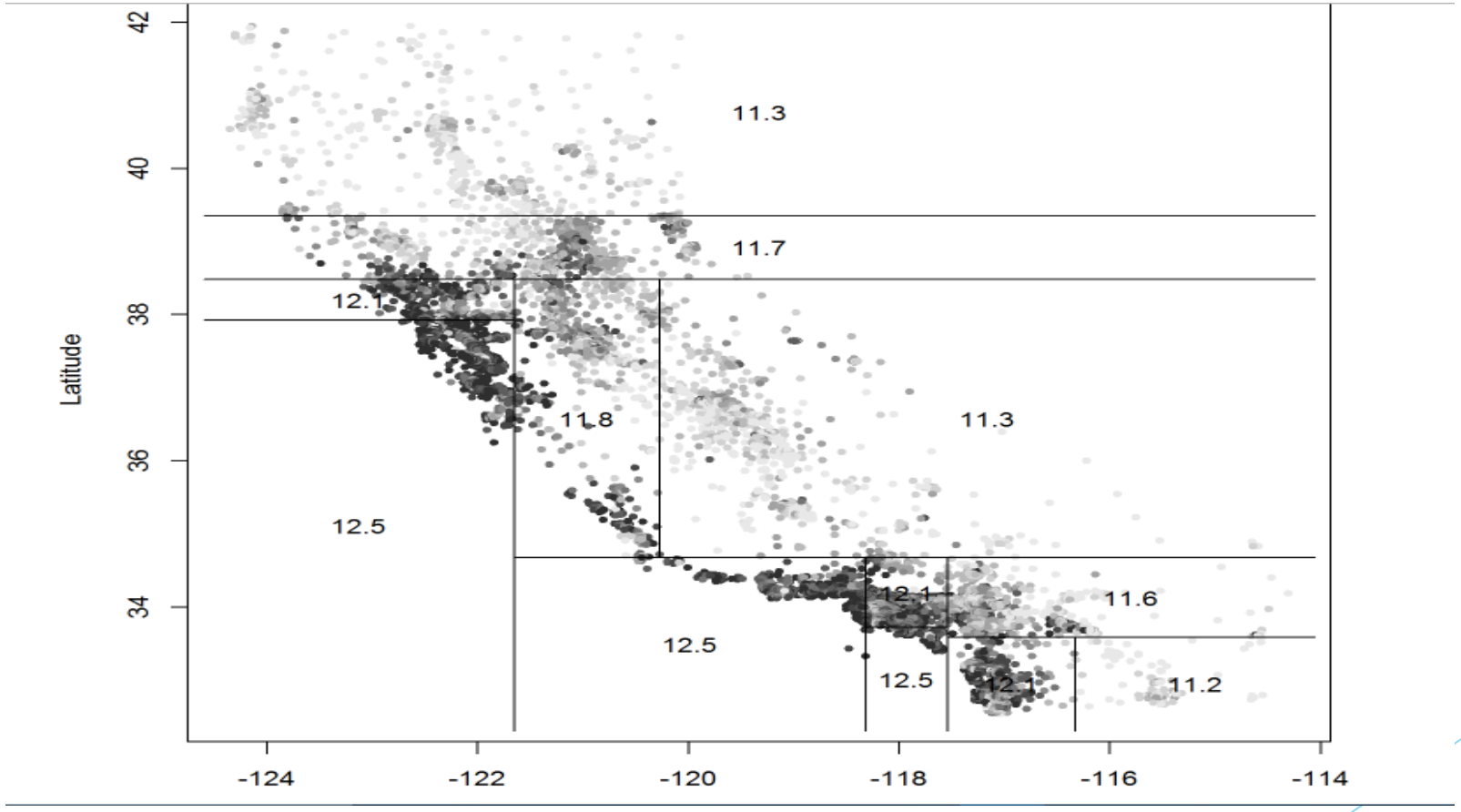
Regression Trees like say linear regression, outputs an expected
certain output.

```
library(tree)

real.estate <- read.table("cadata.dat", header=TRUE)
tree.model <- tree(log(MedianHouseValue) ~ Longitude + Latitude, data=real.estate)
plot(tree.model)
text(tree.model, cex=.75)
```


Latitude < 38.485
Longitude < -121.655
Latitude < 39.355
Latitude < 37.925
Latitude < 34.675
11.73
11.32
12.48
12.10
Longitude < -118.315
Longitude < -120.275
11.75
11.26
12.53
Longitude < -117.545
Latitude < 33.725
Latitude < 33.59
12.04
12.14
Longitude < -116.33
11.63
12.09
11.16

We can compare the predictions with the dataset (darker is more expensive) which seem to capture the global price trend:

```
price.deciles <- quantile(real.estate$MedianHouseValue, 0:10/10)
cut.prices    <- cut(real.estate$MedianHouseValue, price.deciles, include.lowest=TRUE)
plot(real.estate$Longitude, real.estate$Latitude, col=grey(10:2/11)[cut.prices], pch=20, xlab="Longitude",ylab="Latitude")
partition.tree(tree.model, ordvars=c("Longitude","Latitude"), add=TRUE)
```



```
summary(tree.model)
```

```
##
## Regression tree:
## tree(formula = log(MedianHouseValue) ~ Longitude + Latitude,
##     data = real.estate)
## Number of terminal nodes:  12
## Residual mean deviance:  0.1662 = 3429 / 20630
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -2.75900 -0.26080 -0.01359  0.00000  0.26310  1.84100
```

Deviance means here the mean squared error.
The flexibility of a tree is basically controlled by how many leaves they have, since that's how many cells they partition things into. The tree fitting function has a number of controls settings which limit how much it will grow | each node has to contain a certain number of points, and adding a node has to reduce the error by at least a certain amount. The default for the latter, min.dev, is 0:01; let's turn it down and see what happens:

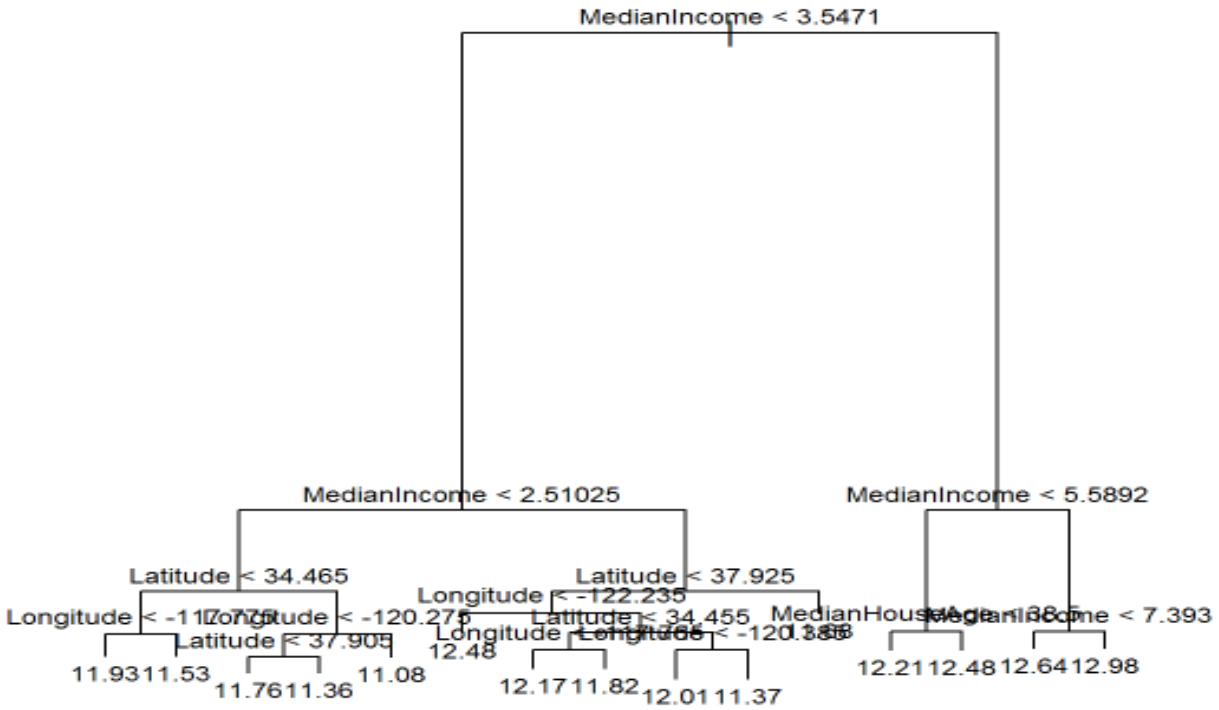
```
tree.model2 <- tree(log(MedianHouseValue) ~ Longitude + Latitude, data=real.estate, mindev=0
plot(tree.model2)
text(tree.model2, cex=.75)
```

```
summary(tree.model2)
```

```
##
## Regression tree:
## tree(formula = log(MedianHouseValue) ~ Longitude + Latitude,
##     data = real.estate, mindev = 0.001)
## Number of terminal nodes:  68
## Residual mean deviance:  0.1052 = 2164 / 20570
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -2.94700 -0.19790 -0.01872  0.00000  0.19970  1.60600
```

It’s obviously much finer-grained than the previous example (68 leaf's against 12), and does a better job of matching the actual prices (lower error).
Also, I can include all the variables, not only the latitude and longitude:

```
tree.model3 <- tree(log(MedianHouseValue) ~ ., data=real.estate)
plot(tree.model3)
text(tree.model3, cex=.75)
```



```
summary(tree.model3)
```

```
##
## Regression tree:
## tree(formula = log(MedianHouseValue) ~ ., data = real.estate)
## Variables actually used in tree construction:
## [1] "MedianIncome"   "Latitude"       "Longitude"      "MedianHouseAge"
## Number of terminal nodes:  15
## Residual mean deviance:  0.1321 = 2724 / 20620
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -2.86000 -0.22650 -0.01475  0.00000  0.20740  2.03900
```

Classification Trees

Classification trees output the predicted class for a given sample. Let's use here the iris dataset (and split it into train and test sets):

```
set.seed(101)
alpha     <- 0.7 # percentage of training set
inTrain   <- sample(1:nrow(iris), alpha * nrow(iris))
train.set <- iris[inTrain,]
test.set  <- iris[-inTrain,]
```

There are two options for the output: + Point prediction: simply gives the predicted class + Distributional prediction: gives a probability for each class

```r
library(tree)

tree.model <- tree(Species ~ Sepal.Width + Petal.Width, data=train.set)
tree.model
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 105 230.200 versicolor ( 0.33333 0.36190 0.30476 )
##    2) Petal.Width < 0.8 35   0.000 setosa ( 1.00000 0.00000 0.00000 ) *
##    3) Petal.Width > 0.8 70  96.530 versicolor ( 0.00000 0.54286 0.45714 )
##      6) Petal.Width < 1.7 40  21.310 versicolor ( 0.00000 0.92500 0.07500 )
##       12) Petal.Width < 1.35 20   0.000 versicolor ( 0.00000 1.00000 0.00000 ) *
##       13) Petal.Width > 1.35 20  16.910 versicolor ( 0.00000 0.85000 0.15000 )
##         26) Sepal.Width < 3.05 14  14.550 versicolor ( 0.00000 0.78571 0.21429 ) *
##         27) Sepal.Width > 3.05 6   0.000 versicolor ( 0.00000 1.00000 0.00000 ) *
##      7) Petal.Width > 1.7 30   8.769 virginica ( 0.00000 0.03333 0.96667 )
##       14) Petal.Width < 1.85 8   6.028 virginica ( 0.00000 0.12500 0.87500 ) *
##       15) Petal.Width > 1.85 22   0.000 virginica ( 0.00000 0.00000 1.00000 ) *
```

```
summary(tree.model)
```

```
##
## Classification tree:
## tree(formula = Species ~ Sepal.Width + Petal.Width, data = train.set)
## Number of terminal nodes:  6
## Residual mean deviance:  0.2078 = 20.58 / 99
## Misclassification error rate: 0.0381 = 4 / 105
```

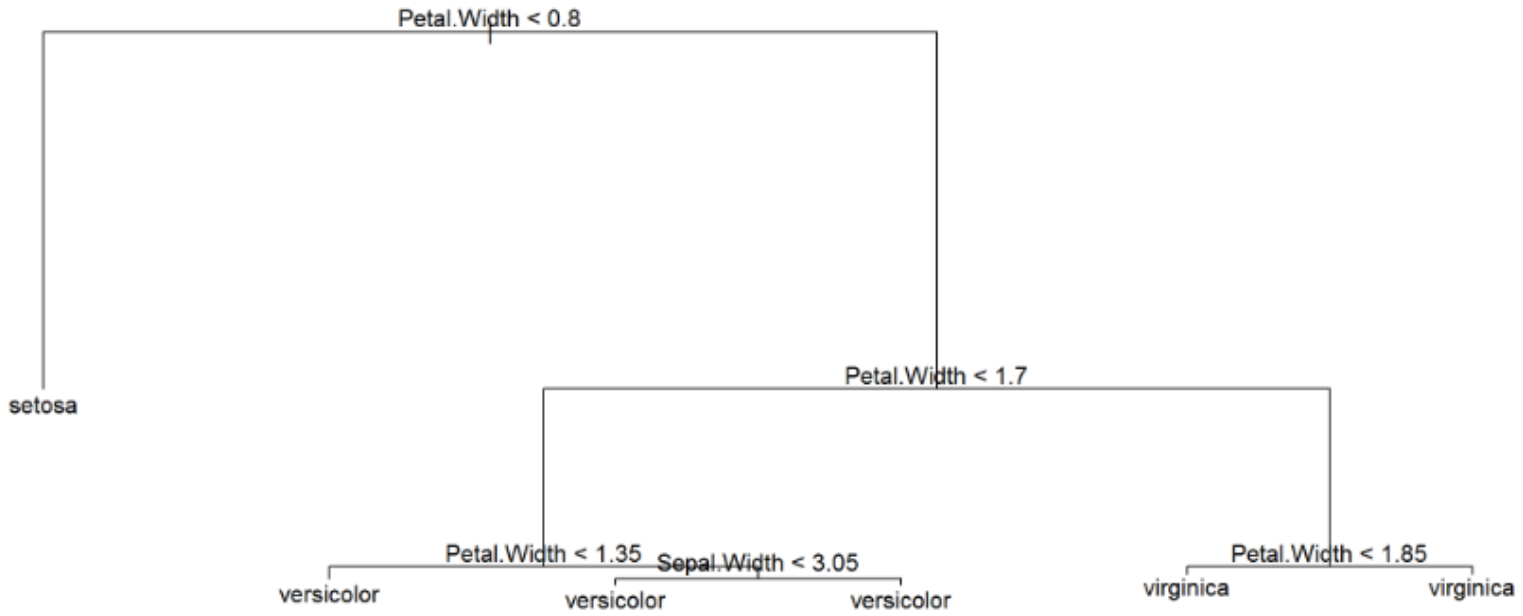
```
# Distributional prediction
my.prediction <- predict(tree.model, test.set) # gives the probability for each class
head(my.prediction)
```

```
##    setosa versicolor virginica
## 5       1          0         0
## 10      1          0         0
## 12      1          0         0
## 15      1          0         0
## 16      1          0         0
## 18      1          0         0
```

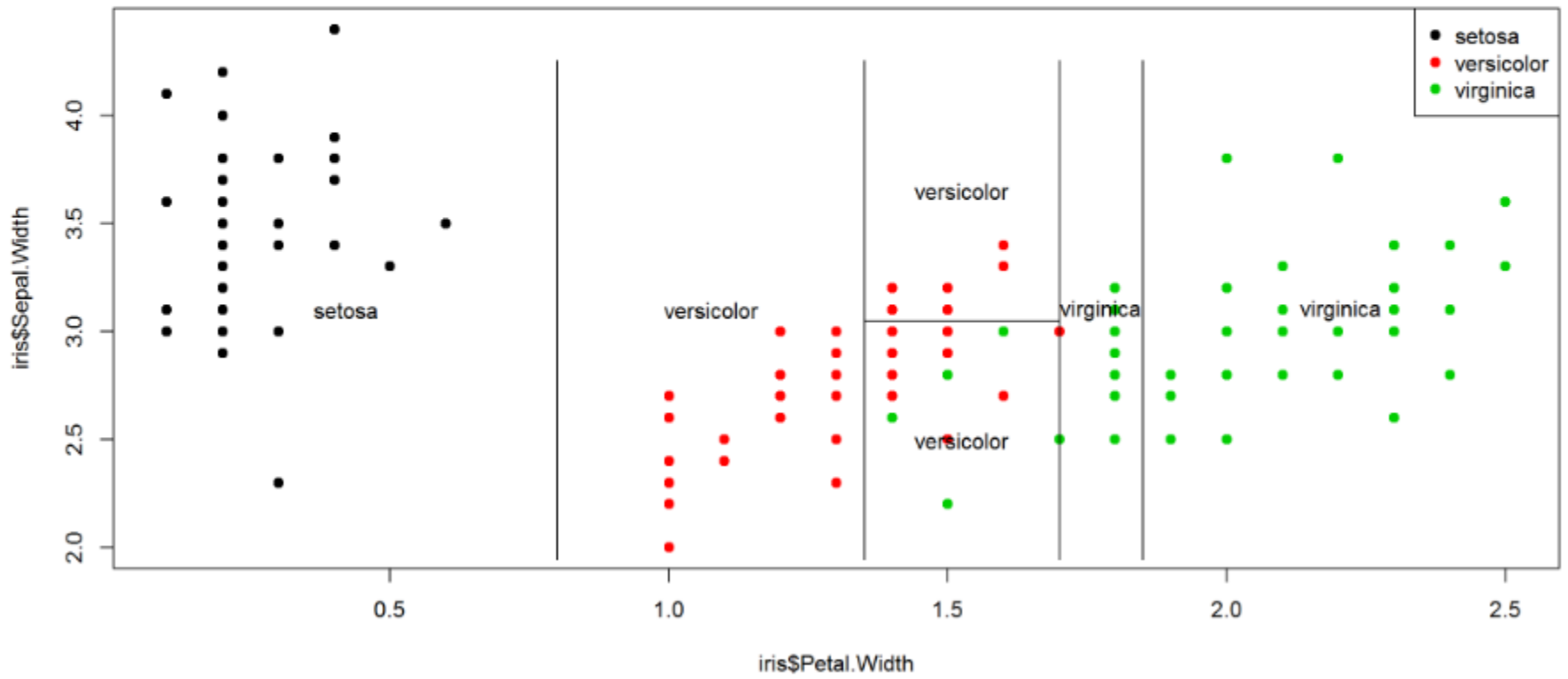
```
# Point prediction
# Let's translate the probability output to categorical output
maxidx <- function(arr) {
    return(which(arr == max(arr)))
}
idx <- apply(my.prediction, c(1), maxidx)
prediction <- c('setosa', 'versicolor', 'virginica')[idx]
table(prediction, test.set$Species)
```

```
##
## prediction   setosa versicolor virginica
##   setosa         15          0         0
##   versicolor      0         11         1
##   virginica       0          1        17
```

```
plot(tree.model)
text(tree.model)
```



```
# Another way to show the data:
plot(iris$Petal.Width, iris$Sepal.Width, pch=19, col=as.numeric(iris$Species))
partition.tree(tree.model, label="Species", add=TRUE)
legend("topright",legend=unique(iris$Species), col=unique(as.numeric(iris$Species)), pch=19)
```

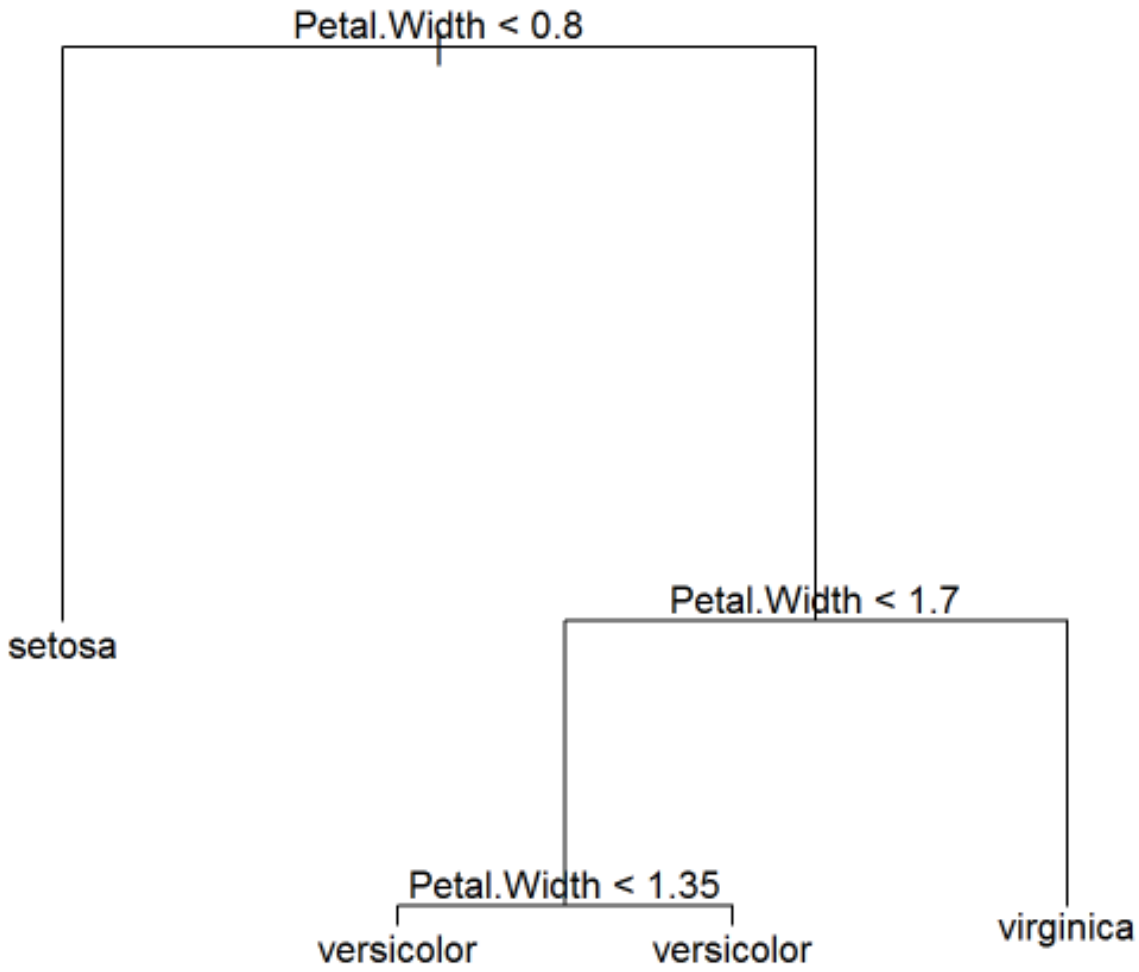


```
summary(tree.model)
```

```
##
## Classification tree:
## tree(formula = Species ~ Sepal.Width + Petal.Width, data = train.set)
## Number of terminal nodes:  6
## Residual mean deviance:  0.2078 = 20.58 / 99
## Misclassification error rate: 0.0381 = 4 / 105
```

For classification trees I can also use argument method="misclass" so that the pruning measure should be the number of misclassifications.

```
pruned.tree <- prune.tree(tree.model, best=4)
plot(pruned.tree)
text(pruned.tree)
```


Petal.Width < 0.8
setosa
Petal.Width < 1.7
Petal.Width < 1.35
versicolor
versicolor
virginica

```
pruned.prediction <- predict(pruned.tree, test.set, type="class") # give the
table(pruned.prediction, test.set$Species)
```

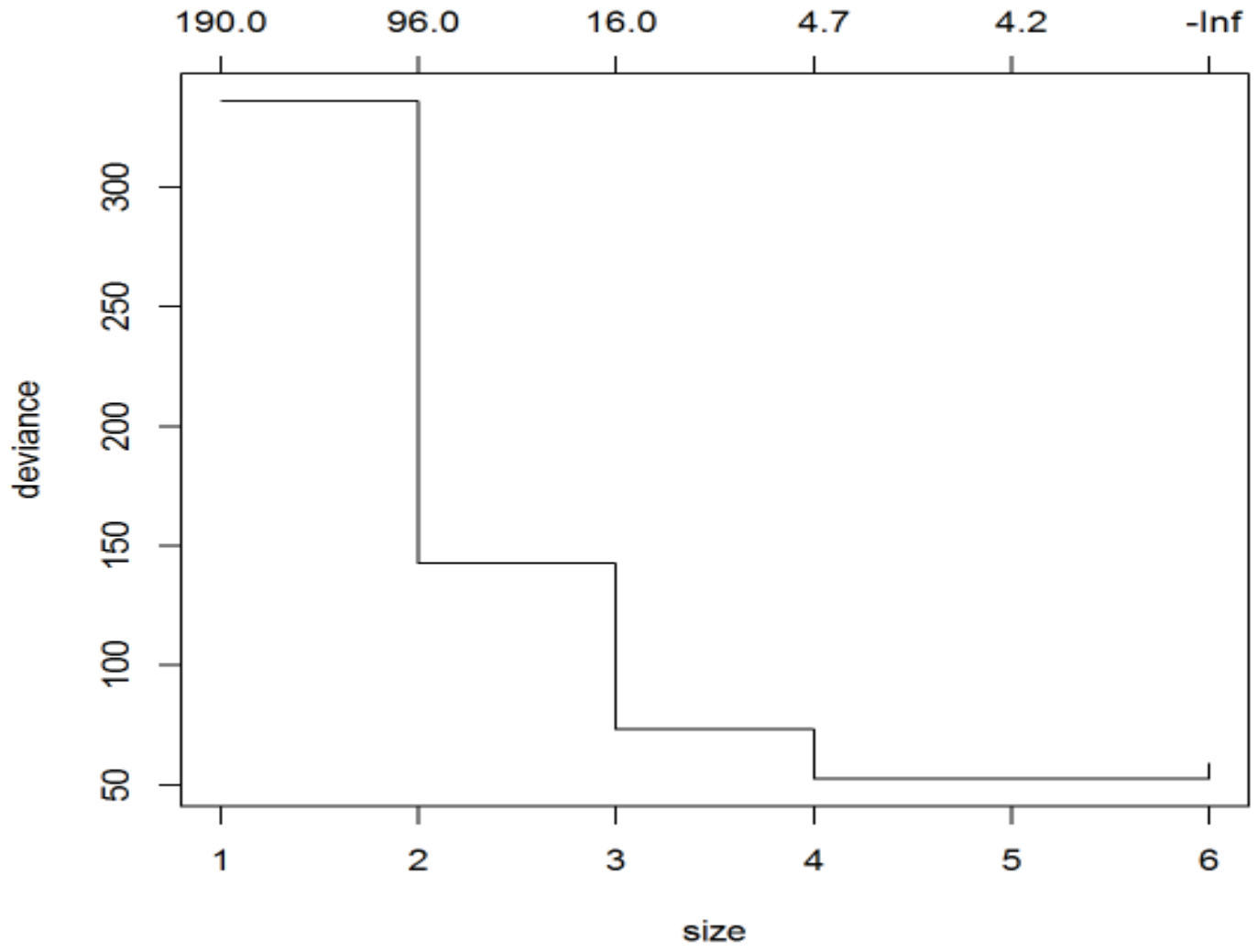
```
##
## pruned.prediction setosa versicolor virginica
##        setosa         15          0         0
##        versicolor      0         11         1
##        virginica       0          1        17
```

This package can also do K-fold cross-validation using `cv.tree()` to find the best tree:

```
# here, let's use all the variables and all the samples
tree.model <- tree(Species ~ ., data=iris)
summary(tree.model)
```

```
##
## Classification tree:
## tree(formula = Species ~ ., data = iris)
## Variables actually used in tree construction:
## [1] "Petal.Length" "Petal.Width"  "Sepal.Length"
## Number of terminal nodes:  6
## Residual mean deviance:  0.1253 = 18.05 / 144
## Misclassification error rate: 0.02667 = 4 / 150
```

```
cv.model <- cv.tree(tree.model)
plot(cv.model)
```


190.0
96.0
16.0
4.7
4.2
-Inf
deviance
50
100
150
200
250
300
1
2
3
4
5
6
size

```
cv.model$dev  # gives the deviance for each K (small is better)
```

```
## [1]  59.08966  52.62454  52.58414  73.33613 142.82350 336.21863
```

```
best.size <- cv.model$size[which(cv.model$dev==min(cv.model$dev))] # which size is better?
best.size
```

```
## [1] 4
```

```
# let's refit the tree model (the number of leafs will be no more than best.size)
cv.model.pruned <- prune.misclass(tree.model, best=best.size)
summary(cv.model.pruned)
```

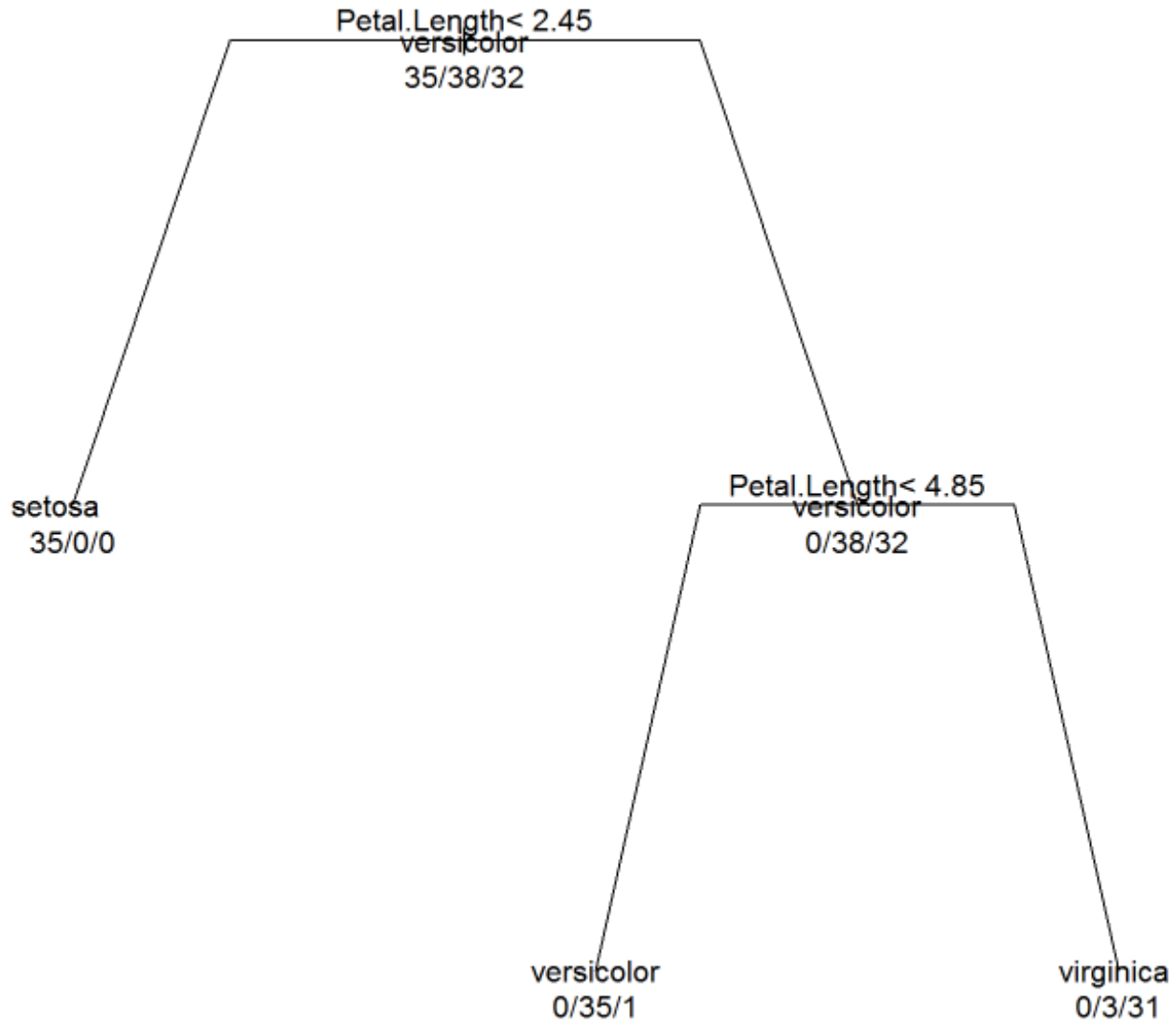
```
##
## Classification tree:
## snip.tree(tree = tree.model, nodes = c(7L, 12L))
## Variables actually used in tree construction:
## [1] "Petal.Length" "Petal.Width"
## Number of terminal nodes:  4
## Residual mean deviance:  0.1849 = 26.99 / 146
## Misclassification error rate: 0.02667 = 4 / 150
```

The misclassification rate has just slighty increased with the pruning of the tree.

Package rpart

This package is faster than tree.

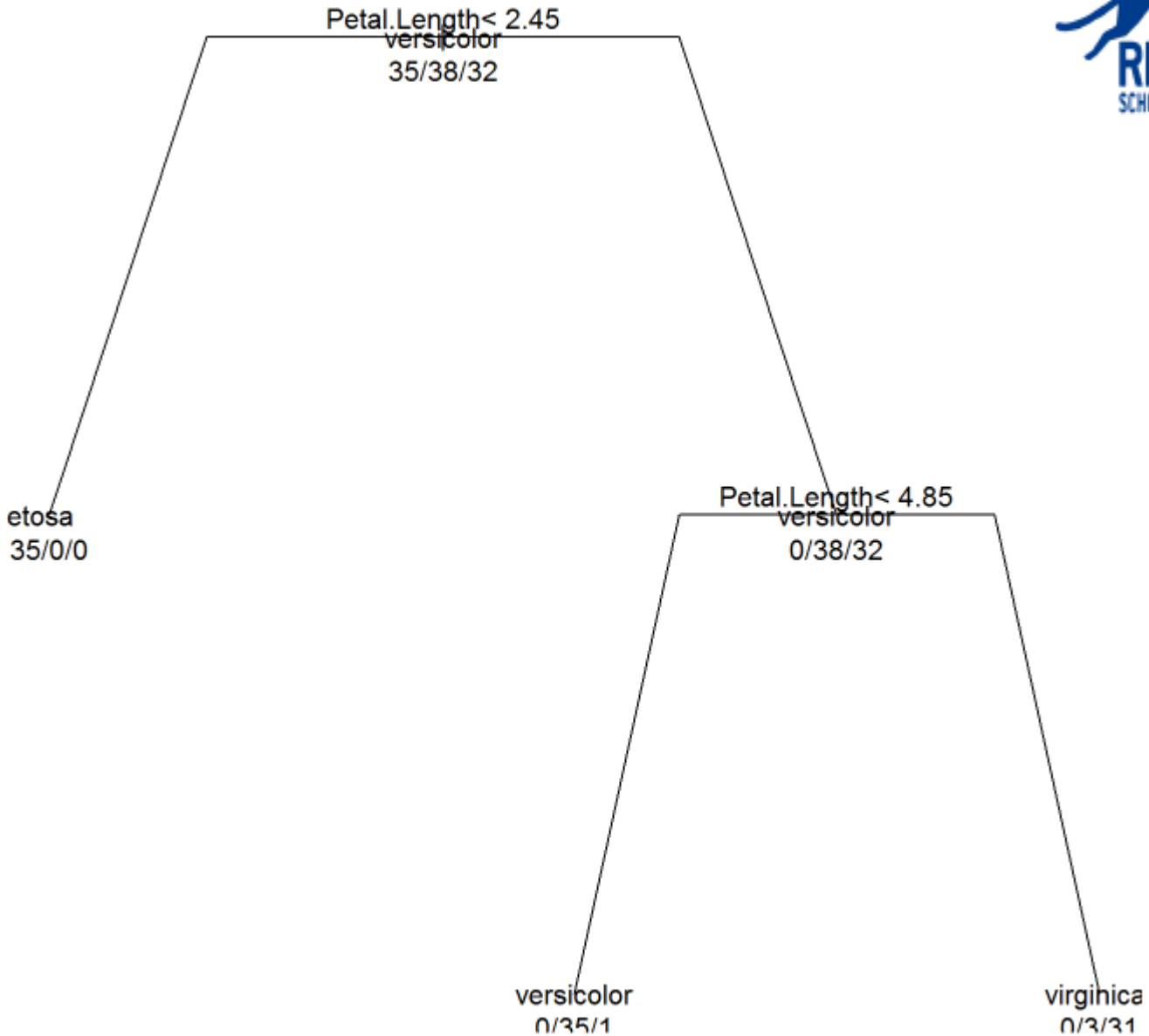
```
library(rpart)

rpart.tree <- rpart(Species ~ ., data=train.set)
plot(rpart.tree, uniform=TRUE, branch=0.6, margin=0.05)
text(rpart.tree, all=TRUE, use.n=TRUE)
title("Training Set's Classification Tree")
```


Petal.Length< 2.45
versicolor
35/38/32
setosa
35/0/0
Petal.Length< 4.85
versicolor
0/38/32
versicolor
0/35/1
virginica
0/3/31

```r
predictions <- predict(rpart.tree, test.set, type="class")
table(test.set$Species, predictions)
```

```
##              predictions
##               setosa versicolor virginica
##    setosa         15          0         0
##    versicolor      0         11         1
##    virginica       0          2        16
```

```r
prune.rpart.tree <- prune(rpart.tree, cp=0.02) # pruning the tree
plot(prune.rpart.tree, uniform=TRUE, branch=0.6)
text(prune.rpart.tree, all=TRUE, use.n=TRUE)
```


Petal.Length< 2.45
versicolor
35/38/32
etosa
35/0/0
Petal.Length< 4.85
versicolor
0/38/32
versicolor
0/35/1
virginica
0/3/31

An eg with different costs for errors:

```
lmat <- matrix(c(0,1,2,
                 1,0,100,
                 2,100,0), ncol = 3)
lmat
```

```
##      [,1] [,2] [,3]
## [1,]    0    1    2
## [2,]    1    0  100
## [3,]    2  100    0
```

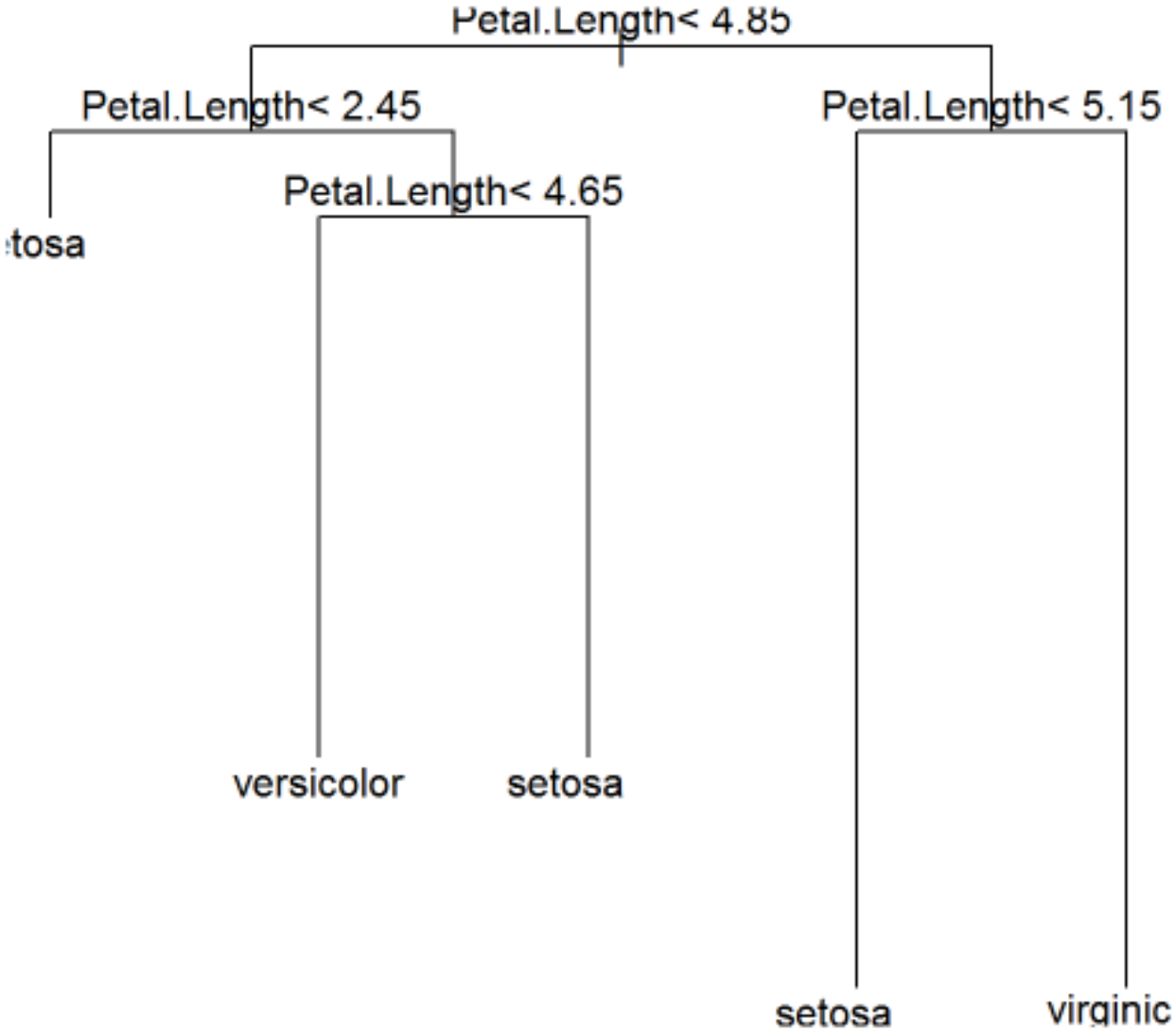
So, misclassifying the 2nd class for the 3rd (or vice-versa) is highly costly.

```
rpart.tree <- rpart(Species ~ ., data=train.set, parms = list(loss = lmat))
predictions <- predict(rpart.tree, test.set, type="class")
table(test.set$Species, predictions)
```

```
##             predictions
##              setosa versicolor virginica
##   setosa         15          0         0
##   versicolor      2         10         0
##   virginica       6          1        11
```

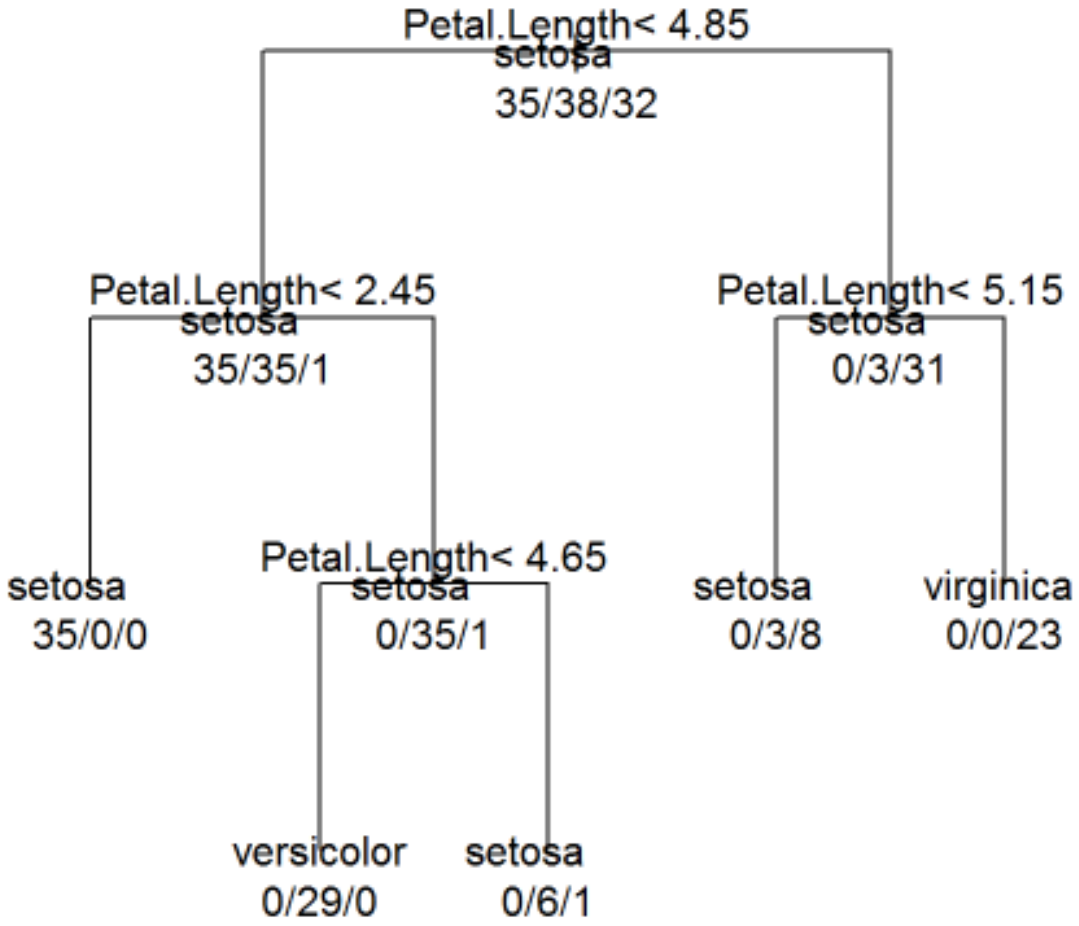
As we see, the algorithm made a different tree to minimize the costly errors.

```
plot(rpart.tree)
text(rpart.tree)
```


Petal.Length< 4.85
Petal.Length< 2.45
Petal.Length< 5.15
Petal.Length< 4.65
tosa
versicolor
setosa
setosa
virginic

A plotting function to better control the parameters:

```r
## Define a plotting function with decent defaults
plot.rpart.obj <- function(rpart.obj, font.size = 0.8) {
    ## plot decision tree
    plot(rpart.obj,
         uniform   = T,    # if 'TRUE', uniform vertical spacing of the nodes is used
         branch    = 1,    # controls the shape of the branches from parent to child node
         compress  = F,    # if 'FALSE', the leaf nodes will be at the horizontal plot
         nspace    = 0.1,
         margin    = 0.1, # an extra fraction of white space to leave around the borders
         minbranch = 0.3)  # set the minimum length for a branch

    ## Add text
    text(x      = rpart.obj,   #
         splits = T,           # If tree are labeled with the criterion for the split
         all    = T,           # If 'TRUE', all nodes are labeled, otherwise just terminal nodes
         use.n  = T,           # Use numbers to annotate
         cex    = font.size)   # Font size
}

plot.rpart.obj(rpart.tree, 1)
```


Petal.Length< 4.85
setosa
35/38/32
Petal.Length< 2.45
setosa
35/35/1
Petal.Length< 5.15
setosa
0/3/31
setosa
35/0/0
Petal.Length< 4.65
setosa
0/35/1
setosa
0/3/8
virginica
0/0/23
versicolor
0/29/0
setosa
0/6/1

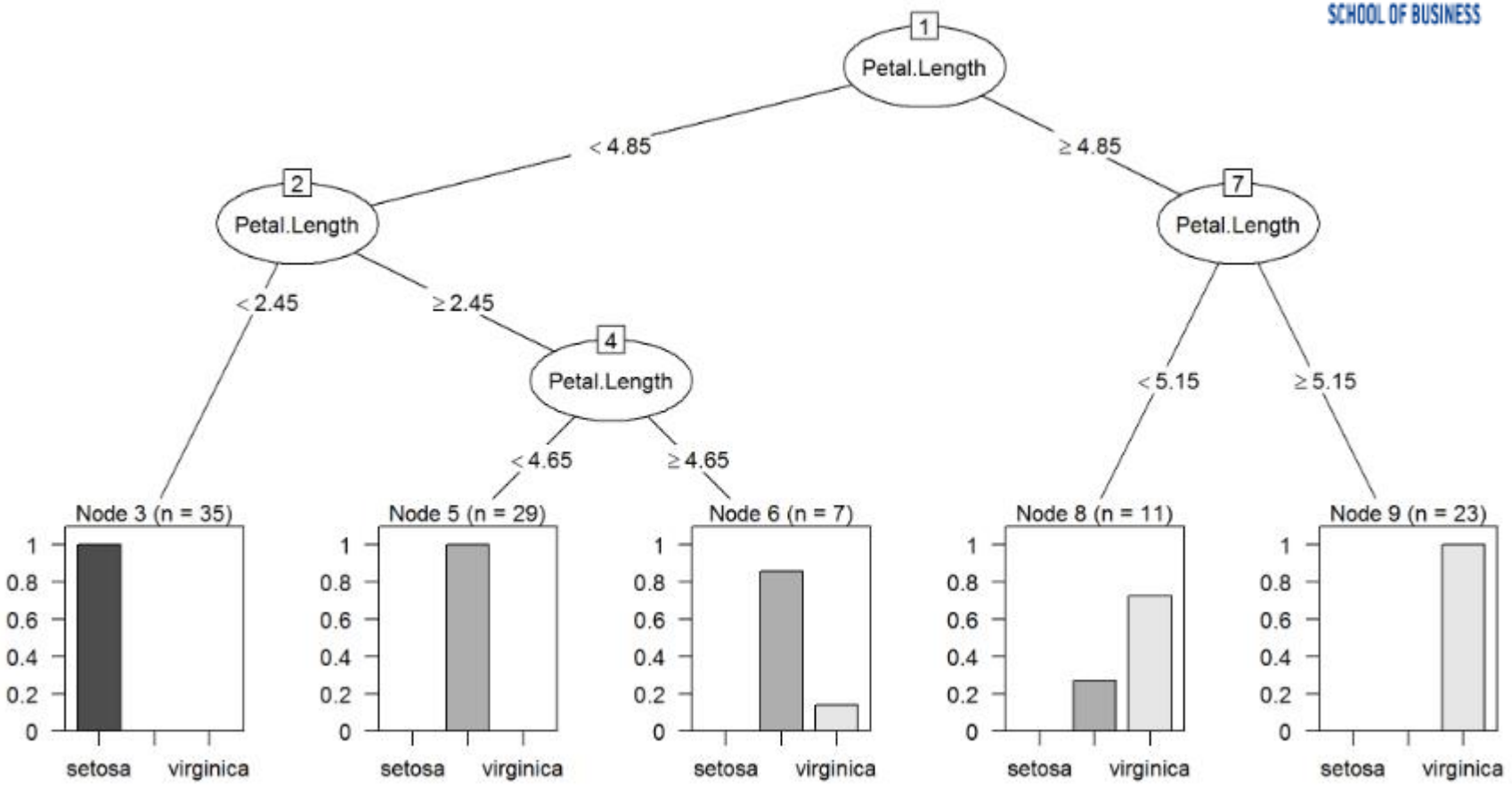
```
library(partykit)
```

```
## Loading required package: grid
```

```
rparty.tree <- as.party(rpart.tree)
rparty.tree
```

```
##
## Model formula:
## Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width
##
## Fitted party:
## [1] root
## |   [2] Petal.Length < 4.85
## |   |   [3] Petal.Length < 2.45: setosa (n = 35, err = 0.0%)
## |   |   [4] Petal.Length >= 2.45
## |   |   |   [5] Petal.Length < 4.65: versicolor (n = 29, err = 0.0%)
## |   |   |   [6] Petal.Length >= 4.65: versicolor (n = 7, err = 14.3%)
## |   [7] Petal.Length >= 4.85
## |   |   [8] Petal.Length < 5.15: virginica (n = 11, err = 27.3%)
## |   |   [9] Petal.Length >= 5.15: virginica (n = 23, err = 0.0%)
##
## Number of inner nodes:    4
## Number of terminal nodes: 5
```

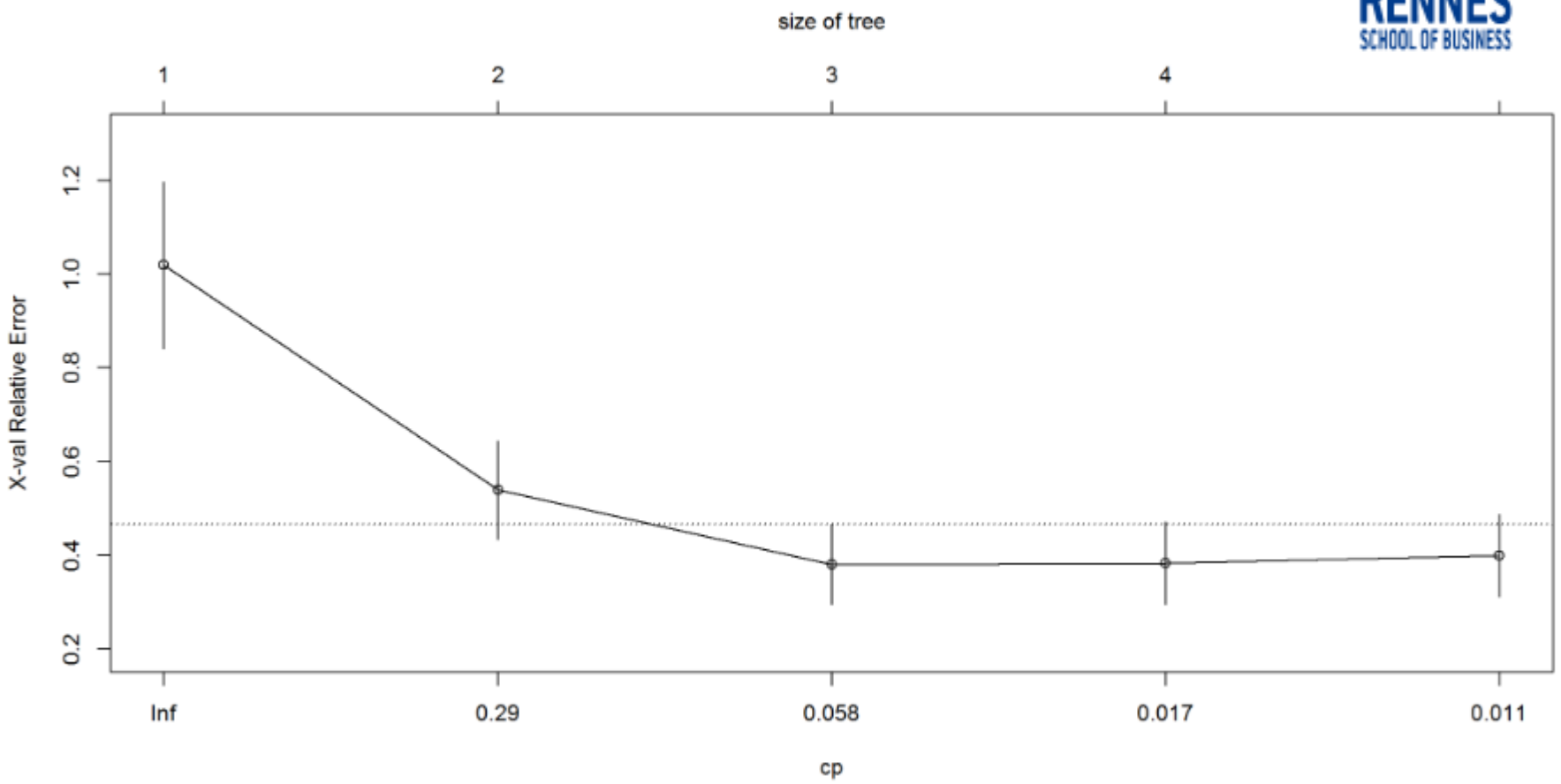
```
plot(rparty.tree)
```


1
Petal.Length
< 4.85
≥ 4.85
2
Petal.Length
7
Petal.Length
< 2.45
≥ 2.45
4
Petal.Length
< 5.15
≥ 5.15
< 4.65
≥ 4.65
Node 3 (n = 35)
Node 5 (n = 29)
Node 6 (n = 7)
Node 8 (n = 11)
Node 9 (n = 23)
1
0.8
0.6
0.4
0.2
0
setosa
virginica

```r
fit <- rpart(Mileage~Price + Country + Reliability + Type, method="anova", data
printcp(fit) # display the results
```

```
##
## 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 1.01935 0.178205
## 2 0.132061      1   0.37711 0.53891 0.104409
## 3 0.025441      2   0.24505 0.38064 0.085444
## 4 0.011604      3   0.21961 0.38348 0.087662
## 5 0.010000      4   0.20801 0.39990 0.087645
```

```r
plotcp(fit) # visualize cross-validation results
```


size of tree
1
2
3
4
X-val Relative Error
0.2
0.4
0.6
0.8
1.0
1.2
Inf
0.29
0.058
0.017
0.011
cp

```
summary(fit) # detailed summary of splits
```

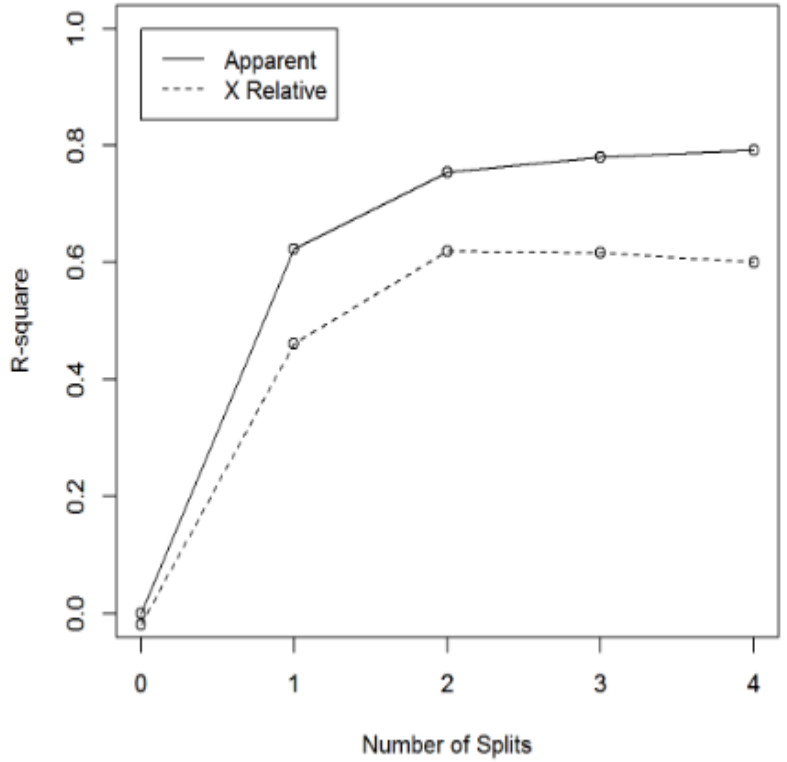
```
## Call:
## rpart(formula = Mileage ~ Price + Country + Reliability + Type,
##     data = cu.summary, method = "anova")
##   n=60 (57 observations deleted due to missingness)
##
##           CP nsplit rel error    xerror       xstd
## 1 0.62288527      0 1.0000000 1.0193513 0.17820468
## 2 0.13206061      1 0.3771147 0.5389072 0.10440911
## 3 0.02544094      2 0.2450541 0.3806396 0.08544376
## 4 0.01160389      3 0.2196132 0.3834794 0.08766209
## 5 0.01000000      4 0.2080093 0.3998985 0.08764468
##
## Variable importance
##   Price    Type Country
##      48      42      10
##
## Node number 1: 60 observations,    complexity param=0.6228853
##   mean=24.58333, MSE=22.57639
##   left son=2 (48 obs) right son=3 (12 obs)
##   Primary splits:
##       Price       < 9446.5  to the right, improve=0.6228853, (0 missing)
##       Type        splits as  LLLRLL,      improve=0.5044405, (0 missing)
##       Reliability splits as  LLLRR,       improve=0.1263005, (11 missing)
##       Country     splits as  --LRLRRRLL,  improve=0.1243525, (0 missing)
##   Surrogate splits:
##       Type    splits as  LLLRLL,     agree=0.950, adj=0.750, (0 split)
##       Country splits as  --LLLLRRLL, agree=0.833, adj=0.167, (0 split)
##
```

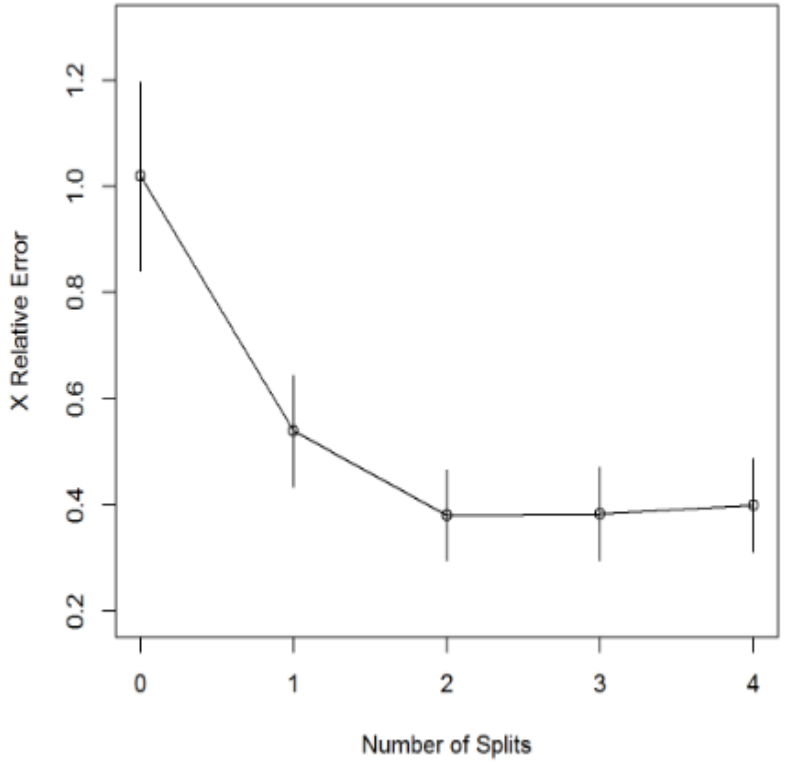
```
##   left son=4 (23 obs) right son=5 (25 obs)
##   Primary splits:
##       Type        splits as  RLLRRL,      improve=0.43853830, (0 missing)
##       Price       < 12154.5 to the right, improve=0.25748500, (0 missing)
##       Country     splits as  --RRLRL-LL,  improve=0.13345700, (0 missing)
##       Reliability splits as  LLLRR,       improve=0.01637086, (10 missing)
##   Surrogate splits:
##       Price   < 12215.5 to the right, agree=0.812, adj=0.609, (0 split)
##       Country splits as  --RRLRL-RL,  agree=0.646, adj=0.261, (0 split)
##
## Node number 3: 12 observations
##   mean=32.08333, MSE=8.576389
##
## Node number 4: 23 observations,    complexity param=0.02544094
##   mean=20.69565, MSE=2.907372
##   left son=8 (10 obs) right son=9 (13 obs)
##   Primary splits:
##       Type    splits as  -LR--L,      improve=0.515359600, (0 missing)
##       Price   < 14962   to the left,  improve=0.131259400, (0 missing)
##       Country splits as  ----L-R--R,  improve=0.007022107, (0 missing)
##   Surrogate splits:
##       Price < 13572   to the right, agree=0.609, adj=0.1, (0 split)
##
## Node number 5: 25 observations,    complexity param=0.01160389
##   mean=24.56, MSE=6.4864
##   left son=10 (14 obs) right son=11 (11 obs)
##   Primary splits:
##       Price       < 11484.5 to the right, improve=0.09693168, (0 missing)
##       Reliability splits as  LLRRR,       improve=0.07767167, (4 missing)
##       Type        splits as  L--RR-,      improve=0.04209834, (0 missing)
##       Country     splits as  --LRRR--LL,  improve=0.02201687, (0 missing)
##   Surrogate splits:
##       Country splits as  --LLLL--LR, agree=0.80, adj=0.545, (0 split)
##       Type    splits as  L--RL-,     agree=0.64, adj=0.182, (0 split)
##
```

```
##
## Node number 8: 10 observations
##   mean=19.3, MSE=2.21
##
## Node number 9: 13 observations
##   mean=21.76923, MSE=0.7928994
##
## Node number 10: 14 observations
##   mean=23.85714, MSE=7.693878
##
## Node number 11: 11 observations
##   mean=25.45455, MSE=3.520661
```

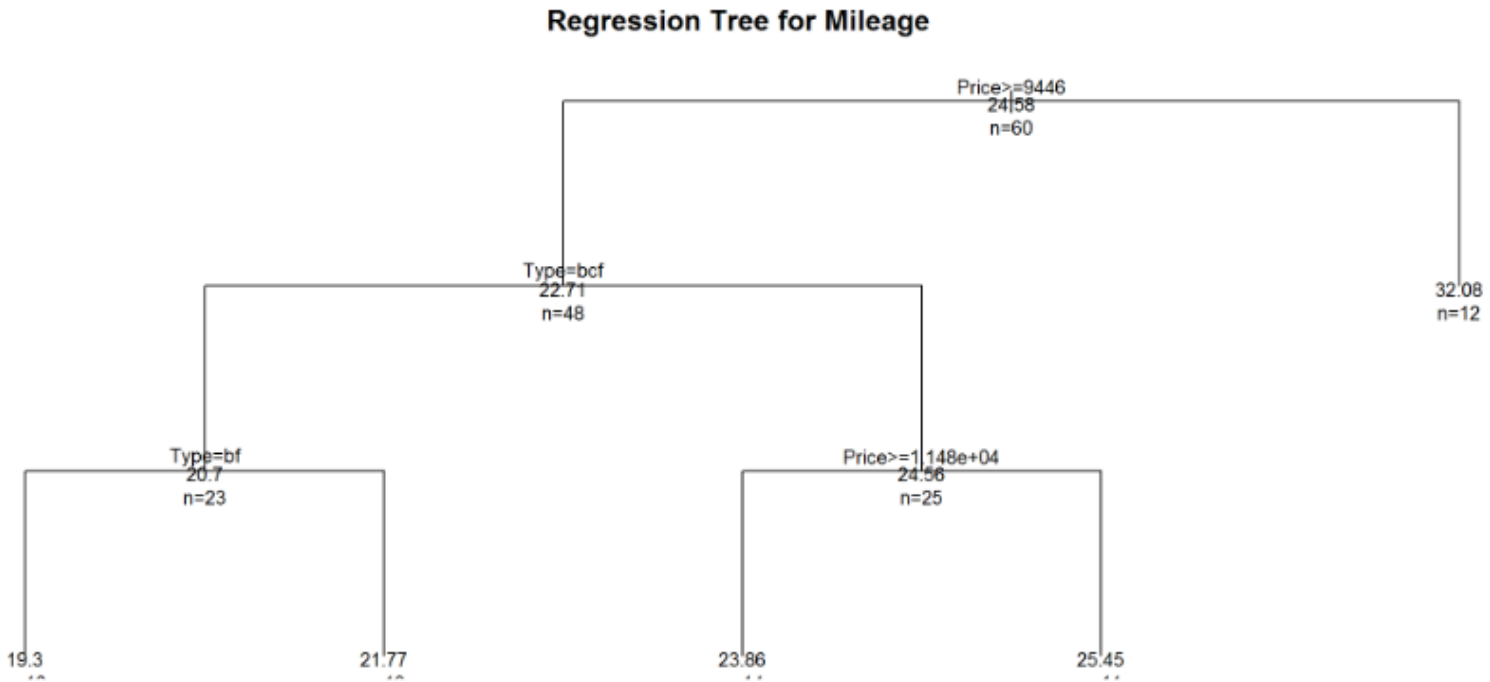
```r
# create additional plots
par(mfrow=c(1,2)) # two plots on one page
rsq.rpart(fit) # visualize cross-validation results
```

```
##
## 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 1.01935 0.178205
## 2 0.132061      1   0.37711 0.53891 0.104409
## 3 0.025441      2   0.24505 0.38064 0.085444
## 4 0.011604      3   0.21961 0.38348 0.087662
## 5 0.010000      4   0.20801 0.39990 0.087645
```


Apparent
X Relative
R-square
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
Number of Splits


X Relative Error
0.2
0.4
0.6
0.8
1.0
1.2
0
1
2
3
4
Number of Splits

```r
par(mfrow=c(1,1))

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



Random Forests

Random forests are an ensemble learning method for classification (and regression) that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes output by individual trees

Library(“randomForest”)

Random forest

Print(r)

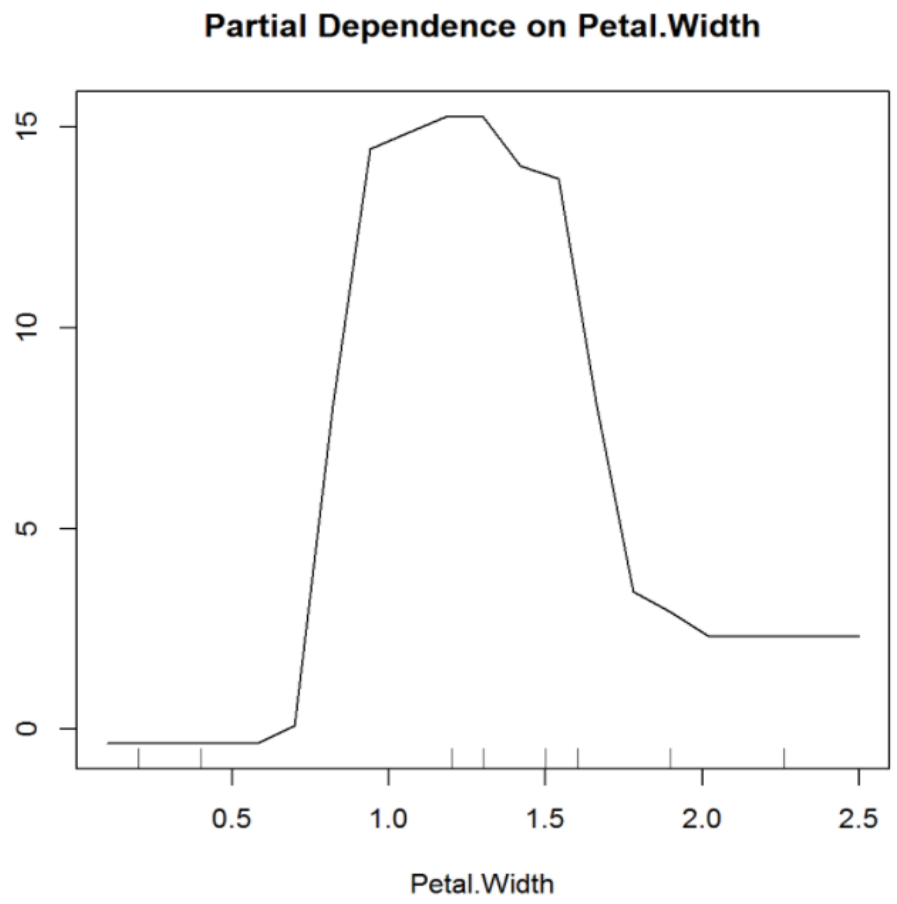
```
##
## Call:
## randomForest(formula = Species ~ ., data = train.set, importance = TRUE,
do.trace = 100, ntree = 100)
## Type of random forest: classification
## Number of trees: 100
## No. of variables tried at each split: 2
## ## OOB estimate of error rate: 4.76%
 ## Confusion matrix:
## setosa versicolor virginica class.error
## setosa 35 0 0 0.00000000
## versicolor 0 36 2 0.05263158
## virginica 0 3 29 0.09375000
```

Random Forests predictions

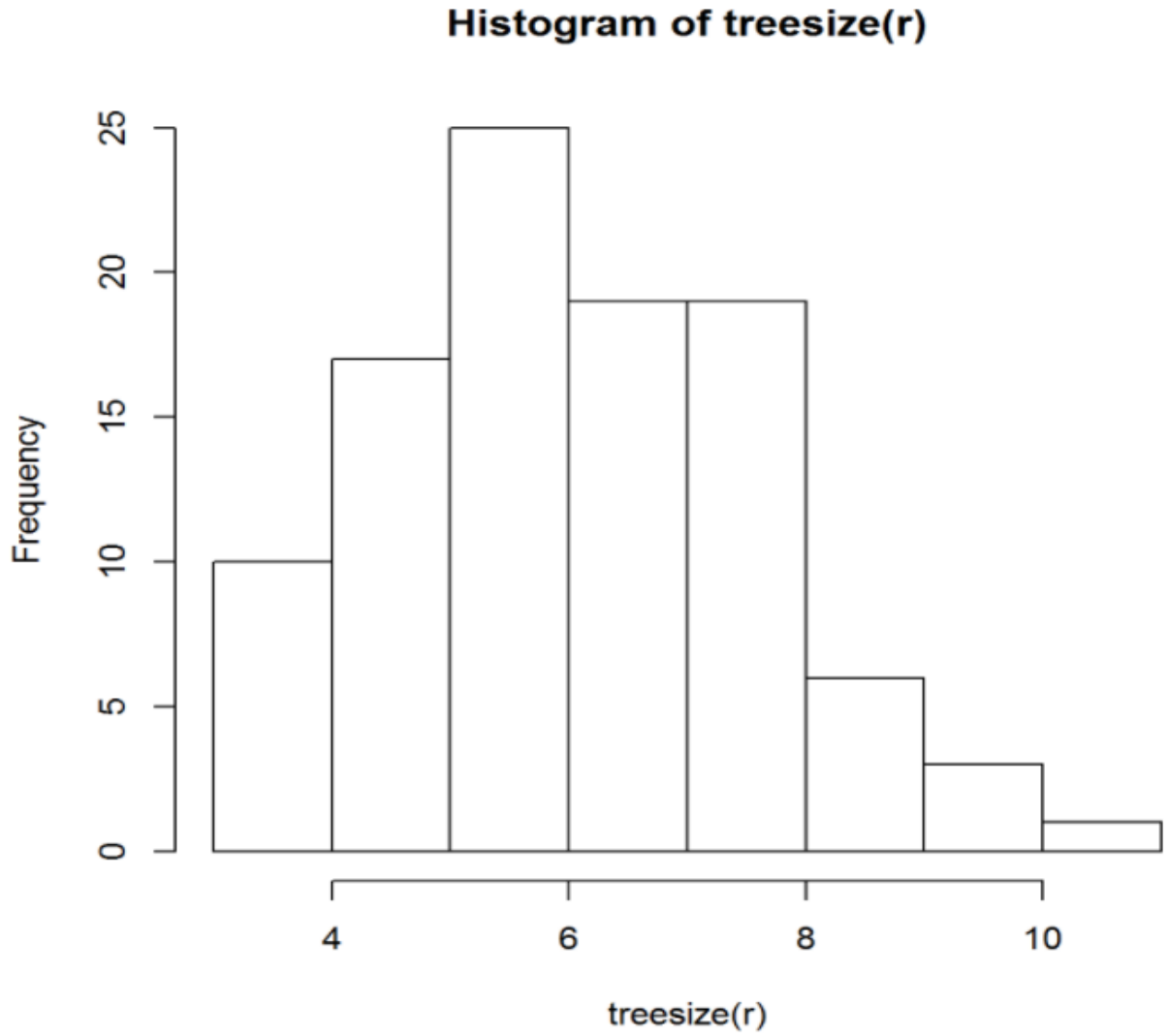
```
predictions <- predict(r, test.set)
table(test.set$Species, predictions)
```

```
##              predictions
##               setosa versicolor virginica
##   setosa          15          0         0
##   versicolor       0         12         0
##   virginica        0          2        16
```

```r
# next function gives a graphical depiction of the marginal effect of a variable on the class probability
r response (regression).
partialPlot(r, train.set, Petal.Width, "versicolor")
```



```
hist(treesize(r))
```



We can also tune the structure, ie, finding the best hyperparameters of the method via grid

```
library("e1071") # to access 'tune' method

tuned.r <- tune(randomForest, train.x = Species ~ .,
                data = train.set,
                validation.x = test.set)

best.model <- tuned.r$best.model
predictions <- predict(best.model, test.set)
table.random.forest <- table(test.set$Species, predictions)
table.random.forest
```

```
##             predictions
##              setosa versicolor virginica
##   setosa         15          0         0
##   versicolor      0         11         1
##   virginica       0          3        15
```

```
# computing overall error:
error.rate <- 1 - sum(diag(as.matrix(table.random.forest))) / sum(table.random.forest)
error.rate
```

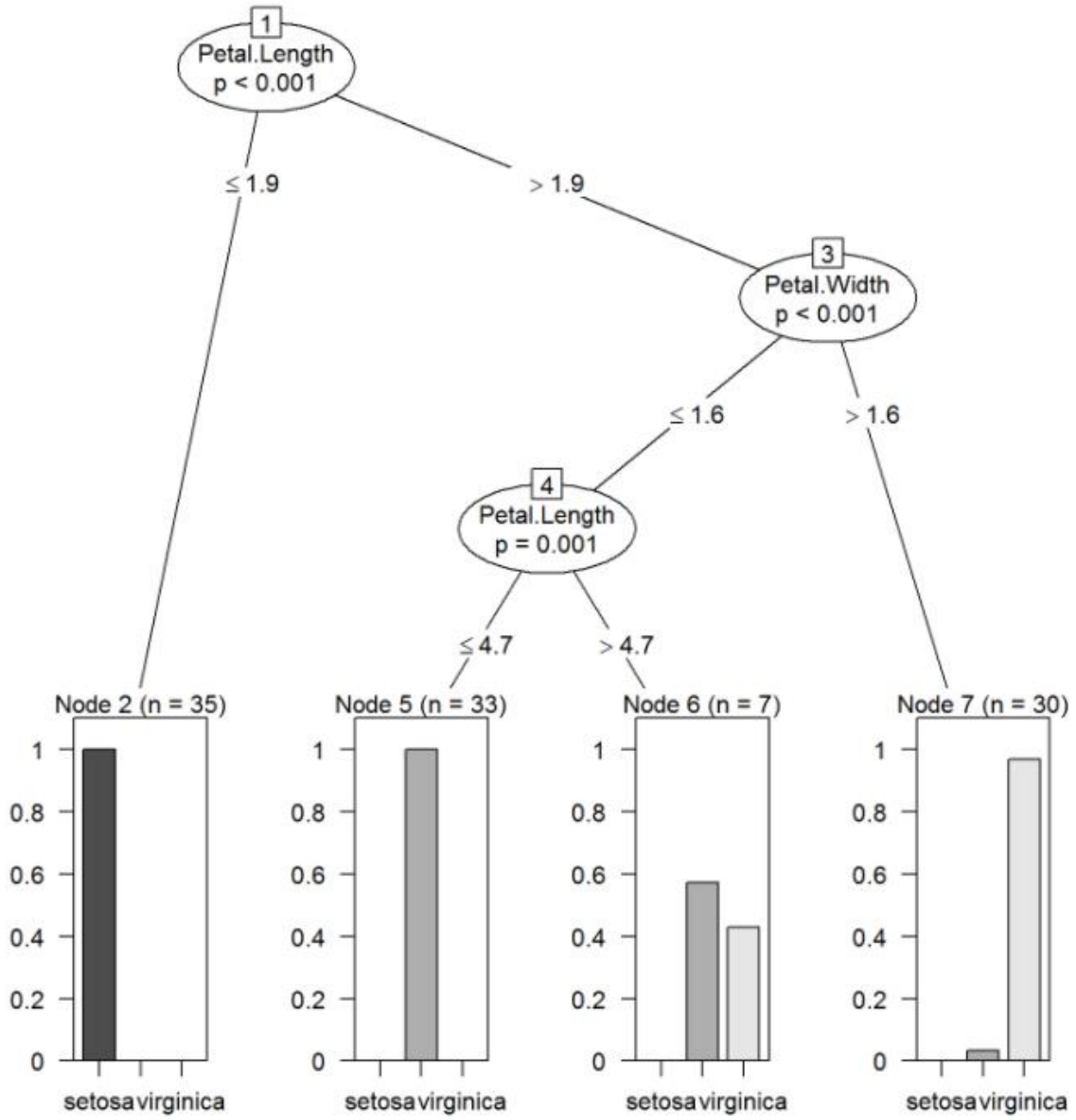
```
## [1] 0.08888889
```

Conditional Inference Trees

Conditional inference trees estimate a regression relationship by binary recursive partitioning in a conditional inference framework. Roughly, the algorithm works as follows: 1) Test the global null hypothesis of independence between any of the input variables and the response (which may be multivariate as well). Stop if this hypothesis cannot be rejected. Otherwise select the input variable with strongest association to the resonse. This association is measured by a p-value corresponding to a test for the partial null hypothesis of a single input variable and the response.

Library(party)

```
iris.model <- ctree(Species ~ . , data = train.set)
plot(iris.model)
```



```r
# The package is able to format the plot tree. Eg:
innerWeights <- function(node){
  grid.circle(gp = gpar(fill = "White", col = 1))
  mainlab <- paste( node$psplit$variableName, "\n(n = ")
  mainlab <- paste(mainlab, sum(node$weights),")" , sep = "")
  grid.text(mainlab,gp = gpar(col='red'))
}
plot(iris.model, type='simple', inner_panel = innerWeights)
```