Stat 8053, Fall 2013: Recursive Partitioning

Source: Chu, Singfat (2001), Pricing the C's of Diamond Stones, *Journal of Statistics Education Volume* 9, Number 2 (2001), `http://www.amstat.org/publications/jse/v9n2/datasets.chu.html`. Data on diamond pricing

```
loc <- "http://www.stat.umn.edu/~sandy/courses/8053/Data/diamonds.txt"
str(diamonds <- read.table(url(loc), header=TRUE))
```

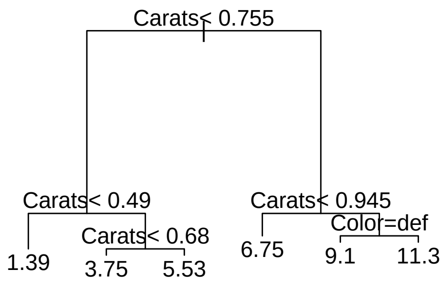
```
'data.frame':	308 obs. of  5 variables:
 $ Price  : int  823 765 803 803 705 725 967 1050 967 863 ...
 $ Carats : num  0.18 0.18 0.18 0.18 0.18 0.18 0.19 0.19 0.19 0.19 ...
 $ Cert   : Factor w/ 3 levels "GIA","HRD","IGI": 3 3 3 3 3 3 3 3 3 3 ...
 $ Clarity: Factor w/ 5 levels "IF","VS1","VS2",..: 4 5 1 1 5 1 5 1 1 4 ...
 $ Color  : Factor w/ 6 levels "D","E","F","G",..: 3 3 4 4 4 5 1 2 3 3 ...
```

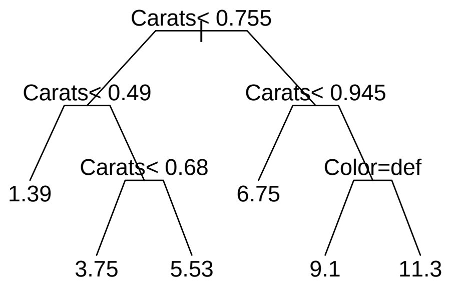
I will do the fitting with 200 cases and use 108 for validation.

```
set.seed(10131985)
est.set <- sample(1:308, 200)
val.set <- (1:308)[-est.set]
```

Fit using `rpart`, which implements the methodology in L Breiman, J. Friedman, R. Olshen and C. Stone (1984) *Classification and Regression Trees*, Wadsworth.

```
library(rpart)
r1 <- rpart(I(Price/1000) ~., diamonds, method = "anova", subset=est.set)
par(mfrow=c(1, 2), xpd = NA)
plot(r1, margin=.1)
text(r1, digits=3)
plot(r1, compress=TRUE, uniform=TRUE, branch=0.4, margin=.1)
text(r1, digits=3)
```

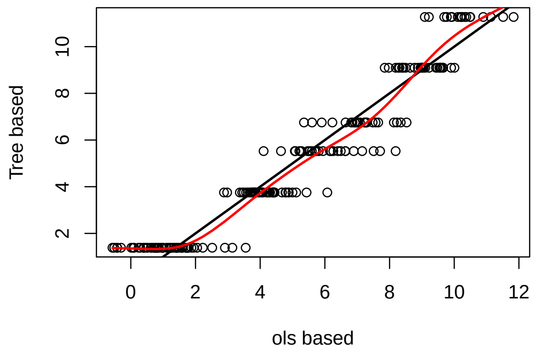




The `method = "anova"`, which is the default for a continuous response, specifies the criterion for the splitting based on within-node residual sum of squares. The numbers printed at the terminal nodes are the predicted values (I divided `Price` by 1000 so these would not be in scientific notation, and then used `text(d1, digits=3)`). The variable `Color` appears to be relevant only if `Carats` > 0.945 and thus the tree model "automatically" included an interaction. One version of tree-building is even called *automatic interaction detection.*

Compare the fit of `rpart` to `ols` fitting:

```
m1 <- lm(I(Price/1000)~ Carats + Color + Cert + Clarity, diamonds, subset=est.set)
plot(predict(r1) ~ predict(m1), ylab="Tree based", xlab="ols based")
abline(0, 1, lwd=2)
or <- order(predict(m1))
library(mgcv)
lines(predict(m1)[or], predict(gam(predict(r1)[or] ~ s(predict(m1)[or]))), col="red", lwd=2)
```



Tree-based methods predict the same value for all observations in a terminal node; this tree had only 6 nodes. `ols` can predict different values for each observation. Observations in the node with fitted value 5.3, for example, have `ols` fitted values between about 4 and 9.

Tree pruning

The more nodes, the more complex the tree, and the lower the within-node residual sum of squares. Let $\texttt{RSS}(k)$ be the residual sum of squares for the tree with k terminal nodes, for $k = 1, 2, \ldots, n$, where n_m is the tree with one node for each combination of the predictors. There is a trade-off between complexity and error. The textbook, following Brieman *et al.*, Sec 3.3, suggest ranking trees according to their *cost-complexity.*

$$\texttt{CC}(k) \quad = \quad \texttt{RSS}_k + \lambda k$$

λ is like a smoothing parameter with large λ or $\texttt{CP} = \lambda/\texttt{RSS}_1$ penalizing large trees and small λ penalizing small trees. Brieman *et al.* suggest an algorithm that fixes λ, and then finds the tree that minimizes cost complexity. This turns out to be a step function: For λ large enough, the tree with 1 node is selected. Eventually λ is small enough to prefer two nodes, and this stays true until a λ decreases to make three nodes preferable, and so on.

A long series of trees can be summarized using

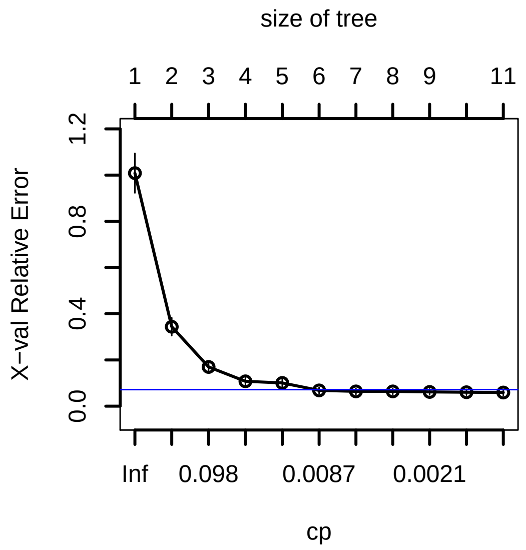
```
r2 <- update(r1, cp=0.001)
printcp(r2)
```

```
Regression tree:
rpart(formula = I(Price/1000) ~ ., data = diamonds, subset = est.set,
    method = "anova", cp = 0.001)

Variables actually used in tree construction:
[1] Carats Color

Root node error: 2229/200 = 11

n= 200

       CP nsplit rel error xerror  xstd
1  0.6931      0     1.000  1.009 0.086
2  0.1346      1     0.307  0.344 0.039
3  0.0717      2     0.172  0.170 0.026
```

```
4  0.0239      3     0.101  0.108 0.020
5  0.0237      4     0.077  0.101 0.020
6  0.0032      5     0.053  0.068 0.013
7  0.0030      6     0.050  0.064 0.013
8  0.0027      7     0.047  0.064 0.013
9  0.0016      8     0.044  0.062 0.013
10 0.0015      9     0.042  0.060 0.012
11 0.0010     10     0.041  0.059 0.012
```

The value of CP is the smallest value that produces a tree of a given number of nodes. The column nsplit $= k - 1$. The rel.error is $\texttt{RSS}(k)/\texttt{RSS}(1)$. Each of these trees is then examined using 10-fold cross-validation, in which the data are divided into 10 equal segments; the tree is built using 9 of the 10 segments, and error is assessed on the tenth segment. This is repeated leaving off each segment in turn and errors are then averaged and scaled to give xerror. The xstd is the variation between the 10 subsample estimates.

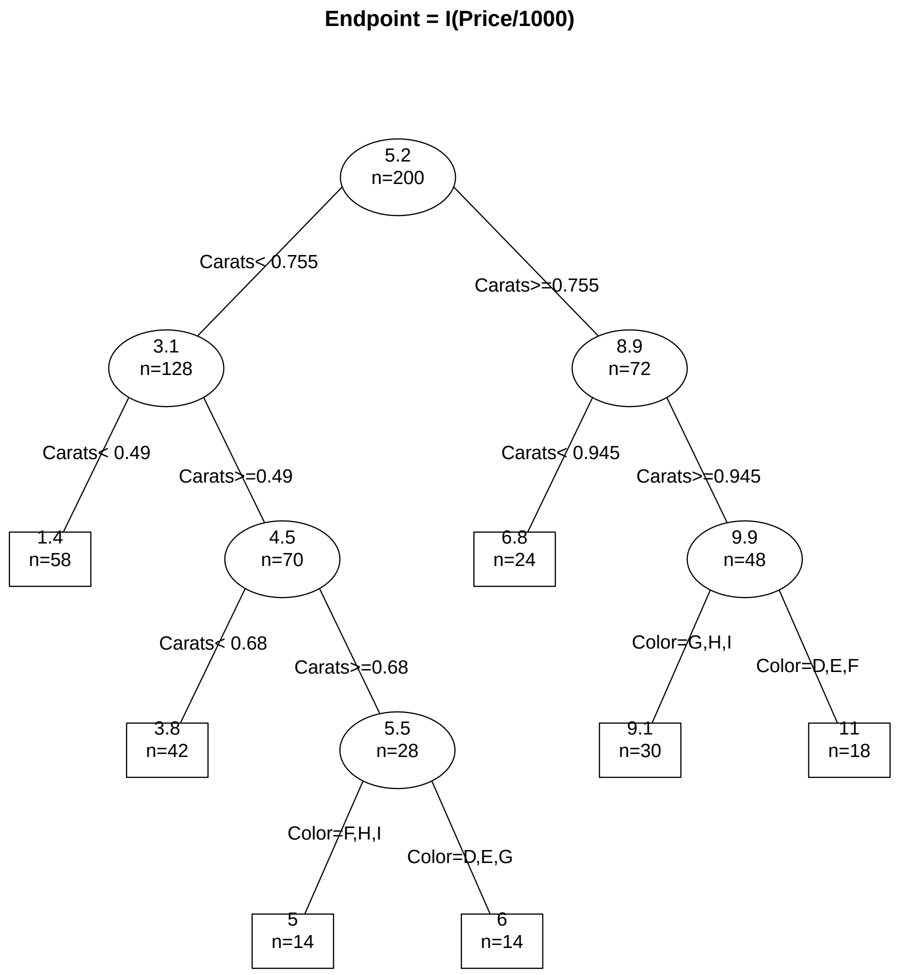
```
plotcp(r2, minline=TRUE, col="blue", lwd=2, lty=1)  # draw line 1 SD above minimum rel.error
```



The horizontal axis in the graph is really the size of the tree, nsplit $+ 1$, indicated across the top. The corresponding cp values are shwon across the bottom. Solutions of at least 6 nodes seem useful.

Here are the pruned tree and the original tree:

```
par(xpd = NA)
r3 <- prune(r2, cp=r2$cptable[7, 1])
post(r3, filename="")   # Fancy summary plot
```



How well does this work? Let's compute in-sample and validation sample errors for the tree, for ols, and for stepwise regression:

```
get.error <- function(m, which){
                 with(diamonds, sqrt(sum(((Price[which]/1000) -
                     predict(m, diamonds[which, ]))^2)/length(which)))}
summary(m2 <- step(m1, trace=0))$coef

             Estimate Std. Error  t value   Pr(>|t|)
(Intercept)  0.02858     0.2836   0.1008  9.198e-01
Carats      12.43507     0.1997  62.2662 6.791e-128
ColorE      -1.00668     0.2616  -3.8475  1.631e-04
ColorF      -1.67597     0.2512  -6.6710  2.729e-10
ColorG      -1.89443     0.2533  -7.4799  2.719e-12
ColorH      -2.48034     0.2545  -9.7446  1.948e-18
ColorI      -2.93257     0.2622 -11.1849  1.295e-22
ClarityVS1  -1.33436     0.1793  -7.4430  3.376e-12
ClarityVS2  -1.67492     0.1935  -8.6543  2.144e-15
ClarityVVS1 -0.57052     0.1873  -3.0463  2.648e-03
ClarityVVS2 -1.11086     0.1796  -6.1867  3.722e-09

(ans <- data.frame(rpart=c(get.error(r1, est.set), c(get.error(r1,val.set))),
                   pruned=c(get.error(r3, est.set), c(get.error(r3,val.set))),
                   ols = c(get.error(m1, est.set), c(get.error(m1, val.set))),
                   ols.step= c(get.error(m2, est.set), c(get.error(m2, val.set))),
                   row.names=c("Estimation", "Validation")))

             rpart pruned    ols ols.step
Estimation 0.7691 0.7454 0.6627   0.6655
Validation 0.8811 0.8810 0.7861   0.7829
```

Recursive partitioning, at least in the simple-minded way done here, actually does considerably worse than `ols` or `ols` with stepwise selection.

Classification

For this example we use data collected on 98 patients diagnosed with Glaucoma and 98 patients without Glaucoma, an eye disease. On each patient there are 62 variables measured, all of which are continuous. The data are in the `TH.data` package.

```
data(GlaucomaM, package="TH.data")
```

Squared error loss is not appropriate for classification. We let p_{ik} be the fraction of points in node i that fall in response category k, so $\sum_k p_{ik} = 1$. A node will be *pure* if one of the $p_{ik} = 1$. The worst case for prediction occurs if all the p_{ik} for each i are all equal.

By default `rpart` uses the *Gini index* to measure lack of fit. For node i,

$$D_i = 1 - \sum p_{ik}^2$$

$D_I = 0$ for a pure node, and $D_i = 1/2$ for a binary response with $p_{i1} = p_{i2}$. Large values of $D = \sum D_i$ correspond to poor trees. Other choices of error measures are available in `rpart` but I haven't been able to figure out how to get them with the terrible documentation that is available.

```
par(mfrow=c(1, 2), xpd=NA)
set.seed(123456)
const <- sample(1:170, 125)
g1 <- rpart(Class ~ . , GlaucomaM, subset=const)
xtabs(~ predict(g1, type="class") + GlaucomaM$Class[const])
```

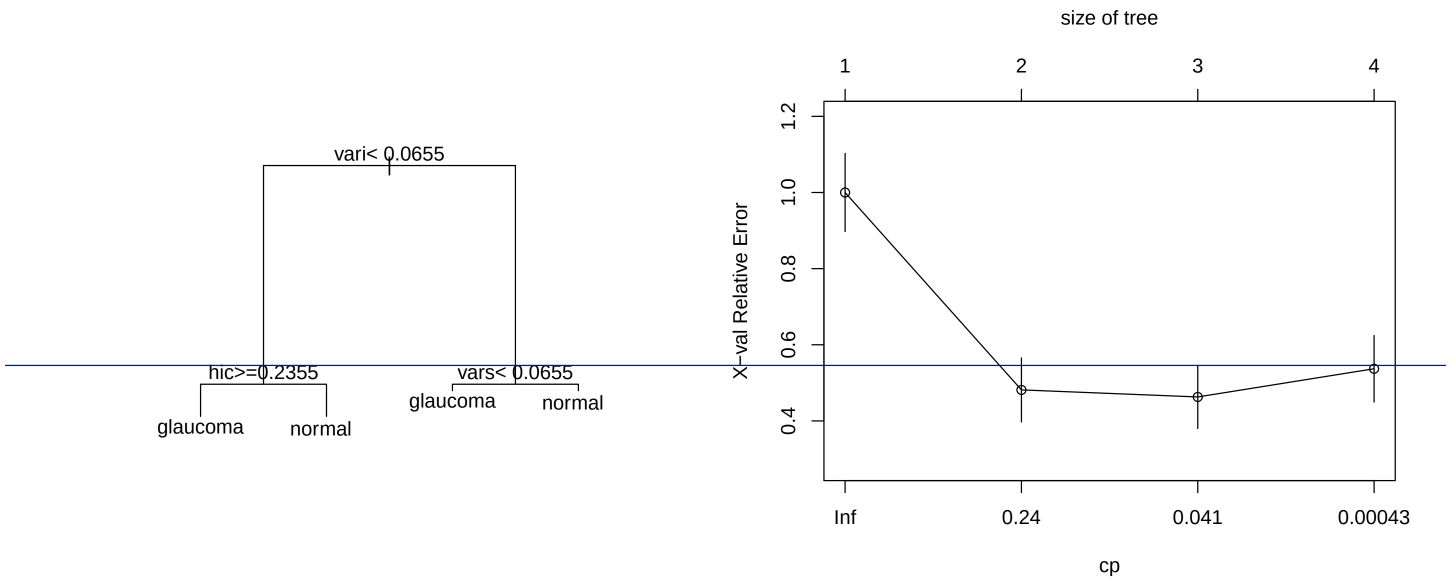
```
                           GlaucomaM$Class[const]
predict(g1, type = "class") glaucoma normal
                   glaucoma       48      8
                   normal          6     63
```

```
plot(g1, margin=.2)
text(g1)
g2 <- update(g1, cp = 0.00001)
printcp(g2)
```

```
Classification tree:
rpart(formula = Class ~ ., data = GlaucomaM, subset = const,
    cp = 1e-05)
```

```
Variables actually used in tree construction:
[1] hic  vari vars

Root node error: 54/125 = 0.43

n= 125

       CP nsplit rel error xerror  xstd
1 0.62963      0      1.00   1.00 0.103
2 0.09259      1      0.37   0.48 0.084
3 0.01852      2      0.28   0.46 0.083
4 0.00001      3      0.26   0.54 0.087
```

```
plotcp(g2, lty=1, col="blue")
```



A summary of fit for a classification problem is to count correct and incorrect classifications. We do this for the construction and for the validation sets.

```
show <- function(tt){
    print(tt)
```

```
     cat(paste("Misclassification rate =",round(1-sum(diag(tt))/sum(tt), 2),"\n"))
     invisible()}
 show(with(GlaucomaM, table(actual=Class[const],
           predicted=predict(g1, type="class"))))
```

```
           predicted
actual      glaucoma normal
  glaucoma        48      6
  normal           8     63
Misclassification rate = 0.11
```

```
 show(with(GlaucomaM, table(actual=Class[-const],
           predicted=predict(g1, newdata=GlaucomaM[-const,], type="class"))))
```

```
           predicted
actual      glaucoma normal
  glaucoma        37      7
  normal           6     21
Misclassification rate = 0.18
```

For comparison, let's do the same thing with logistic regression.

```
 m1 <- glm(Class ~., binomial, GlaucomaM, subset=const)
 show(with(GlaucomaM, table(actual=Class[const],
            predicted=ifelse(predict(m1) <= 0, 0 ,1))))
```

```
           predicted
actual       0  1
  glaucoma 54  0
  normal    0 71
Misclassification rate = 0
```

```
 show(with(GlaucomaM, table(actual=Class[-const],
            predicted=ifelse(predict(m1, newdata=GlaucomaM[-const,]) <= 0, 0, 1))))
```

```
          predicted
actual       0  1
  glaucoma 34 10
  normal   12 15
Misclassification rate = 0.31
```

```
m2 <- update(m1, ~ hic + phcg + tms + varg)
show(with(GlaucomaM, table(actual=Class[-const],
           predicted=ifelse(predict(m2, newdata=GlaucomaM[-const,]) <= 0, 0, 1))))
```

```
          predicted
actual       0  1
  glaucoma 35  9
  normal    4 23
Misclassification rate = 0.18
```

Bagging

Recursive partitioning solutions are rather fragile because only one split is considered at a time. Thus, small perturbations in the data are likely to lead to completely different solutions. One possible way to deal with this problem is to recompute the tree based on slightly perturbed data, and then average over the perturbations. Leo Brieman proposed this idea, and called it bootstrap aggregation, or *bagging*.

Here is the general idea. We start with data $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ from which we can estimate a function $\hat{f}(x)$ that can return predictions at the point x. For example, $\hat{f}(x)$ can be from regression tree, an additive model or almost any other fitting procedure. Suppose that $\mathcal{D}^b$ is a bootstrap sample of size n from $\mathcal{D}$, and $\hat{f}^b(x)$ is the corresponding prediction. Brieman's idea was to replace $\hat{f}(x)$ by $\mathrm{E}(\hat{f}^b(x))$, the average over all possible bootstrap samples, as a more stable prediction. In practice this is estimated by

$$\hat{f}_{\text{bag}} = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^b(x)$$

the average of B bootstrap samples.

The *cost* of bagging is that there is no longer a simple tree representation of the result, as predictions are now done by averaging B trees which can have different complexities, number of nodes, and so on.

Bagging is implemented in the package `ipred`.

```
library(ipred)
b1 <- bagging(Class ~. , GlaucomaM, subset=const, nbagg=100)
show(with(GlaucomaM, table(actual=Class[const],predicted=predict(b1))))
```

```
          predicted
actual     glaucoma normal
  glaucoma       42     12
  normal         11     60
Misclassification rate = 0.18
```

```
show(with(GlaucomaM, table(actual=Class[-const],
      predicted=predict(b1, newdata=GlaucomaM[-const,]))))
```

```
          predicted
actual     glaucoma normal
  glaucoma       36      8
  normal          3     24
Misclassification rate = 0.15
```

Bagging presents a considerable improvement over fitting a single tree.

Random Forests

Random forests are an elaboration of bagging due to Breiman in which each tree is grown using an algorithm that is more complicated that simply using recursive partitioning. The package `randomForest` is used for the fitting and includes some interesting summary graphs. Slightly paraphrased from `http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#overview`, each tree is grown as follows:

1. The training sample consists of N observations. Create a "bootstrap" sample of size N by sampling with replacement from the N observations. This will divide the data into the cases included in the bootstrap sample and a random number of observations that are "out of bootstrap" samples.

2. If there are M input variables, a number $m << M$ is specified [and fixed]. At each node, m variables are selected at random out of the M and the best split on these m is used to split the node.

3. Each tree is grown to the largest extent possible. There is no pruning.

For each tree grown, lack of fit is measured either using the usual within-sample error, or else it is based on the "out of bootstrap" error. By default 500 trees are grown. The final solution is a combination of the 500 trees.

```
library(randomForest)
b2 <- randomForest(Class ~., GlaucomaM, subset=const)
show(with(GlaucomaM, table(actual=Class[const], predicted=predict(b2))))
```

```
          predicted
actual     glaucoma normal
  glaucoma       45      9
  normal          5     66
Misclassification rate = 0.11
```

```
show(with(GlaucomaM, table(actual=Class[-const],
      predicted=predict(b2, newdata=GlaucomaM[-const, ]))))
```

```
          predicted
actual     glaucoma normal
  glaucoma       34     10
  normal          2     25
Misclassification rate = 0.17
```

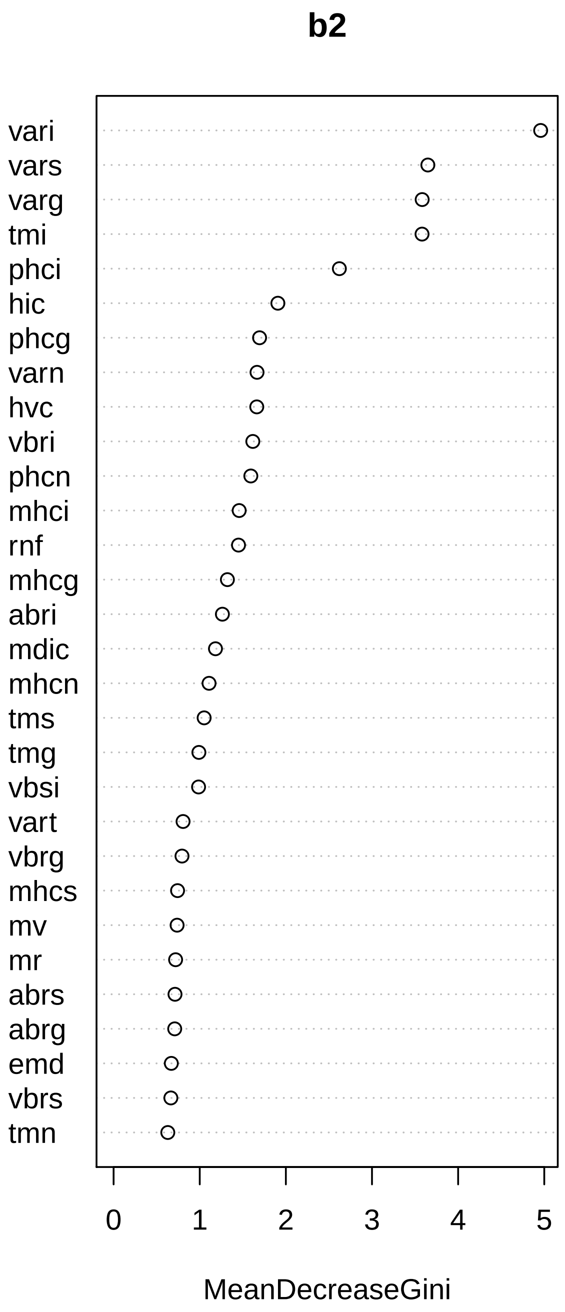
Variable importance

Consider predictor ℓ and a particular node. Let $I(v(t) = \ell)$ be an indicator function whose value is 1 if the t-th node in the tree is a split on variable ℓ, and equal to zero otherwise. Let $\hat{i}_t^2$ be the decrease in risk (generally squared error loss, but any loss function you like) for including node $v(t)$. Then the importance of variable ℓ in the tree is

$$I_\ell^2 = \sum_{i=1}^{J-1} \hat{i}_t^2 I(v(t) = \ell)$$

where J is the number of nodes in the tree. For many trees, average I over all the trees. Large values of I_ℓ^2 correspond to important variables.

```
varImpPlot(b2,type=2)
```


b2
vari
vars
varg
tmi
phci
hic
phcg
varn
hvc
vbri
phcn
mhci
rnf
mhcg
abri
mdic
mhcn
tms
tmg
vbsi
vart
vbrg
mhcs
mv
mr
abrs
abrg
emd
vbrs
tmn
0
1
2
3
4
5
MeanDecreaseGini

For this example there are perhaps 13 predictors with some “importance”, with the top 4 much more important

Visualizing Variable Importance: Partial Dependence Plots

Suppose that X_k is a variable of interest, and $X_{\backslash k}$ is the vector of remaining predictors. Consider the problem of display of the dependence of the response on X_k that is implied by the model. If the true mean function is additive

$$E_T(Y|X_k, X_{\backslash k}) = h_1(X_k) + h_2(X_{\backslash k}) \tag{1}$$

then a plot of $\widehat{h_1}(X_k)$ versus X_k, as is done in the `mgcv` package is the appropriate plot. This of course assumes that we have an estimated the function h_1, but in methods that average over models like random forests, bagging or other model aggregation methods, we may not have an explicit estimate for h_1. If the true mean function is not additive,

$$E_T(Y|X_k, X_{\backslash k}) = h_1(X_k) + h_2(X_{\backslash k}) + h_{12}(x_k, x_{\backslash k}) \tag{2}$$

then the idea that there is an “effect” due to X_1 alone is in doubt, and conditioning on the value of $x_{\backslash k}$ may be required to get a summary graph. We are used to these ideas in the *interaction graphs* that are commonly used with factorial models. Similar graphs are implemented in the `effects` package in R, but not (yet) for tree models.

Partial Response Plot

If (1) holds, then:

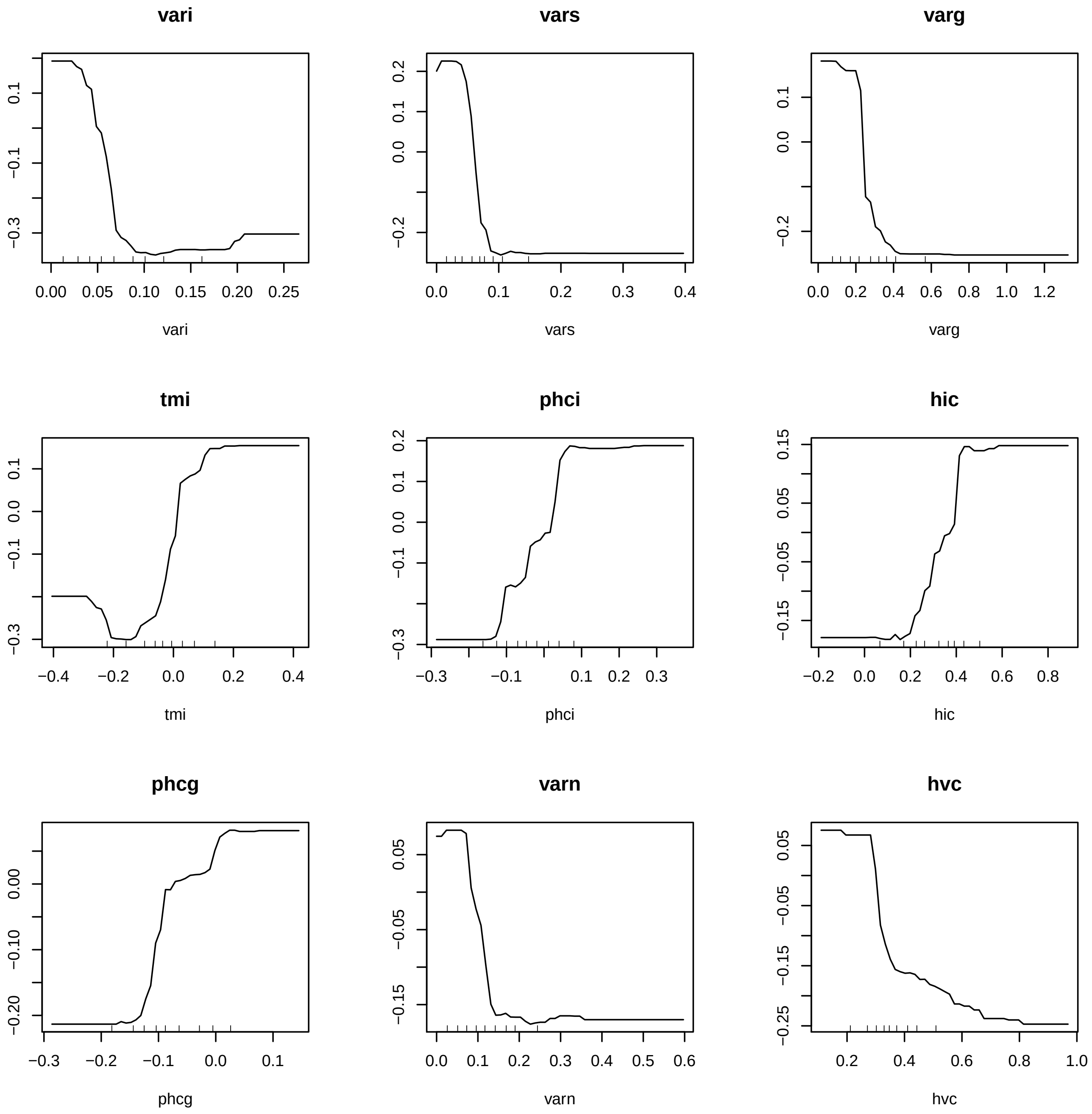
$$\begin{aligned} \mathrm{E}(Y|X_k) &= \mathrm{E}(E_T(Y|X_k, X_{\backslash k})|X_k) \\ &= \mathrm{E}(h_1(X_k) + h_2(X_{\backslash k})|X_k) \\ &= h_1(X_k) + \mathrm{E}(h_2(X_{\backslash k})|X_k) \end{aligned}$$

Friedman, (2001 *Annals of Stats*, 29, 1189–1232) suggested the the right side of this last equation can be estimated by a sum,

$$\frac{1}{n}\sum_{i=1}^{n} \hat{E}(Y|X_k, X_{\backslash k} = x_{i,\backslash k}) \tag{3}$$

where the $x_{i,\backslash k}$ are the rows of $X_{\backslash k}$, and n is the sample size. If more information is available, one could average over observations from the conditional distribution of $X_{\backslash k}|X_k$. The plot with X_k on the horizontal axis and (3) on the vertical axis is called a partial response plot.

```
importanceOrder <- order(-b2$importance)
names <- rownames(b2$importance)[importanceOrder][1:9]
par(mfrow=c(3, 3), xpd=NA)
for (name in names)
    partialPlot(b2, GlaucomaM, eval(name), main=name, xlab=name)
```


vari
vars
varg
tmi
phci
hic
phcg
varn
hvc