DECISION TREES

The first chunk of code below simply reads in the UCI Breast Cancer data and creates a training and test dataset. We've skipped the validation step because we're just building one default tree and seeing how it performs on a holdout sample.

```
> options(digits=2)
> load("breast_cancer.Rdata")
> set.seed(7515)
> perm=sample(1:699)
> BC_randomOrder=BCdata[perm,]
> train = BC_randomOrder[1:floor(0.75*699),]
> test = BC_randomOrder[(floor(0.75*699)+1):699,]
```

Now we'll build a default decision tree model using entropy as the metric for building the tree. The rpart package is the most popular for building decision trees.

```
> library("rpart")
> tree = rpart(Target ~ . - Target, data=train, method='class',
+              parms = list(split='entropy'))
```

To display the decision tree (and then label it), use the plot() and text() functions. The option uniform=T causes all the branches of the tree to have the same length. The first two commands and the last command in this chunk of code are merely used to set and reset the margins of the plot window. You will notice that the default tree plots leave much to be desired. We'll fancify them later in this code!

```
> .pardefault = par()
> par(mai=c(.2,.2,.2,.2))
> plot(tree, uniform=T)
> text(tree)
> #text(tree, use.n=T)
> par(.pardefault)
```

All we see in the resulting tree (Figure 1) is the *decision* at every leaf. While this tells us the most prevalent target class of each leaf, it does not display the predicted probability of that outcome in that leaf. We can get that information by adding the option use.n=T to the text() function, but it is almost invariably unreadable!

Unfortunately the tree does not even tell us *how* the split should be used. For example, the first split in the tree is '$V14 \geq 52.5$.' However, the tree does not indicate whether observations meeting that constraint should proceed down the left or right branches of the tree! I'll save you some trouble and let you know that the 'TRUE' values, i.e. those that meet the given condition, are sent down the LEFT branch of the tree.

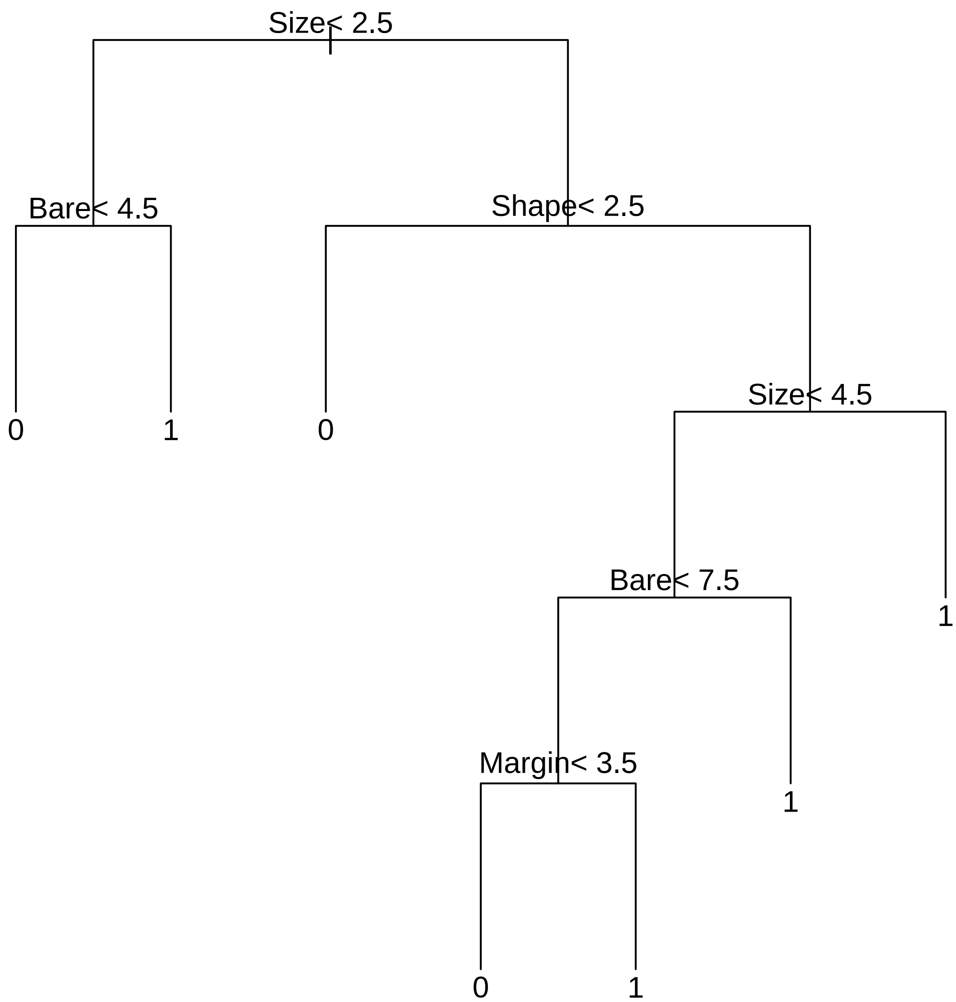


Figure 1: Default Decision Tree for Breast Cancer Data

Variable Importance

We can also obtain statistics describing variable importance. In general, variable importance is typically calculated using some impurity function, $i(\dot{})$ that measures the impurity of a given node like *entropy* or the *Gini coefficient*. Recall that we decide to split some node if the change in that impurity measure is significant. If t_L is the left child of a split and t_R is the right child of a given node t, and N_L, N_R, N are the number of observations in the left, right and root node respectively, then the change in that impurity measure is calculated as

$$\Delta i(t) = i(t) - \frac{N_L}{N} i(t_L) - \frac{N_R}{N} i(t_R).$$

We use that change in impurity measure then to calculate the **variable importance** of variable $\mathbf{x}_k$, denoted $I(\mathbf{x}_k)$ by adding up the weighted impurity decreases for all nodes t where $\mathbf{x}_k$ is used. If $\mathbf{x}_k$ is only used in one split, then the variable importance is just the impurity decrease for that one split. (Note: if using random forests, we can average this importance over all trees in the forest.)

$$I(\mathbf{x}_k) = \sum_t \frac{N_t}{N} \Delta i(t)$$

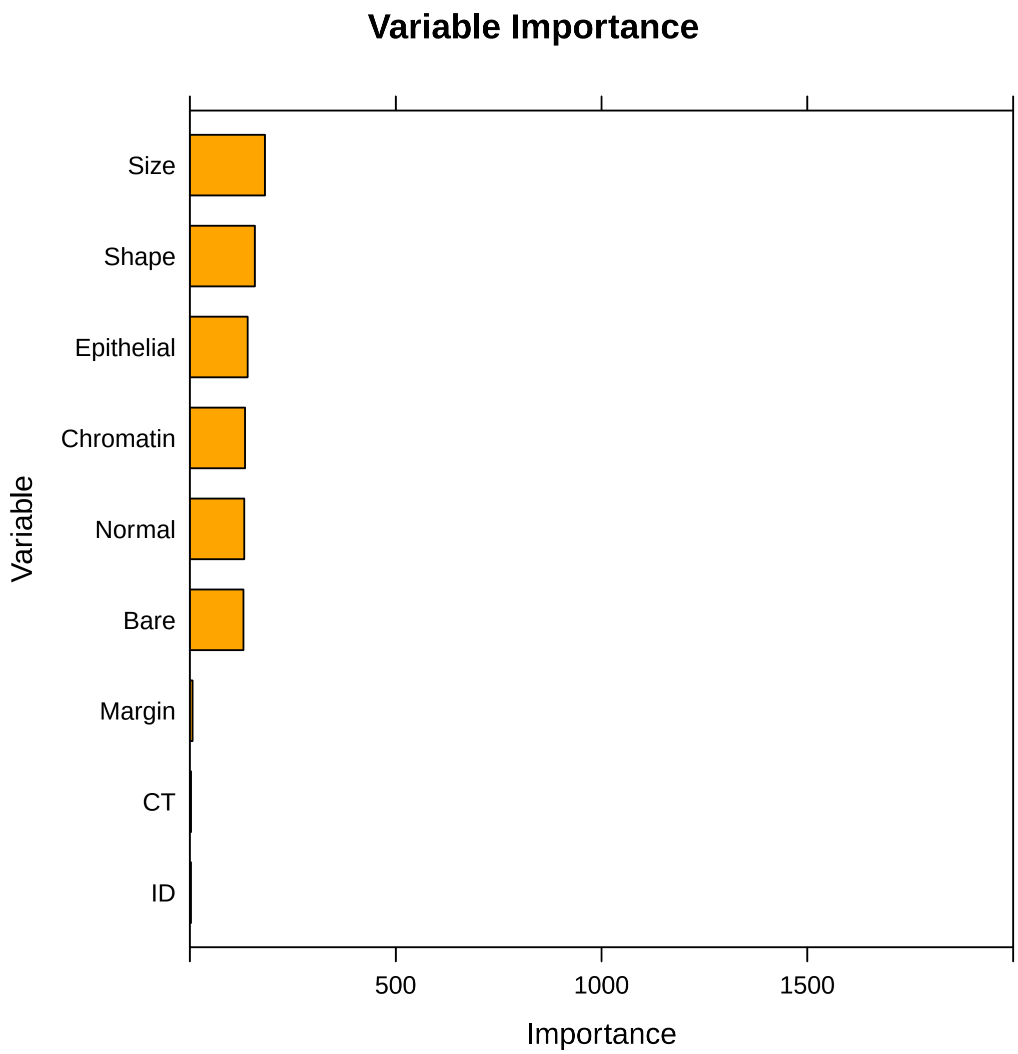
For our BreastCancer dataset, we can examine the variable importance measures using the following code:

```
> tree$variable.importance
```

```
     Size       Shape Epithelial  Chromatin     Normal       Bare     Margin
    182.5       157.8      140.1      134.1      132.1      130.0        6.3
       CT          ID
      2.8         2.6
```

Or create a bar plot to quickly investigate the relative differences,

```
> library('lattice')
> barchart(tree$variable.importance[order(tree$variable.importance)],
+         xlab = 'Importance', horiz=T, xlim=c(0,2000),ylab='Variable',
+         main = 'Variable Importance',cex.names=0.8, las=2, col = 'orange')
```



Computing Validation Misclassification

R has a nice function, predict(), which allows us to apply a model to a new set of data. The resulting output is just a vector or matrix of predictions. We can either output an $n \times 2$ matrix of predicted probabilites (for each observation we predict the probability that it belongs to each possible class (malignant or benign)) *or* we can just predict a class (0 or 1) for each observation – these options are chosen by the type= option. We'll then work with those predictions to determine the validation misclassification rate ourselves. Let's compute training misclassification while we're at it.

```
> tscores = predict(tree,type='class')
> scores = predict(tree, test, type='class')
> cat('Training Misclassification Rate:',
+     sum(tscores!=train$Target)/nrow(train))
```

```
Training Misclassification Rate: 0.038
```

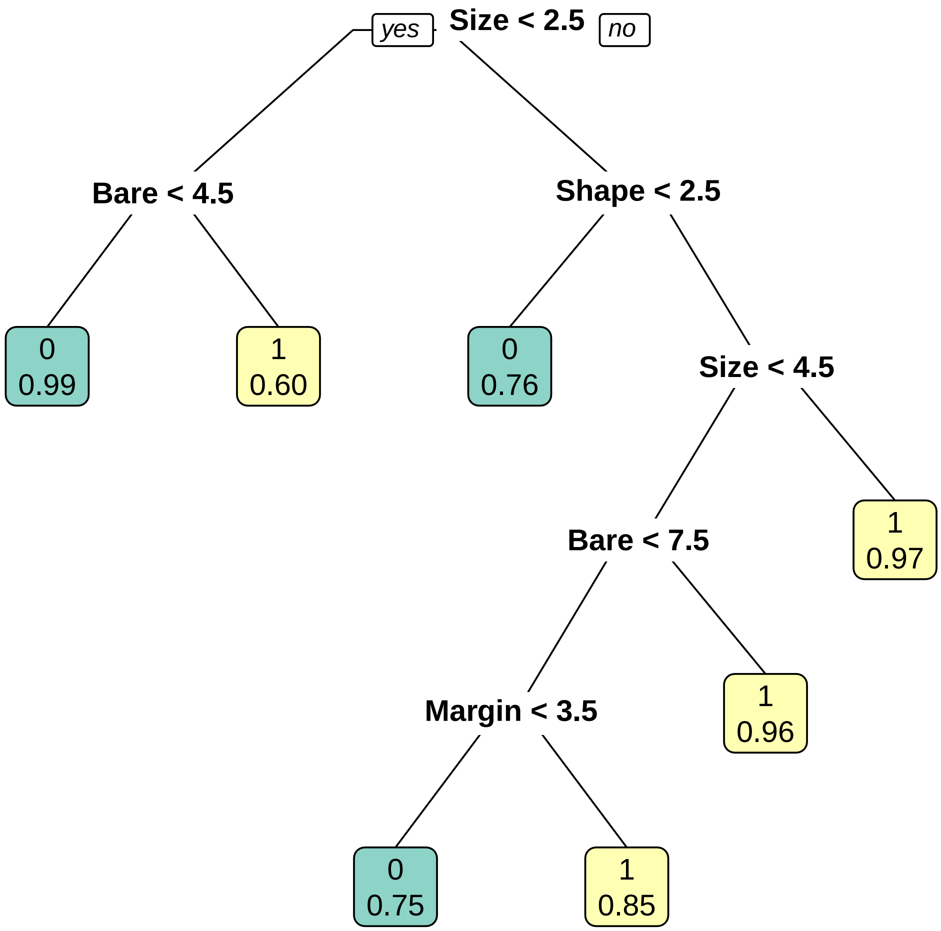
```
> cat('Validation Misclassification Rate:',
+     sum(scores!=test$Target)/nrow(test))
```

```
Validation Misclassification Rate: 0.074
```

Visual Aesthetics

As you've already noticed, the tree plots straight out of the rpart package leave much to be desired. Luckily others have put some work into giving us more plot options. Check out the documentation for the prp() function in the rpart.plot library. In that documentation you will find a variety of descriptions for potential display options as specified by the type= and extra= options. Below we look at some quick examples of how these plots can look a little more appealing.

```
> library("rattle")                              # Fancy tree plot
> library("rpart.plot")                          # Enhanced tree plots
> library("RColorBrewer")                  # Color selection for fancy tree plot
> library("party")                                          # Alternative decision tree algorithm
> library("partykit")                                  # Convert rpart object to BinaryTree
> # fancyRpartPlot(tree)       # Looks completely terrible but has
> #                            # potential for smaller trees, fewer classes
> #
> # prp(tree)
> # prp(tree, type =3, extra=100) # label branches, label nodes with % of obs
> # prp(tree, type =3, extra=2) # label branches, label nodes with misclass rate
> # prp(tree, type =3, extra=8) # label branches, label nodes with pred prob of class
> # # BEWARE WITH BINARY TREES WHERE WE TYPICALLY WANT TO SHOW PROB OF SUCCESS/FAILURE
> # # FOR EVERY NODE IN THE TREE!
> prp(tree, type =0, extra=8, leaf.round=1, border.col=1,
+     box.col=brewer.pal(10,"Set3")[tree$frame$yval], )
```


yes
Size < 2.5
no
Bare < 4.5
Shape < 2.5
0
0.99
1
0.60
0
0.76
Size < 4.5
Bare < 7.5
1
0.97
Margin < 3.5
1
0.96
0
0.75
1
0.85