6 Tree-Based Methods

Classification and regression trees are similar *supervised* techniques which are used to analyse problems in a step-by-step approach. We start (even yet again) with the iris data where the objective is to find a set of rules, based on the four measurements we have, of classifying the flowers into on of the fours species. The rules will be of the form:

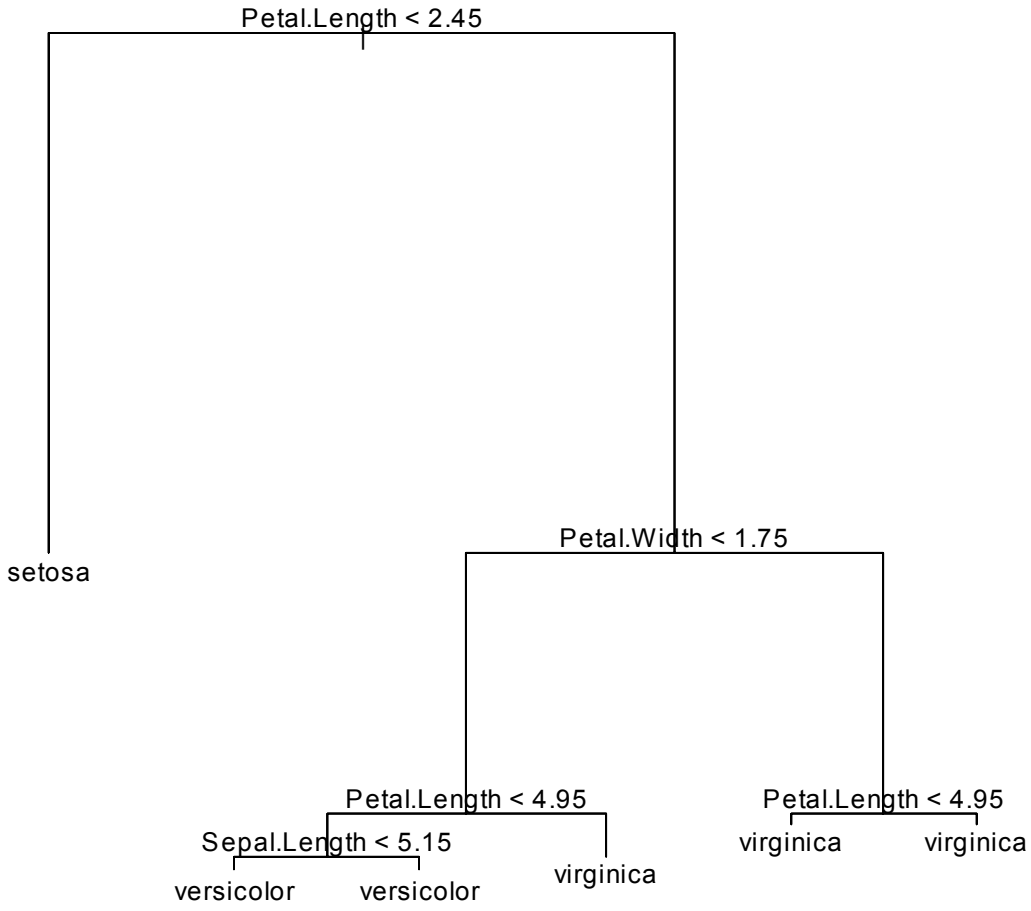
'if petal length>x then…. , but if petal length $\leq$ x then something else'

i.e. the rules are based on the values of one variable at a time and gradually partition the data set into groups.

```
> data(iris)
> attach(iris)
> ir.tr<-tree(Species~.,iris)
> plot(ir.tr)
> summary(ir.tr)

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
> text(ir.tr,all=T,cex=0.5)
```

Now look at the graphical representation:



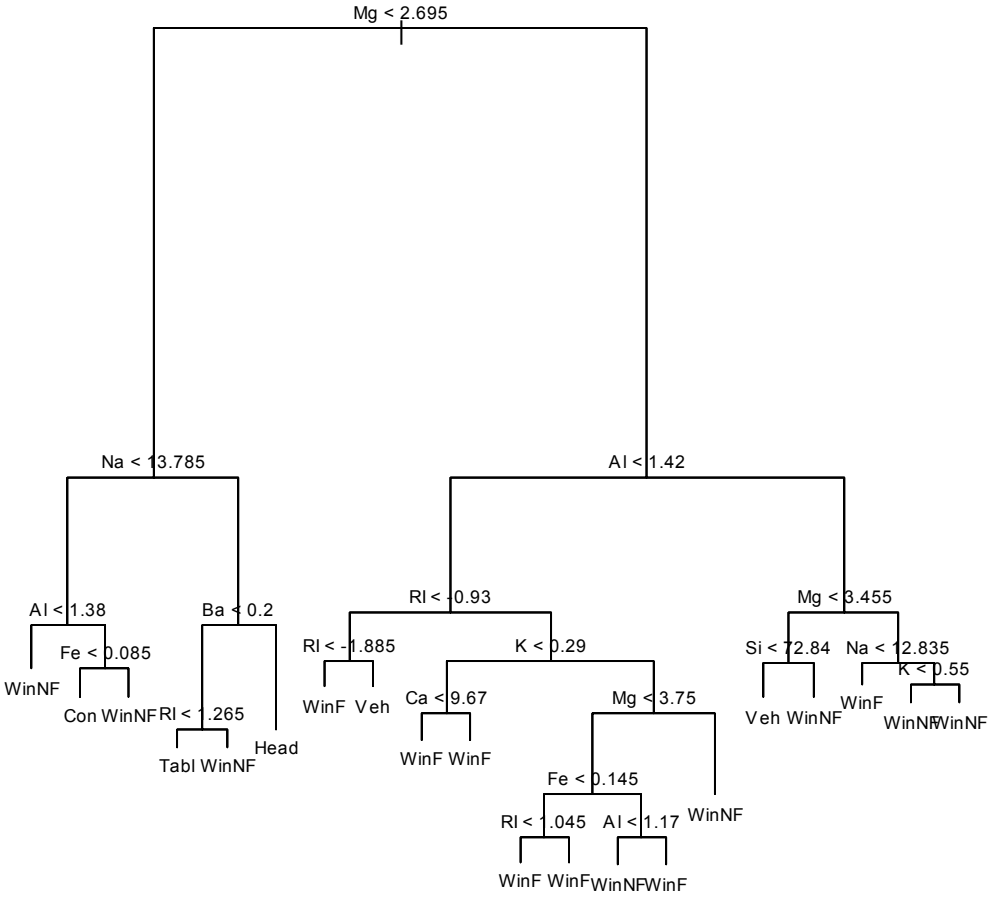

```
> ir.tr
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

 1) root 150 329.600 setosa ( 0.33333 0.33333 0.33333 )
   2) Petal.Length < 2.45 50   0.000 setosa ( 1.00000 0.00000 0.00000 ) *
   3) Petal.Length > 2.45 100 138.600 versicolor ( 0.00000 0.50000 0.50000 )
     6) Petal.Width < 1.75 54  33.320 versicolor ( 0.00000 0.90741 0.09259 )
      12) Petal.Length < 4.95 48   9.721 versicolor ( 0.00000 0.97917 0.02083 )
        24) Sepal.Length < 5.15 5   5.004 versicolor ( 0.00000 0.80000 0.20000 ) *
        25) Sepal.Length > 5.15 43   0.000 versicolor ( 0.00000 1.00000 0.00000 ) *
      13) Petal.Length > 4.95 6   7.638 virginica ( 0.00000 0.33333 0.66667 ) *
     7) Petal.Width > 1.75 46   9.635 virginica ( 0.00000 0.02174 0.97826 )
      14) Petal.Length < 4.95 6   5.407 virginica ( 0.00000 0.16667 0.83333 ) *
      15) Petal.Length > 4.95 40   0.000 virginica ( 0.00000 0.00000 1.00000 ) *
>
```


Another example: the forensic glass data `fgl`. the data give the refractive index and oxide content of six types of glass.

```
> data(fgl)
> attach(fgl)
> summary(fgl)
       RI               Na              Mg              Al        
 Min.   :-6.8500   Min.   :10.73   Min.   :0.000   Min.   :0.290  
 1st Qu.:-1.4775   1st Qu.:12.91   1st Qu.:2.115   1st Qu.:1.190  
 Median :-0.3200   Median :13.30   Median :3.480   Median :1.360  
 Mean   : 0.3654   Mean   :13.41   Mean   :2.685   Mean   :1.445  
 3rd Qu.: 1.1575   3rd Qu.:13.82   3rd Qu.:3.600   3rd Qu.:1.630  
 Max.   :15.9300   Max.   :17.38   Max.   :4.490   Max.   :3.500  
       Si              K               Ca              Ba        
 Min.   :69.81   Min.   :0.0000   Min.   : 5.430   Min.   :0.0000  
 1st Qu.:72.28   1st Qu.:0.1225   1st Qu.: 8.240   1st Qu.:0.0000  
 Median :72.79   Median :0.5550   Median : 8.600   Median :0.0000  
 Mean   :72.65   Mean   :0.4971   Mean   : 8.957   Mean   :0.1750  
 3rd Qu.:73.09   3rd Qu.:0.6100   3rd Qu.: 9.172   3rd Qu.:0.0000  
 Max.   :75.41   Max.   :6.2100   Max.   :16.190   Max.   :3.1500  
       Fe              type    
 Min.   :0.00000   WinF :70  
 1st Qu.:0.00000   WinNF:76  
 Median :0.00000   Veh  :17  
 Mean   :0.05701   Con  :13  
 3rd Qu.:0.10000   Tabl : 9  
 Max.   :0.51000   Head :29  
> fgl.tr<-tree(type~.,fgl)
> summary(fgl.tr)

Classification tree:
tree(formula = type ~ ., data = fgl)
Number of terminal nodes:  20 
Residual mean deviance:  0.6853 = 133 / 194 
Misclassification error rate: 0.1542 = 33 / 214 
> plot(fgl.tr)
> text(fgl.tr,all=T,cex=0.5))
Error: syntax error
> text(fgl.tr,all=T,cex=0.5)
```



Mg < 2.695
Na < 13.785
Al < 1.42
Al < 1.38
Ba < 0.2
RI < -0.93
Mg < 3.455
Fe < 0.085
RI < -1.885
K < 0.29
Si < 72.84
Na < 12.835
K < 0.55
WinNF
Con WinNF
RI < 1.265
WinF Veh
Ca < 9.67
Mg < 3.75
Veh WinNF
WinF
WinNF WinNF
Tabl WinNF
Head
WinF WinF
Fe < 0.145
RI < 1.045
Al < 1.17
WinNF
WinF WinF
WinNF WinF

Decision Trees:

One common use of classification trees is as an aid to decision making — not really different from classification but sometimes distinguished.

Data shuttle gives guidance on whether to use autolander or manual control on landing the space shuttle under various conditions such as head or tail wind of various strengths, good or poor visibility (always use auto in poor visibility!) etc, 6 factors in all. There are potentially 256 combinations of conditions and these can be tabulated and completely enumerated but displaying the correct decision as a tree is convenient and attractive.

```
> data(shuttle)
> attach(shuttle)
> summary(shuttle)
 stability   error   sign       wind         magn     vis          use
 stab :128   LX:64   nn:128   head:128   Light :64   no :128   auto  :145
 xstab:128   MM:64   pp:128   tail:128   Medium:64   yes:128   noauto:111
             SS:64                       Out   :64
             XL:64                       Strong:64
> table(use,vis)
        vis
use       no yes
  auto   128  17
  noauto   0 111
> table(use,wind)
        wind
use      head tail
  auto     72   73
  noauto   56   55
```

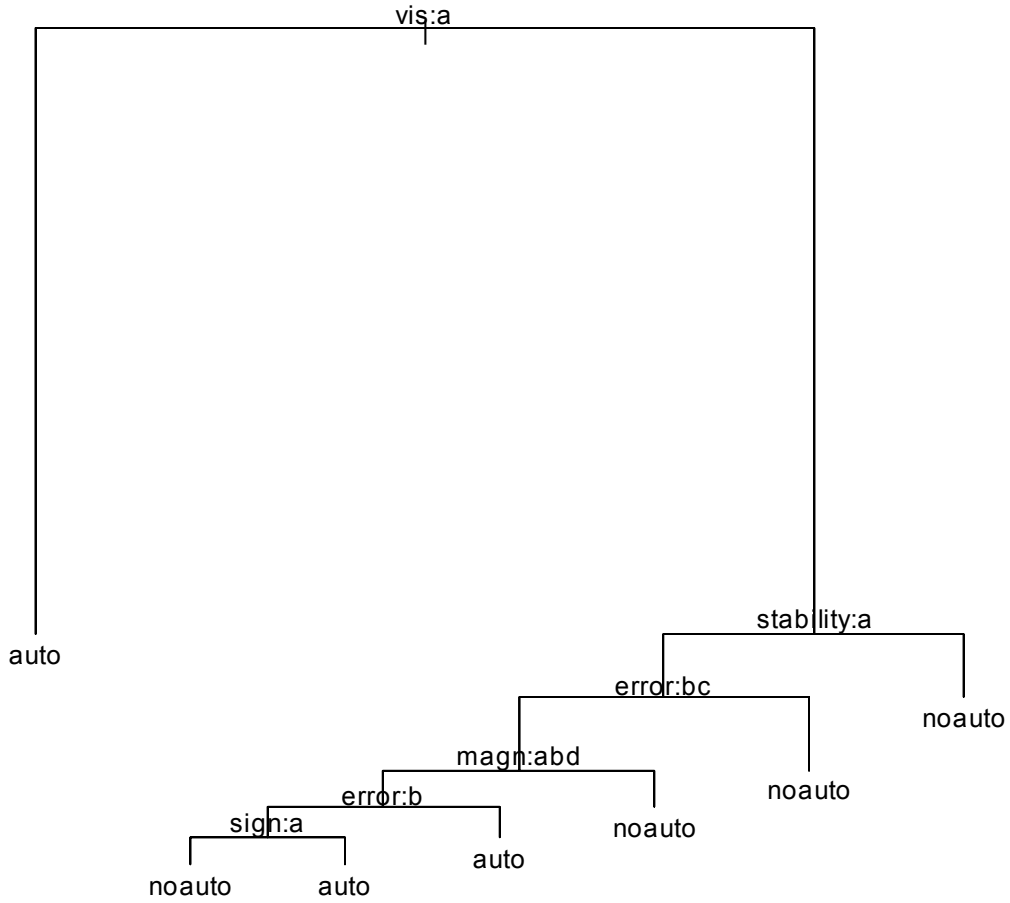

```
> table(use,magn,wind)
, , wind = head

        magn
use      Light Medium Out Strong
  auto      19     19  16     18
  noauto    13     13  16     14

, , wind = tail

        magn
use      Light Medium Out Strong
  auto      19     19  16     19
  noauto    13     13  16     13

> shuttle
    stability error sign wind   magn vis    use
1       xstab    LX   pp head  Light  no   auto
2       xstab    LX   pp head Medium  no   auto
3       xstab    LX   pp head Strong  no   auto
…         …          …          …          …          …
…         …          …          …          …          …
…         …          …          …          …          …
…         …          …          …          …          …
255      stab    MM   nn head Medium yes noauto
256      stab    MM   nn head Strong yes noauto
>
> shuttle.tr<-tree(use~.,shuttle)
> plot(shuttle.tr)
> text(shuttle.tr)
```

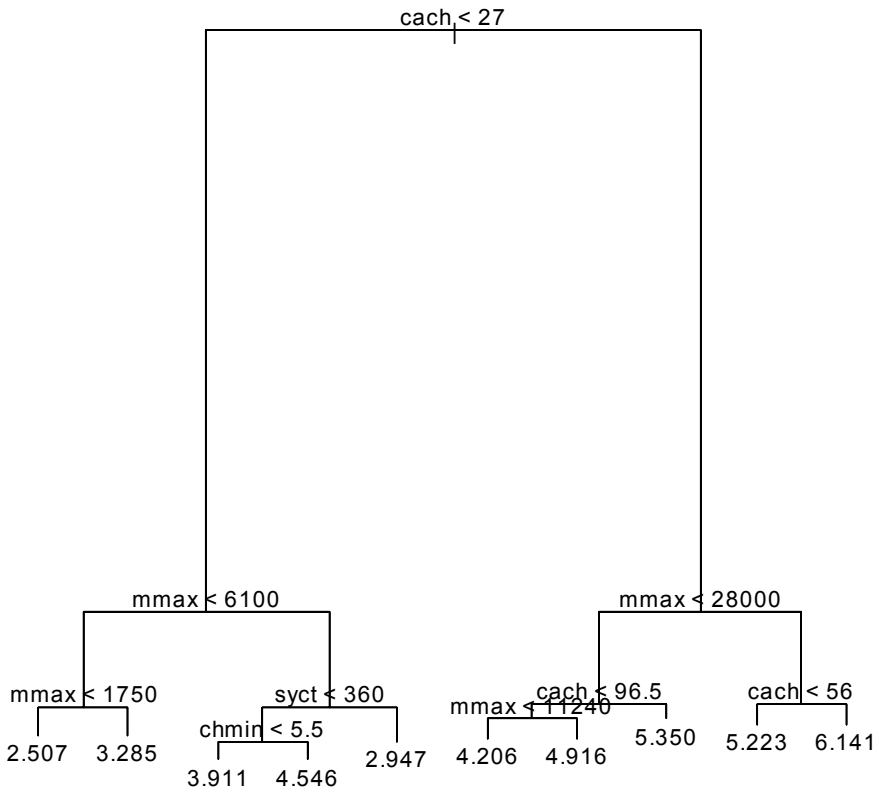
In this default display, the levels of the factors are indicated by a,b,…. alphabetically and the tree is read so that levels indicated are to the left branch and others to the right, e.g. at the first branching `vis:a` indicates `no` for the left branch and `yes` for the right one. At the branch labelled `magn:abd` the right branch is for level c which is '`out of range`'; all other levels take the left branch. The plot can of course be enhanced with better labels.

Regression Trees:

We can think of classification trees as modelling a ***discrete*** factor or outcome as depending on various explanatory variables, either continuous or discrete**.** For example, the *iris species* depended upon values of the continuous variables giving the dimensions of the sepals and petals. In an analogous way we could model a ***continuous*** outcome on explanatory variables using tree-based methods, i.e. *regression trees*. The analysis can be thought of as categorizing the continuous outcome into discrete levels, i.e. turning the continuous outcome into a discrete factor. The number of distinct levels can be controlled by specifying the minimum number of observations (`minsize`) at a node that can be split and the reduction of variance produced by splitting a node (`mindev`). This is illustrated on the hills data, but first an example of data on c.p.u. performance of 209 different processors in data set `cpus` contained in the `MASS` library. The measure of performance is perf and we model the log of this variable.

```
> library(MASS)
> library(tree)
> data(cpus)
> attach(cpus)
> summary(cpus)
                  name          syct            mmin            mmax
 WANG VS10           :  1   Min.   :  17.0   Min.   :   64   Min.   :   64
 WANG VS 90          :  1   1st Qu.:  50.0   1st Qu.:  768   1st Qu.: 4000
 STRATUS 32          :  1   Median : 110.0   Median : 2000   Median : 8000
 SPERRY 90/80 MODEL 3:  1   Mean   : 203.8   Mean   : 2868   Mean   :11796
 SPERRY 80/8         :  1   3rd Qu.: 225.0   3rd Qu.: 4000   3rd Qu.:16000
 SPERRY 80/6         :  1   Max.   :1500.0   Max.   :32000   Max.   :64000
 (Other)             :203
      cach           chmin           chmax            perf          estperf
 Min.   :  0.00 Min.   : 0.000 Min.   :  0.00 Min.   :   6.0 Min.   :  15.0
 1st Qu.:  0.00 1st Qu.: 1.000 1st Qu.:  5.00 1st Qu.:  27.0 1st Qu.:  28.0
 Median :  8.00 Median : 2.000 Median :  8.00 Median :  50.0 Median :  45.0
 Mean   : 25.21 Mean   : 4.699 Mean   : 18.27 Mean   : 105.6 Mean   :  99.3
 3rd Qu.: 32.00 3rd Qu.: 6.000 3rd Qu.: 24.00 3rd Qu.: 113.0 3rd Qu.: 101.0
 Max.   :256.00 Max.   :52.000 Max.   :176.00 Max.   :1150.0 Max.   :1238.0
```

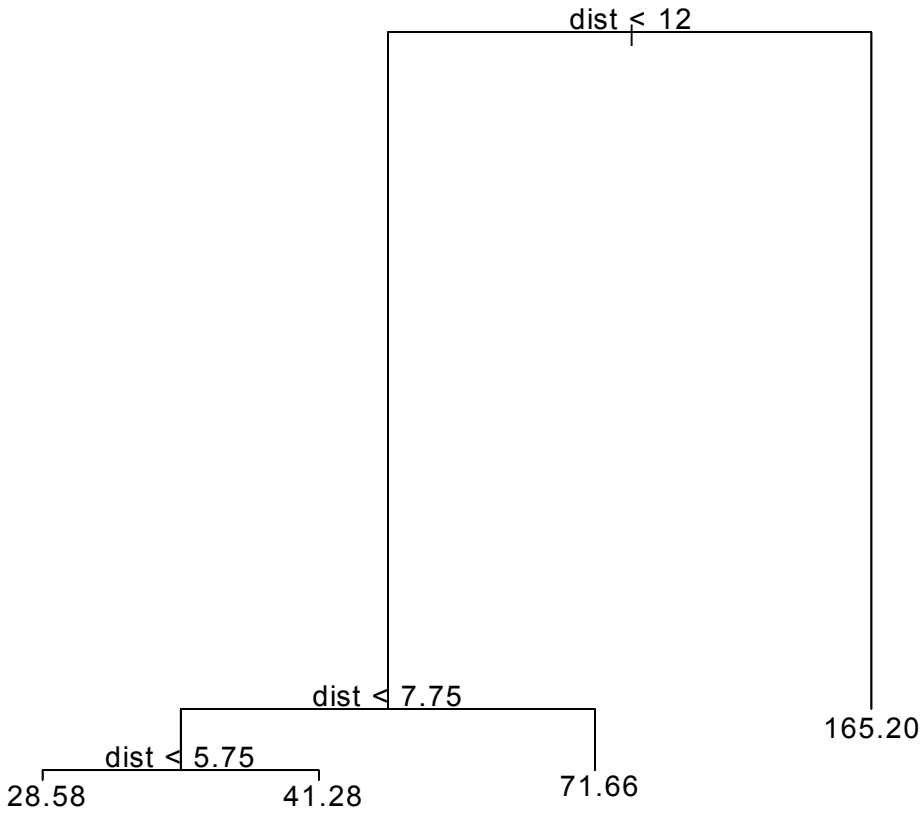

```
> cpus.tr<-tree(log(perf)~.,cpus[,2:8])
> plot(cpus.tr)
> text(cpus.tr)
```



The attraction of the display is that it gives a quick way of predicting cpu performance for a processor with specified characteristics. The accuracy of the predictions can be increased by increasing the number of terminal nodes (or leaves). However, this does not offer a substitute for more investigative modelling and outlier identification.

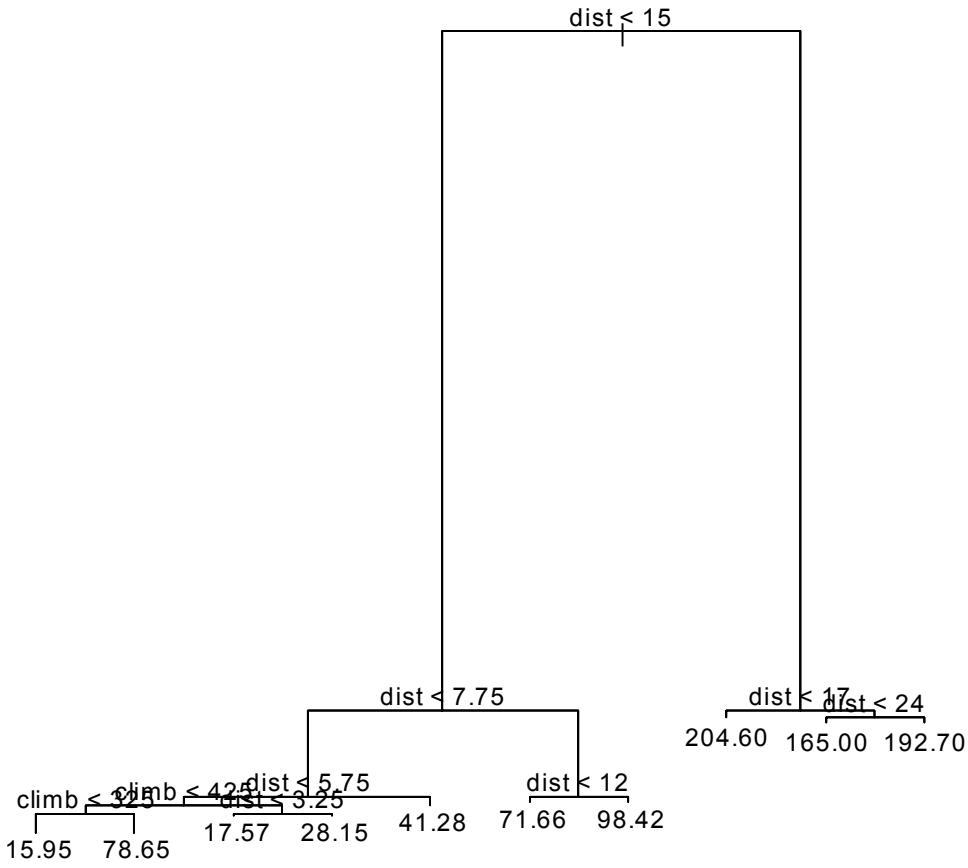
Now we illustrate some of the refinements with the more familiar Scottish Hill race data `hills`

```
> data(hills)
> hills.tr<-tree(time~.,hills)
> plot(hills.tr)
> text(hills.tr,cex=1.25)
```



Next, increase the number of terminal nodes, i.e. increase the resolution of forecasting.

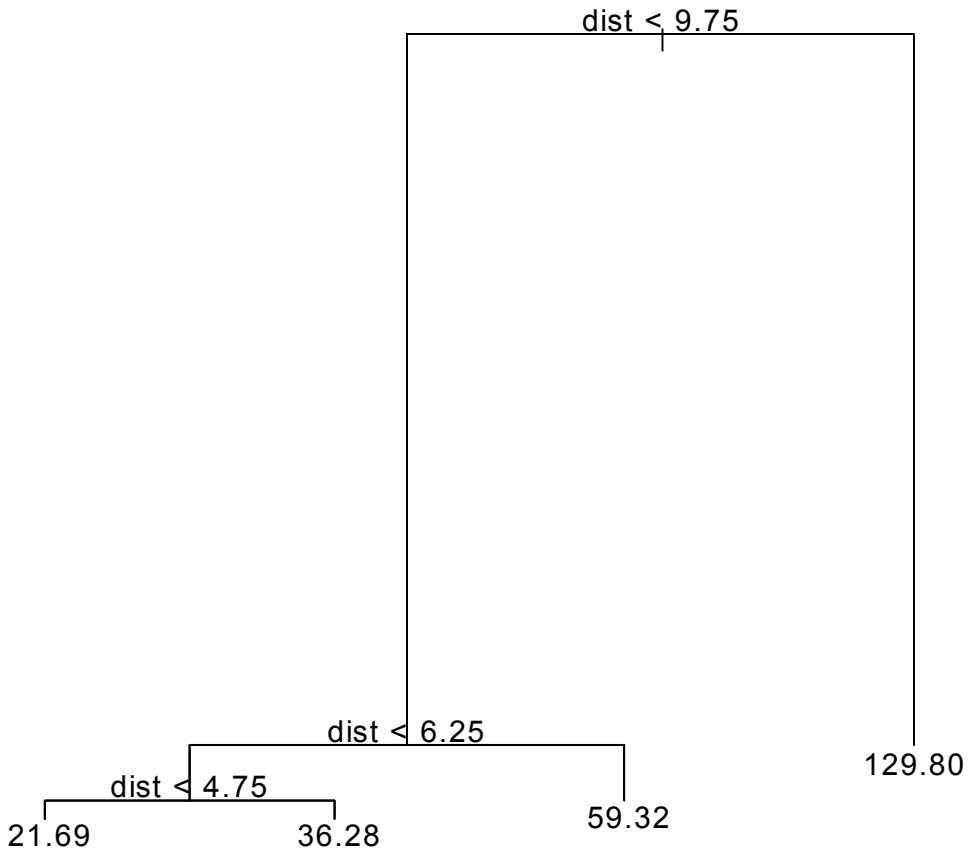
```
> hills.tr1<-tree(time~.,hills,control=tree.control(nobs=35,
+ minsize=2,mindev=.003))
> plot(hills.tr1)
> text(hills.tr1)
```



With function `tree`, the maximum number of terminal nodes is 32.

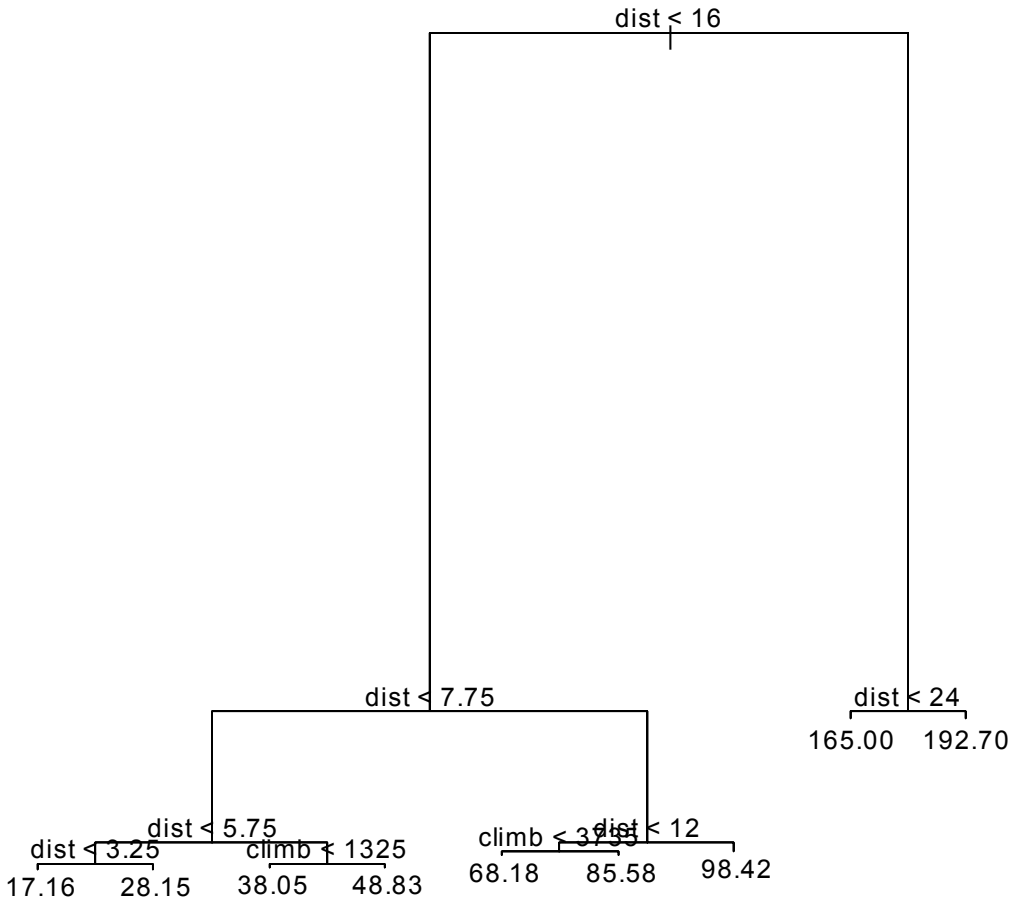
Next, for comparison we repeat the above analyses removing the known outliers observations 7 and 18.

```
> hills.tr2<-tree(time~.,hills[-c(7,18),])
> plot(hills.tr2)
> text(hills.tr2,cex=1.25)
```



And with more nodes:

```
> hills.tr3<-tree(time~., hills[-c(7,18),],
+ control=tree.control(nobs=33,
+ minsize=2,mindev=.003))
> plot(hills.tr3)
> text(hills.tr3)
```



It is clear that there differences in the trees after dropping the outliers but these are not quite so great as they might appear, taking into account the resolution of the end nodes.

More refined facilities are available in library `rpart` (pruning etc).