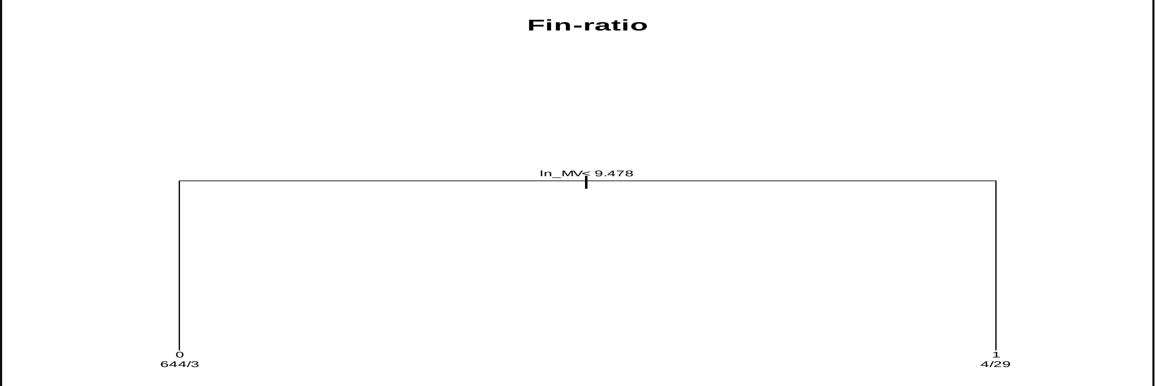
Chapter 6 Classification Tree

The binary logistic regression and the multinomial logit model introduced in Chapter 5 aim at classifying observations based on a linear combination of variables *x*. To most ordinary user, these models may be difficult to understand or hard to compute. They are looking for simple rules that can be used to classify the observations. **Classification Tree** is a method to generate a set of simple rules that can be applied to classify the observations. This is useful in **Data Mining** and has many potential applications. For example, it can be used to derive simple rules for credit approval, medical diagnosis, etc.

6.1 Classification Tree

Classification Tree method was first developed by Breiman, Friedman, Olshen, and Stone in 1984. It has been an active research area since then. This method has been implemented in R's built-in library *rpart* (stands for **Recursive Partitioning and Regression Trees**). This method will build a classification tree based on the binary splitting of variables, one at a time. The tree is constructed such that each terminal node is as "**pure**" as possible. That is, in each terminal node, most of the **"close"** observations are belonging to the same group. Once the classification tree is built, a set of simple classification rules can be easily obtained. Since this method will search for all possible binary splitting of the variables, it is computational intensive and may be time-consuming. Let us first illustrate this method using our *HSI* example.

```
> d<-read.csv("fin-ratio.csv")                # read in data in csv format
> library(rpart)                              # load rpart library
> ctree<-rpart(HSI~EY+CFTP+ln_MV+DY+BTME+DTE,data=d,method="class")
> plot(ctree,asp=0.5,main=" Fin-ratio" )   # plot ctree
> text(ctree,use.n=T,cex=0.6)                 # add text
```



This classification is very simple. The data set is split into two classes (0 or 1) according to the variable `ln_MV`. The classification rules are:

R1: *If* ln_MV < 9.478 *then* class = 0.
R2: *If* ln_MV>=*9.478* *then* class = 1.

The numbers in the terminal nodes represent the number of cases. For example, in the group *ln_MV<9.478*, there are 644 “zeroes” and 3 “ones” while in the group *ln_MV>=9.478*, there are 4 “zeroes” and 29 “ones”. A more detailed output can be obtained by using the command: `> print(ctree)`

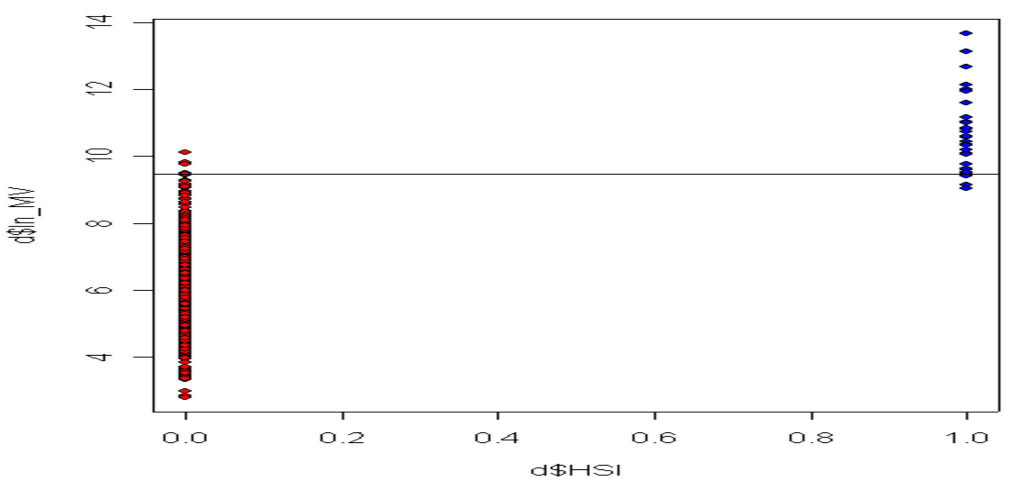
```
> print(ctree)
n= 680
node), split, n, loss, yval, (yprob)
      * denotes terminal node
1) root 680 32 0 (0.952941176 0.047058824)
  2) d$ln_MV< 9.4776 647  3 0 (0.995363215 0.004636785) *
  3) d$ln_MV>=9.4776 33  4 1 (0.121212121 0.878787879) *
```

This output shows the details at each node:

1. At node 1 (root), there is totally n=680 cases. Among them 32 is group 1 (loss). Since the majority is from group 0, the group label is yval=0 and the percentage of group 0 and 1 are given in the parenthesis; 0.0470588=32/680.
2. At node 2 (left of the split), the split condition is d$ln_MV<9.4776. n=647, cases 3 are from group 1 and the group label is yval=0; 0.004636785=3/647.
3. At node 3 (right of the split), n=33, 4 cases are from group 0 and the group label is yval=1; 0.121212=4/33.

What this output actually implies that we can predict or classify if a stock is Blue Chip based on only the condition `ln_MV<9.4776`. Since the classification tree in this example is very simple, we can actually plot the observation to see the full picture.

```
# plot ln_MV versus HSI with color, red=group 0 and blue=group 1
> plot(d$HSI,d$ln_MV,pch=21,bg=c("red","blue")[d$HSI+1])
> abline(h=9.478)       # add a horizontal line at y=9.478
```



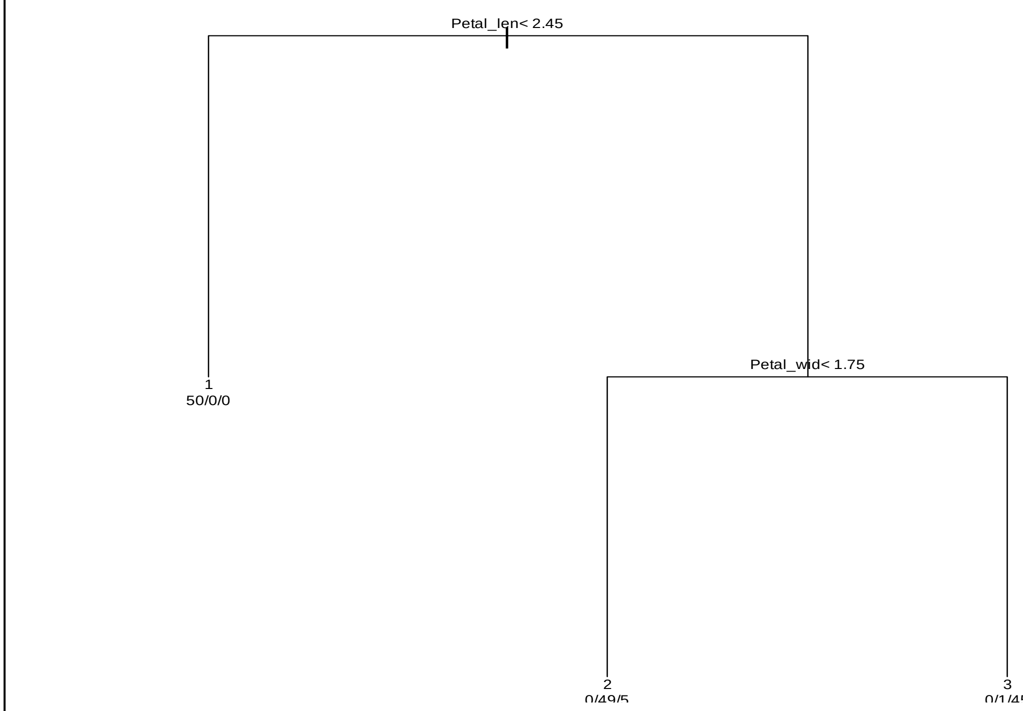
We can produce the classification table using

```
> pr<-predict(ctree)              # pr has 2 columns of prob. in group 0 or 1
> cl<-0*(pr[,1]>0.5)+1*(pr[,2]>0.5)   # assign group label if prob>0.5
> table(cl,d$HSI)                 # cross tabluation

cl   0    1
  0 644   3
  1   4  29
```

From the output, there are 3 Blue Chips misclassified as non-Blue Chips while there are 4 non-Chips misclassified as Blue-Chips. Let us look at Fisher's Iris data we have considered in Chapter 4.

```
> d<-read.csv("iris.csv")
> library(rpart)
> names(d)
[1] "Sepal_len" "Sepal_wid" "Petal_len" "Petal_wid" "Species"
> ctree<-rpart(Species~Sepal_len+Sepal_wid+Petal_len+Petal_wid,data=d,method="class")
> plot(ctree,asp=2)
> text(ctree,use.n=T,cex=0.6)
```



From the output, the classification rules are:

R1: *If* Petal_len < 2.45 *then* species = 1 (setosa) (50/0/0).
R2: *If* (Petal_len>=2.45) and (Petal_wid < 1.75) *then* species = 2 (versicolor) (0/49/5).
R3: *If* (Petal_len>=2.45) and (Petal_wid >= 1.75) *then* species = 3 (virginica) (0/1/45).

We can also display the detailed information about each node using:

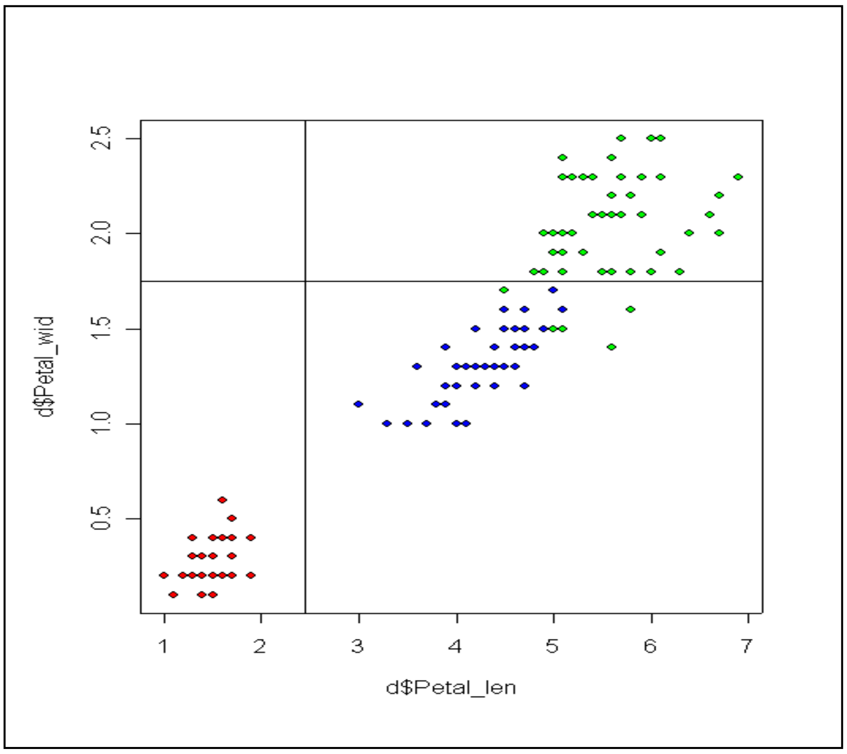
```
> print(ctree)
n= 150

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 150 100 1 (0.33333333 0.33333333 0.33333333)
  2) Petal_len< 2.45 50   0 1 (1.00000000 0.00000000 0.00000000) *
  3) Petal_len>=2.45 100  50 2 (0.00000000 0.50000000 0.50000000)
    6) Petal_wid< 1.75 54   5 2 (0.00000000 0.90740741 0.09259259) *
    7) Petal_wid>=1.75 46   1 3 (0.00000000 0.02173913 0.97826087) *
```

Since this classification tree is relatively simple and the rule also depends on two variables, we can actually plot the observations for illustration.

```
# plot Petal_wid versus Petal_len with different color for each species
> plot(d$Petal_len,d$Petal_wid,pch=21,bg=c("red","blue","green")[d$Species])
> abline(h=1.75)        # add a horizontal line
> abline(v=2.45)        # add a vertical line
```



From this plot, we can clearly see how this classification rule works. Finally, we can produce the classification table as follows:

```
> pr<-predict(ctree)                    # pr has 3 columns of prob.
> cl<-max.col(pr)                  # cl is the col. no. such that pr is the maximum.
> table(cl,d$Species)                   # cross tabulation table

cl  1   2  3
  1 50  0  0
  2  0 49  5
  3  0  1 45
```

From the table, there are five group 3 flowers misclassified into group 2 while there is one group 2 flower misclassified into group 3. The error rate is (1+5)/150 = 4%. There are three classification rules. The quality of these rules is assessed by **support (or coverage), confidence (or accuracy)** and **capture**:

Support = total number of cases in that rule / total number of cases in the entire dataset.

Confidence = proportion of correctly classified cases in that rule.

Capture = total number of cases of the majority group in that rule / total number of cases of that group in the entire dataset.

In Iris data, there are 150 cases. Among them there are 50 in each species.

R1: If Petal_len < 2.45 *then* species = 1 (setosa) (50/0/0).
Support=50/150=0.33, Confidence=50/50=1, Capture=50/50=1.

R2: If (Petal_len>=2.45) and (Petal_wid < 1.75) *then* species = 2 (versicolor) (0/49/5).
Support=54/150=0.36, Confidence=49/54=0.907, Capture=49/50=0.98.

R3: If (Petal_len>=2.45) and (Petal_wid >= 1.75) *then* species = 3 (virginica) (0/1/45).
Support=46/150=0.31, Confidence=45/46=0.978, Capture=45/50=0.9.

Although the *rpart()* function is quite easy to use, there are many options hidden in this function (see *help(rpart.control)*, in particular, *minsplit* and *maxdepth)*, see [2] for more details. There is one important option need to mention: the *method* option in *rpart* can be one of *"anova"*, *"poisson"*, *"class"* or *"exp"*. For classification problem, the target variable is a categorical variable; therefore *method="class"* is used.

6.2 Training and Testing data

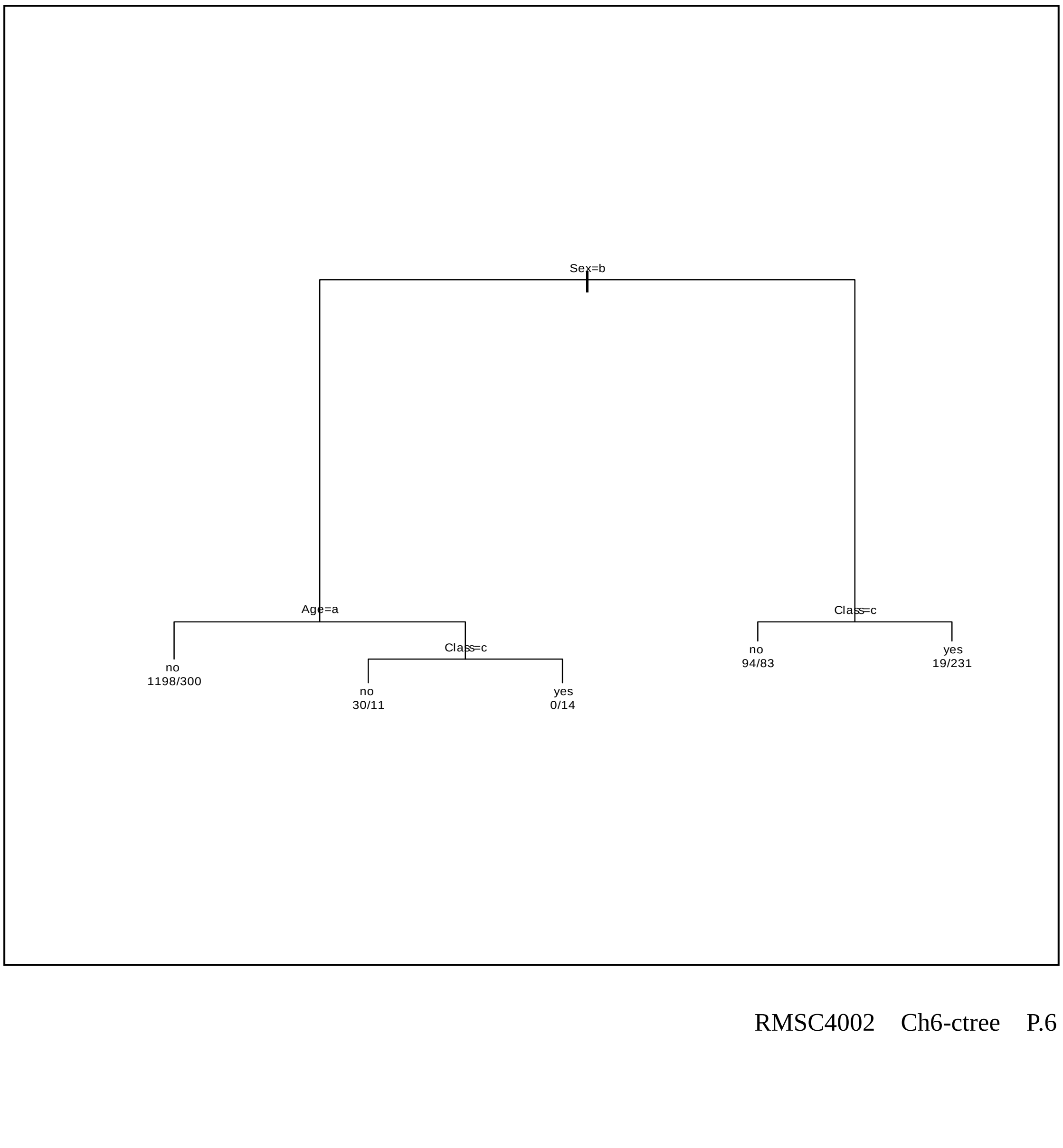
We can select a random subset from the dataset as the training dataset and build a classification tree model. The rest of the data will form the testing dataset to test the classification tree model. Let us illustrate this again using the *fin-ratio1.csv* data.

```
d<-read.csv("fin-ratio1.csv")     # read in data
n<-nrow(d)                        # no. of obs
set.seed(12345)                   # set random seed
id<-sample(1:n,size=560)          # generate id
d1<-d[id,]                        # training dataset
d2<-d[-id,]                       # testing dataset
ctree<-rpart(HSI~EY+CFTP+ln_MV+DY+BTME+DTE,data=d1,method="class")
print(ctree)                      # print ctree
n= 560
node), split, n, loss, yval, (yprob)
      * denotes terminal node
1) root 560 28 0 (0.950000000 0.050000000)
  2) ln_MV< 9.50415 532  3 0 (0.994360902 0.005639098) *
  3) ln_MV>=9.50415 28  3 1 (0.107142857 0.892857143) *

pr<-predict(ctree)                # in-sample
cl<-max.col(pr)                   # column index of max. pr
table(cl,d1$HSI)                  # classification table for d1
cl    0   1                       # error rate = 6/560=0.0107
  1 529   3
  2   3  25
pr<-predict(ctree,d2)             # out-sample
cl<-max.col(pr)
table(cl,d2$HSI)                  # classification table for d2
cl   0  1                         # error rate=1/98=0.0102
  1 94  1
  2  0  3
```

Let us illustrate this by an interesting example. The file "*Titanic.csv*" contains the information about the *2201* people on board of Titanic. There are 4 columns: Class (1st, 2nd, 3rd, crew), Age(adult, child), Sex(Female, Male) and Survive(Yes, No). We want to build a classification tree to "predict" who is going to survive in the crash. Instead of using the whole dataset, we randomly sampled 1980 cases (about 90%) as our training data and the remaining *221* cases as testing data.

```
> d<-read.csv("titanic.csv")                        # read in data
> names(d)                                          # display variables
[1] "Class"   "Age"     "Sex"     "Survive"
> set.seed(12345)                                   # set random seed
> dim(d)                                            # display dimension
[1] 2201    4
> id<-sample(1:2201,1980)                           # create random sample
> d1<-d[id,]                                        # select training data
> d2<-d[-id,]                                       # select testing data
> dim(d1)
[1] 1980    4
> dim(d2)
[1] 221   4
> library(rpart)
> ctree<-rpart(Survive~Class+Age+Sex,data=d1,method="class")
> plot(ctree,asp=10)
> text(ctree,cex=0.6,use.n=T)
```



The classification rules from the training data is as follow:
R1*: If* (Sex=male) *and* (Age=adult) *then* NO (1198/300).
R2*: If* (Sex=male) *and* (Age=child) *and* (Class=3rd) *then* NO (30/11).
R3*: If* (Sex=male) *and* (Age=child) *and* (Class≠3rd) *then* YES (0/14).
R4: *If* (Sex=female) *and* (Class=3rd) *then* NO (94/83).
R5: *If* (Sex=female) *and* (Class≠3rd) *then* YES (19/231).

Note that the representation of rule is not unique. An equivalent form of the rule is
R1: *If* Class=3rd *then* No.
R2: *If* (Class≠3rd) and (Sex=female) *then* YES.
R3: *If* (Class≠3rd) and (Sex=male) *then* NO.
R4: *If* (Class≠3rd) and (Age=child) *then* YES.
R5: *If* (Class≠3rd) and (Age=adult) *then* NO.

We apply this rule to the training dataset (d1) and then test the dataset (d2) by using the built-in function `predict()` to assess this rule.

```
> prob<-predict(ctree)    # return 2 col. of prob.
> pr<-prob[,1]<0.5        # create suitable label
> table(pr,d1$Survive)    # classification table

pr         no  yes
  FALSE 1322  394
  TRUE    19  245

> prob<-predict(ctree,d2) # prediction for testing dataset
> pr<-prob[,1]<0.5
> table(pr,d2$Survive)

pr        no yes
  FALSE 148  47
  TRUE    1  25
```

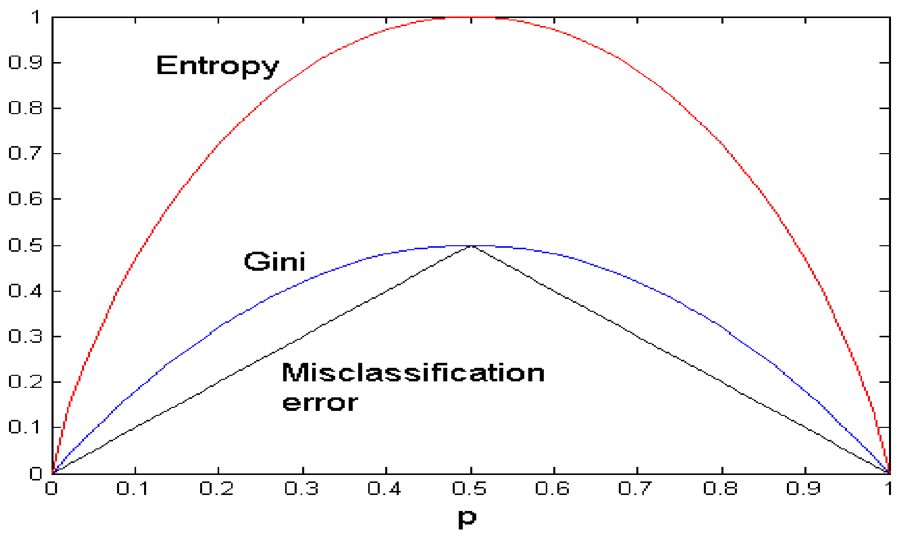
There are totally (394+19)=413 error cases out of 1980 training cases, so that the error rate is 413/1980=20.86%; while there are totally 48 error cases out of 221 testing cases, so that the error rate is 48/221=21.7%.

6.3 Algorithm for C-Tree building

The objective of C-Tree is to find a set of rules such that the distribution of each terminal node is as "pure" as possible. To understand how to build a C-Tree, we need some measure of the impurity of the distribution. Suppose that there are c classes and $p(i|t)$ is the proportion of records belonging to class i at node t, for $i=0,1,2,...,c-1$. There are several commonly used measures of impurity:

1. $Entropy(t) = -\sum_{i=0}^{c-1} p(i\,|\,t)\log_2 p(i\,|\,t)$, here we adopt the custom $0\log_2 0 = 0$;
2. $Gini(t) = 1 - \sum_{i=0}^{c-1} [p(i\,|\,t)]^2$;
3. $Classification\ error(t) = 1 - \max_i [p(i\,|\,t)]$.

The following is the plot of these functions for $c = 2$. Note that for all impurity measures, the maxima are attained at $p(i\,|\,t) = 1/2$ when both $i = 0,1$ are equally likely and the node is at the "most impurest" state.



In general, for c categories, all impurities measure should attain maximum at $p(i\,|\,t) = 1/c$ when all $i = 0,1,2,\ldots,c-1$ are equally likely. The maximum values for various impurity measures are: (1) $\max(Entropy) = -\log_2(c)$, (2) $\max(Gini) = 1 - 1/c$, or (3) $\max(Misclassification) = 1 - 1/c$. As $c \to \infty$, the entropy diverges to infinity, while the Gini index and misclassification error are approaching and are bounded at 1.

Now we turn to the problem of how to choose the best variable for splitting. We need to compare the impurity of the parent node (before splitting) and the impurity of the child node (after splitting). We define the goodness of a splitting variable *v* by

$$\Delta(v) = \mathrm{Im}(parent) - \mathrm{Im}(v) = \mathrm{Im}(parent) - \sum_{j=1}^{k} N(v_j)\,\mathrm{Im}(v_j)/N \qquad (6.1)$$

where *Im(.)* is the impurity measure of a given node, *N* is the total number of records in the parent node, *k* is the number of attribute value of *v*, $N(v_j)$ is the number of records associated with the child node v_j. We choose the variable *v* such that $\Delta(v)$ is largest, or equivalently the impurity measure *Im(v)* is smallest.

In fact, Eq. (6.1) is of the form of the Laplacian operator acting over the impurity measure, which is shown in the Appendix. The Laplacian is also the fundamental equation to model the flow of hot lava. Analogous to the stop of lava flow when its temperature gradient becomes too small and then it solidifies, while for a *c*-tree, further downward branching will stop when the change in impurity measure Δv is smaller than a pre-set tolerance level.

https://www.youtube.com/watch?v=uh4dTLJ9q9o

Let us first consider a very simple example to illustrate the calculations. Assume that we need to choose between two binary variables *A* and *B* such that

root (6,6)
  node 1:A=0 (4,3) *
  node 2:A=1 (2,3) *

root(6,6)
  node 1:B=0 (1,4) *
  node 2:B=1 (5,2) *

We first use Gini index as the impurity measure. Now, Im(parent)=$1-(6/12)^2-(6/12)^2$ =0.5, while for the child nodes:

Using A,
at node 1: Gini=1-(4/7)^2-(3/7)^2=0.4898
at node 2: Gini=1-(2/5)^2-(3/5)^2=0.48
Δ =0.5-(7/12)(0.4898)-(5/12)(0.48)=0.5-0.486

Using B,
at node 1: Gini=1-(1/5)^2-(4/5)^2=0.32
at node 2: Gini=1-(5/7)^2-(2/7)^2=0.408
Δ =0.5-(5/12)(0.32)-(7/12)(0.408)=0.5-0.3715

Hence variable *B* is preferred to *A* since *Im(A)=0.486* and *Im(B)=0.3715*.

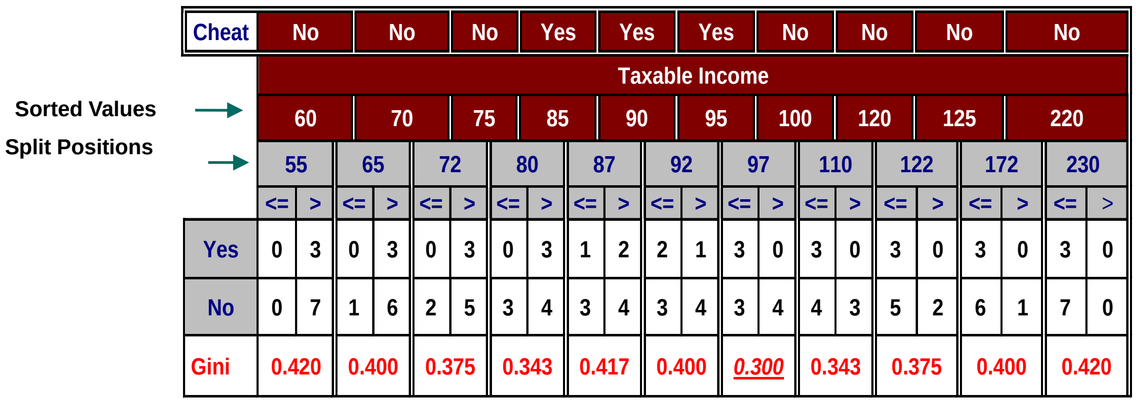
Now we repeat the calculation using entropy as the impurity measure. Now, Im(parent) $=-(6/12)\log_2(6/12)-(6/12)\log_2(6/12)=1.0$. while for the child notes:

Using A,
at node 1:
Entropy= $-(4/7)\log_2(4/7)-(3/7)\log_2(3/7)=0.9852$
at node 2:
Entropy=$-(2/5)\log_2(2/5)-(3/5)\log_2(3/5)=0.9710$
Δ =1.0-(7/12)(0.9852)-(5/12)(0.9710)=1.0-0.9793

Using B,
at node 1:
Entropy=$-(1/5)\log_2(1/5)-(4/5)\log_2(4/5)=0.7219$
at node 2:
Entropy=$-(5/7)\log_2(5/7)-(2/7)\log_2(2/7)=0.8631$
Δ =1.0-(5/12)(07219)-(7/12)(0.8631)=1.0-0.8043

Still variable *B* is preferred to *A* since *Im(A)=0.9793* and *Im(B)=0.8043*.

The calculation of *Im(v)* can be easily extended to find the best splitting condition for continuous variable where we divide the value of *v* into intervals and obtain the distribution (probability mass) over the interval. For example:



Cheat		No		No		No		Yes		Yes		Yes		No		No		No		No		
	Taxable Income																					
Sorted Values		60		70		75		85		90		95		100		120		125		220		
Split Positions	55		65		72		80		87		92		97		110		122		172		230	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini	0.420		0.400		0.375		0.343		0.417		0.400		0.300		0.343		0.375		0.400		0.420	

The value of income is sorted and the mid-point between two successive values is used as the split position. The distribution of the target variable is obtained and the impurity measure is computed accordingly. In the above example, the splitting condition *Income<=97* provides the best (smallest) impurity measure.

Child node: *<=97,* *(Y,N)=(3,3),* *Gini=1-(3/6)^2-(3/6)^2=1/2*
>97, *(Y,N)=(0,4)* *Gini=1-(0/4)^2-(4/4)^2=0*

Impurity measure *Im* = *(6/10)(1/2)+(4/10)(0)* = *3/10* = *0.3*.

6.4 Classification Tree using EXCEL

There is a good implementation of Classification tree using EXCEL found in the internet developed by Dr. Saha. The file *ch6-ctree-iris.xls* contains the iris data. Inside the EXCEL file, there are seven separate sheets.

1. ReadMe sheet contains the basic information for using this file. Basically, this file can handle up to 10,000 observations and 50 predictor variables.
2. Data are stored in L24 of the Data sheet. We can specify which variable is class, continuous, categorical or omit. We can copy and paste your own data here to build our own classification tree.
3. In the UserInput sheet, we can specify the parameters to build the tree: **minimum node size**, **maximum purity** at each node, and **maximum depth** of the tree. The data set is divided into two subsets: training data set and testing data set. We can specify the percentage of data points used as testing data. Once the parameters are specified, we can click the Build Tree button to build the tree.
4. The result sheet contains the overall information about this tree as well as the classification (confusion) table of the training data and testing data as follow:

Training Data

	Predicted Class			
True Class	setosa	verginica	versicolor	
setosa	48			48
verginica		44	1	45
versicolor		5	41	46
	48	49	42	139

Test Data

	Predicted Class			
True Class	setosa	verginica	versicolor	
setosa	2			2
verginica		5		5
versicolor			4	4
	2	5	4	11

5. Tree sheet contains the classification tree and the Node View contains information in each node. There is a button View node link to the Node view sheet. Information about the **splitting condition, label, confidence** and **support** of each terminal node is given. In node 1, there are 48 records out of total 139 records, therefore the **support** is 48/139=34.5%. The majority class is setosa (node ID=1). There is no misclassification; therefore the **confidence** is 100%. In node 3, the node ID=2. There are 49 records, therefore the support is 49/139=35.3%. The misclassification rate is 5/49=10.2%; therefore the confidence is 44/49=89.87%. Finally, in node 4, the node ID=3. There are 42 records and the support is 42/139=30.2%. The misclassification rate is 1/42=2.4% and hence the confidence is 41/42=97.6%.
6. Finally a set of classification rules is induced from the tree. The support and confidence of each rule is given as in the Tree sheet. There is an additional **Capture** is also given. The capture is the percentage of correctly classified records captured by the rule. For example, the capture of rule 1 is 100% since all 48 setosa species is captured by rule 1. For rule 2, there are 44 out of 45 (=97.8%) virginica species are captured. For rule 3, there are 41 out of 46 (=89.1%) versicolor species are captured.

We can apply this *Ctree.xls* on our *HSI* example. The file *ctree-fin.xls* contains the *HSI* data. *10* stocks are randomly chosen as testing data while the remaining *670* stocks are used as training data set. The classification rule is very similar to R's *rpart()* output:

R1: *If* (ln_MV<9.4076) *then* HSI=0

R2: *If* (ln_MV>=9.4076) *then* HSI=1.

Reference:

[1] Section 11.8 of Applied Multivariate Statistical Analysis, 5th ed., Richard Johnson and Dean Wichern, Prentice Hall.

[2] Friedman, J., Hastie, T. and Tibshirani, R., The Elements of Statistical Learning.

Springer. (2001)

Appendix: Laplacian and goodness of a splitting variable *v* in *c*-trees

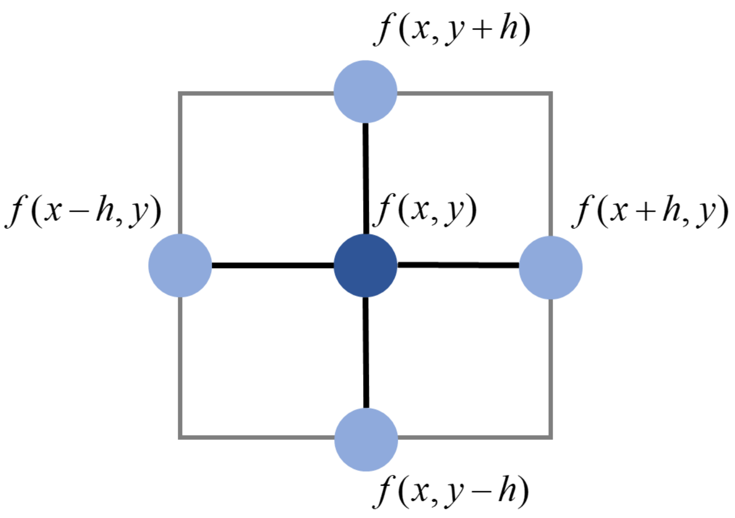
As an illustration, the Laplacian acting on a scalar function in two dimension is

$$\Delta f(x,y) = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) f(x,y)$$

Numerically, the Laplacian operator on the plane R^2 can be approximated by a discretized 2-D special finite differences over a sequence of areas h^2.

$$\Delta f(x,y) = \frac{f(x+h,y) - 2f(x,y) + f(x-h,y)}{h^2} + \frac{f(x,y+h) - 2f(x,y) + f(x,y-h)}{h^2}$$

$$= \frac{4}{h^2}\left[\frac{f(x+h,y) + f(x-h,y) + f(x,y+h) + f(x,y-h)}{4} - f(x,y)\right]$$



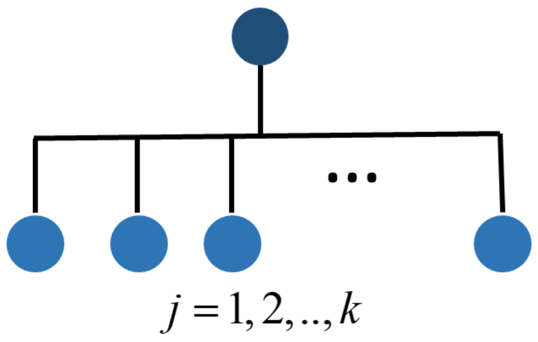
It can be seen that the Laplacian operator gives the increase in the average function values at the periphery to the function value at the original point without branching out.

Generalizing the result from R^2, the branching of classification tree is analogous to the Laplacian operator acting on a scalar function. Recall Eq. (6.1)

$$\Delta(v) = \mathrm{Im}(parent) - \sum_{j=1}^{k} \frac{N(v_j)}{N} \mathrm{Im}(v_j)$$

$$= -\Delta[\mathrm{Im}(parent)]$$

The second term is a weighted average of the impurity measure at the further downward branches. $\Delta(v)$ gives the reduction (hence the negative sign) in impurity measure at all the lower branches compared to the impurity measure of the parent.



Further branching of the child nodes will be stopped if the potential change in impurity measures Δv is smaller than a pre-set tolerance level.