Classification and Regression Trees (CART) Theory and Applications

A Master Thesis Presented

by

Roman Timofeev

(188778)

to

Prof. Dr. Wolfgang Härdle

CASE - Center of Applied Statistics and Economics

Humboldt University, Berlin

in partial fulfillment of the requirements

for the degree of

Master of Art

Berlin, December 20, 2004

Declaration of Authorship

I hereby confirm that I have authored this master thesis independently and without use of others than the indicated resources. All passages, which are literally or in general matter taken out of publications or other resources, are marked as such.

Roman Timofeev

Berlin, January 27, 2005

Abstract

This master thesis is devoted to Classification and Regression Trees (CART). CART is classification method which uses historical data to construct decision trees. Depending on available information about the dataset, classification tree or regression tree can be constructed. Constructed tree can be then used for classification of new observations.

The first part of the thesis describes fundamental principles of tree construction, different splitting algorithms and pruning procedures. Second part of the paper answers the questions why should we use or should not use the CART method. Advantages and weaknesses of the method are discussed and tested in detail.

In the last part, CART is applied to real data, using the statistical software XploRe. Here different statistical macros (quantlets), graphical and plotting tools are presented.

Keywords: CART, Classification method, Classification tree, Regression tree, Statistical Software

Contents

List of Figures

Notation

X	$[N \times M]$ matrix of variables in learning sample
Y	$[M \times 1]$ vector of classes/response values in learning sample
N	number of observations in learning sample
M	number of variables in learning sample
t	index of node
t_p	parent node
t_l	child left node
t_r	child right node
P_l	probability of left node
P_r	probability of right node
$i(t)$	impurity function
$i(t_p)$	impurity value for parent node
$i(t_c)$	impurity value for child nodes
$i(t_l)$	impurity value for left child node
$i(t_r)$	impurity value for right child node
$\Delta i(t)$	change of impurity
K	number of classes
k	index of class
$p(k\|t)$	conditional probability of class k provided we are in node t
T	decision tree
$R(T)$	misclassification error of tree T
$\tilde{T}$	number of terminal nodes in tree T
$\alpha(\tilde{T})$	complexity measure which depends on number of terminal nodes $\tilde{T}$
x_j^R	best splitting value of variable x_j.
N_{min}	minimum number of observations parameter, which is used for pruning

1 Introduction

Classification and Regression Trees is a classification method which uses historical data to construct so-called *decision trees*. *Decision trees* are then used to classify new data. In order to use CART we need to know number of classes a priori.

CART methodology was developed in 80s by Breiman, Freidman, Olshen, Stone in their paper "Classification and Regression Trees" (1984). For building *decision trees*, CART uses so-called *learning sample* - a set of historical data with pre-assigned classes for all observations. For example, *learning sample* for credit scoring system would be fundamental information about previous borrows (variables) matched with actual payoff results (classes).

Decision trees are represented by a set of questions which splits the *learning sample* into smaller and smaller parts. CART asks only yes/no questions. A possible question could be: "Is age greater than 50?" or "Is sex male?". CART algorithm will search for all possible variables and all possible values in order to find the best split - the question that splits the data into two parts with maximum homogeneity. The process is then repeated for each of the resulting data fragments. Here is an example of simple classification tree, used by San Diego Medical Center for classification of their patients to different levels of risk:

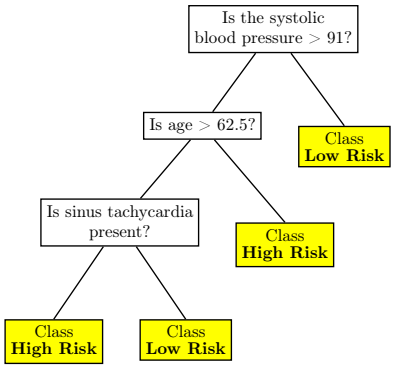

Figure 1.1: Classification tree of San Diego Medical Center patients.

In practice there can be much more complicated *decision trees* which can include dozens of levels and hundreds of variables. As it can be seen from figure 1.1, CART can easily handle both numerical and categorical variables. Among other advantages of CART method is its robustness to outliers. Usually the splitting algorithm will isolate outliers in individual node or nodes. An important practical property of CART is that the structure of its classification or regression trees is invariant with respect to monotone transformations of independent variables. One can replace any variable with its logarithm or square root value, the structure of the tree will not change.

CART methodology consists of tree parts:

1. Construction of *maximum tree*
2. Choice of the right tree size
3. Classification of new data using constructed tree

2 Construction of Maximum Tree

This part is most time consuming. Building the *maximum tree* implies splitting the learning sample up to last observations, i.e. when terminal nodes contain observations only of one class. Splitting algorithms are different for classification and regression trees. Let us first consider the construction of classification trees.

2.1 Classification tree

Classification trees are used when for each observation of learning sample we know the class in advance. Classes in learning sample may be provided by user or calculated in accordance with some exogenous rule. For example, for stocks trading project, the class can be computed as a subject to real change of asset price.

Let t_p be a parent node and t_l,t_r - respectively left and tight child nodes of parent node t_p. Consider the learning sample with variable matrix X with M number of variables x_j and N observations. Let class vector Y consist of N observations with total amount of K classes.

Classification tree is built in accordance with *splitting rule* - the rule that performs the splitting of learning sample into smaller parts. We already know that each time data have to be divided into two parts with maximum homogeneity:

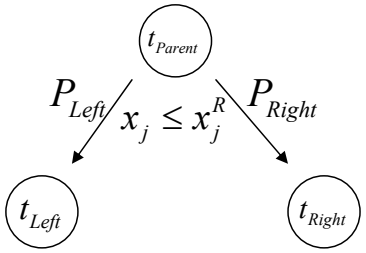

Figure 2.1: Splitting algorithm of CART

where t_p, t_l, t_r - parent, left and right nodes; x_j - variable j; x_j^R - best splitting value of variable x_j.

Maximum homogeneity of child nodes is defined by so-called *impurity function* $i(t)$. Since the impurity of parent node t_p is constant for any of the possible splits $x_j \leq x_j^R,\ j = 1, \ldots, M$, the maximum homogeneity of left and right child nodes will be equivalent to the maximization of change of impurity function $\Delta i(t)$:

$$\Delta i(t) = i(t_p) - E[i(t_c)]$$

where t_c - left and right child nodes of the parent node t_p. Assuming that the P_l, P_r - probabilities of right and left nodes, we get:

$$\Delta i(t) = i(t_p) - P_l i(t_l) - P_r i(t_r)$$

Therefore, at each node CART solves the following maximization problem:

$$\underset{x_j \leq x_j^R,\ j=1,\ldots,M}{\arg\max} [i(t_p) - P_l i(t_l) - P_r i(t_r)] \tag{2.1}$$

Equation 2.1 implies that CART will search through all possible values of all variables in matrix X for the best split question $x_j < x_j^R$ which will maximize the change of impurity measure $\Delta i(t)$.

The next important question is how to define the impurity function $i(t)$. In theory there are several impurity functions, but only two of them are widely used in practice: Gini splitting rule and Twoing splitting rule.

2.1.1 Gini splitting rule

Gini splitting rule (or Gini index) is most broadly used rule. It uses the following impurity function $i(t)$:

$$i(t) = \sum_{k \neq l} p(k|t)p(l|t) \tag{2.2}$$

where $k, l\ 1, \ldots, K$ - index of the class; $p(k|t)$ - conditional probability of class k provided we are in node t.

Applying the Gini impurity function 2.2 to maximization problem 2.1 we will get the following change of impurity measure $\Delta i(t)$:

$$\Delta i(t) = -\sum_{k=1}^{K} p^2(k|t_p) + P_l \sum_{k=1}^{K} p^2(k|t_l) + P_r \sum_{k=1}^{K} p^2(k|t_r)$$

Therefore, Gini algorithm will solve the following problem:

$$\underset{x_j \leq x_j^R,\ j=1,\ldots,M}{\arg\max} \left[-\sum_{k=1}^{K} p^2(k|t_p) + P_l \sum_{k=1}^{K} p^2(k|t_l) + P_r \sum_{k=1}^{K} p^2(k|t_r) \right] \tag{2.3}$$

Gini algorithm will search in learning sample for the largest class and isolate it from the rest of the data. Ginni works well for noisy data.

2.1.2 Twoing splitting rule

Unlike Gini rule, Twoing will search for two classes that will make up together more then 50% of the data. Twoing splitting rule will maximize the following change-of-impurity measure:

$$\Delta i(t) = \frac{P_l P_r}{4} \left[\sum_{k=1}^{K} \left| p(k|t_l) - p(k|t_r) \right| \right]^2$$

which implies the following maximization problem:

$$\underset{x_j \leq x_j^R,\ j=1,\ldots,M}{\arg\max} \left(\frac{P_l P_r}{4} \left[\sum_{k=1}^{K} \left| p(k|t_l) - p(k|t_r) \right| \right]^2 \right) \tag{2.4}$$

Although Twoing splitting rule allows us to build more balanced trees, this algorithm works slower than Gini rule. For example, if the total number of classes is equal to K, than we will have 2^{K-1} possible splits.

Besides mentioned Gini and Twoing splitting rules, there are several other methods. Among most used are Enthropy rule, χ^2 rule, maximum deviation rule. But it has been proved [1] that final tree is insensitive to the choice of splitting rule. It is pruning procedure which is much more important.

We can compare two trees, build on the same dataset but using different splitting rules. With the help of `cartdrawpdfclass` command in XploRe, one can construct classification tree in PDF, specifying which splitting rule should be used (setting the third parameter to 0 for Gini rule or to 1 - for Twoing):

- `cartdrawpdfclass(x,y, 0, 1)` - Gini splitting rule
- `cartdrawpdfclass(x,y, 1, 1)` - Twoing splitting rule

[1] Breiman, Leo; Friedman, Jerome; Olshen, Richard; Stone, Charles (1984). *Classification and Regression Trees*, Chapman & Hall, page 95

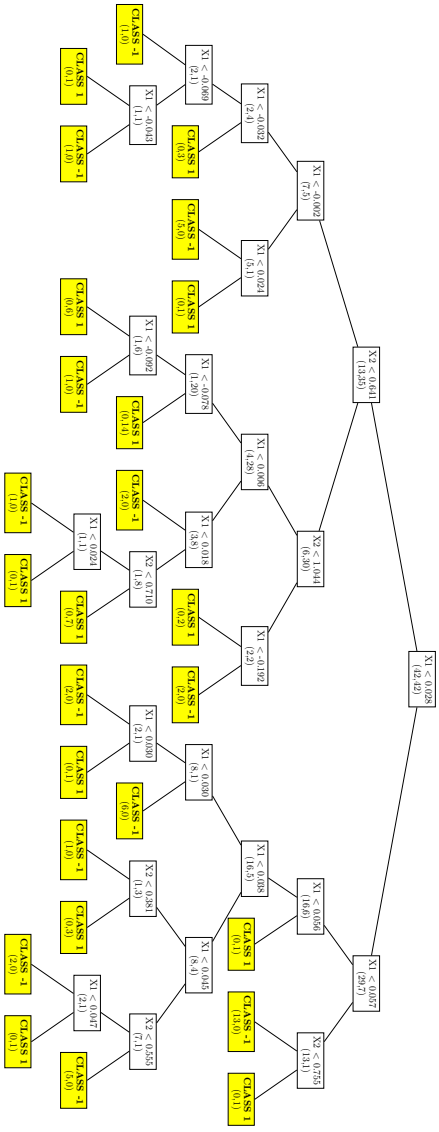

Figure 2.2: Maximum classification tree for bancruptcy dataset, constructed using Gini splitting rule.

CARTGiniTree1.xpl

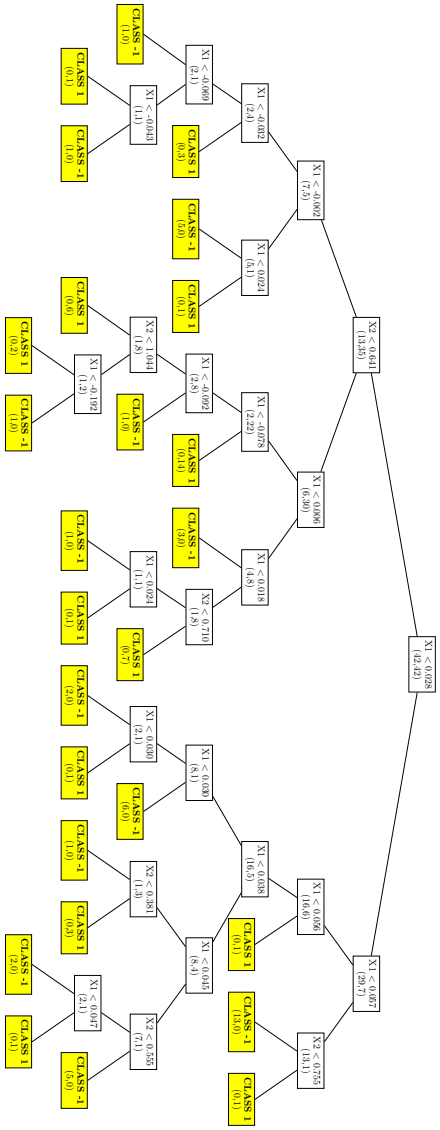

Figure 2.3: Maximum classification tree for bancruptcy dataset, constructed using Twoing splitting rule.

CARTTwoingTree1.xpl

It can be seen that although there is a small difference between tree contructed using Gini and tree constructed via Twoing rule, the difference can be seen only at the bottom of the tree where the variables are less significant in comparison with top of the tree.

2.2 Regression tree

Regression trees do not have classes. Instead there are response vector Y which represents the response values for each observation in variable matrix X. Since regression trees do not have pre-assigned classes, classification splitting rules like Gini 2.3 or Twoing 2.4 can not be applied.

Splitting in regression trees is made in accordance with *squared residuals minimization algorithm* which implies that expected sum variances for two resulting nodes should be minimized.

$$\underset{x_j \leq x_j^R,\ j=1,\ldots,M}{\arg\min} [P_l \mathrm{Var}(Y_l) + P_r \mathrm{Var}(Y_r)] \tag{2.5}$$

where $\mathrm{Var}(Y_l), \mathrm{Var}(Y_r)$ - reponse vectors for corresponding left and right child nodes; $x_j \leq x_j^R,\ j = 1, \ldots, M$ - optimal splitting question which satisfies the condition 2.5.

Squared residuals minimization algorithm is identical to Gini splitting rule. Gini impurity function 2.2 is simple to interpret through variances notation. If we assign to objects of class k the value 1, and value 0 to objects of other classes, then sample variance of these values would be equal to $p(k|t)[1 - p(k|t)]$. Summarizing by number of classes K, we will get the following impurity measure $i(t)$:

$$i(t) = 1 - \sum_{k=1}^{K} p^2(k|t)$$

Up to this point so-called maximum tree was constructued which means that splitting was made up to the last observations in learning sample. Maximum tree may turn out to be very big, especially in the case of regression trees, when each response value may result in a separate node. Next chapter is devoted to different prunning methods - procedure of cutting off insignificant nodes.

3 Choice of the Right Size Tree

Maximum trees may turn out to be of very high complexity and consist of hundreds of levels. Therefore, they have to be optimized before being used for classification of new data. Tree optimization implies choosing the right size of tree - cutting off insignificant nodes and even subtrees. Two pruning algorithms can be used in practice: optimization by number of points in each node and cross-validation.

3.1 Optimization by minimum number of points

In this case we say that splitting is stopped when number of observations in the node is less than predefined required minimum N_{min} . Obviously the bigger N_{min} parameter, the smaller the grown tree. On the one hand this approach works very fast, it is easy to use and it has consistent results. But on the other hand, it requires the calibration of new parameter N_{min}. In practice N_{min} is usually set to 10% of the learning sample size.

While defining the size of the tree, there is a trade-off between the measure of tree impurity and complexity of the tree, which is defined by total number of terminal nodes in the tree $\tilde{T}$. Using the command `cartsplitclass` one can build the tree structure in XploRe and then define different parameters of the tree, namely:

- `cartimptree(tree)` - calculated the impurity measure of the tree as the sum of classification error for all terminal nodes.
- `cartleafnum(tree)` - returns number of terminal nodes in the tree

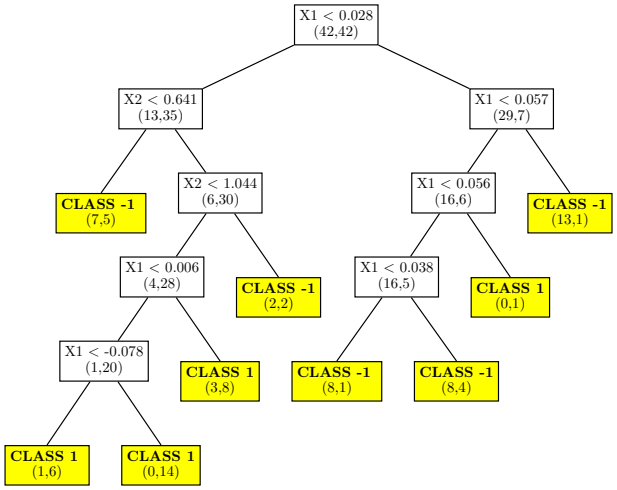

Figure 3.1: Classification tree for bancruptcy dataset, parameter N_{min} is equal to 15. Tree impurity is equal to 0.20238. Number of terminal nodes 9

CARTPruning1.xpl

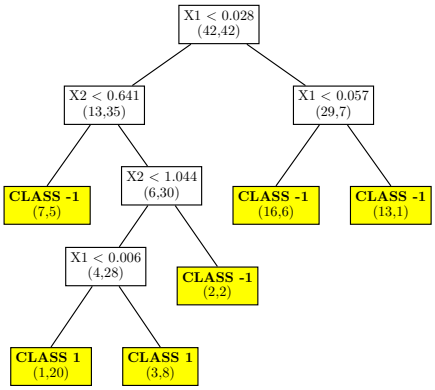

Figure 3.2: Classification tree for bancruptcy dataset, parameter N_{min} is equal to 30. Tree impurity is equal to 0.21429. Number of terminal nodes 6

CARTPruning2.xpl

We can see, that with the increase of the tree parameter N_{min}, on the one hand, the impurity increases (for $N_{min} = 15$, impurity is equal to 0.20238 and for $N_{min} = 30$ impurity is equal to 0.21429). On the other hand, the complexity of the tree decreases (for $N_{min} = 15$, number of terminal nodes $\tilde{T}$ is equal 9 and for $N_{min} = 30$ $\tilde{T} = 6$). For the maximum tree, the impurity measure will be minimum and equal to 0, but number of terminal nodes $\tilde{T}$ will be maximum. To find the optimal tree size, one can use cross-validation procedure.

3.2 Cross-validation

The procedure of cross validation is based on optimal proportion between the complexity of the tree and misclassification error. With the increase in size of the tree, misclassification error is decreasing and in case of maximum tree, misclassification error is equal to 0. But on the other hand, complex decision trees poorly perform on independent data. Performance of decision tree on independent data is called *true predictive power* of the tree. Therefore, the primary task - is to find the optimal proportion between the tree complexity and misclassification error. This task is achieved through cost-complexity function:

$$R_\alpha(T) = R(T) + \alpha(\tilde{T}) \longrightarrow \min_T \tag{3.1}$$

where $R(T)$ - misclassification error of the tree T; $\alpha(\tilde{T})$ - complexity measure which depends on $\tilde{T}$ - total sum of terminal nodes in the tree. α - parameter is found through the sequence of in-sample testing when a part of learning sample is used to build the tree, the other part of the data is taken as a testing sample. The process repeated several times for randomly selected learning and testing samples.
Although cross-validation does not require adjustment of any parameters, this process is time consuming since the sequence of trees is constructed. Because the testing and learning sample are chosen randomly, the final tree may differ from time to time.

4 Classification of New Data

As the classification or regression tree is constructed, it can be used for classification of new data. The output of this stage is an assigned class or response value to each of the new observations. By set of questions in the tree, each of the new observations will get to one of the terminal nodes of the tree. A new observation is assigned with the *dominating class/response value* of terminal node, where this observation belongs to.

Dominating class - is the class, that has the largest amount of observations in the current node. For example, the node with 5 observations of class 1, two observation of class 2 and 0 observation of class 3, will have class 1 as a dominating class.

5 Advantages and Disadvantages of CART

This chapter answers an important questions: "Why should we use CART?". Before applying CART to real sector, it is important to compare CART with other statistical classification methods, identify its advantages and possible pitfalls.

5.1 CART as a classification method

- **CART is nonparametric**. Therefore this method does not require specification of any functional form
- **CART does not require variables to be selected in advance.**

  CART algorithm will itslef identify the most significant variables and eleminate non-significant ones.

  To test this property, one can inlude insignificant (random) variable and compare the new tree with tree, built on initial dataset. Both trees should be grown using the same parameters (splitting rule and N_{min} parameter). We can see that the final tree 5.1, build on new dataset of three variables, is the identical to tree 3.2, built on two-dimentional dataset.

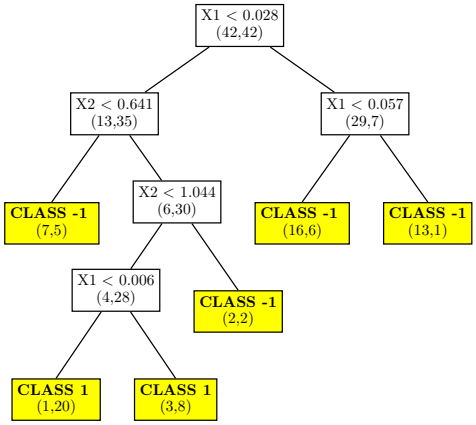

Figure 5.1: Classification tree for bancruptcy dataset after including a third random variable

CARTRandom.xpl

- **CART results are invariant to monotone transformations of its independent variables.**

  Changing one or several variables to its logarithm or square root will not change the structure of the tree. Only the splitting values (but not variables) in the questions will be different.

  I have replaced the values of the first variable with corresponding $log(x_1 + 100)$ values. It can be seen that the structure of the tree did not change, but changed the splitting values in questions with the first variable.

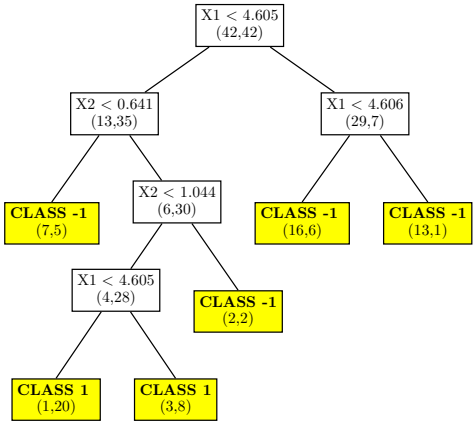

Figure 5.2: Classification tree for bancruptcy dataset after monotone transformation of the first variable $log(x_1 + 100)$

CARTLog.xpl

- **CART can easily handle outliers**.

  Outliers can negatively affect the results of some statistical models, like Principal Component Analysis (PCA) and linear regression. But the splitting algorithm of CART will easily handle noisy data: CART will isolate the outliers in a separate node. This property is very important, because financial data very often have outliers due to financial crisises or defaults.

  In order to test CART ability to handle outliers, I have included instead of usual x_1 of about $[-0.5; 0.5]$ two observation with x_1 equal to 25 and 26. Let us see how the tree structure changed:

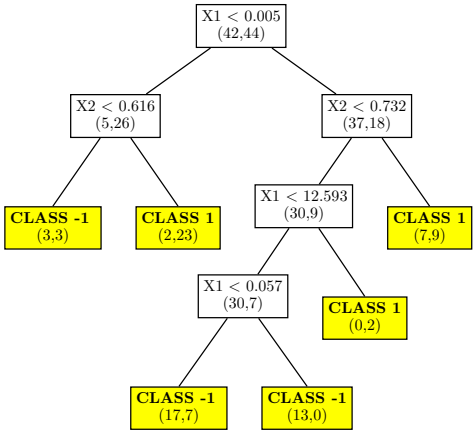

Figure 5.3: Classification tree for bancruptcy dataset after two outliers

CARTOutlier.xpl

We can see that both outliers were isolated to a separate node.

5.2 CART in financial sector

- **CART has no assumptions and computationally fast.**

  There are plenty of models that can not be applied to real life due to its complexity or strict assumptions. In the table 5.2 there is an computational efficiency of CART modul in XploRe. One can see that a dataset with 50 variables and 50000 observations is processed in less than 3 minutes.

Number of observations	Number of variables				
	2	10	20	40	50
1000	0	0	0	0	0
5000	1	2	2	4	6
10000	1	4	6	13	14
30000	13	25	38	62	71
50000	33	60	85	142	170

Table 5.1: Computational efficiency of CART module in XploRe (in seconds)

- **CART is flexible and has an ability to adjust in time.**

  The main idea is that learning sample is consistently replanished with new observations. It means that CART tree has an important ability to adjust to current situatation in the market.

  Many banks are using the Basel II credit scoring system to classify different companies to risk levels, which uses a group of coefficients and inidicators. This approach, on the other hand, requires continuous correction of all indicators and coefficients in order to adjsut to market changes.

5.3 Disadvantages of CART

As any model, method of classification and regression trees has its own weaknesses.

- **CART may have unstable decision trees.**

  Insignificant modification of learning sample, such as elminating several observations, could lead to radical changes in decision tree: increase or descease of tree complexity, changes in splitting variables and values.

  At figure 5.4 one can see how the decision tree constructed for 90% of bacruptcy data is different from initial classification tree.

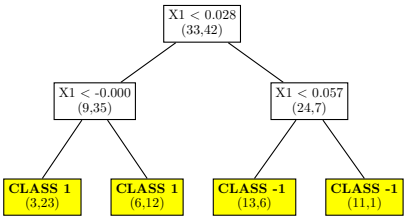

Figure 5.4: Classification tree, countrcuted on 90% data of bancruptcy dataset

CARTStab.xpl

  One can notice that tree complexity decreased from 5 levels to 3 levels. In the new classification tree only x_1 participate1 in splitting questions, therefore x_2 is not considered significant anymore. Obviously, classification results of data will change with the use of new classification tree. Therefore, instability of trees can negatively influence the financial results.

- **CART splits only by one variable.**

  In other words, all splits are perpendicular to axis. Let us consider two different examples of data structure. At the first picture 5.5 there are 3 classes (red, grey and green). CART will easily handle the splits and it can be seen on the right picture.

  Although, if data have more complex structure, as for example at figure 5.6, then CART may not catch the correct structure of the data. From example 5.6 it can be seen that CART can not correctly identify question $x_1 - x_2 \leq 0$ because in split question can participate only one variable. In order to capture the data structure, splitting algorithm will generate many splits (nodes) at the border of $x_1 - x_2 \leq 0$ line.

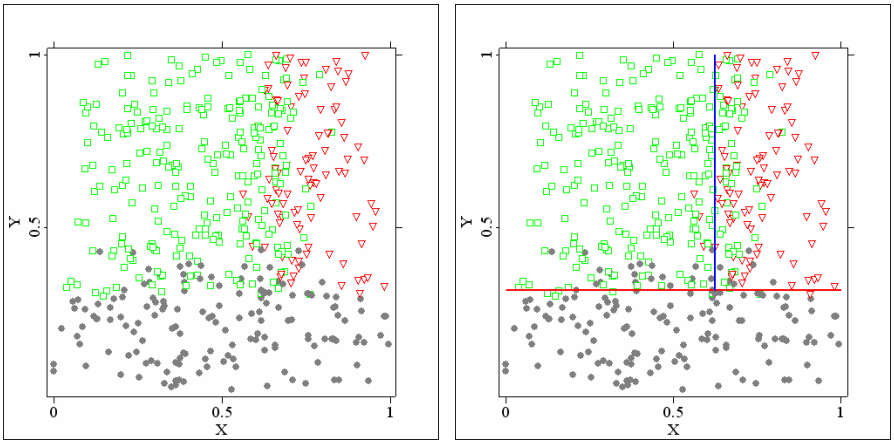

Figure 5.5: Dataset of three classes with linear structure

In the end CART will grow a huge tree where almost each observation at the border will be in a separate node. But despite the big tree, classification will be done correctly and all observations that belong to red class will be classified as a red, green observation as green and etc.

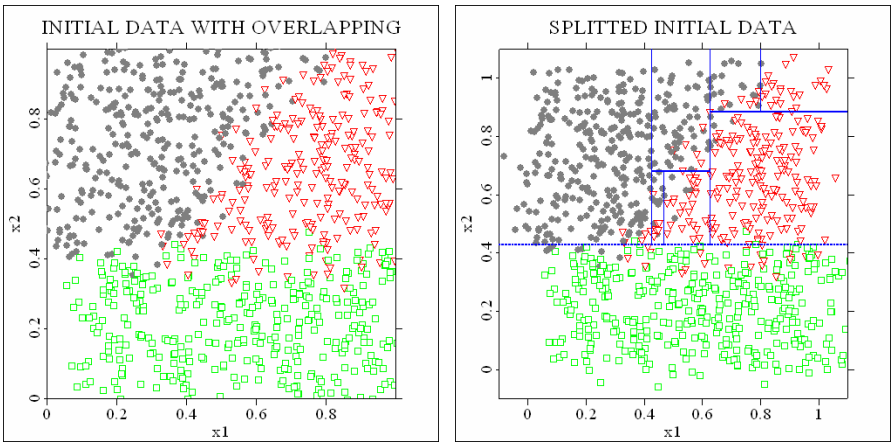

Figure 5.6: Dataset of three classes with non-linear structure

6 Examples

6.1 Simulated example

Let's simulate a simple example with three classes.

```
proc(y) = simulate(seed, n)
    randomize(seed)
    // generating data with colour layout
    xdat=uniform(n,2)
    index = (xdat[,2]<=0.5) + (xdat[,2]>0.5).*(xdat[,1]<=0.5)*2
    color = 2.*(index==1)+1.*(index==0) + 4.*(index==2)
    layout= 4.*(index==1)+2.*(index==0) + 3.*(index==2)
    // return the list
    y = list(xdat, index, color, layout)
endp
library("xclust")
d = createdisplay(1,1)
data = simulate(1, 500)
x = data.xdat
setmaskp(x, data.color, data.layout)
show(d, 1, 1, x)
```

The generated data will have the following structure:

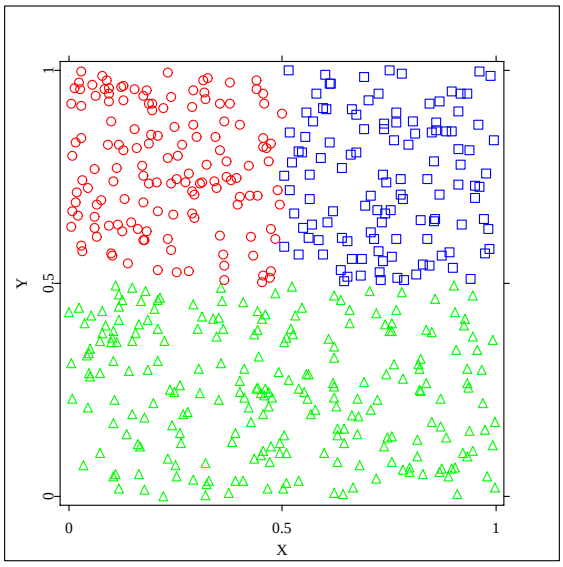

Figure 6.1: Simulated uniform data with 3 classes

CARTSim1.xpl

Each colour indicates a class (red - class 2, blue - class 0, green - class 1). Therefore we have 500 observations with three classes. Visually it can be seen that data can be perfectly splitted at $y \leq 0.5$ and then at $x \leq 0.5$. Let's see how classification tree will identify the data structure.

```
library("xclust")
data = simulate(1, 500)
tr = cartsplitclass(x, data.index, 0, 1)
cartdisptree(tr)
```

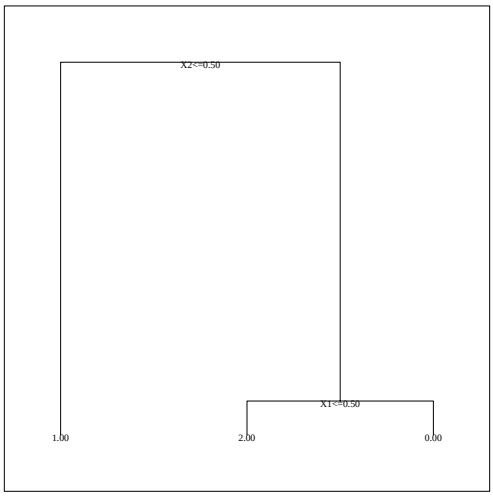

Figure 6.2: Classification tree for generated data of 500 observations

CARTSimTree1.xpl

But in real life, the data usually is not perfectly split. We can try to simulate the real life example with overlapping over different classes.

```
proc(y) = simulate(seed, n)
    randomize(seed)
    // generating data with color layout
    xdat=uniform(n,2)
    index = (xdat[,2]<=0.5) + (xdat[,2]>0.5).*(xdat[,1]<=0.5)*2
    color = 2.*(index==1)+1.*(index==0) + 4.*(index==2)
    layout= 4.*(index==1)+2.*(index==0) + 3.*(index==2)
    // generating overlapping
    overlapping = 0.01
    xdat[,2]=xdat[,2]+(index==1)*overlapping
    xdat[,2]=xdat[,2]-(index==2)*overlapping-(index==0)*overlapping
    xdat[,1]= xdat[,1]+(index==2)*overlapping
    xdat[,1]= xdat[,1]-(index==0)*overlapping
    // return the list
    y = list(xdat, index, color, layout)
endp
library("xclust")
d = createdisplay(1,1)
data = simulate(1, 500)
x = data.xdat
setmaskp(x, data.color, data.layout)
show(d, 1, 1, x)}
```

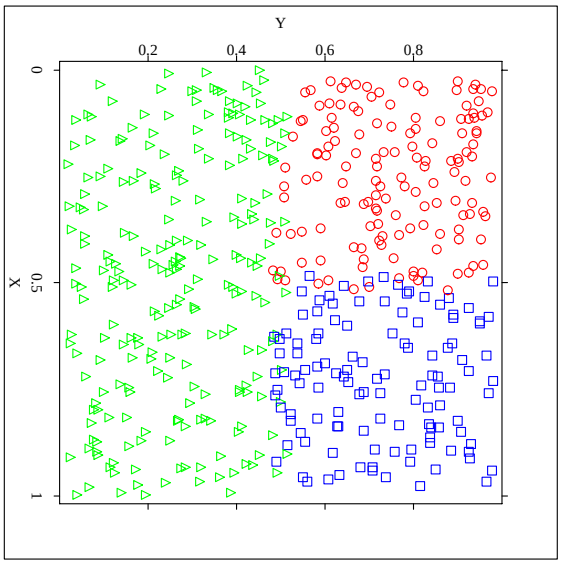

Figure 6.3: Simulated uniform data with three classes and overlapping

CARTSim2.xpl

This time maximum tree will consist of much more levels, since CART algorithm tries to split all observations of learning sample.

```
data = simulate(1, 500)
x = data.xdat
tr = cartsplitclass(x, data.index, 0, 1)
catdisptree(tr)}
```

In order to see number of observations for each split, run the following code:

```
data = simulate(1, 500)
x = data.xdat
tr = cartsplitclass(x, data.index, 0, 1)
catdisptree(tr, "NumberOfPoints")
```

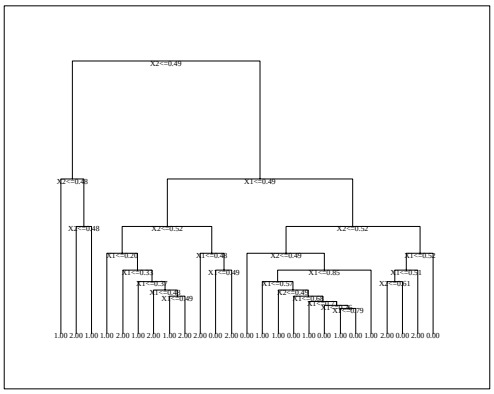

Figure 6.4: Maximum classification tree for simulated two dimensional data with overlapping

CARTSimTree2.xpl

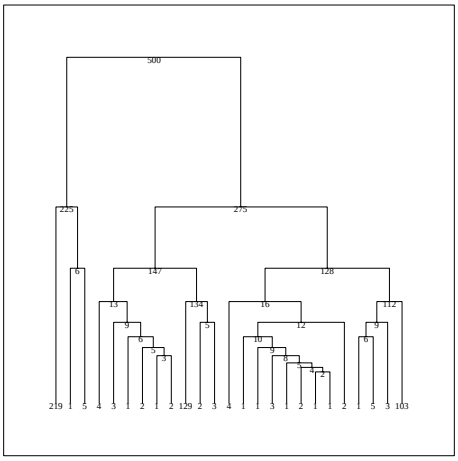

Figure 6.5: Maximum tree for simulated two dimensional data with overlapping and "NumberOfPoints" option

CARTSimTree3.xpl

It can be seen at figures 6.4 and 6.5, that CART first makes more important splits - splits $x_2 \leq 0.49$ and then $x_1 \leq 0.49$. Afterwards CART tries to capture the overlapping structure which we simulated: CART isolates the observations of the same class in separate nodes.

For building and analysing big tree, `cartdrawpdfclass` command can be used. The following code will generate TEX file with classification tree:

```
data = simulate(1, 500)
x = data.xdat
cartdrawpdfclass(x, data.index, 0, 1, "c:\textbackslash
  ClassificationTree.tex")
```

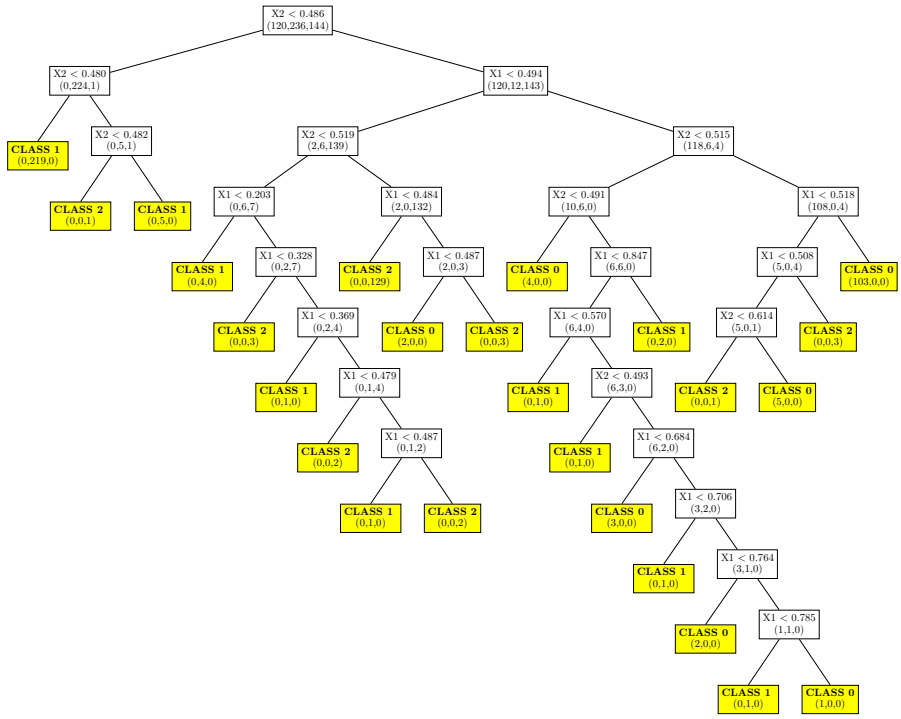

Figure 6.6: PDF tree generated via `cartdrawpdfclass` command

`CARTSimTree4.xpl`

6.2 Boston housing example

Boston Housing is a classical dataset which can be easily used for regression trees. On the one hand, we have 13 independent variables, on the other hand, there is response variable - value of house (variable number 14).

Boston housing dataset consists of 506 observations and includes the following variables:

1. crime rate
2. percent of land zoned for large lots
3. percent of non-retail business
4. Charles river indicator, 1 if on Charles river, 0 otherwise
5. nitrogen oxide concentration
6. average number of rooms
7. percent built before 1980
8. weighted distance to employment centers
9. accessibility to radial highways
10. tax rate
11. pupil-teacher ration
12. percent black
13. percent lower status
14. median value of owner-occupied homes in thousands of dollars

Let us choose a small sample of 100 observations and then build the regression tree with minimum 10 observations in terminal node:

```
library("xclust")
boston = read("bostonh")
data = boston[1:100,]
Var = data[,1:13]
Class = data[,14]
tr = cartsplitregr(Var, Class, 10)
cartdisptree(tr)
NumLeaf = cartleafnum(tr)
NumLeaf
```

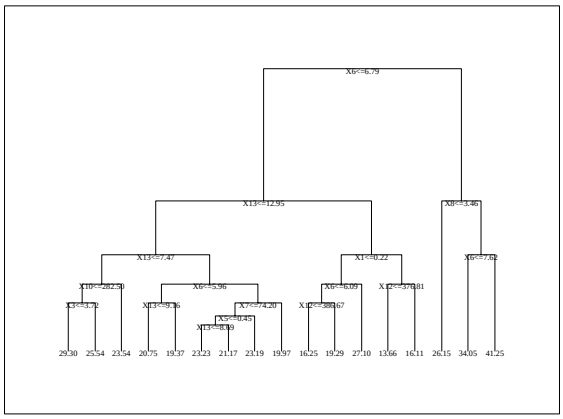

Figure 6.7: Regression tree for 100 observations of Boston housing dataset, minimum number of observations N_{min} is set to 10.

CARTBoston1.xpl

It is also important to notice, that at the upper levels of the tree there are more significant variables, and less significant at the bottom of the tree. Therefore, we can state, that for Boston housing dataset, average number of rooms ($x6$) is most significant, since it is located in the root node of the tree. Then come variable $x13$ (percent lower status) and $x8$ (weighted distance to employment center).

Taking into account the fact that in regression trees we do not have classes, but have response values, maximum tree will contain as many terminal nodes as there are observation in the dataset, because each observation has a different response value. On the contrary, classification tree approach uses classes instead of response values, therefore splitting can be automatically finished when the node contains observations of one class.

If we try to count number of terminal nodes for the maximum tree, then we find out that there are 100 terminal nodes in the regression tree:

```
library("xclust")
boston = read("bostonh")
data = boston[1:100,]
Var = data[,1:13]
Class = data[,14]
tr = cartsplitregr(Var, Class, 1)
NumLeaf = cartleafnum(tr)
NumLeaf
```

As we found out, the optimization (pruning) of the tree even more important for regression trees, since the maximum tree is too big. There are several methods for optimal tree pruning. Most used method - cross-validation which uses cost-complexity function 3.1 to determine the optimal complexity of the tree. Tree complexity is defined by number of terminal nodes $\tilde{T}$.

The other method - adjusting the parameter of minimum number of observations. By increasing the parameter, the size of the tree will decrease and vice versa. Let us build the regression tree with at least 30 observations in each of the terminal nodes:

```
library("xclust")
boston = read("bostonh")
data = boston[1:100,]
Var = data[,1:13]
Class = data[,14]
tr = cartsplitregr(Var, Class, 30)
cartdisptree(tr)
```

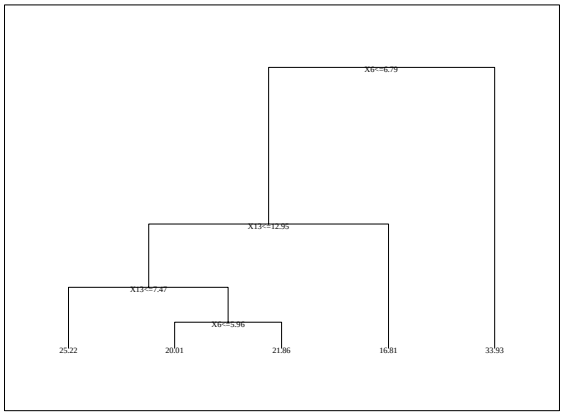

Figure 6.8: Regression tree for 100 observations of Boston housing dataset, N_{min} parameter is set to 30.

CARTBoston2.xpl

From this tree it can be seen that only two variables are found significant: $x6$ - number of rooms and $x13$ - percentage lower status. The parameter minimum number of observation in the terminal node has to be adjusted by iterative testing procedure when a part of dataset is used as an learning sample and the rest of the data - as an out-of-sample testing sample. By this procedure one can determine the tree of which size performs in the best way. We will illustrate this example on bankruptcy dataset.

6.3 Bankruptcy data example

The bankruptcy dataset - two-dimensional dataset, which includes the following variables

1. Net income to total assets ratio
2. Total liabilities to total assets ratio

For each of 84 observations there is a third column - bankruptcy class. In case of the company went bankrupt - the class 1, otherwise class -1. For this dataset we shall use classification procedures, since we do have only classes, no response values.

Let us build the classification tree with minimum number of observations N_{min} equal to 30.

```
library("xclust");
data = read("bankruptcy.dat")
x = data[,1:2];
y = data[,3]
cartdrawpdfclass(x,y,0,30, "C:\textbackslash ClassificationTree.tex")
```

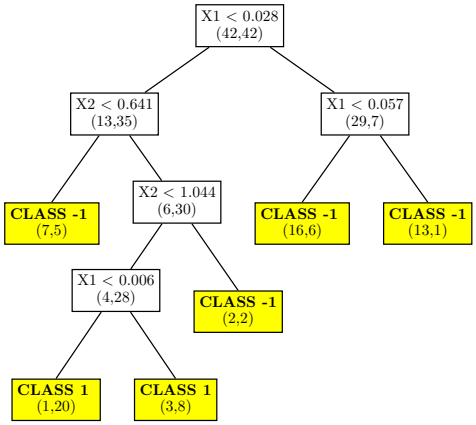

Figure 6.9: Classification tree of bankruptcy dataset, minimum number of observations N_{min} is set to 30.

CARTBan1.xpl

If we change the parameter of minimum number of observations N_{min} to 10, then we will get a classification tree of higher complexity:

```
library("xclust");
data = read("bankruptcy.dat")
x = data[,1:2];
y = data[,3]
cartdrawpdfclass(x,y,0,10, "C:\textbackslash ClassificationTree.tex")
```

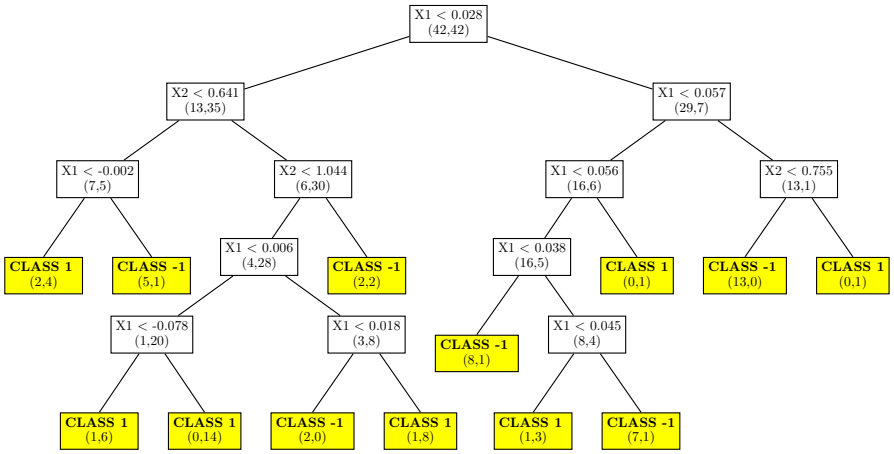

Figure 6.10: Classification tree of bankruptcy dataset, minimum number of observations N_{min} is set to 10.

CARTBan2.xpl

The question is which tree to choose. The answer can be found by iterative procedure of calculation of each tree performance. Let us take 83 observations as a learning sample, and 1 left observation as a testing sample. We will use 83 observations to construct the tree and then using constructed tree, classify the out-of-sample testing sample (1 observation). Since we actually know the classes of testing sample, we can determine the so-called classification ratio - ratio of correct classifications to total number of observations in testing sample. We can choose the learning sample and testing sample in 84 different ways, therefore, we can loop the procedure over possible number of iterations which is in our case 84.

```
proc() = bancrupcy()
data = read("bankrupcy.dat")
x = data[,1:2];
y = data[,3]
NumberOfLines = rows(data)
index = (1:NumberOfLines)
CorrectAnswers = 0
MinSize = 1
i = 1
while (i <= NumberOfLines)
    newx = paf(x,index<>i)
    newy = paf(y,index<>i)
    testingData = data[i,1:14];
    realClass = data[i,15];]
    tr = cartsplitclass(newx,newy, 0, 1);
    predictedClass = cartpredict(tr, testingData)
    if (predictedClass == realClass)
    CorrectAnswers = CorrectAnswers + 1
    endif
    i = i + 1
endo
CorrectAnswers/NumberOfLines
endp
bancrupcy()
```

The quantlet will return 73.81% value of classification ratio for maximum tree ($MinSize = 1$) and gini rule ($SplitRule = 0$). We can run this procedure for different parameters and build the distribution of classification ratio over tree size.

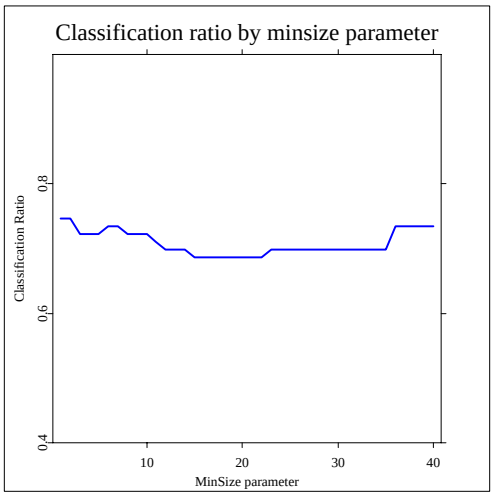

Figure 6.11: Distribution of classification ratio by N_{min} parameter

So in this case, it turned out that maximum tree gives the best classification ratio at rate of 73.81%. One can see that the dependence of classification performance over different tree sizes is not monotone: at first it decreases, but beginning with $N_{min} = 15$ the performance improves. It can be explained by the fact that simpler trees may reflect the actual data structure and do not capture any small dependencies which may be in fact misleading.

Let us depict the tree with N_{min} equal to 40 which performs at the ratio of 72.619%:

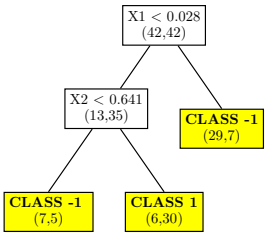

Figure 6.12: Classification tree of bankruptcy dataset, minimum number of observations N_{min} is set to 40.

CARTBan3.xpl

```
library("xclust");
data = read("bankruptcy.dat")
x = data[,1:2];
y = data[,3]
tr = cartsplitclass(x,y,0,40)
impurity = carttrimp(tr)
impurity
```

Analyzing this very simple tree we can state that the overall impurity of the tree, which is claculated as the sum of misclassification ratios for all terminal nodes, is equal to 21.429%. Impurity is an inverse value of successful classification ratio, i.e. the higher the impurity measure, the less classification power has the tree. In this example we can see that despite tree simplicity, the impurity is quite low and this is probably why this tree performs well on independent data.