Classification and regression trees

From a course by Anne Ruiz-Gazen

Camille Mondon

1 About trees

1.1 Introduction

- **CART** are also called **decision trees**.
- One variable to explain, that can be:
  - qualitative (categorical) → **classification trees**;
  - or quantitative (numerical) → **regression trees**.

  This means that decision tree learning is a **supervized** method (contrarily to PCA and clustering that are unsupervized methods): part of machine learning algorithms that "learn" from the data.
- Predictors are either numerical or categorical.
- A decision tree is defined using a **training** sample and tested on a **test** sample (split the original sample in two samples).
- The method is nonlinear.
- Different algorithms exist. We consider only **CART** developed by Breiman et al. (1984).

1.2 Examples

1. Predict the **land use** of a given area (agriculture, forest, etc.) given satellite information, meteorological data, socio-economic information, prices information, etc.
2. Predict high risk for **heart attack**:
   - University of California: a study into patients after admission for a heart attack.
   - 19 variables collected during the first 24 hours for 215 patients (for those who survived the 24 hours)
   - Question: Can the high risk (will not survive 30 days) patients be identified?
3. Determine how **computer performance** is related to a number of variables which describe the features of a PC (the size of the cache, the cycle time of the computer, the memory size and the number of channels. Both the last two were not measured but minimum and maximum values obtained).

1.2.1 Risk of heart attack

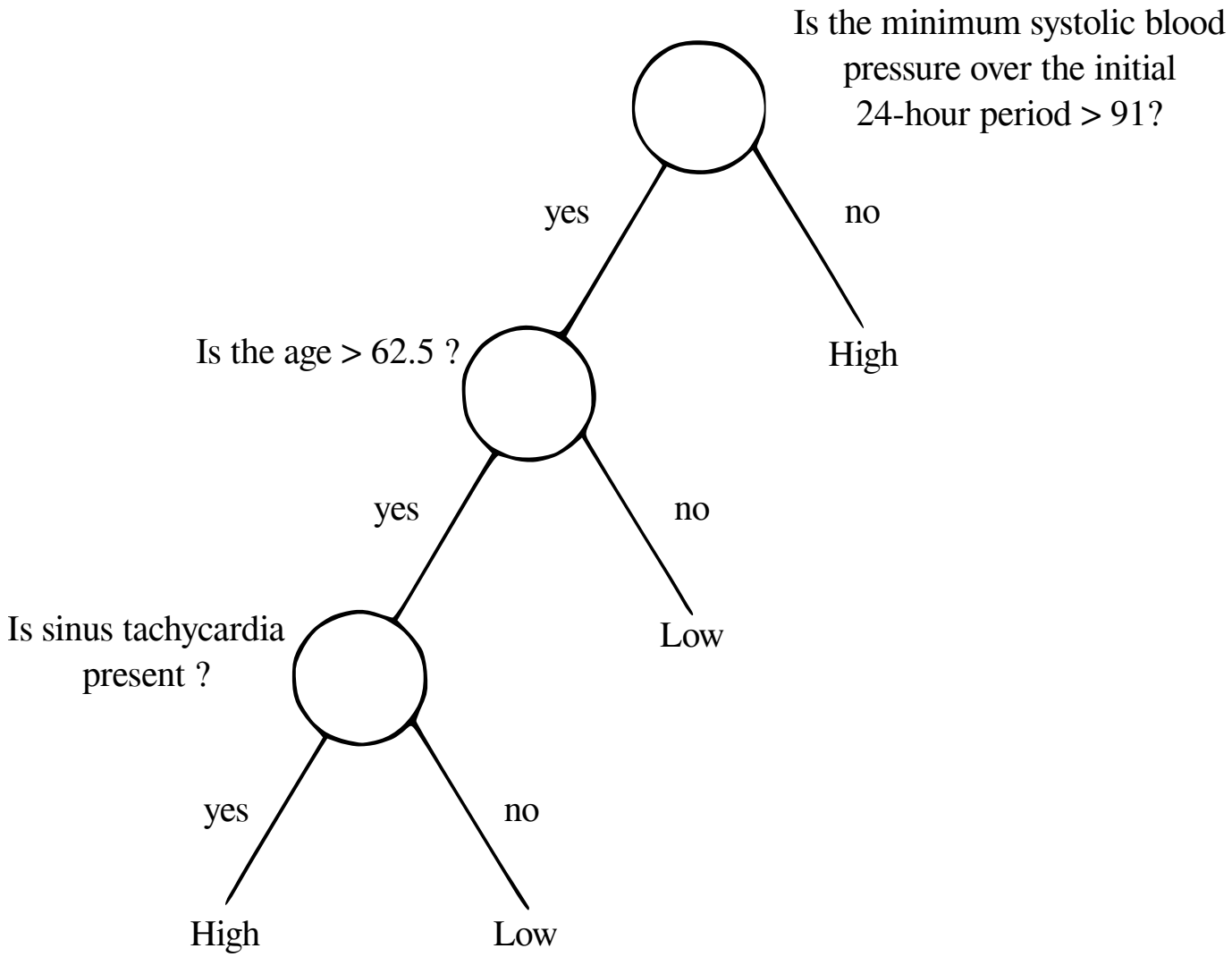


Figure 1: A binary tree to identify patients at high risk of heart attack. Source: (Breiman et al. 1984, fig. 1.1).

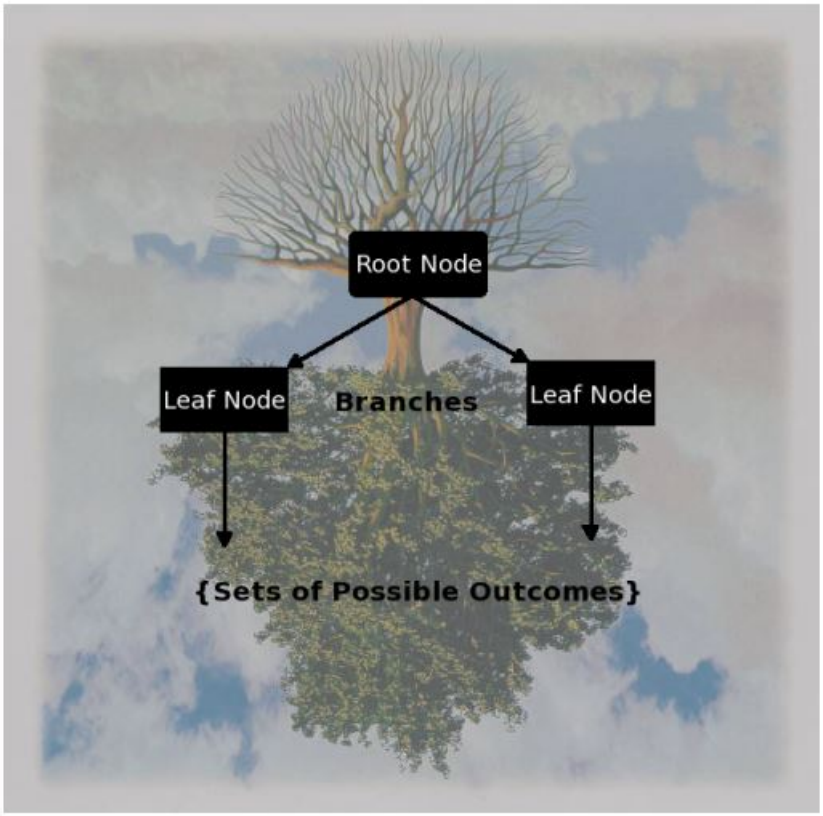


Figure 2: A (reversed) true tree.

1.3 Vocabulary

- The top, or first node, is called the **root node**.
- The last level of nodes are the **leaf nodes** and contain the final classification.
- The intermediate nodes are **child nodes**.
- **Binary trees** (two child nodes) are the most popular type of tree. However, M-ary trees (M branches at each node) are possible.
- Nodes contain one question. In a binary tree, by convention if the answer to a question is “yes", the left branch is selected. Note that the same question can appear in multiple places in the tree.
- Key questions include **how to grow the tree, how to stop growing, and how to prune the tree** to increase generalization (good results should not be obtained only on the training set).

1.4 When to use decision trees?

Decision trees are well designed for

- problems with a **big number of variables** (a variable selection is included in the method);
- providing an **intuitive interpretation** with a simple graphic;
- solving **either regression and classification problems with quantitative or qualitative predictors**, hence the name **CART**.

but they face several difficulties:

- they **need a large training dataset** to be efficient;
- as they select the variables hierarchically, they can **fail to find a global optimum**.

2 Procedure

2.1 Overview

The algorithm is based on two steps:

- a **recursive binary partitioning** of the feature space spanned by the predictor variables into increasingly homogeneous groups with respect to the response variable as measured by:
  - For regression: Mean squared error.
  - For classification (with classes $k = 1, 2, \ldots$): the **Gini index** (or entropy or misclassification rate).
- a **pruning** algorithm of the tree when it is too complex.

```
library(rpart)
library(rpart.plot)

cars_fit <- rpart(Price ~ ., data = cu.summary, minbucket = 1, cp = 0)

rpart.plot(cars_fit, tweak = 2)
```

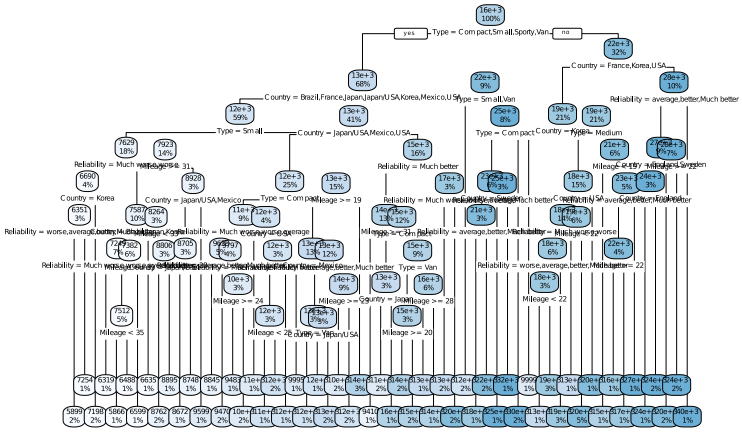


Figure 3: This tree is too complex, it needs to be pruned!

The subgroups of data created by the partitioning are called **nodes**.

- For **regression** trees the predicted (fitted) value at a node is just the **average** value of the response variable for all observations in that node.
- For **classification** trees the predicted value is the class in which there is the **largest** number of the observations at the node.
- For **classification** trees one also gets the estimated probability of membership in each of the classes from the proportion of the observations at the node in each of the classes.

The subgroups of data created by the partitioning are called **nodes**.

- At each step, an optimization is carried out to select:
  - A node,
  - A predictor variable,
  - A cut-off value if the predictor variable is numerical, or a group of modalities if the predictor variable is categorical,
- That maximizes the (percentage) decrease in the objective function (homogeneity measure).

2.2 Notations

The data is built from a pair of random variables, (X, Y), where X is made of p **qualitative or quantitative predictors,** $X^1, \ldots, X^p$, and Y **is a qualitative (classification) or quantitative (regression) variable to predict from** X.

The data consists in n i.i.d. observations $(x_1, y_1), \ldots, (x_n, y_n)$ of (X, Y).

As explained previously, building a decision tree aims at finding a series of:

- **nodes**, which are defined by the **choice of a variable** X^k,
- **splits** deduced from this variable and a **threshold** τ, dividing the training sample into two subsamples: $\{i : x_i^k < \tau\}$ and $\{i : x_i^k > \tau\}$ (or in two groups of modalities of x is categorical).

In the following, we will note the nodes with the following convention:

- the **root** will be denoted by $\mathcal{N}^0$;
- the **child nodes of the root** will be denoted by $\mathcal{N}_k^1$ with $k = 1, 2$;
- the **child nodes of** $\mathcal{N}_k^1$ will be denoted by $\mathcal{N}_{2k-1+j}^2$ with $k = 1, 2$ and $j = 0, 1$ (hence these nodes are indexed from 1 to 4);
- ...
- the **nodes at step** m will be denoted by $\mathcal{N}_k^m$ for $k = 1, \ldots, 2^m$ (where, eventually, some of the indexes can be unused): $\mathcal{N}_1^m$ and $\mathcal{N}_2^m$ are the child nodes of $\mathcal{N}_1^{m-1}$, $\mathcal{N}_3^m$ and $\mathcal{N}_4^m$ are the child nodes of $\mathcal{N}_2^{m-1}$, ...
- ...

2.3 Growing the tree

2.3.1 The need of an homogeneity criterion

When splitting $(y_i)_i$ into two groups, we aim at having two non empty groups with Y values as **homogeneous** as possible.

Then, we have to define an homogeneity criterion for each node, i.e. a function

$$H : \mathcal{N} \to \mathbb{R}_+$$

that is

- **null if and only if** all the individuals that belong to the node have the same Y value;
- **large when** all the Y values are very different.

The split of a node $\mathcal{N}_k^m$ is chosen, among all possible splits, by **minimizing** the sum of the homogeneity of the corresponding child nodes:

$$\arg \max_{\text{splits of } \mathcal{N}_k^m} H(\mathcal{N}_k^m) - \left(n_1 H(\mathcal{N}_{2k-1}^{m+1}) + n_2 H(\mathcal{N}_{2k}^{m+1}\right) / (n_1 + n_2).$$

2.3.2 Homogeneity of a split

Consider a given node $\mathcal{N}_k^m$ that has to be split into two classes (two new child nodes) and denote by

- $n_1 = \sum_{i=1}^n \mathbb{1}_{\{y_i \in \mathcal{N}_{2k-1}^{m+1}\}}$ and $n_2 = \sum_{i=1}^n \mathbb{1}_{\{y_i \in \mathcal{N}_{2k}^{m+1}\}}$;
- for $j \in \{-1, 0\}$ and a regression tree $H(\mathcal{N}_{2k+j}^{m+1}) = \sum_{i=1}^n (y_i - \mathcal{Y}_{2k+j})^2 \mathbb{1}_{\{y_i \in \mathcal{N}_{2k+j}^{m+1}\}}$ where $\mathcal{Y}_{2k+j}$ is the predicted value in node $\mathcal{N}_{2k+j}^{m+1}$: $\mathcal{Y}_{2k+j} = \sum_{i=1}^n y_i \mathbb{1}_{\{y_i \in \mathcal{N}_{2k+j}^{m+1}\}}$. The sum of squared criterion is replaced by the Gini index for classification trees.

2.3.3 Building a split: small classification example detailed

```
library(ggplot2)
set.seed(0)

a <- data.frame(X_1 = rnorm(800), X_2 = rnorm(800), Y = as.factor("A"))
b <- data.frame(X_1 = rnorm(200, 2), X_2 = rnorm(200, 2), Y = as.factor("B"))
ab <- rbind(a, b)

ggplot(ab, aes(X_1, X_2, color = Y, label = Y)) +
  geom_point()
```

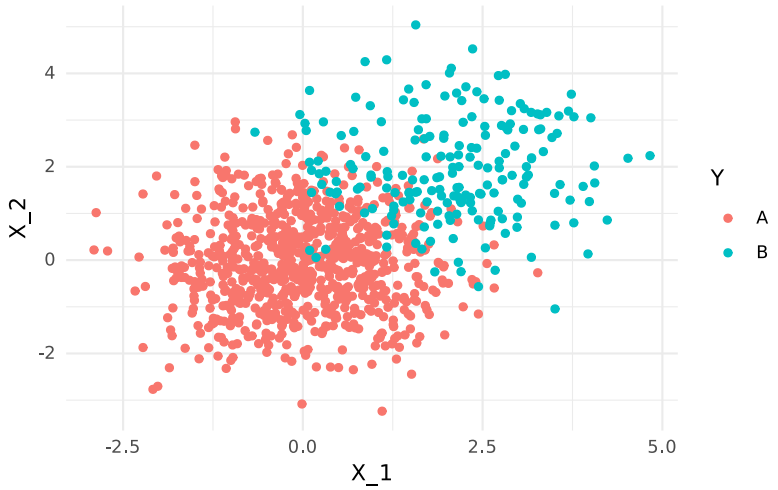


Figure 4: A two-classes variable to predict with two quantitative predictors X_1 and X_2.

```
library(parttree)

ab_fit <- rpart(Y ~ ., data = ab, minbucket = nrow(ab), cp = 0)
rpart.plot(ab_fit, box.palette = "RdBu")

ggplot(ab, aes(X_1, X_2, color = Y, label = Y)) +
  geom_point()

for (i in 1:3) {
  ab_fit <- rpart(Y ~ ., data = ab, maxdepth = i, cp = -Inf)
```

```
  rpart.plot(ab_fit, box.palette = "RdBu")

  p <- ggplot(ab, aes(X_1, X_2, color = Y, fill = Y)) +
    geom_parttree(data = ab_fit, alpha = 0.1, flip = i == 1) +
    geom_point()
  print(p)
}
```

- In order to split a node, we need to choose a criterion of inhomogeneity or measure of **impurity**.
- The measure should be at a maximum when a node is equally divided amongst the two classes.
- The impurity should be zero if the node is all one class.
- The Gini index is the most widely used measure of impurity (at least by CART):

$$1 - p_A^2 - p_B^2$$

  where p_A and p_B are respectively the proportion of observations in class A and in class B for a given node
- we compute the decrease of the Gini index when splitting a given node in two child nodes by calculating the Gini index at the given node and substract one half of the sum of the Gini indices at the child nodes as illustrated in the small example below.

- We select the split (a variable and a threshold) that most decreases the Gini Index. This is done over all possible places for a split and all possible variables to split.
- There are two possible variables to split on and each of those can split for a range of values of τ i.e.: $x < \tau$ or $y < \tau$.
- We keep splitting until the terminal nodes have very few cases or are all pure. Note that this seems an unsatisfactory answer to when to stop growing a tree, but it was realized that the best approach is to grow a larger tree than required and then to prune it.

2.3.4 Computing Gini indices

We consider the split: $X_1 < 1.64$.

```
ab$split <- ifelse(ab$X_1 < ab_fit$splits[1, 4], "Left", "Right")
dplyr::sample_n(ab, 10)
```

X_1	X_2	Y	split
-1.5017872	1.6803619	A	Left
-0.9233743	-0.8534551	A	Left
1.5462761	3.3774228	B	Left
-0.8320433	1.1174336	A	Left
-1.0854224	2.2061932	A	Left
0.9922884	0.0171323	A	Left
2.6586580	0.3255937	A	Right
-0.1726686	-0.2422814	A	Left
-0.8874201	-0.9109120	A	Left

X_1	X_2	Y	split
-1.1729836	-1.6521118	A	Left

```
ab_ct <- table(ab$split, ab$Y) |>
  addmargins() |>
  as.data.frame.matrix()
ab_ct
```

	A	B	Sum
Left	766	63	829
Right	34	137	171
Sum	800	200	1000

```
ab_freqs <- ab_ct$Sum / ab_ct$Sum[3]

(ab_probs <- ab_ct / ab_ct$Sum)
```

	A	B	Sum
Left	0.9240048	0.0759952	1
Right	0.1988304	0.8011696	1
Sum	0.8000000	0.2000000	1

```
ab_probs$Freq <- ab_freqs
ab_probs$Gini <- 1 - ab_probs$A^2 - ab_probs$B^2
ab_probs
```

	A	B	Sum	Freq	Gini
Left	0.9240048	0.0759952	1	0.829	0.1404398
Right	0.1988304	0.8011696	1	0.171	0.3185938
Sum	0.8000000	0.2000000	1	1.000	0.3200000

The total difference in Gini index of this split is:

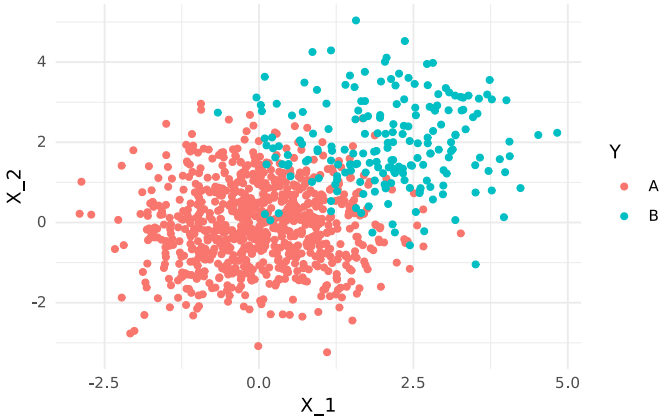
```
with(ab_probs, Freq[3] * Gini[3] - Freq[2] * Gini[2] - Freq[1] * Gini[1])
```

```
[1] 0.1490959
```

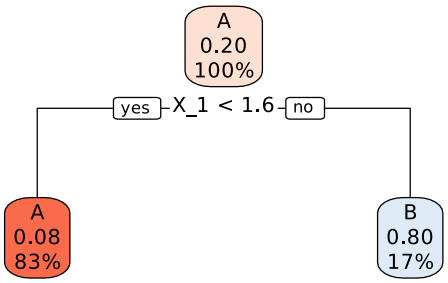
2.4 Stopping the tree

The growing of the tree is stopped if the **obtained node is homogeneous** or if the **number of observations in the nodes is smaller than a fixed number** (generally chosen between 1 and 5).

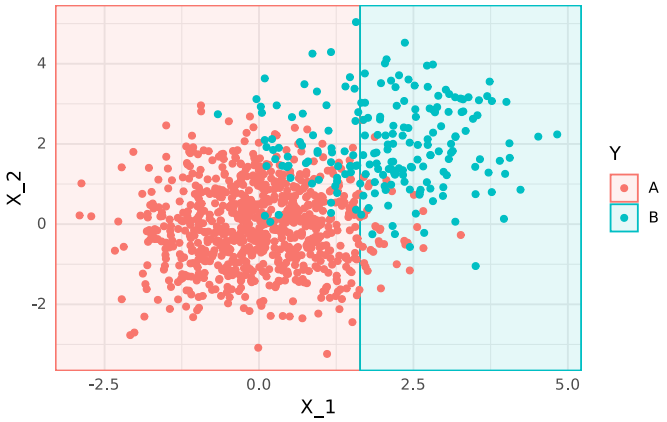
(a) Depth 1: a root node.



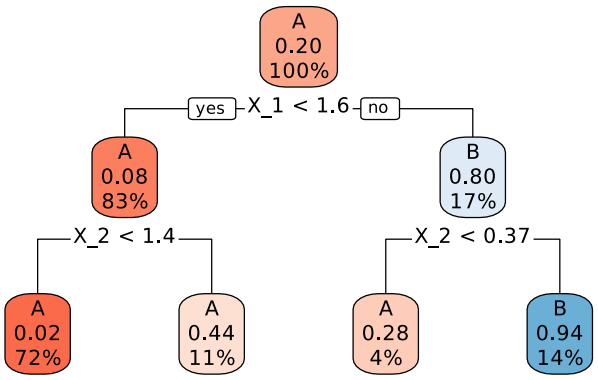
(b) Depth 1: the feature space.



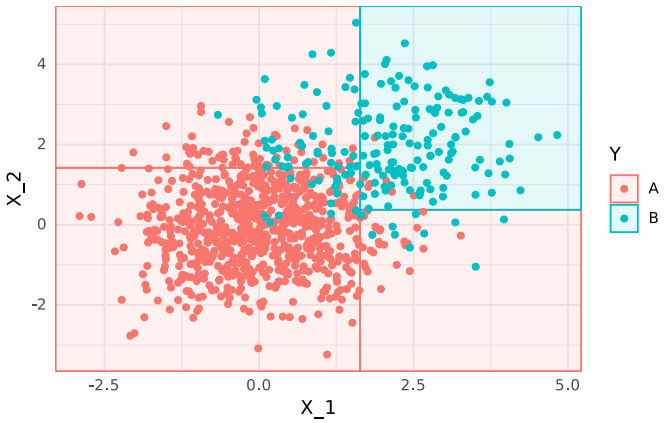
(c) Depth 2: the first split.



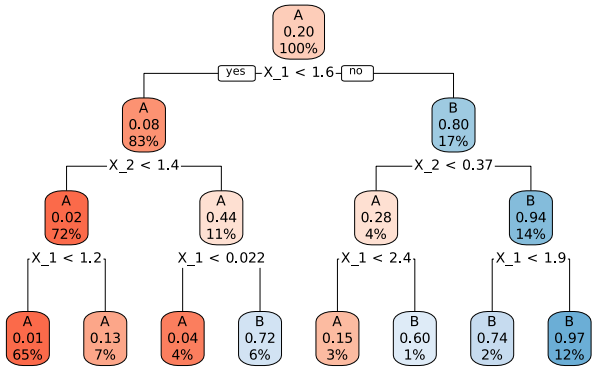
(d) Depth 2: partitioning the feature space.



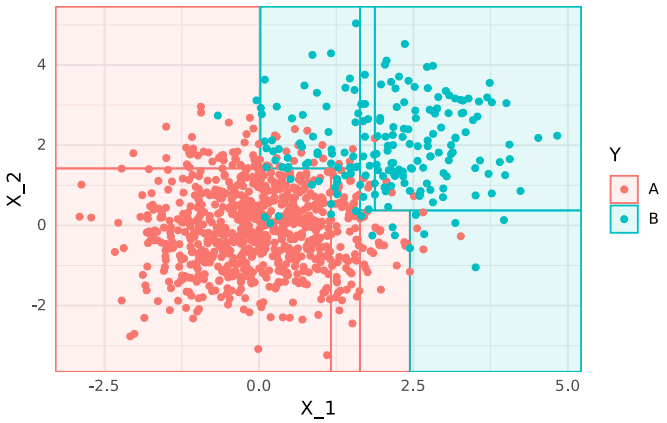
(e) Depth 3: a taller binary tree.



(f) Depth 3: partitioning the feature space again.



(g) Depth 4: an even taller binary tree.



(h) Depth 4: partitioning the feature space even further.

Figure 5: Building splits recursively.

2.5 Pruning the tree

2.5.1 Why and how pruning?

As said earlier it has been found that the best method of arriving at a suitable size for the tree is to grow an overly complex one then to **prune** it back. The pruning is based on the **misclassification rate** (number of observations misclassified divided by the total number of observations, see also **confusion table**). However the error rate will always drop (or at least not increase) with every split. This does not mean however that the error rate on the test data will improve.

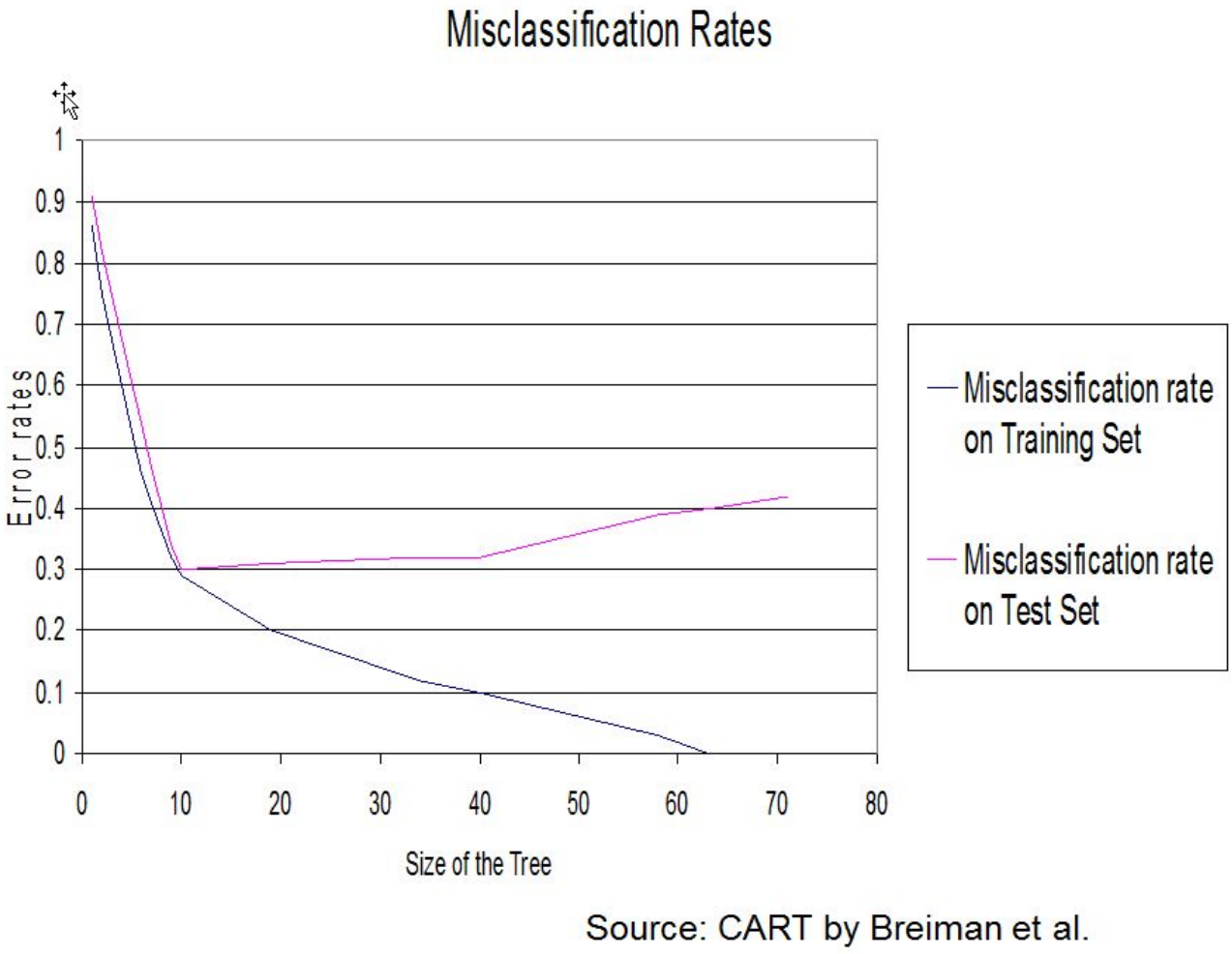


Figure 6: Why pruning?

To overcome the **overfitting** problem (good results on the training set but bad results on the test set).

A solution is to choose one of the subtree of the maximal tree obtained by **iterative pruning** and to choose the **optimal subtree** for a given quality criterion. This step-by-step methodology do not necessarily lead to a global optimal subtree but it is a computationally plausible solution.

2.5.2 A penalized criterion

The idea of pruning is to estimate an **error criterion penalized by the complexity of the model**.

More precisely, suppose that we have obtained the tree $\mathscr{T}$ with leafs $\mathscr{F}_1, \ldots, \mathscr{F}_K$ (so K is the number of leafs) having predicting values $\mathscr{Y}_1, \ldots, \mathscr{Y}_K$. Then, the **error of** $\mathscr{T}$ is

$$L\mathscr{T} = \sum_{k=1}^{K} L\mathscr{F}_k \qquad \text{where } L\mathscr{F}_k = \sum_{i:\ x_i \in \mathscr{F}_k} (y_i - \mathscr{Y}_k)^2$$

for the regression case and the misclassification rate for the classification case.

Hence, a penalized version of the error that takes into account the complexity of the tree can be defined through:

$$L_\gamma \mathscr{T} = L\mathscr{T} + \gamma K.$$

where K is the size of the tree (number of terminal nodes).

2.5.3 Iterative pruning

When $\gamma = 0$, $L_\gamma \mathscr{T} = L\mathscr{T}$ and hence the tree optimizing this criterion is $\mathscr{T}$ (which has been designed for).

When γ **is increasing**, one of the nodes' split, $\mathscr{S}_j$ appears such that:

$$L_\gamma \mathscr{T} > L_\gamma \mathscr{T}^{-\mathscr{S}_j}$$

where $\mathscr{T}^{-\mathscr{S}_j}$ is the tree where the split $\mathscr{S}_j$ has been removed. Let us call $\mathscr{T}_{K-1}$ this new tree.

This process is iterated to obtain a **sequence of trees**

$$\mathscr{T} \supset \mathscr{T}_{K-1} \supset \ldots \mathscr{T}_1$$

where $\mathscr{T}_1$ is the tree restricted to its root.

2.5.4 Choosing the optimal subtree

The optimal subtree is chosen by validation or cross-validation by the following algorithm: Algorithm for cross validation choice

1. Build the maximal tree, $\mathscr{T}$
2. Build the sequence of subtrees $(\mathscr{T}_k)_{k=1,\ldots,K}$
3. **By validation (or cross validation), evaluate the errors** $L\mathscr{T}_k$ for $k = 1, \ldots, K$
4. Build the graphic of $L\mathscr{T}_k$ in function of $k = 1, \ldots, K$
5. Finally, **choose** k **which minimizes** $L\mathscr{T}_k$

2.5.5 Pruning a tree with the `rpart` R function

- One version of the method carries out a **10 fold cross validation** where the data is divided into 10 subsets of equal size (at random) and then the tree is grown leaving out one of the subsets and the performance assessed on the subset left out from growing the tree. This is done for each of the 10 sets. The average performance is then assessed.
- This is all done by the command `rpart` in R and the results are accessible using `printcp` and `plotcp` where cp denotes the complexity parameter (γ for `printcp` but geometric mean of the interval bounds for `plotcp`).

- We can then use this information to decide how complex (determined by the size of cp) the tree needs to be.
- The possible rules are to **minimize the cross validation relative error (xerror)**, or to use the "1-SE rule" which uses the largest value of cp with the "xerror" within one standard deviation of the minimum. This rule is often preferred (see the dashed line in the "plotcp" function).
- More details can be found on pages 12, 13 and 14 of Therneau and Atkinson (2023).

2.5.6 Pruning a tree: small regression example detailed

```
rpart.plot(cars_fit)
```

```
Warning: labs do not fit even at cex 0.15, there may be some overplotting
```

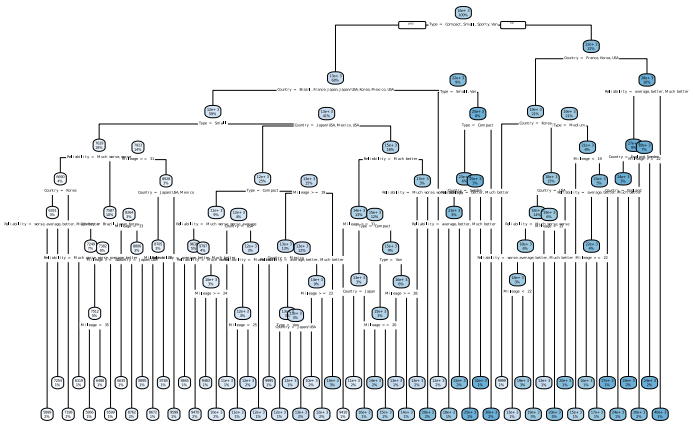


Figure 7: A regression tree that is too complex.

```
printcp(cars_fit)
```

```
Regression tree:
rpart(formula = Price ~ ., data = cu.summary, minbucket = 1,
    cp = 0)

Variables actually used in tree construction:
[1] Country     Mileage     Reliability Type

Root node error: 7407472615/117 = 63311732

n= 117

           CP nsplit rel error  xerror    xstd
1  2.5052e-01      0   1.00000 1.01587 0.16050
2  1.4836e-01      1   0.74948 0.86176 0.16636
3  8.7654e-02      2   0.60112 0.78607 0.15611
```

```
4  6.2818e-02      3   0.51347 0.75405 0.14418
5  3.4875e-02      4   0.45065 0.65157 0.12648
6  2.4396e-02      5   0.41577 0.64501 0.12503
7  1.1966e-02      8   0.34259 0.63270 0.12229
8  1.0640e-02     14   0.27079 0.65914 0.12669
9  9.9092e-03     15   0.26015 0.66715 0.12744
10 8.8587e-03     16   0.25024 0.65961 0.12804
11 7.3572e-03     20   0.21480 0.70312 0.13287
12 7.2574e-03     22   0.20009 0.70568 0.13278
13 3.8972e-03     28   0.15655 0.77327 0.14843
14 1.9968e-03     31   0.14334 0.79289 0.14972
15 1.9131e-03     33   0.13935 0.80349 0.15072
16 1.6070e-03     34   0.13744 0.80366 0.15069
17 1.1151e-03     35   0.13583 0.80803 0.15074
18 9.0617e-04     36   0.13471 0.81686 0.15078
19 4.7736e-04     42   0.12928 0.81161 0.15086
20 1.4084e-04     43   0.12880 0.81322 0.15109
21 1.0325e-04     45   0.12852 0.81495 0.15102
22 1.0187e-04     46   0.12841 0.81536 0.15100
23 8.0922e-05     47   0.12831 0.81536 0.15100
24 6.9751e-05     48   0.12823 0.81531 0.15100
25 6.0368e-05     49   0.12816 0.81482 0.15102
26 5.2584e-05     50   0.12810 0.81371 0.15084
27 5.1761e-05     52   0.12800 0.81371 0.15084
28 2.5252e-05     53   0.12794 0.81394 0.15083
29 1.2155e-05     54   0.12792 0.81436 0.15082
30 7.9655e-06     55   0.12791 0.81441 0.15082
31 1.5920e-06     56   0.12790 0.81441 0.15082
32 4.1013e-07     57   0.12790 0.81424 0.15081
33 0.0000e+00     58   0.12790 0.81427 0.15081
```

```
plotcp(cars_fit)
```

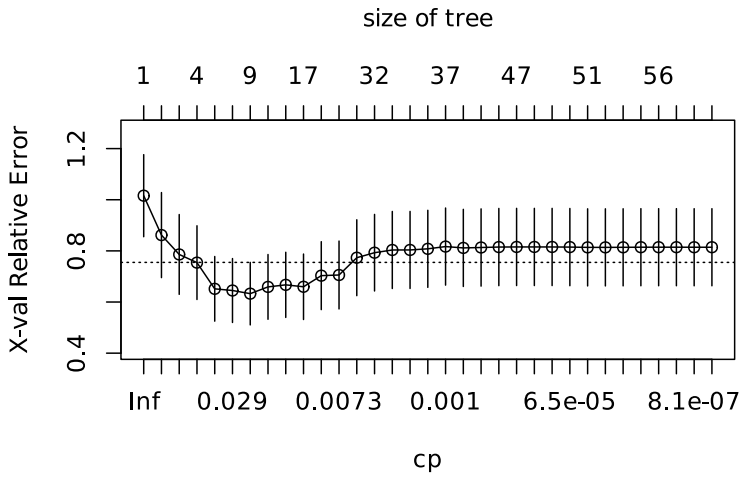


Figure 8: 1-SE method.

```
cars_fit <- prune(cars_fit, cp = 0.074)
rpart.plot(cars_fit)
```

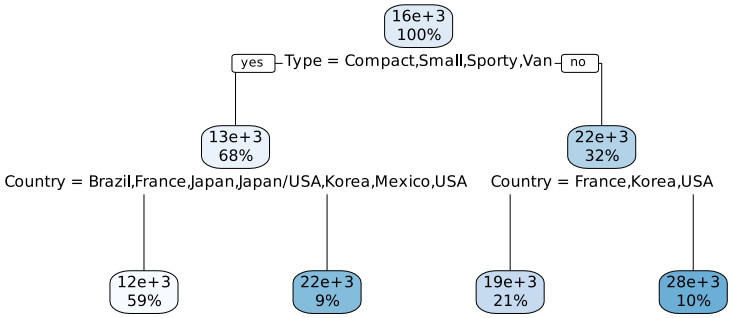


Figure 9: The resulting pruned tree.

3 CART in practice

```
# Data
candidates <- read.table("data_dt.txt", header = TRUE)
# Transformation of the variable to predict into a factor
candidates$Res <- as.factor(candidates$Res)
candidates
```

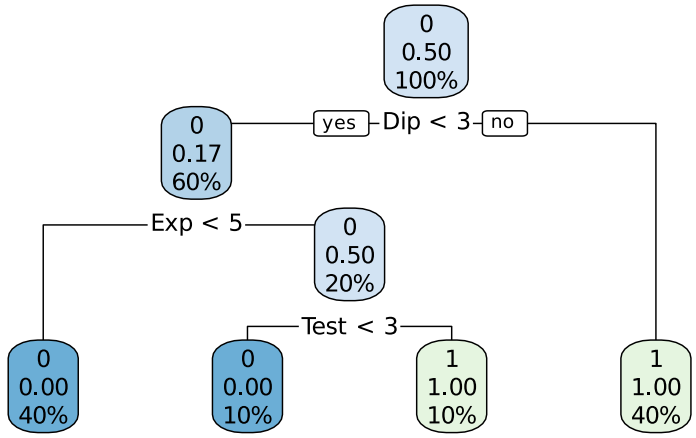
Table 5: Data set on candidates for a job.

Obs	Id	Dip	Test	Exp	Res
1	A	1	5	4	0
2	B	2	3	3	0
3	C	1	4	5	1
4	D	2	3	4	0
5	E	1	4	4	0
6	F	4	3	4	1
7	G	3	4	4	1
8	H	1	1	5	0
9	I	3	2	5	1
10	J	5	4	4	1

```
# Classification tree
library(rpart)
library(rpart.plot)

# Building the tree (prune is not useful here)
candidates_fit <- rpart(Res ~ Dip + Test + Exp,
  data = candidates, method = "class",
```

```
  parms = list(split = "gini"), control = rpart.control(minsplit = 2)
)
```

```
# Plot
rpart.plot(candidates_fit)
```

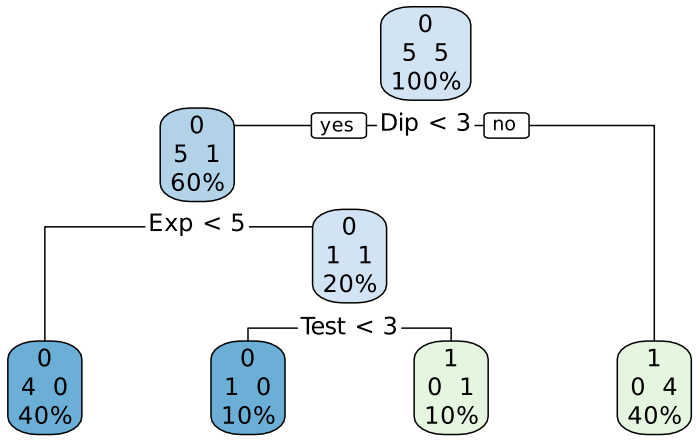



```
summary(candidates_fit)
```

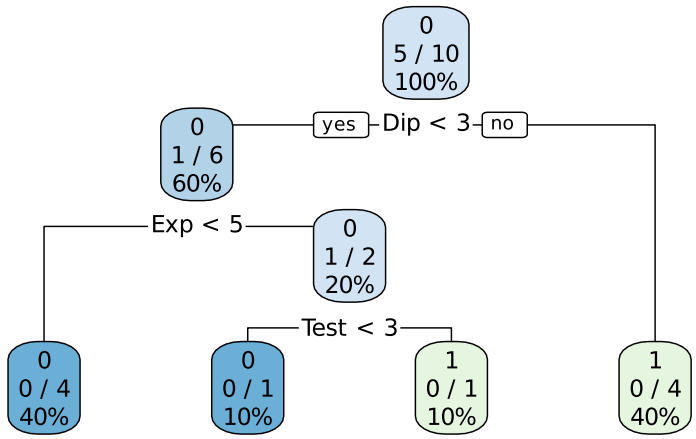
```
Call:
rpart(formula = Res ~ Dip + Test + Exp, data = candidates, method = "class",
    parms = list(split = "gini"), control = rpart.control(minsplit = 2))
  n= 10

    CP nsplit rel error xerror      xstd
1 0.80      0       1.0    2.0 0.0000000
2 0.10      1       0.2    0.2 0.1897367
3 0.01      3       0.0    0.6 0.2898275

Variable importance
 Dip Test  Exp
  67   20   13

Node number 1: 10 observations,    complexity param=0.8
  predicted class=0  expected loss=0.5  P(node) =1
    class counts:     5     5
   probabilities: 0.500 0.500
  left son=2 (6 obs) right son=3 (4 obs)
  Primary splits:
      Dip  < 2.5 to the left,  improve=3.3333330, (0 missing)
      Test < 1.5 to the left,  improve=0.5555556, (0 missing)
      Exp  < 3.5 to the left,  improve=0.5555556, (0 missing)

Node number 2: 6 observations,    complexity param=0.1
  predicted class=0  expected loss=0.1666667  P(node) =0.6
```

```
    class counts:     5     1
   probabilities: 0.833 0.167
  left son=4 (4 obs) right son=5 (2 obs)
  Primary splits:
      Exp  < 4.5 to the left,  improve=0.6666667, (0 missing)
      Test < 3.5 to the left,  improve=0.3333333, (0 missing)
      Dip  < 1.5 to the right, improve=0.1666667, (0 missing)

Node number 3: 4 observations
  predicted class=1  expected loss=0  P(node) =0.4
    class counts:     0     4
   probabilities: 0.000 1.000

Node number 4: 4 observations
  predicted class=0  expected loss=0  P(node) =0.4
    class counts:     4     0
   probabilities: 1.000 0.000

Node number 5: 2 observations,    complexity param=0.1
  predicted class=0  expected loss=0.5  P(node) =0.2
    class counts:     1     1
   probabilities: 0.500 0.500
  left son=10 (1 obs) right son=11 (1 obs)
  Primary splits:
      Test < 2.5 to the left,  improve=1, (0 missing)

Node number 10: 1 observations
  predicted class=0  expected loss=0  P(node) =0.1
    class counts:     1     0
   probabilities: 1.000 0.000

Node number 11: 1 observations
  predicted class=1  expected loss=0  P(node) =0.1
    class counts:     0     1
   probabilities: 0.000 1.000
```

```
rpart.plot(candidates_fit, extra = 101)
```



```
rpart.plot(candidates_fit, extra = 103)
```



```
# Rules
rpart.rules(candidates_fit)
```

	Res											
4	0.00	when	Dip	<	3	&	Exp	<	5			
10	0.00	when	Dip	<	3	&	Exp	>=	5	&	Test	<	3
11	1.00	when	Dip	<	3	&	Exp	>=	5	&	Test	>=	3
3	1.00	when	Dip	>=	3							

```
# Prediction
predict(candidates_fit, candidates, type = "class")
```

```
 1  2  3  4  5  6  7  8  9 10
 0  0  1  0  0  1  1  0  1  1
Levels: 0 1
```

```r
predict(candidates_fit, candidates, type = "prob")
```

```
   0 1
1  1 0
2  1 0
3  0 1
4  1 0
5  1 0
6  0 1
7  0 1
8  1 0
9  0 1
10 0 1
```

```r
predict(candidates_fit, candidates, type = "matrix")
```

```
   [,1] [,2] [,3] [,4] [,5] [,6]
1     1    4    0    1    0  0.4
2     1    4    0    1    0  0.4
3     2    0    1    0    1  0.1
4     1    4    0    1    0  0.4
5     1    4    0    1    0  0.4
6     2    0    4    0    1  0.4
7     2    0    4    0    1  0.4
8     1    1    0    1    0  0.1
9     2    0    4    0    1  0.4
10    2    0    4    0    1  0.4
```

```r
# confusion table
candidates_predict <- predict(candidates_fit, candidates, type = "vector")
table(candidates_predict, candidates$Res)
```

```
candidates_predict 0 1
                 1 5 0
                 2 0 5
```

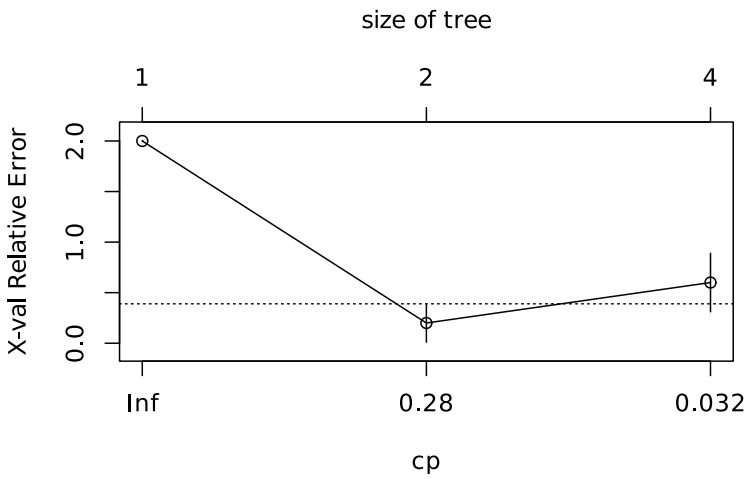
```r
# prediction of a new observations
predict(candidates_fit, newdata = data.frame(Dip = c(1, 2, 2), Exp = c(5, 5, 3), Test = c(3, 1, 5)
```

```
1 2 3
2 1 1
```

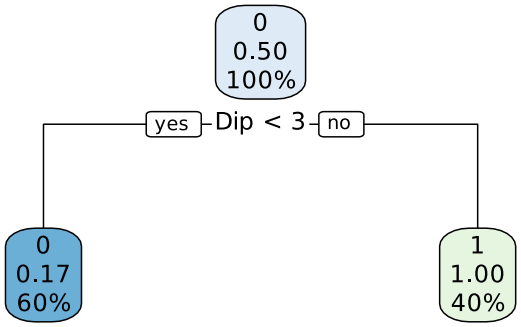
```r
# Pruning
# print and plot the complexity parameter (cp)
printcp(candidates_fit)
```

```
Classification tree:
rpart(formula = Res ~ Dip + Test + Exp, data = candidates, method = "class",
    parms = list(split = "gini"), control = rpart.control(minsplit = 2))

Variables actually used in tree construction:
[1] Dip  Exp  Test

Root node error: 5/10 = 0.5

n= 10

    CP nsplit rel error xerror    xstd
1 0.80      0       1.0    2.0 0.00000
2 0.10      1       0.2    0.2 0.18974
3 0.01      3       0.0    0.6 0.28983
```

```r
plotcp(candidates_fit)
```




```r
candidates_fit_prune <- prune(candidates_fit, cp = 0.2)
rpart.plot(candidates_fit_prune)
```



```
# Prediction on the pruned data set
predict(candidates_fit_prune, candidates, type = "matrix")
```

```
   [,1] [,2] [,3]      [,4]      [,5] [,6]
1     1    5    1 0.8333333 0.1666667  0.6
2     1    5    1 0.8333333 0.1666667  0.6
3     1    5    1 0.8333333 0.1666667  0.6
4     1    5    1 0.8333333 0.1666667  0.6
5     1    5    1 0.8333333 0.1666667  0.6
6     2    0    4 0.0000000 1.0000000  0.4
7     2    0    4 0.0000000 1.0000000  0.4
8     1    5    1 0.8333333 0.1666667  0.6
9     2    0    4 0.0000000 1.0000000  0.4
10    2    0    4 0.0000000 1.0000000  0.4
```

References

Breiman, Leo, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. 1984. *Classification And Regression Trees*. 1st ed. Routledge. https://doi.org/10.1201/9781315139470.

Therneau, Terry M, and Elizabeth J Atkinson. 2023. "An Introduction to Recursive Partitioning Using the RPART Routines." https://cran.r-project.org/web/packages/rpart/vignettes/longintro.pdf.