Decision Tree Classifier for Beginners in R

The Decision Tree algorithm is a member of the family of supervised machine learning algorithms. These algorithms typically rely on the concept of ground truth, which involves actual labels indicating, for instance, whether a person has diabetes or not. By comparing the model’s predictions with these ground truth labels, metrics can be generated to evaluate the model’s performance. Essentially, this underlines that Decision Tree is a supervised learning algorithm.

Decision Trees can be utilized for both classification and regression tasks. In a supervised learning context, the problem can be either classification or regression, and Decision Trees are capable of handling both. The primary objective of the Decision Tree algorithm is to develop a model that predicts the value of a target variable 𝑦 y. The algorithm uses a tree structure to solve the problem, where the leaf nodes represent class labels, and the internal nodes represent attributes.

For a more comprehensive understanding, please refer to the illustration below.

## Task One: Getting Started

In this task, you will get started by getting an overview of the project  The Decision Tree algorithm is a powerful tool within the realm of supervised machine learning, known for its ability to handle both classification and regression tasks. A Decision Tree is structured with several key components: the root node, decision nodes (also called internal nodes), and terminal nodes (or leaf nodes).

As depicted in the diagram, the root node represents the initial decision point, from which the dataset is split based on certain criteria. Decision nodes represent further splits within the tree, while terminal nodes indicate the final outcomes or class labels.

The algorithm selects splits based on the principle of creating the most homogeneous partitions, meaning it aims to group similar instances together. This process involves iteratively dividing the dataset into smaller, more homogeneous subsets. Each split is based on one of the input variables, and the goal is to maximize the homogeneity within each resulting partition.

Starting from the root node, the tree grows by evaluating the homogeneity of the outcome within each group and continuing to split as necessary. This process continues until the groups are sufficiently homogeneous or cannot be split further without reducing their similarity.

One of the significant advantages of Decision Trees is their interpretability. They simplify complex decision-making processes by breaking them down into a series of straightforward, smaller decisions, following a divide-and-conquer strategy. The tree structure effectively represents a set of if-else conditions that split the data based on the values of the variables.

Additionally, Decision Trees often perform better than linear regression in non-linear settings due to their ability to capture non-linear relationships between variables. However, a major drawback is the potential for overfitting, especially if the tree grows too large without pruning or cross-validation. Pruning and cross-validation techniques are essential to mitigate overfitting and ensure the model generalizes well to new data.

## Task Two: Import Required Libraries

In this task, you will learn how to import the required packages for this project

## 2.1: Import the required packages  
#install.packages("tree")  
#install.packages("caret")  
#install.packages("rpart")  
#install.packages("rpart.plot")  
#install.packages("mlbench")  
#install.packages("e1071")  
library(tree)

## Warning: package 'tree' was built under R version 4.3.3

library(caret)

## Warning: package 'caret' was built under R version 4.3.3

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 4.3.3

## Loading required package: lattice

library(rpart)   
library(rpart.plot)

## Warning: package 'rpart.plot' was built under R version 4.3.3

library(mlbench)

## Warning: package 'mlbench' was built under R version 4.3.3

library(e1071)

The ‘mlbench’ package is a collection of benchmark data sets and functions for machine learning in R. We will use a sonar data set in that package. To load the data we will use the data function and put the name of the data set in quotes. First thing to always do is to now what this data really contain. You can use the ?sonar to review the details and understand what is in Sonar data.

Description of Sonar Data “This is the data set used by Gorman and Sejnowski in their study of the classification of sonar signals using a neural network [1]. The task is to train a network to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock.”

## Task Three: Import and Explore Dataset

In this task, you will learn how to import and explore the data set

## 3.1: Load the dataset in R  
data("Sonar")

Let’s now proceed the our exploration of data. One thing we can do quickly is to use the View() function to open up a new tab consists of all variables of the data.

## 3.2: View the dataset  
View(Sonar)

As you can see, we have a table consist of 61 variables and 208 values. It says that each pattern is a set of 60 numbers in the range 0.0 to 1.0. Each number represents the energy within a particular frequency. And if you scroll to the last part, you can see the class label. The only, nominal column, namely Class with values M and R. Now let’s try to obtain the frequency of the target variable or the Class column for us to be sure that our data set labels are balanced. That is one thing you should always do when you talk about a machine learning task, because if the labels are not balanced, your model that you would train will learn more of the one value, example if the M has 60 counts and 20 for R, the model will learn more of the M than of R. Unless we try to do some of the different tools for balancing the Class.

## 3.3: Obtain the frequency of the target variable  
table(Sonar$Class)

##   
## M R   
## 111 97

As you can we, we have 111 for M and 97 for R. Which is quite close. Though it is not equal, it is still quite balance unlike if we have let’s say 111 and 50. So this is still kind of balanced. So we can actually proceed and build classification model. But before that, let’s first check for missing values. We can use this function sum then is.na which says is there any missing value in this Sonar data?

## 3.4: Check for missing values   
sum(is.na(Sonar))

## [1] 0

Ans this shows us that the result is zero, meaning we have no missing values. Alternatively, we can use the anyNA() function to check for missing values.

## OR  
anyNA(Sonar)

## [1] FALSE

And it shows, FALSE, meaning there is no any single missing values in our data set. Let’s proceed with our exploration, another thing we want to do is to check the summary of the data, there is this summary() function in R that helps with getting this summary of the data.

## 3.5: Summarise the data set  
summary(Sonar)

## V1 V2 V3 V4   
## Min. :0.00150 Min. :0.00060 Min. :0.00150 Min. :0.00580   
## 1st Qu.:0.01335 1st Qu.:0.01645 1st Qu.:0.01895 1st Qu.:0.02438   
## Median :0.02280 Median :0.03080 Median :0.03430 Median :0.04405   
## Mean :0.02916 Mean :0.03844 Mean :0.04383 Mean :0.05389   
## 3rd Qu.:0.03555 3rd Qu.:0.04795 3rd Qu.:0.05795 3rd Qu.:0.06450   
## Max. :0.13710 Max. :0.23390 Max. :0.30590 Max. :0.42640   
## V5 V6 V7 V8   
## Min. :0.00670 Min. :0.01020 Min. :0.0033 Min. :0.00550   
## 1st Qu.:0.03805 1st Qu.:0.06703 1st Qu.:0.0809 1st Qu.:0.08042   
## Median :0.06250 Median :0.09215 Median :0.1070 Median :0.11210   
## Mean :0.07520 Mean :0.10457 Mean :0.1217 Mean :0.13480   
## 3rd Qu.:0.10028 3rd Qu.:0.13412 3rd Qu.:0.1540 3rd Qu.:0.16960   
## Max. :0.40100 Max. :0.38230 Max. :0.3729 Max. :0.45900   
## V9 V10 V11 V12   
## Min. :0.00750 Min. :0.0113 Min. :0.0289 Min. :0.0236   
## 1st Qu.:0.09703 1st Qu.:0.1113 1st Qu.:0.1293 1st Qu.:0.1335   
## Median :0.15225 Median :0.1824 Median :0.2248 Median :0.2490   
## Mean :0.17800 Mean :0.2083 Mean :0.2360 Mean :0.2502   
## 3rd Qu.:0.23342 3rd Qu.:0.2687 3rd Qu.:0.3016 3rd Qu.:0.3312   
## Max. :0.68280 Max. :0.7106 Max. :0.7342 Max. :0.7060   
## V13 V14 V15 V16   
## Min. :0.0184 Min. :0.0273 Min. :0.0031 Min. :0.0162   
## 1st Qu.:0.1661 1st Qu.:0.1752 1st Qu.:0.1646 1st Qu.:0.1963   
## Median :0.2640 Median :0.2811 Median :0.2817 Median :0.3047   
## Mean :0.2733 Mean :0.2966 Mean :0.3202 Mean :0.3785   
## 3rd Qu.:0.3513 3rd Qu.:0.3862 3rd Qu.:0.4529 3rd Qu.:0.5357   
## Max. :0.7131 Max. :0.9970 Max. :1.0000 Max. :0.9988   
## V17 V18 V19 V20   
## Min. :0.0349 Min. :0.0375 Min. :0.0494 Min. :0.0656   
## 1st Qu.:0.2059 1st Qu.:0.2421 1st Qu.:0.2991 1st Qu.:0.3506   
## Median :0.3084 Median :0.3683 Median :0.4350 Median :0.5425   
## Mean :0.4160 Mean :0.4523 Mean :0.5048 Mean :0.5630   
## 3rd Qu.:0.6594 3rd Qu.:0.6791 3rd Qu.:0.7314 3rd Qu.:0.8093   
## Max. :1.0000 Max. :1.0000 Max. :1.0000 Max. :1.0000   
## V21 V22 V23 V24   
## Min. :0.0512 Min. :0.0219 Min. :0.0563 Min. :0.0239   
## 1st Qu.:0.3997 1st Qu.:0.4069 1st Qu.:0.4502 1st Qu.:0.5407   
## Median :0.6177 Median :0.6649 Median :0.6997 Median :0.6985   
## Mean :0.6091 Mean :0.6243 Mean :0.6470 Mean :0.6727   
## 3rd Qu.:0.8170 3rd Qu.:0.8320 3rd Qu.:0.8486 3rd Qu.:0.8722   
## Max. :1.0000 Max. :1.0000 Max. :1.0000 Max. :1.0000   
## V25 V26 V27 V28   
## Min. :0.0240 Min. :0.0921 Min. :0.0481 Min. :0.0284   
## 1st Qu.:0.5258 1st Qu.:0.5442 1st Qu.:0.5319 1st Qu.:0.5348   
## Median :0.7211 Median :0.7545 Median :0.7456 Median :0.7319   
## Mean :0.6754 Mean :0.6999 Mean :0.7022 Mean :0.6940   
## 3rd Qu.:0.8737 3rd Qu.:0.8938 3rd Qu.:0.9171 3rd Qu.:0.9003   
## Max. :1.0000 Max. :1.0000 Max. :1.0000 Max. :1.0000   
## V29 V30 V31 V32   
## Min. :0.0144 Min. :0.0613 Min. :0.0482 Min. :0.0404   
## 1st Qu.:0.4637 1st Qu.:0.4114 1st Qu.:0.3456 1st Qu.:0.2814   
## Median :0.6808 Median :0.6071 Median :0.4904 Median :0.4296   
## Mean :0.6421 Mean :0.5809 Mean :0.5045 Mean :0.4390   
## 3rd Qu.:0.8521 3rd Qu.:0.7352 3rd Qu.:0.6420 3rd Qu.:0.5803   
## Max. :1.0000 Max. :1.0000 Max. :0.9657 Max. :0.9306   
## V33 V34 V35 V36   
## Min. :0.0477 Min. :0.0212 Min. :0.0223 Min. :0.0080   
## 1st Qu.:0.2579 1st Qu.:0.2176 1st Qu.:0.1794 1st Qu.:0.1543   
## Median :0.3912 Median :0.3510 Median :0.3127 Median :0.3211   
## Mean :0.4172 Mean :0.4032 Mean :0.3926 Mean :0.3848   
## 3rd Qu.:0.5561 3rd Qu.:0.5961 3rd Qu.:0.5934 3rd Qu.:0.5565   
## Max. :1.0000 Max. :0.9647 Max. :1.0000 Max. :1.0000   
## V37 V38 V39 V40   
## Min. :0.0351 Min. :0.0383 Min. :0.0371 Min. :0.0117   
## 1st Qu.:0.1601 1st Qu.:0.1743 1st Qu.:0.1740 1st Qu.:0.1865   
## Median :0.3063 Median :0.3127 Median :0.2835 Median :0.2781   
## Mean :0.3638 Mean :0.3397 Mean :0.3258 Mean :0.3112   
## 3rd Qu.:0.5189 3rd Qu.:0.4405 3rd Qu.:0.4349 3rd Qu.:0.4244   
## Max. :0.9497 Max. :1.0000 Max. :0.9857 Max. :0.9297   
## V41 V42 V43 V44   
## Min. :0.0360 Min. :0.0056 Min. :0.0000 Min. :0.0000   
## 1st Qu.:0.1631 1st Qu.:0.1589 1st Qu.:0.1552 1st Qu.:0.1269   
## Median :0.2595 Median :0.2451 Median :0.2225 Median :0.1777   
## Mean :0.2893 Mean :0.2783 Mean :0.2465 Mean :0.2141   
## 3rd Qu.:0.3875 3rd Qu.:0.3842 3rd Qu.:0.3245 3rd Qu.:0.2717   
## Max. :0.8995 Max. :0.8246 Max. :0.7733 Max. :0.7762   
## V45 V46 V47 V48   
## Min. :0.00000 Min. :0.00000 Min. :0.00000 Min. :0.00000   
## 1st Qu.:0.09448 1st Qu.:0.06855 1st Qu.:0.06425 1st Qu.:0.04512   
## Median :0.14800 Median :0.12135 Median :0.10165 Median :0.07810   
## Mean :0.19723 Mean :0.16063 Mean :0.12245 Mean :0.09142   
## 3rd Qu.:0.23155 3rd Qu.:0.20037 3rd Qu.:0.15443 3rd Qu.:0.12010   
## Max. :0.70340 Max. :0.72920 Max. :0.55220 Max. :0.33390   
## V49 V50 V51 V52   
## Min. :0.00000 Min. :0.00000 Min. :0.000000 Min. :0.000800   
## 1st Qu.:0.02635 1st Qu.:0.01155 1st Qu.:0.008425 1st Qu.:0.007275   
## Median :0.04470 Median :0.01790 Median :0.013900 Median :0.011400   
## Mean :0.05193 Mean :0.02042 Mean :0.016069 Mean :0.013420   
## 3rd Qu.:0.06853 3rd Qu.:0.02527 3rd Qu.:0.020825 3rd Qu.:0.016725   
## Max. :0.19810 Max. :0.08250 Max. :0.100400 Max. :0.070900   
## V53 V54 V55 V56   
## Min. :0.000500 Min. :0.001000 Min. :0.00060 Min. :0.000400   
## 1st Qu.:0.005075 1st Qu.:0.005375 1st Qu.:0.00415 1st Qu.:0.004400   
## Median :0.009550 Median :0.009300 Median :0.00750 Median :0.006850   
## Mean :0.010709 Mean :0.010941 Mean :0.00929 Mean :0.008222   
## 3rd Qu.:0.014900 3rd Qu.:0.014500 3rd Qu.:0.01210 3rd Qu.:0.010575   
## Max. :0.039000 Max. :0.035200 Max. :0.04470 Max. :0.039400   
## V57 V58 V59 V60   
## Min. :0.00030 Min. :0.000300 Min. :0.000100 Min. :0.000600   
## 1st Qu.:0.00370 1st Qu.:0.003600 1st Qu.:0.003675 1st Qu.:0.003100   
## Median :0.00595 Median :0.005800 Median :0.006400 Median :0.005300   
## Mean :0.00782 Mean :0.007949 Mean :0.007941 Mean :0.006507   
## 3rd Qu.:0.01043 3rd Qu.:0.010350 3rd Qu.:0.010325 3rd Qu.:0.008525   
## Max. :0.03550 Max. :0.044000 Max. :0.036400 Max. :0.043900   
## Class   
## M:111   
## R: 97   
##   
##   
##   
##

This shows us all the five number summary of all the variables 1 to 60 with the mean values. With minimum, 1st quartile, median, 3rd quartile, and maximum. While for categorical variables, it gives us the frequency of the variable.

## Exercise 3.1: Check the number of rows and columns  
nrow(Sonar)

## [1] 208

ncol(Sonar)

## [1] 61

We have 208 number of rows and 61 columns. Now the next thing we want to do, is to do a visualization to have an idea of the data and i will randomly select 9 variables only because we have 60 variables, and that is too many. I will set seet to help this project be reproducible. So what ever answer i get now, you’ll get that too if you try to reproduce this.

## Set seed to 11  
set.seed(11)

Now what we want to do is to randomly select nine features that we will plot using box plot. To do that, let’s create a variable column and sample 9 from 60 variables using the sample() function with two parameters, the x which is the whole data and the size of 9. Then next is we will now create a new variable which contains the 9 column-subset of data.

## 3.6: Randomly choose nine features from the data set  
columns <- sample(x = 1:60, size = 9)  
columns

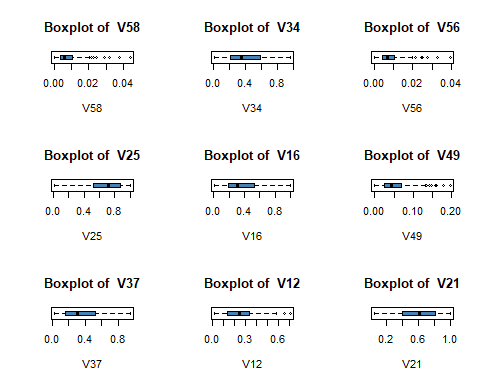
## [1] 58 34 56 25 16 49 37 12 21

pre\_var <- Sonar[, columns]  
pre\_var

## V58 V34 V56 V25 V16 V49 V37 V12 V21  
## 1 0.0084 0.7547 0.0167 0.6711 0.2273 0.0383 0.6692 0.1582 0.5783  
## 2 0.0049 0.2341 0.0191 0.3250 0.9444 0.0409 0.3835 0.6552 0.5212  
## 3 0.0164 0.2709 0.0244 0.5331 0.6931 0.0130 0.6116 0.7060 0.7974  
## 4 0.0044 0.3497 0.0073 0.3140 0.2131 0.0681 0.5408 0.1992 0.2741  
## 5 0.0048 0.0981 0.0015 0.3161 0.5326 0.0230 0.4604 0.3952 0.4148  
## 6 0.0027 0.2198 0.0089 0.2043 0.9988 0.0264 0.3807 0.4250 0.2074  
## 7 0.0143 0.1873 0.0138 0.5268 0.5440 0.0507 0.6153 0.0658 0.7723  
## 8 0.0047 0.4006 0.0097 0.7613 0.4376 0.0285 0.1915 0.3086 0.7832  
## 9 0.0093 0.3305 0.0049 0.3374 0.1136 0.0777 0.2463 0.1654 0.9802  
## 10 0.0035 0.0306 0.0068 0.9207 0.1133 0.0092 0.1594 0.1056 0.6068  
## 11 0.0003 0.6055 0.0093 0.6654 0.0907 0.0228 0.3947 0.0996 0.1847  
## 12 0.0092 0.2641 0.0118 0.5973 0.2032 0.0134 0.3994 0.0847 0.7852  
## 13 0.0058 0.2351 0.0019 0.6833 0.2723 0.0152 0.3382 0.1215 0.9674  
## 14 0.0053 0.3601 0.0152 0.6009 0.5326 0.0146 0.3858 0.2547 0.3542  
## 15 0.0196 0.3082 0.0054 0.6446 0.2390 0.0450 0.5501 0.1901 0.5801  
## 16 0.0049 0.8889 0.0076 0.9371 0.4335 0.0452 0.7615 0.3645 0.3932  
## 17 0.0015 0.8697 0.0048 0.2561 0.3308 0.0216 0.5789 0.2237 0.3158  
## 18 0.0037 0.3785 0.0024 0.5780 0.5761 0.0520 0.8852 0.1601 0.5399  
## 19 0.0132 0.6083 0.0039 0.8132 0.5354 0.0325 0.5715 0.1370 0.4430  
## 20 0.0094 0.2009 0.0006 0.4562 0.6985 0.0372 0.1212 0.5419 1.0000  
## 21 0.0082 0.7870 0.0054 0.4122 0.2430 0.0436 0.6385 0.2909 0.8319  
## 22 0.0162 0.6942 0.0026 0.2484 0.4039 0.0578 0.8912 0.4129 0.3246  
## 23 0.0160 0.6234 0.0139 0.2476 0.2458 0.1981 0.8712 0.1892 0.1441  
## 24 0.0037 0.6152 0.0026 0.4466 0.2584 0.0704 0.5324 0.0622 0.4449  
## 25 0.0160 0.9435 0.0075 0.4609 0.0809 0.0567 0.5516 0.0472 0.2111  
## 26 0.0024 0.9647 0.0011 0.8675 0.0162 0.0337 0.2718 0.0942 0.3825  
## 27 0.0019 0.2962 0.0054 0.6993 0.4527 0.0219 0.0476 0.1518 0.6736  
## 28 0.0057 0.3076 0.0124 1.0000 0.7201 0.0356 0.0794 0.3476 0.4865  
## 29 0.0020 0.3341 0.0047 0.7043 0.4441 0.0589 0.5534 0.3282 0.9857  
## 30 0.0092 0.1588 0.0140 0.6053 0.4183 0.0108 0.1605 0.3710 0.7473  
## 31 0.0019 0.1615 0.0094 0.7781 0.4997 0.0144 0.4681 0.2434 0.5053  
## 32 0.0031 0.1693 0.0060 0.8673 0.5121 0.0106 0.1388 0.1167 0.3733  
## 33 0.0095 0.5294 0.0089 0.7732 0.4602 0.0343 0.4041 0.0639 0.8631  
## 34 0.0105 0.4120 0.0073 0.6626 0.2036 0.0719 0.1336 0.2375 0.8017  
## 35 0.0182 0.1260 0.0101 0.3178 0.3463 0.0592 0.5382 0.2555 0.7672  
## 36 0.0180 0.3325 0.0096 0.4718 0.2307 0.0221 0.2607 0.1785 0.4497  
## 37 0.0087 0.9039 0.0129 0.6905 0.0717 0.0174 0.4027 0.0273 0.4925  
## 38 0.0132 0.6859 0.0063 0.8677 0.1678 0.0021 0.2750 0.0522 0.3485  
## 39 0.0058 0.5255 0.0018 0.6776 0.2077 0.0272 0.1279 0.0520 0.1354  
## 40 0.0043 0.8741 0.0097 0.7663 0.2032 0.0367 0.3446 0.0898 0.2310  
## 41 0.0052 0.5686 0.0068 0.7026 0.2652 0.0093 0.2959 0.0922 0.2493  
## 42 0.0050 0.6211 0.0054 0.6217 0.2799 0.0085 0.4889 0.1599 0.4300  
## 43 0.0049 0.7610 0.0020 0.8099 0.3283 0.0171 0.9123 0.1935 0.3363  
## 44 0.0116 0.8773 0.0092 0.8589 0.5367 0.0176 0.9419 0.3738 0.4330  
## 45 0.0199 0.5092 0.0198 0.5161 0.2871 0.0956 0.6062 0.0861 0.4978  
## 46 0.0102 0.8050 0.0198 0.8367 0.3731 0.0475 0.6253 0.3332 0.8835  
## 47 0.0122 0.6963 0.0051 0.8218 0.7295 0.0422 0.5468 0.2940 1.0000  
## 48 0.0054 0.6539 0.0073 0.7710 0.1563 0.0317 0.4897 0.2565 0.6079  
## 49 0.0022 0.5308 0.0140 1.0000 0.0572 0.0325 0.4260 0.3473 0.6484  
## 50 0.0073 0.5621 0.0069 1.0000 0.1136 0.0304 0.5025 0.3720 0.7468  
## 51 0.0096 0.6807 0.0108 0.5756 0.1240 0.0559 0.9497 0.0979 0.1268  
## 52 0.0013 0.4366 0.0039 0.6948 0.2934 0.0108 0.4746 0.0844 0.4084  
## 53 0.0064 0.0688 0.0050 0.8856 0.2568 0.0159 0.2216 0.1862 0.5665  
## 54 0.0055 0.1721 0.0102 0.8504 0.2321 0.0397 0.2735 0.1209 0.5402  
## 55 0.0140 0.2298 0.0048 0.8282 0.1985 0.0325 0.2395 0.1475 0.4893  
## 56 0.0029 0.2196 0.0037 0.8930 0.2097 0.0490 0.2630 0.1387 0.5328  
## 57 0.0011 0.1744 0.0046 0.9500 0.2023 0.0412 0.2459 0.2115 0.5525  
## 58 0.0018 0.6168 0.0122 0.7973 0.1134 0.0179 0.6253 0.0780 0.5438  
## 59 0.0035 0.5967 0.0057 0.9282 0.0844 0.0119 0.4786 0.0948 0.5958  
## 60 0.0041 0.4994 0.0048 0.9065 0.0742 0.0102 0.4577 0.0500 0.5959  
## 61 0.0022 0.5389 0.0045 0.9192 0.1145 0.0073 0.4826 0.1100 0.6673  
## 62 0.0028 0.5159 0.0024 0.8420 0.2215 0.0192 0.5144 0.1112 0.6797  
## 63 0.0050 0.6954 0.0029 0.8052 0.1304 0.0101 0.5917 0.1101 0.6275  
## 64 0.0034 0.5516 0.0019 0.4958 0.1297 0.0306 0.4561 0.0909 0.5911  
## 65 0.0032 0.8897 0.0105 0.0395 0.1227 0.0355 0.8459 0.1554 0.3031  
## 66 0.0179 0.8443 0.0050 0.0616 0.1688 0.0563 0.7766 0.1854 0.1976  
## 67 0.0081 0.8747 0.0097 0.0904 0.2232 0.0524 0.6960 0.2060 0.2224  
## 68 0.0067 0.8708 0.0070 0.0240 0.1669 0.0566 0.7712 0.2545 0.2069  
## 69 0.0042 0.6547 0.0022 0.5643 0.0890 0.0155 0.4679 0.1322 0.2297  
## 70 0.0024 0.2092 0.0021 0.8440 0.2138 0.0357 0.1058 0.1102 0.2005  
## 71 0.0036 0.2176 0.0036 0.8167 0.1666 0.0276 0.0589 0.0827 0.2146  
## 72 0.0049 0.2227 0.0004 0.8505 0.1732 0.0336 0.0795 0.1245 0.2257  
## 73 0.0019 0.1582 0.0044 0.9061 0.1918 0.0265 0.0429 0.1179 0.2429  
## 74 0.0059 0.2907 0.0022 0.9048 0.1396 0.0445 0.3186 0.0828 0.3797  
## 75 0.0056 0.2112 0.0077 0.8980 0.6570 0.0182 0.2149 0.0501 0.8297  
## 76 0.0054 0.3000 0.0065 0.9725 0.7973 0.0235 0.2541 0.0269 0.9413  
## 77 0.0026 0.3172 0.0078 0.9009 0.8278 0.0201 0.2692 0.0259 1.0000  
## 78 0.0015 0.2847 0.0169 0.9118 0.6225 0.0259 0.1192 0.0689 1.0000  
## 79 0.0038 0.1529 0.0072 0.9066 0.6847 0.0497 0.1073 0.1382 1.0000  
## 80 0.0023 0.3336 0.0058 0.8658 0.4754 0.0262 0.0379 0.1412 0.9493  
## 81 0.0029 0.2484 0.0056 0.4106 0.7213 0.0309 0.1338 0.3358 0.9116  
## 82 0.0019 0.3992 0.0060 0.7159 0.8468 0.0773 0.2370 0.3565 0.8142  
## 83 0.0061 0.4786 0.0087 0.6948 0.7555 0.0665 0.2043 0.3323 0.7561  
## 84 0.0030 0.3481 0.0094 0.7745 0.8073 0.0506 0.2130 0.4113 0.8106  
## 85 0.0062 0.3894 0.0065 0.5946 0.7868 0.0423 0.1312 0.3118 0.8443  
## 86 0.0094 0.0401 0.0063 0.8762 0.7257 0.1022 0.7076 0.0387 0.5605  
## 87 0.0057 0.0588 0.0057 0.8457 0.7807 0.1366 0.7319 0.1523 0.5148  
## 88 0.0102 0.1997 0.0051 0.9372 0.6699 0.1326 0.7775 0.0946 0.5297  
## 89 0.0161 0.0765 0.0074 0.7283 0.7312 0.0748 0.4827 0.0906 0.5103  
## 90 0.0068 0.1107 0.0030 0.8240 0.6804 0.0929 0.6423 0.0837 0.4019  
## 91 0.0034 0.4393 0.0022 0.8109 0.9751 0.0172 0.4801 0.1207 0.5001  
## 92 0.0039 0.2767 0.0116 0.6338 0.7152 0.0114 0.1356 0.3223 0.5280  
## 93 0.0008 0.4268 0.0015 0.6094 0.2978 0.0208 0.0535 0.2187 0.3598  
## 94 0.0085 0.6973 0.0151 0.8918 0.2231 0.0163 0.1598 0.3259 0.4460  
## 95 0.0016 0.5110 0.0085 0.7272 0.2410 0.0344 0.0947 0.2250 0.3613  
## 96 0.0081 0.6279 0.0137 0.1500 0.2050 0.0000 0.6359 0.0794 0.3470  
## 97 0.0048 0.4479 0.0041 0.0747 0.1231 0.0317 0.4480 0.0671 0.2109  
## 98 0.0140 0.5727 0.0119 0.5228 0.3323 0.0762 0.6903 0.2497 0.3264  
## 99 0.0127 0.1437 0.0013 0.5070 0.3956 0.0665 0.3725 0.2259 0.2243  
## 100 0.0224 0.3974 0.0090 0.3779 0.5553 0.0641 0.1023 0.2615 0.2395  
## 101 0.0057 0.1956 0.0187 0.2717 0.8705 0.0117 0.1073 0.5205 0.6908  
## 102 0.0377 0.3112 0.0193 0.2554 0.7391 0.0370 0.1834 0.5771 0.5760  
## 103 0.0101 0.4441 0.0122 0.2270 0.7221 0.0767 0.1681 0.4257 0.7326  
## 104 0.0100 0.5050 0.0074 0.3711 0.7487 0.0187 0.2319 0.4754 0.7685  
## 105 0.0121 0.3998 0.0152 0.3150 0.7339 0.0245 0.4609 0.4992 0.8240  
## 106 0.0054 0.2292 0.0277 0.6102 0.6464 0.0102 0.3201 0.3988 0.8646  
## 107 0.0044 0.3776 0.0129 0.6177 0.5020 0.0436 0.4716 0.1734 0.8073  
## 108 0.0084 0.5111 0.0131 0.8555 0.5392 0.0453 0.4234 0.2473 0.8416  
## 109 0.0114 0.4132 0.0081 0.7327 0.4665 0.0342 0.4532 0.2023 0.9144  
## 110 0.0077 0.2472 0.0068 0.9575 0.4191 0.0628 0.2909 0.0840 0.8759  
## 111 0.0107 0.2618 0.0099 0.8496 0.4998 0.0293 0.3248 0.1741 0.8906  
## 112 0.0146 0.2392 0.0151 0.8337 0.4144 0.0469 0.4441 0.2226 0.9171  
## 113 0.0321 0.5234 0.0084 0.6705 0.5411 0.0380 0.4516 0.3323 1.0000  
## 114 0.0033 0.3947 0.0073 0.7007 0.5036 0.0244 0.3619 0.2869 0.9275  
## 115 0.0145 0.1390 0.0041 0.6367 0.6755 0.0728 0.2818 0.2973 1.0000  
## 116 0.0077 0.4040 0.0165 0.7434 0.6610 0.0177 0.3704 0.3025 0.8804  
## 117 0.0055 0.0431 0.0055 0.9074 0.5107 0.0703 0.2120 0.3106 0.9891  
## 118 0.0045 0.0968 0.0095 1.0000 0.4499 0.0649 0.2250 0.3109 0.9349  
## 119 0.0114 0.1969 0.0050 0.9536 0.4359 0.0492 0.0711 0.2154 0.8379  
## 120 0.0021 0.2066 0.0124 0.9716 0.5398 0.0399 0.1302 0.3650 0.8396  
## 121 0.0024 0.1566 0.0102 0.9738 0.5325 0.0386 0.2824 0.2390 0.7084  
## 122 0.0036 0.0212 0.0157 0.9376 0.5335 0.0636 0.2373 0.2568 0.8941  
## 123 0.0244 0.0955 0.0133 0.9776 0.6087 0.0463 0.2554 0.2961 0.8995  
## 124 0.0094 0.0798 0.0092 1.0000 0.5512 0.0598 0.2626 0.2399 0.7811  
## 125 0.0068 0.1046 0.0111 0.7884 0.4823 0.0098 0.1017 0.3934 0.6918  
## 126 0.0093 0.3290 0.0125 0.9357 0.6573 0.0242 0.2413 0.3643 0.7999  
## 127 0.0099 0.1743 0.0042 0.5893 0.9718 0.0658 0.1803 0.5249 0.8980  
## 128 0.0288 0.1403 0.0101 0.7806 0.8292 0.1069 0.1271 0.1334 0.8920  
## 129 0.0113 0.1510 0.0087 0.6718 0.8272 0.0395 0.1709 0.3999 0.8890  
## 130 0.0079 0.3016 0.0067 0.4860 0.7939 0.0531 0.1757 0.5484 0.7960  
## 131 0.0115 0.1124 0.0055 0.2076 0.9016 0.0513 0.2562 0.5364 0.7156  
## 132 0.0141 0.6627 0.0110 0.1348 0.3602 0.1092 0.7848 0.0991 0.9777  
## 133 0.0201 0.6116 0.0212 0.2419 0.5312 0.1057 0.7356 0.1700 1.0000  
## 134 0.0073 0.8043 0.0249 0.2318 0.4579 0.0777 0.7056 0.2583 0.9939  
## 135 0.0122 0.8703 0.0032 0.2086 0.6632 0.0334 0.9386 0.2251 0.7992  
## 136 0.0168 0.8681 0.0326 0.0885 0.3828 0.1129 0.8327 0.0834 0.9779  
## 137 0.0205 0.6253 0.0167 0.2737 0.4659 0.0274 0.8420 0.2362 0.9312  
## 138 0.0162 0.6553 0.0049 0.6665 0.3737 0.1069 0.4118 0.5033 0.6527  
## 139 0.0194 0.1906 0.0161 0.8079 0.2580 0.1474 0.5496 0.3160 0.2604  
## 140 0.0094 0.3763 0.0185 0.6352 0.3234 0.0824 0.3319 0.2681 0.5606  
## 141 0.0225 0.2866 0.0166 0.5091 0.5079 0.1794 0.3087 0.2671 0.5400  
## 142 0.0131 0.4048 0.0138 0.6695 0.5543 0.0983 0.3652 0.1764 0.7464  
## 143 0.0103 0.6092 0.0174 0.6534 0.2285 0.0546 0.1666 0.2807 0.5562  
## 144 0.0109 0.6898 0.0129 0.5100 0.2764 0.0364 0.1580 0.2778 0.5187  
## 145 0.0130 0.5227 0.0046 0.4464 0.3892 0.0698 0.1309 0.0904 0.3112  
## 146 0.0147 0.4950 0.0117 0.5825 0.3308 0.1200 0.5647 0.3920 0.5342  
## 147 0.0214 0.5962 0.0087 0.4106 0.3129 0.1608 0.1188 0.2631 0.3184  
## 148 0.0440 0.5254 0.0394 0.7756 0.4331 0.1299 0.0970 0.1377 0.5248  
## 149 0.0101 0.0927 0.0048 0.1773 0.1930 0.0565 0.1781 0.3985 0.8958  
## 150 0.0090 0.3977 0.0121 0.6299 0.3990 0.0911 0.1429 0.1787 0.8159  
## 151 0.0047 0.1845 0.0029 0.5557 0.3827 0.0115 0.3837 0.1556 0.9264  
## 152 0.0091 0.2398 0.0114 0.3952 0.5595 0.0455 0.4331 0.1520 0.9921  
## 153 0.0066 0.3411 0.0065 0.1934 0.5343 0.0615 0.2699 0.1990 0.7352  
## 154 0.0101 0.3791 0.0061 0.5103 0.6707 0.0596 0.0351 0.0236 1.0000  
## 155 0.0062 0.1893 0.0029 0.5067 0.6592 0.0305 0.0888 0.0754 0.9615  
## 156 0.0039 0.3864 0.0031 0.3439 0.6122 0.0449 0.1112 0.0367 0.9162  
## 157 0.0045 0.6720 0.0068 0.7906 0.2401 0.0990 0.6241 0.3263 0.1557  
## 158 0.0020 0.6643 0.0092 0.6926 0.1610 0.1085 0.6938 0.3037 0.0521  
## 159 0.0060 0.7040 0.0097 0.5734 0.2080 0.0941 0.7673 0.3431 0.0576  
## 160 0.0052 0.5986 0.0023 0.5771 0.3019 0.0933 0.8084 0.2484 0.1624  
## 161 0.0063 0.6390 0.0079 0.7531 0.1193 0.0884 0.5983 0.2601 0.0625  
## 162 0.0042 0.6200 0.0089 0.6577 0.1237 0.1427 0.7337 0.2621 0.0512  
## 163 0.0022 0.4255 0.0049 0.9104 0.1967 0.0769 0.1451 0.2087 0.5116  
## 164 0.0036 0.5961 0.0064 1.0000 0.2808 0.0392 0.2369 0.2880 0.6753  
## 165 0.0108 0.9536 0.0020 0.7140 0.1139 0.1219 0.1249 0.3750 0.7894  
## 166 0.0070 0.8726 0.0058 0.9655 0.2666 0.0764 0.1546 0.2984 0.8099  
## 167 0.0050 0.4012 0.0132 0.3452 0.1196 0.1259 0.3356 0.1335 0.2836  
## 168 0.0035 0.4169 0.0058 0.5424 0.0906 0.0922 0.7963 0.1084 0.5810  
## 169 0.0010 0.5012 0.0029 0.8949 0.1951 0.0289 0.8195 0.2575 0.7420  
## 170 0.0009 0.4853 0.0058 0.7758 0.1533 0.0292 0.7536 0.2389 0.6663  
## 171 0.0024 0.6036 0.0008 0.8587 0.1100 0.0286 0.8800 0.2325 0.6508  
## 172 0.0155 0.6901 0.0126 0.8729 0.0422 0.0355 0.5060 0.4507 0.8550  
## 173 0.0050 0.6220 0.0025 0.4974 0.1308 0.0241 0.3809 0.3035 0.6960  
## 174 0.0042 0.1436 0.0053 0.7115 0.2808 0.0108 0.1644 0.3161 0.4539  
## 175 0.0122 0.3308 0.0132 0.9932 0.3889 0.0600 0.3208 0.5239 0.7271  
## 176 0.0006 0.3095 0.0054 0.9138 0.2956 0.0112 0.4072 0.5044 0.7044  
## 177 0.0085 0.2863 0.0095 1.0000 0.3492 0.0311 0.4649 0.5632 0.7702  
## 178 0.0105 0.2894 0.0035 0.9637 0.2007 0.0233 0.4773 0.3108 0.7209  
## 179 0.0045 0.4476 0.0072 0.7853 0.2056 0.0369 0.1772 0.2470 0.4282  
## 180 0.0068 0.4988 0.0062 0.7745 0.2252 0.0588 0.2244 0.3404 0.6474  
## 181 0.0056 0.3156 0.0037 0.7152 0.1784 0.0783 0.2062 0.5611 0.8202  
## 182 0.0021 0.2575 0.0075 0.5984 0.3172 0.1271 0.3039 0.4292 0.6802  
## 183 0.0030 0.3244 0.0040 0.6484 0.3496 0.1054 0.3590 0.4499 0.7650  
## 184 0.0036 0.2755 0.0012 0.7748 0.3447 0.0947 0.3167 0.5148 0.8147  
## 185 0.0027 0.2911 0.0045 0.7201 0.2856 0.1085 0.1869 0.5679 0.8959  
## 186 0.0051 0.2908 0.0042 0.7287 0.3075 0.0906 0.1602 0.5341 0.8751  
## 187 0.0087 0.3409 0.0031 0.6207 0.3002 0.1584 0.1433 0.3669 0.4890  
## 188 0.0086 0.3968 0.0092 0.4808 0.2534 0.0971 0.2010 0.2935 0.3504  
## 189 0.0069 0.3842 0.0044 0.6409 0.1452 0.0716 0.1570 0.3094 0.3355  
## 190 0.0046 0.3263 0.0079 0.7221 0.1745 0.0490 0.1670 0.2719 0.4268  
## 191 0.0056 0.3524 0.0038 0.7519 0.1892 0.0511 0.1592 0.2783 0.4102  
## 192 0.0035 0.2294 0.0007 0.7868 0.1485 0.0310 0.1829 0.2536 0.3786  
## 193 0.0024 0.1729 0.0084 0.7490 0.1405 0.0516 0.1770 0.2712 0.3830  
## 194 0.0036 0.3009 0.0042 0.7812 0.1786 0.0336 0.2200 0.2721 0.3906  
## 195 0.0044 0.3204 0.0042 0.8878 0.2127 0.0496 0.1234 0.3049 0.5071  
## 196 0.0017 0.1969 0.0027 0.9308 0.2347 0.0546 0.0756 0.3134 0.7531  
## 197 0.0063 0.1653 0.0062 0.9510 0.2055 0.0558 0.0740 0.3484 0.8346  
## 198 0.0017 0.2196 0.0016 0.9498 0.1863 0.0549 0.0948 0.2648 0.7766  
## 199 0.0013 0.2017 0.0026 0.8813 0.1735 0.0477 0.0580 0.3100 0.7616  
## 200 0.0044 0.2413 0.0026 0.9327 0.1791 0.0321 0.0693 0.2882 0.7934  
## 201 0.0009 0.2190 0.0071 0.9865 0.2122 0.0633 0.0521 0.3738 0.8010  
## 202 0.0022 0.1140 0.0046 0.8810 0.2794 0.0501 0.0702 0.3553 0.7969  
## 203 0.0051 0.2175 0.0098 0.8440 0.2043 0.0732 0.1708 0.3309 0.2802  
## 204 0.0115 0.1472 0.0101 0.8919 0.2200 0.0858 0.0969 0.2933 0.4130  
## 205 0.0032 0.2131 0.0063 0.9888 0.1534 0.0647 0.1159 0.3425 0.3272  
## 206 0.0138 0.1786 0.0089 0.9424 0.1503 0.0535 0.1066 0.2374 0.3164  
## 207 0.0079 0.0969 0.0035 0.9242 0.1444 0.0489 0.1572 0.2812 0.5685  
## 208 0.0036 0.1549 0.0061 0.9432 0.1708 0.0154 0.2655 0.2442 0.5700

As you can see, this results in values of the columns we extract from the data. Since we have 9 columns selected that we will visualize we have 9 spaced to put our visuals of the distributions. We will going to use the par() function, this produces a number of plots we want to show. Lastly, is the boxplot of the selected 9 columns. I’ve written a simple functions that shows the boxplots of each of the 9 columns and plot it in a 3x3.

par(mfrow = c(3,3)) # This produces a 3 by 3 plot of the box plots  
## 3.7: Show box plots of the predictor variables  
for (i in 1:ncol(pre\_var)){  
 boxplot(pre\_var[, i], xlab = names(pre\_var[i]),  
 main = paste("Boxplot of ",names(pre\_var[i])),  
 horizontal = TRUE, col = "steelblue")  
}

 We can now see that there are some outliers in some variables, some of the variables are quite normally distributed, while some are quite skewed to the right and some are skewed to the left. And in V12, V49, V56, and V58, we can see that there’s an outliers. This gives us an idea of what our variables mean.

## Task Four: Create Train and Test Sets

In this task, you will learn how to create the train and test data sets to build the decision tree model

In machine learning, generally, the common task is to build algorithm that can learn and make predictions from data. Such algorithms function by making data driven predictions or decisions through building a mathematical model from the input data. So the idea of the train and test set is that the train set is a subset of the data used to train the model, while the test set is a subset of the data used to test the trained model. Generally, the training data set are samples used to create the model, while the test or the validation set is used to qualify the performance of the model. Traditionally, the data set used to evaluate the final model’s performance is called the test set. Usually, the goal of machine learning is that we want to build a model that can generalize on real life data. Not just something that can learn only and give accuracy just while we are building the model. So we want to split the whole data set into two different data, the train data and the test data. Let’s now create a partition of our data. To do that lets create new variable index with createDataPartition() function.

## 4.1: Set Random Seed to 42  
set.seed(42)  
## 4.2: Partition the dataset in the ratio 70:30  
index <- createDataPartition(y=Sonar$Class, p=0.7, list=FALSE)

What it does is it creates a partition of data set which consists of 70 percent of the data and another 30 percent of the data. The arguments in the function includes the y which is the variable class, p which is the percentage of the data that will go to the training set, and list FALSE so it does not return a list. Let’s now create a new variable for training and test sets.

## 4.3: Create a training dataset for indexes  
train\_set <- Sonar[index,]

Since the index variable has 70 percent of the data, we will inserted it inside the train\_set variable, then for the other 30, it will be in the test\_set. Just use the negative of the index.

## 4.4: Create a testing dataset for indexes  
test\_set <- Sonar[-index,]

To check for the dimension, we will just use the dim() function and use the train and test\_set as argument.

## Exercise 4.1: Check the dimensions of the train and test sets  
dim(train\_set)

## [1] 146 61

dim(test\_set)

## [1] 62 61

All set, we have 146 rows and 61 columns in train\_set, and we have 62 rows and 61 columns for test\_set.

## Task Five: Train the Decision Tree Model

In this task, you will train a decision tree model and visualize the result of the trained model

To build a model let’s create a variable name model with tree() function from the tree library, and this takes two parameters, the formula and the data. For the formula, since we want all the variables to be the dependent variable, we will just use a short cut that everyone use which is tilde + dot, which says i want all the other variables to be my independent variables. Then the data that we will use is the train\_set. Remember, training set is used for model building. Let’s run this.

## 5.1: Train the model using the tree function  
model <- tree(formula = Class ~ .,   
 data = train\_set)

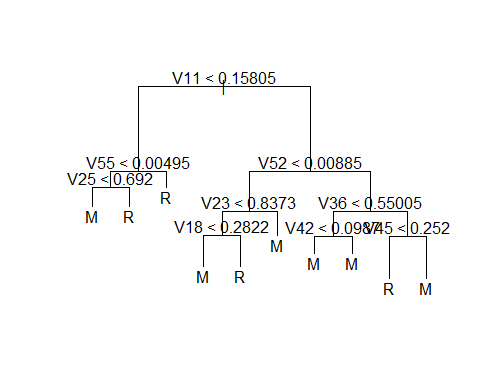
Since the model is done, let’s check the summary of the model using the summary() function. And as you can see, it gives us some little information. It tells us the variable that were used in the model construction. It also tells us the number of terminal nodes, the leaf node or the node where we have the prediction. It also tells us the residual mean deviation and the misclassification error.

## Exercise 5.1: Check the summary of the model  
summary(model)

##   
## Classification tree:  
## tree(formula = Class ~ ., data = train\_set)  
## Variables actually used in tree construction:  
## [1] "V11" "V55" "V25" "V52" "V23" "V18" "V36" "V42" "V45"  
## Number of terminal nodes: 10   
## Residual mean deviance: 0.126 = 17.14 / 136   
## Misclassification error rate: 0.0274 = 4 / 146

Then next thing to do is to visualize the data and see what it looks like. Let’s run this together so it will plot the model at the same time, it has its description.

## 5.2: Visualize the decision tree model  
plot(model)  
text(model)



You can see now what the model looks like. V11 is the root node, if V11 is less than 0.15805, then it goes V55, if V11 is greater than that value, it goes to V52. So, you can see how it splits. Let’s continue, if V55 is less than 0.00495, the it goes to V25 which continue to split further.

Let’s proceed and see this another model using rpart() function

## 5.3: Fit the model using the rpart function  
model1 <- rpart(Class ~ ., data = Sonar,  
 method = "class")

This kind of gives us a better representation of the same model that we have. Let’s build the model the same way using the Class variable using all variables as independent variable, the data we will use is the Sonar data, and the method is Class for classification.

## Exercise 5.2: Print the model and the model summary  
model1

## n= 208   
##   
## node), split, n, loss, yval, (yprob)  
## \* denotes terminal node  
##   
## 1) root 208 97 M (0.5336538 0.4663462)   
## 2) V11>=0.19795 121 30 M (0.7520661 0.2479339)   
## 4) V16< 0.66655 93 13 M (0.8602151 0.1397849)   
## 8) V34< 0.69215 80 6 M (0.9250000 0.0750000) \*  
## 9) V34>=0.69215 13 6 R (0.4615385 0.5384615) \*  
## 5) V16>=0.66655 28 11 R (0.3928571 0.6071429)   
## 10) V31< 0.3998 15 4 M (0.7333333 0.2666667) \*  
## 11) V31>=0.3998 13 0 R (0.0000000 1.0000000) \*  
## 3) V11< 0.19795 87 20 R (0.2298851 0.7701149)   
## 6) V4>=0.0515 21 8 M (0.6190476 0.3809524)   
## 12) V45>=0.16155 10 0 M (1.0000000 0.0000000) \*  
## 13) V45< 0.16155 11 3 R (0.2727273 0.7272727) \*  
## 7) V4< 0.0515 66 7 R (0.1060606 0.8939394) \*

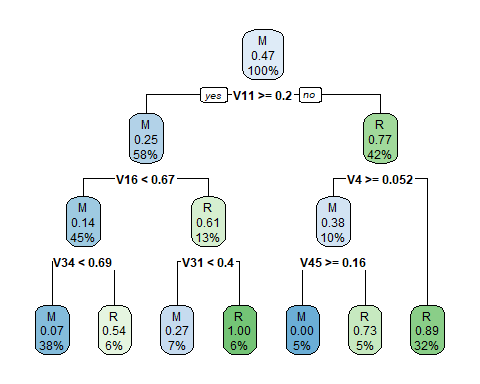
summary(model1)

## Call:  
## rpart(formula = Class ~ ., data = Sonar, method = "class")  
## n= 208   
##   
## CP nsplit rel error xerror xstd  
## 1 0.48453608 0 1.0000000 1.0000000 0.07417268  
## 2 0.06701031 1 0.5154639 0.5773196 0.06594964  
## 3 0.05154639 3 0.3814433 0.6494845 0.06832047  
## 4 0.01030928 5 0.2783505 0.5670103 0.06557279  
## 5 0.01000000 6 0.2680412 0.6288660 0.06768932  
##   
## Variable importance  
## V11 V10 V9 V12 V13 V8 V16 V4 V14 V31 V46 V17 V15 V30 V18 V45 V34 V19 V28 V29   
## 12 8 8 8 6 5 4 4 4 3 3 3 3 3 3 2 2 2 2 2   
## V47 V3 V5 V37 V33 V52 V21 V2 V6 V27 V35   
## 2 1 1 1 1 1 1 1 1 1 1   
##   
## Node number 1: 208 observations, complexity param=0.4845361  
## predicted class=M expected loss=0.4663462 P(node) =1  
## class counts: 111 97  
## probabilities: 0.534 0.466   
## left son=2 (121 obs) right son=3 (87 obs)  
## Primary splits:  
## V11 < 0.19795 to the right, improve=27.60028, (0 missing)  
## V12 < 0.22505 to the right, improve=24.53864, (0 missing)  
## V9 < 0.1164 to the right, improve=20.65490, (0 missing)  
## V10 < 0.16315 to the right, improve=19.89026, (0 missing)  
## V13 < 0.16265 to the right, improve=16.46505, (0 missing)  
## Surrogate splits:  
## V10 < 0.15265 to the right, agree=0.851, adj=0.644, (0 split)  
## V12 < 0.22315 to the right, agree=0.841, adj=0.621, (0 split)  
## V9 < 0.13685 to the right, agree=0.817, adj=0.563, (0 split)  
## V13 < 0.22745 to the right, agree=0.784, adj=0.483, (0 split)  
## V8 < 0.0988 to the right, agree=0.736, adj=0.368, (0 split)  
##   
## Node number 2: 121 observations, complexity param=0.06701031  
## predicted class=M expected loss=0.2479339 P(node) =0.5817308  
## class counts: 91 30  
## probabilities: 0.752 0.248   
## left son=4 (93 obs) right son=5 (28 obs)  
## Primary splits:  
## V16 < 0.66655 to the left, improve=9.401233, (0 missing)  
## V27 < 0.8167 to the right, improve=9.389901, (0 missing)  
## V17 < 0.6686 to the left, improve=8.586237, (0 missing)  
## V18 < 0.53215 to the left, improve=6.527408, (0 missing)  
## V15 < 0.5427 to the left, improve=6.034105, (0 missing)  
## Surrogate splits:  
## V17 < 0.7331 to the left, agree=0.942, adj=0.750, (0 split)  
## V15 < 0.5427 to the left, agree=0.934, adj=0.714, (0 split)  
## V18 < 0.8149 to the left, agree=0.917, adj=0.643, (0 split)  
## V14 < 0.47965 to the left, agree=0.909, adj=0.607, (0 split)  
## V19 < 0.8483 to the left, agree=0.884, adj=0.500, (0 split)  
##   
## Node number 3: 87 observations, complexity param=0.05154639  
## predicted class=R expected loss=0.2298851 P(node) =0.4182692  
## class counts: 20 67  
## probabilities: 0.230 0.770   
## left son=6 (21 obs) right son=7 (66 obs)  
## Primary splits:  
## V4 < 0.0515 to the right, improve=8.384684, (0 missing)  
## V45 < 0.16055 to the right, improve=8.304598, (0 missing)  
## V1 < 0.0392 to the right, improve=7.333079, (0 missing)  
## V16 < 0.3106 to the right, improve=6.912342, (0 missing)  
## V5 < 0.0696 to the right, improve=6.443487, (0 missing)  
## Surrogate splits:  
## V3 < 0.0555 to the right, agree=0.851, adj=0.381, (0 split)  
## V5 < 0.07255 to the right, agree=0.839, adj=0.333, (0 split)  
## V52 < 0.01885 to the right, agree=0.828, adj=0.286, (0 split)  
## V2 < 0.04755 to the right, agree=0.816, adj=0.238, (0 split)  
## V6 < 0.16495 to the right, agree=0.816, adj=0.238, (0 split)  
##   
## Node number 4: 93 observations, complexity param=0.01030928  
## predicted class=M expected loss=0.1397849 P(node) =0.4471154  
## class counts: 80 13  
## probabilities: 0.860 0.140   
## left son=8 (80 obs) right son=9 (13 obs)  
## Primary splits:  
## V34 < 0.69215 to the left, improve=4.804053, (0 missing)  
## V48 < 0.076 to the right, improve=4.301075, (0 missing)  
## V47 < 0.0611 to the right, improve=4.121474, (0 missing)  
## V49 < 0.02315 to the right, improve=3.575468, (0 missing)  
## V52 < 0.00695 to the right, improve=3.129053, (0 missing)  
## Surrogate splits:  
## V33 < 0.73755 to the left, agree=0.935, adj=0.538, (0 split)  
## V27 < 0.3482 to the right, agree=0.914, adj=0.385, (0 split)  
## V35 < 0.7375 to the left, agree=0.903, adj=0.308, (0 split)  
## V7 < 0.2456 to the left, agree=0.892, adj=0.231, (0 split)  
## V24 < 0.1901 to the right, agree=0.892, adj=0.231, (0 split)  
##   
## Node number 5: 28 observations, complexity param=0.06701031  
## predicted class=R expected loss=0.3928571 P(node) =0.1346154  
## class counts: 11 17  
## probabilities: 0.393 0.607   
## left son=10 (15 obs) right son=11 (13 obs)  
## Primary splits:  
## V31 < 0.3998 to the left, improve=7.490476, (0 missing)  
## V27 < 0.50485 to the left, improve=6.526734, (0 missing)  
## V30 < 0.481 to the left, improve=5.592437, (0 missing)  
## V51 < 0.0162 to the right, improve=5.357143, (0 missing)  
## V53 < 0.01095 to the right, improve=5.357143, (0 missing)  
## Surrogate splits:  
## V30 < 0.44475 to the left, agree=0.929, adj=0.846, (0 split)  
## V28 < 0.4678 to the left, agree=0.821, adj=0.615, (0 split)  
## V29 < 0.4884 to the left, agree=0.821, adj=0.615, (0 split)  
## V46 < 0.15915 to the left, agree=0.821, adj=0.615, (0 split)  
## V47 < 0.1269 to the left, agree=0.821, adj=0.615, (0 split)  
##   
## Node number 6: 21 observations, complexity param=0.05154639  
## predicted class=M expected loss=0.3809524 P(node) =0.1009615  
## class counts: 13 8  
## probabilities: 0.619 0.381   
## left son=12 (10 obs) right son=13 (11 obs)  
## Primary splits:  
## V45 < 0.16155 to the right, improve=5.541126, (0 missing)  
## V59 < 0.00695 to the right, improve=4.761905, (0 missing)  
## V46 < 0.16875 to the right, improve=3.750916, (0 missing)  
## V47 < 0.09795 to the right, improve=3.520147, (0 missing)  
## V41 < 0.33645 to the right, improve=2.571429, (0 missing)  
## Surrogate splits:  
## V37 < 0.4561 to the left, agree=0.762, adj=0.5, (0 split)  
## V46 < 0.17375 to the right, agree=0.762, adj=0.5, (0 split)  
## V9 < 0.1236 to the right, agree=0.714, adj=0.4, (0 split)  
## V14 < 0.2945 to the right, agree=0.714, adj=0.4, (0 split)  
## V21 < 0.8116 to the right, agree=0.714, adj=0.4, (0 split)  
##   
## Node number 7: 66 observations  
## predicted class=R expected loss=0.1060606 P(node) =0.3173077  
## class counts: 7 59  
## probabilities: 0.106 0.894   
##   
## Node number 8: 80 observations  
## predicted class=M expected loss=0.075 P(node) =0.3846154  
## class counts: 74 6  
## probabilities: 0.925 0.075   
##   
## Node number 9: 13 observations  
## predicted class=R expected loss=0.4615385 P(node) =0.0625  
## class counts: 6 7  
## probabilities: 0.462 0.538   
##   
## Node number 10: 15 observations  
## predicted class=M expected loss=0.2666667 P(node) =0.07211538  
## class counts: 11 4  
## probabilities: 0.733 0.267   
##   
## Node number 11: 13 observations  
## predicted class=R expected loss=0 P(node) =0.0625  
## class counts: 0 13  
## probabilities: 0.000 1.000   
##   
## Node number 12: 10 observations  
## predicted class=M expected loss=0 P(node) =0.04807692  
## class counts: 10 0  
## probabilities: 1.000 0.000   
##   
## Node number 13: 11 observations  
## predicted class=R expected loss=0.2727273 P(node) =0.05288462  
## class counts: 3 8  
## probabilities: 0.273 0.727

This decision tree model classifies a dataset of 208 observations into classes M and R, with the root node indicating a near-even split (53.37% M and 46.63% R). The first split is based on variable V11 at a threshold of 0.19795. For observations where V11 is greater than or equal to 0.19795 (121 observations), further splits are made on variables V16 and V34, leading to terminal nodes with varying class probabilities. For example, a terminal node where V34 < 0.69215 classifies 80 instances as M with a high probability (92.50%), while another node where V34 >= 0.69215 classifies 13 instances as R (53.85%). Conversely, for observations where V11 is less than 0.19795 (87 observations), additional splits occur on variables V4 and V45, resulting in terminal nodes with distinct class probabilities. For instance, a terminal node where V4 < 0.0515 classifies 66 instances as R with a high probability (89.39%). Terminal nodes, indicated with an asterisk, represent the final classification decisions.

Let’s now visualize the model using the rpart.plot() function. And this gives us a better representation. So let’s run it.

## 5.5: Visualize the new decision tree model  
rpart.plot(model1)



Just like what i explained earlier, V11 greater than 0.2, if yes, go to V16, that means about 58% i’m sure its M. If no, it goes to R. Then it keeps splitting that if V4 is greater than 0.052, if yes, then its 10% sure its M, so on.

## Task Six: Evaluating Model Performance

In this task, you will learn how to evaluate the performance of the decision tree model

First, we want to find the prediction for the first row of the test set. So, I’m going to assess the first row of the test data. That is the data that we have not used in the training. Let’s try to make prediction on the first data to try and see how good the model is and see which class this first data falls. We will use the predict() function, and in this predict() function, we’ll call the model we will be using. In this case we will use the second one. For the second argument we will use the data of the first row of the test set.

## 6.1: Make prediction for the first row of the test set  
test\_set[1, ]

## V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11  
## 3 0.0262 0.0582 0.1099 0.1083 0.0974 0.228 0.2431 0.3771 0.5598 0.6194 0.6333  
## V12 V13 V14 V15 V16 V17 V18 V19 V20 V21 V22  
## 3 0.706 0.5544 0.532 0.6479 0.6931 0.6759 0.7551 0.8929 0.8619 0.7974 0.6737  
## V23 V24 V25 V26 V27 V28 V29 V30 V31 V32 V33  
## 3 0.4293 0.3648 0.5331 0.2413 0.507 0.8533 0.6036 0.8514 0.8512 0.5045 0.1862  
## V34 V35 V36 V37 V38 V39 V40 V41 V42 V43 V44  
## 3 0.2709 0.4232 0.3043 0.6116 0.6756 0.5375 0.4719 0.4647 0.2587 0.2129 0.2222  
## V45 V46 V47 V48 V49 V50 V51 V52 V53 V54 V55  
## 3 0.2111 0.0176 0.1348 0.0744 0.013 0.0106 0.0033 0.0232 0.0166 0.0095 0.018  
## V56 V57 V58 V59 V60 Class  
## 3 0.0244 0.0316 0.0164 0.0095 0.0078 R

predict(model1, newdata = test\_set[1, ])

## M R  
## 3 0 1

So this predicts a 1 for R and 0 for M, that means that the first row of the test data is predicted as R. If you would ask me how R do that, is the predict() function has been built in the back-end to run through the decision tree model and checks that the first row is R. The next thing we want to do is to try to make prediction on all of the data.

## 6.2: Making Predictions on the whole test set  
prediction <- predict(model1, newdata = test\_set, type = "class")

We can see our prediction by building a table of predictions. Which is also called the confusion matrix.

## 6.3: Create a table of predictions  
table(x = prediction, y =test\_set$Class)

## y  
## x M R  
## M 29 3  
## R 4 26

Let me first explain what is in confusion matrix before we interpret the result. The upper part value is the actual value, while the right part is the predicted value. The result is saying that there were 29M’s and the model predicted those 29 correctly. That part is called the True Positives. The second column R, first row, is called the False Positives and that means the value actually was R but the model predicted it to be M. So it made three wrong classifications here. Then, in first column second row, our model value is actually M, which is positive but our model predicted it to be R. These are called the False Negatives, it predicted a wrong value, a negative value when actually it was a positive value. The last value which is 26, is saying that our value was R, and the model predicted it correctly to be R. So the values 26 and 29 are the actual positive. The first one is the True positive while the last one is the True negatives. Let’s now move on to evaluating the results using confusionmatrix() function to make evaluation.

## 6.4: Evaluate the prediction results  
confusionMatrix(data = prediction,  
 reference = test\_set$Class)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction M R  
## M 29 3  
## R 4 26  
##   
## Accuracy : 0.8871   
## 95% CI : (0.7811, 0.9534)  
## No Information Rate : 0.5323   
## P-Value [Acc > NIR] : 2.418e-09   
##   
## Kappa : 0.7737   
##   
## Mcnemar's Test P-Value : 1   
##   
## Sensitivity : 0.8788   
## Specificity : 0.8966   
## Pos Pred Value : 0.9062   
## Neg Pred Value : 0.8667   
## Prevalence : 0.5323   
## Detection Rate : 0.4677   
## Detection Prevalence : 0.5161   
## Balanced Accuracy : 0.8877   
##   
## 'Positive' Class : M   
##

The upper part is what i already explained a while ago, so lets just proceed to the next part. However, those upper values were used to calculate this accuracy. It shows 0.8871, so about 88%, our model can make good predictions. Of course that’s a good one, but it can also be improved. Next thing to see is the Kappa value of Kappa statistics. It is basically used to control only those instances that may have been correctly classified by chance. So what that means is that the Kappa statistics is frequently used to test reliability of our data. Like are they the correct representations of the variables that we have measured? So, the value 0.6-0.79 like in our case is 0.7737, shows a moderate level of agreement and shows that about 35% to 63% of the data is reliable. And i think that’s enough information. We also have the sensitivity value, which is basically the ability of the test to correctly identify the positive class. In our case it is M. And sensitivity is also called the True Positive Rate. That’s the probability that an actual positive tested to be positive, which is 87.88. In this case, sensitivity is of more interest to us, because we want to try as much as possible to reduce the False Negatives. That is the 4. Let me do an analogy for us to really understand why we are trying to reduce the False Negatives. For example if somebody has diabetes and the prediction model is predicting that the person does not have diabetes. We want to reduce that to the minimum and sensitivity is of interest here. The other part is what we call the specificity, and it is the ability to correctly identify the negative class. So sensitivity is just like the opposite of the specificity. Specificity is also called the True Negative Rate which is the probability that an actual negative will test negative. Then, another one we have is the Positive Prediction Value. This third value which is also called the Precision indicates the rate at which the positive predictions are correct. While sensitivity identifies the rate at which our observation from the positive class are correctly predicted, the Precision indicates the rate at which the positive predictions are correct. Another thins is to calculate the F1 score. The F1 score is usually used the most because its like a bridge between Specificity and Sensitivity and the Precision.

## 6.5: Calculate the F1-Score  
## F1 = 2 \* (Recall \* Precision) / (Recall + Precision)  
F1 <- 2 \* (0.8788 \* 0.9062)/(0.8788 + 0.9062)  
F1

## [1] 0.8922897

Our F1 score is 0.8922897 which is quite okay and very interesting. That’s a good value for F1 score. Last thing is to check the prediction error. The misclassification error.

## 6.6: Calculate the prediction error rate  
error\_rate <- round(mean(prediction != test\_set$Class), 2)  
error\_rate

## [1] 0.11

What this is saying is that, look through the prediction, any where the prediction that we had that does not equate to the class from the test data then find the mean. The error is the actual minus the predicted. We want it to find out and round it to two decimal places. This results to 0.11 which is quite low.

Lasly, lets save the model so we could use it somewhere someday. Maybe we want to build a web apps, or you can we can use it in R shiny someday.

## 6.7: Set Working Directory  
getwd()

## [1] "D:/Desktop/DATA ANALYSIS/PROJECTS/R/DECISION TREE CLASSIFIER IN R"

## 6.8: Save the tree model  
save(model1, file = "DecisionTree.RData")