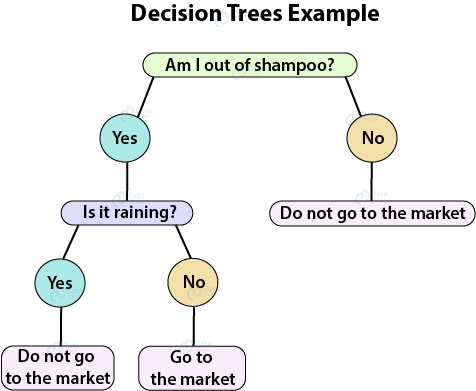
UNIT-IV

Decision tree is a graph to represent choices and their results in form of a tree. The nodes in the graph represent an event or choice and the edges of the graph represent the decision rules or conditions. It is mostly used in Machine Learning and Data Mining applications using R.

Decision Trees can be visualised as follows:

[](https://data-flair.training/blogs/wp-content/uploads/sites/2/2017/07/Decision-Trees-Example.png)

**Applications of Decision Trees**

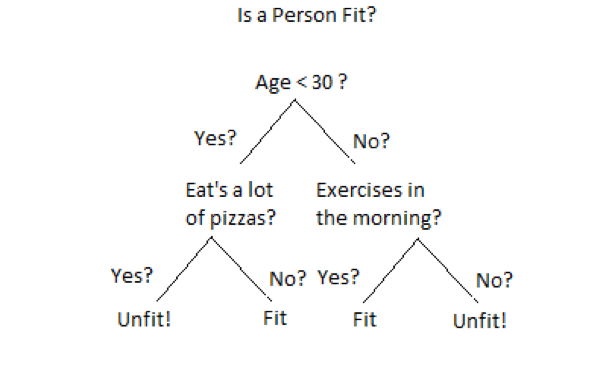
Decision Trees are used in the following areas of applications:

* **Marketing and Sales –**Decision Trees play an important role in a decision-oriented sector like marketing. In order to understand the consequences of marketing activities, organisations make use of Decision Trees to initiate careful measures. This helps in making efficient decisions that help the company to reap profits and minimize losses.
* **Reducing Churn Rate –** Banks make use of [***machine learning algorithms***](https://data-flair.training/blogs/machine-learning-algorithms/) like Decision Trees to retain their customers. It is always cheaper to keep customers than to gain new ones. Banks are able to analyze which customers are more vulnerable to leaving their business. Based on the output, they are able to *make decisions by providing better services, discounts as well as several other features.* This ultimately helps them to reduce the churn rate.
* **Anomaly & Fraud Detection –** Industries like finance and banking suffer from various cases of fraud. In order to filter out anomalous or fraud loan applications, information and insurance fraud, these companies deploy decision trees to provide them with the necessary information to identify fraudulent customers.
* **Medical Diagnosis –** Classification trees identifies patients who are at risk of suffering from serious diseases such as cancer and diabetes.

**Advantages of R Decision Trees**

Decision Trees are highly popular in Data Mining and Machine Learning techniques. Following are the advantages of Decision Trees:

* Decision Trees allow you to comprehend results which convey explicit conditions based on the original variables. Since Decision Trees do not require a lot of computation for processing, the IT staff can easily program the model without any hassle. The calculations comprise numerical comparisons that delineate whether the model is qualitative or quantitative in nature.
* Decision Trees are non-parametric in nature. Therefore, they do not follow any pattern of the probability distribution. Furthermore, the nature of these variables can be collinear.
* Extreme individuals do not affect Decision Trees. Such instances can be isolated in groups of minor nodes as they do not affect classification on a larger scale.
* As opposed to other ML algorithms that suffer from missing data, Decision Trees can very well handle it. With the help of CHAID, Decision Trees can handle missing variables by treating them as an isolated category or merging them into another.
* Trees like CART and C5.0 allow the variables to be handled directly. These type of variables are *continuous, discrete and qualitative in nature.*
* There are various visual representations provided by the Decision Tree for making decisions. This enhances communication and the branches contribute towards greater decision making.



**Disadvantages of R Decision Trees**

Decision Trees possess the following disadvantages:

* According to the definition, the nodes at the level n+1 are dependent on the definition of n level. Therefore, if the condition at level n is true, only then n+1 will be true. Otherwise, it will be false.
* The tree is constrained to local optima. It cannot detect global optima. The values are evaluated sequentially and not simultaneously. Due to this, the node will never revise its choice of division subsequently. The tree detects local, not global optima. It evaluates all the independent variables sequentially, not simultaneously.
* If the variable is located near the top of the true, its modification can alter the structure of the entire tree. This can be overcome with oversampling but it will reduce the readability of the model.

Examples of use of decision tress is − predicting an email as spam or not spam, predicting of a tumor is cancerous or predicting a loan as a good or bad credit risk based on the factors in each of these. Generally, a model is created with observed data also called training data. Then a set of validation data is used to verify and improve the model. R has packages which are used to create and visualize decision trees. For new set of predictor variable, we use this model to arrive at a decision on the category (yes/No, spam/not spam) of the data.

The R package **"party"** is used to create decision trees.

**Install R Package**

Use the below command in R console to install the package. You also have to install the dependent packages if any.

install.packages("party")

The package "party" has the function **ctree()** which is used to create and analyze decison tree.

**Syntax**

The basic syntax for creating a decision tree in R is −

ctree(formula, data)

Following is the description of the parameters used −

* **formula** is a formula describing the predictor and response variables.
* **data** is the name of the data set used.

**Input Data**

We will use the R in-built data set named **readingSkills** to create a decision tree. It describes the score of someone's readingSkills if we know the variables "age","shoesize","score" and whether the person is a native speaker or not.

Here is the sample data.

# Load the party package. It will automatically load other

# dependent packages.

library(party)

# Print some records from data set readingSkills.

print(head(readingSkills))

When we execute the above code, it produces the following result and chart −

nativeSpeaker age shoeSize score

1 yes 5 24.83189 32.29385

2 yes 6 25.95238 36.63105

3 no 11 30.42170 49.60593

4 yes 7 28.66450 40.28456

5 yes 11 31.88207 55.46085

6 yes 10 30.07843 52.83124

Loading required package: methods

Loading required package: grid

...............................

...............................

**Example**

We will use the **ctree()** function to create the decision tree and see its graph.

# Load the party package. It will automatically load other

# dependent packages.

library(party)

# Create the input data frame.

input.dat <- readingSkills[c(1:105),]

# Give the chart file a name.

png(file = "decision\_tree.png")

# Create the tree.

output.tree <- ctree(

nativeSpeaker ~ age + shoeSize + score,

data = input.dat)

# Plot the tree.

plot(output.tree)

# Save the file.

dev.off()

When we execute the above code, it produces the following result −

null device

1

Loading required package: methods

Loading required package: grid

Loading required package: mvtnorm

Loading required package: modeltools

Loading required package: stats4

Loading required package: strucchange

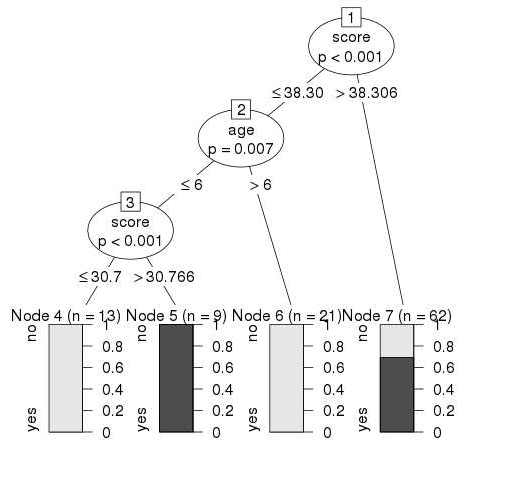
Loading required package: zoo

Attaching package: ‘zoo’

The following objects are masked from ‘package:base’:

as.Date, as.Date.numeric

Loading required package: sandwich



**Conclusion**

From the decision tree shown above we can conclude that anyone whose readingSkills score is less than 38.3 and age is more than 6 is not a native Speaker.

Does decision tree need preprocessing?

One of the benefits of decision trees is that **ordinal (continuous or discrete) input data does not require any significant preprocessing**. In fact, the results should be consistent regardless of any scaling or translational normalization, since the trees can choose equivalent splitting points.

# Building a Decision tree

There are several algorithms to build a decision tree.

1. CART-Classification and Regression Trees
2. ID3-Iterative Dichotomiser 3
3. C4.5
4. CHAID-Chi-squared Automatic Interaction Detection

We will be discussing only CART and ID3 algorithms as they are the ones majorly used.

# CART

CART is a DT algorithm that produces **binary** Classification or Regression Trees, depending on whether the dependent (or target) variable is categorical or numeric, respectively. It handles data in its raw form (no preprocessing needed) and can use the same variables more than once in different parts of the same DT, which may uncover complex interdependencies between sets of variables.

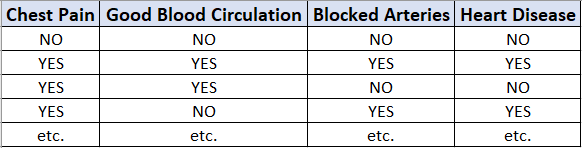


Fig.5- Sample dataset

Now we are going to discuss how to build a decision tree from a raw table of data. In the example given above, we will be building a decision tree that uses chest pain, good blood circulation, and the status of blocked arteries to predict if a person has heart disease or not.

The first thing we have to know is which feature should be on the top or in the root node of the tree. We start by looking at how chest pain alone predicts heart disease.

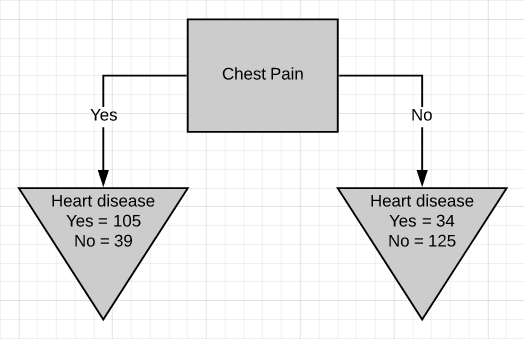


Fig.6-Chest pain as the root node

There are two leaf nodes, one each for the two outcomes of chest pain. Each of the leaves contains the no. of patients having heart disease and not having heart disease for the corresponding entry of chest pain. Now we do the same thing for good blood circulation and blocked arteries.

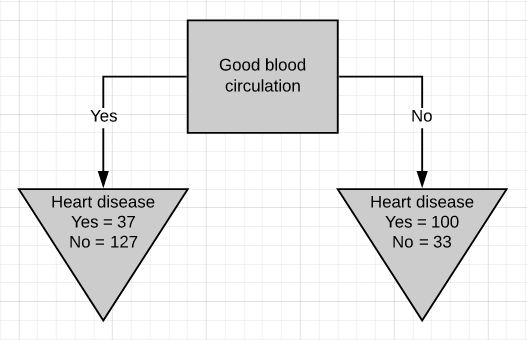


Fig.7-Good blood circulation as the root node

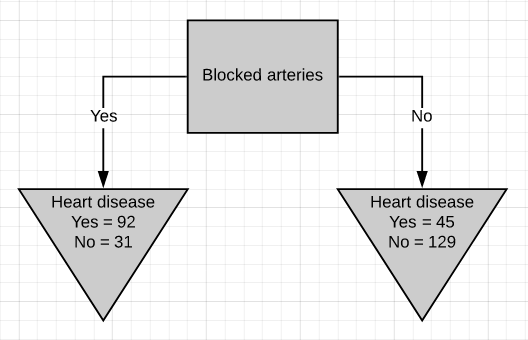


Fig.8-Blocked arteries as the root node

We can see that neither of the 3 features separates the patients having heart disease from the patients not having heart disease perfectly. It is to be noted that the total no. of patients having heart disease is different in all three cases. This is done to simulate the missing values present in real-world datasets.

Because none of the leaf nodes is either 100% ‘yes heart disease’ or 100% ‘no heart disease’, they are all considered **impure.** To decide on which separation is the best, we need a method to measure and compare **impurity.**

The metric used in the CART algorithm to measure impurity is the **Gini impurity score**. Calculating Gini impurity is very easy. Let’s start by calculating the Gini impurity for chest pain.

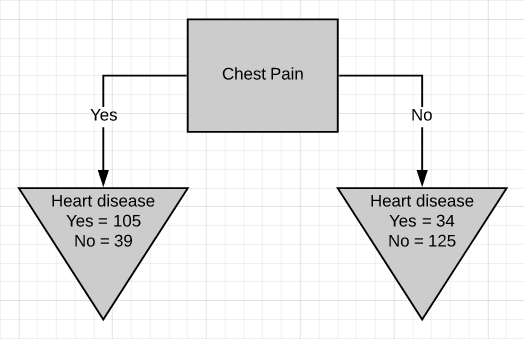


Fig.9- Chest pain separation

For the left leaf,

**Gini impurity = 1 - (probability of ‘yes’)² - (probability of ‘no’)²**  
 = 1 - (105/105+39)² - (39/105+39)²  
Gini impurity = 0.395

Similarly, calculate the Gini impurity for the right leaf node.

**Gini impurity = 1 - (probability of ‘yes’)² - (probability of ‘no’)²**  
 = 1 - (34/34+125)² - (125/34+125)²  
Gini impurity = 0.336

Now that we have measured the Gini impurity for both leaf nodes, we can calculate the total Gini impurity for using chest pain to separate patients with and without heart disease.

The leaf nodes do not represent the same no. of patients as the left leaf represents 144 patients and the right leaf represents 159 patients. Thus the total Gini impurity will be the weighted average of the leaf node Gini impurities.

Gini impurity = (144/144+159)\*0.395 + (159/144+159)\*0.336  
 = 0.364

Similarly the total Gini impurity for ‘good blood circulation’ and ‘blocked arteries’ is calculated as

Gini impurity for ‘good blood circulation’ = 0.360  
Gini impurity for ‘blocked arteries’ = 0.381

‘Good blood circulation’ has the lowest impurity score among the tree which symbolizes that it best separates the patients having and not having heart disease, so we will use it at the root node.

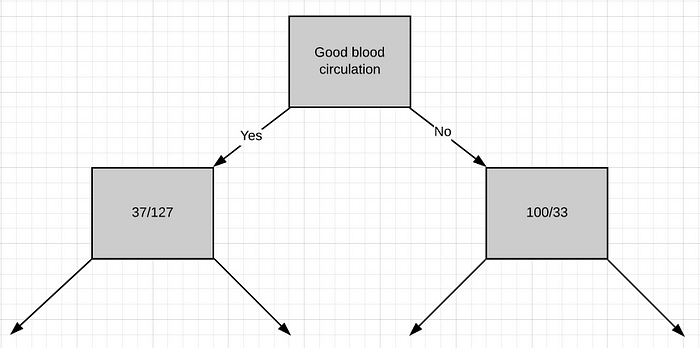


Fig.10-Good blood circulation at the root node

Now we need to figure out how well ‘chest pain’ and ‘blocked arteries’ separate the 164 patients in the left node(37 with heart disease and 127 without heart disease).

Just like we did before we will separate these patients with ‘chest pain’ and calculate the Gini impurity value.

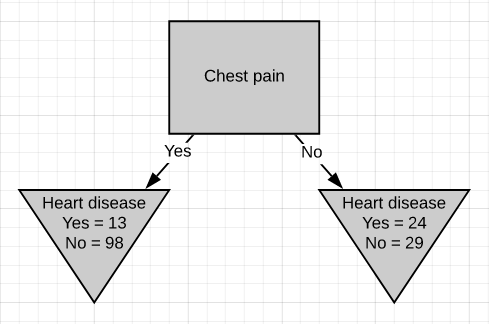


Fig.11- Chest pain separation

The Gini impurity was found to be 0.3. Then we do the same thing for ‘blocked arteries’.

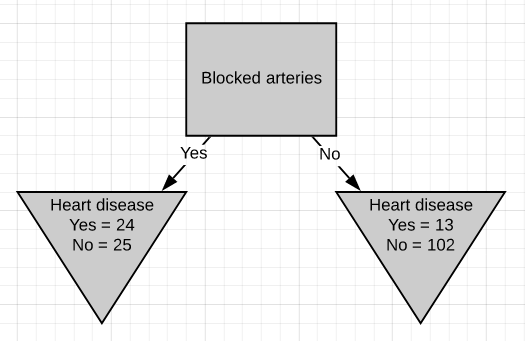


Fig.12-Blocked arteries separation

The Gini impurity was found to be 0.29. Since ‘blocked arteries’ has the lowest Gini impurity, we will use it at the left node in Fig.10 for further separating the patients.

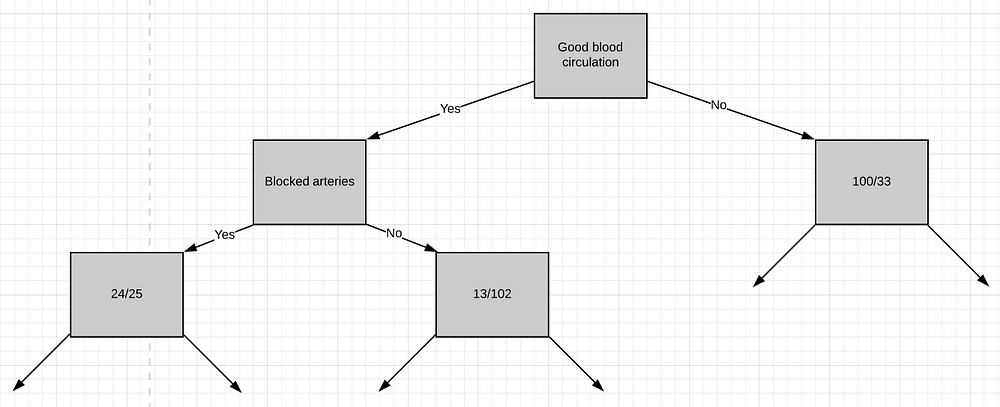


Fig.13-Blocked arteries separation

All we have left is ‘chest pain’, so we will see how well it separates the 49 patients in the left node(24 with heart disease and 25 without heart disease).

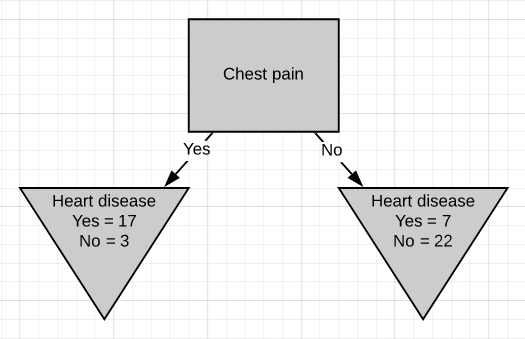


Fig.14-Chest pain separation in the left node

We can see that chest pain does a good job separating the patients.

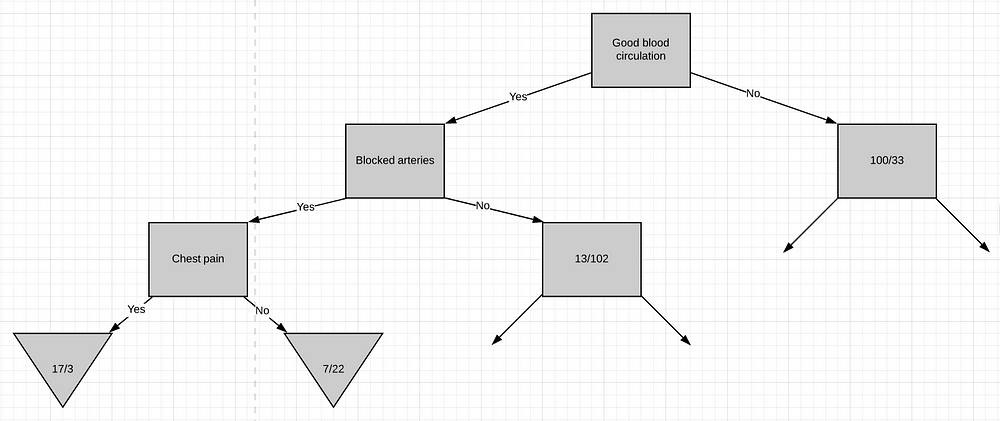


Fig.15-Final chest pain separation

So these are the final leaf nodes of the left side of this branch of the tree. Now let’s see what happens when we try to separate the node having 13/102 patients using ‘chest pain’. Note that almost 90% of the people in this node are not having heart disease.

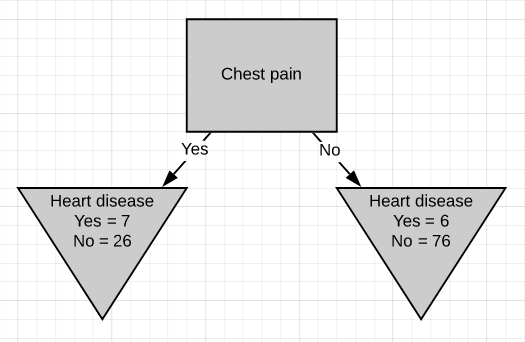


Fig.16-Chest pain separation on the right node

The Gini impurity of this separation is 0.29. But the Gini impurity for the parent-node before using chest-pain to separate the patients is

Gini impurity = 1 - (probability of yes)² - (probability of no)²  
 = 1 - (13/13+102)² - (102/13+102)²  
Gini impurity = 0.2

The impurity is lower if we don’t separate patients using ‘chest pain’. So we will make it a leaf-node.

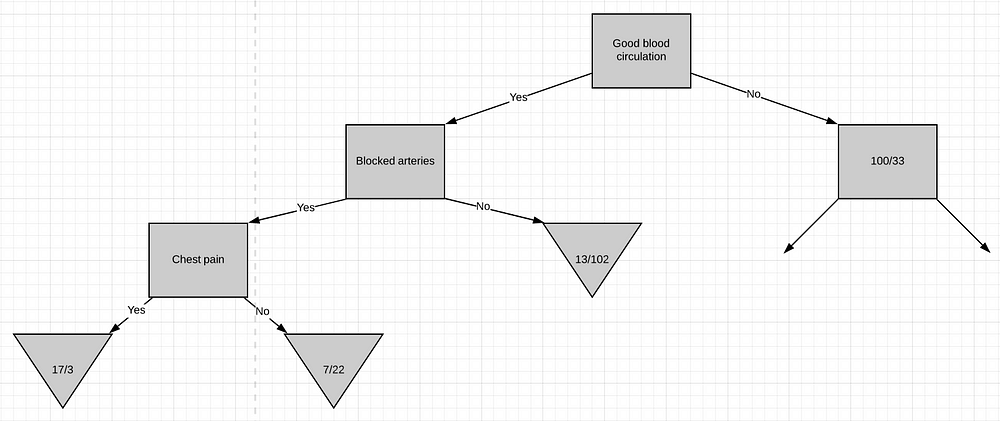
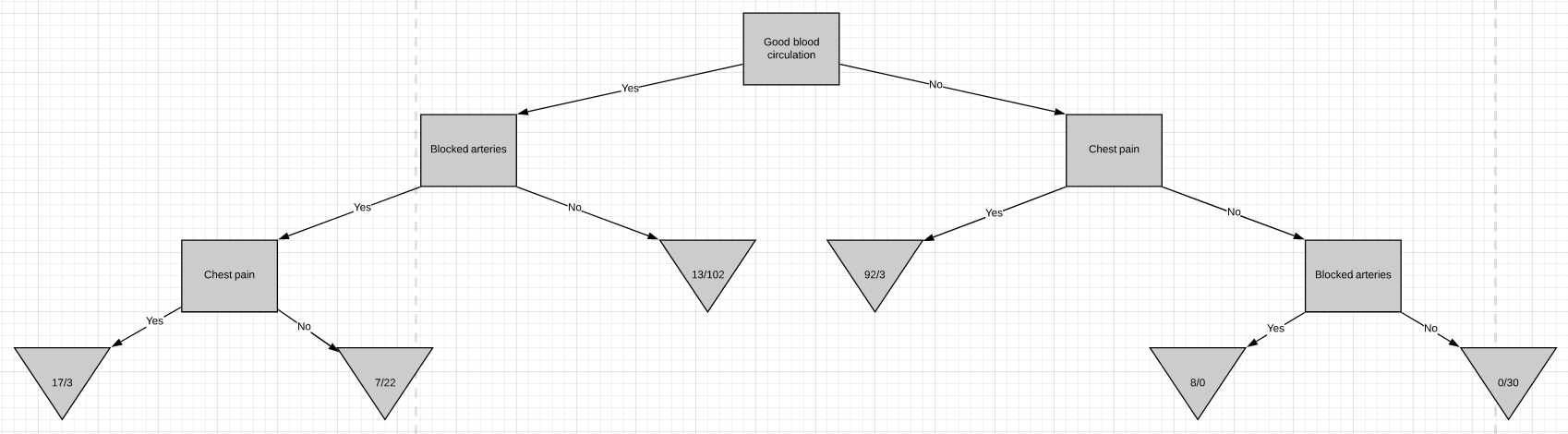


Fig.17-Left side completed

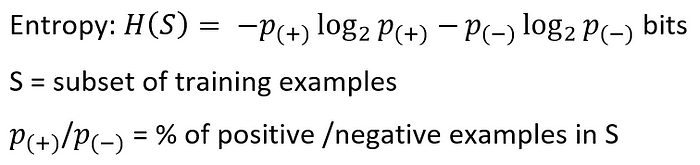
At this point, we have worked out the entire left side of the tree. The same steps are to be followed to work out the right side of the tree.

1. Calculate the Gini impurity scores.
2. If the node itself has the lowest score, then there is no point in separating the patients anymore and it becomes a leaf node.
3. If separating the data results in improvement then pick the separation with the lowest impurity value.

Fig.18-Complete Decision tree

# ID3

The process of building a decision tree using the ID3 algorithm is almost similar to using the CART algorithm except for the method used for measuring purity/impurity. The metric used in the ID3 algorithm to measure purity is called **Entropy**.



Entropy is a way to measure the uncertainty of a class in a subset of examples. Assume item belongs to subset S having two classes positive and negative. Entropy is defined as the no. of bits needed to say whether x is positive or negative.

Entropy always gives a number between 0 and 1. So if a subset formed after separation using an attribute is pure, then we will be needing zero bits to tell if is positive or negative. If the subset formed is having equal no. of positive and negative items then the no. of bits needed would be 1.

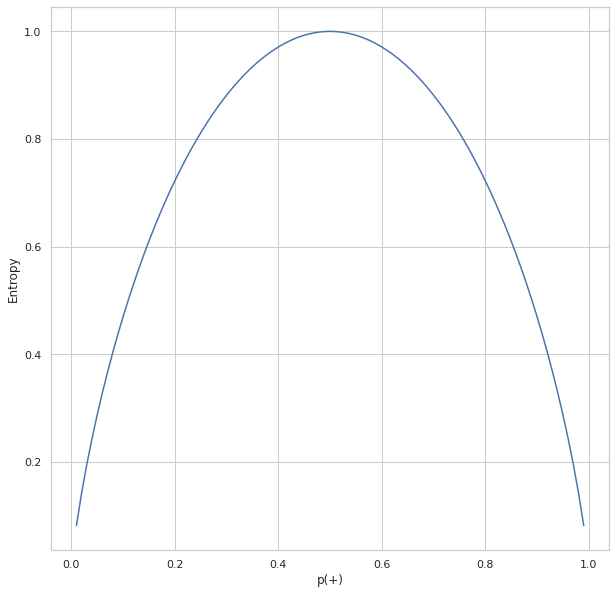
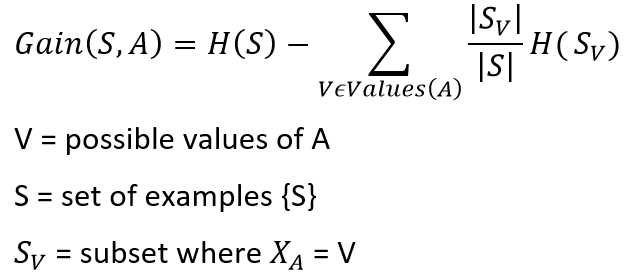


Fig-19.Entropy vs. p(+)

The above plot shows the relation between entropy and i.e., the probability of positive class. As we can see, the entropy reaches 1 which is the maximum value when which is there are equal chances for an item to be either positive or negative. The entropy is at its minimum when p(+) tends to zero(symbolizing x is negative) or 1(symbolizing x is positive).

Entropy tells us how pure or impure each subset is after the split. What we need to do is aggregate these scores to check whether the split is feasible or not. This is done by **Information gain**.



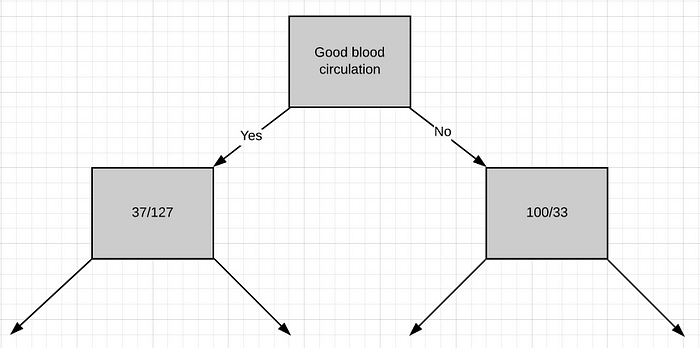
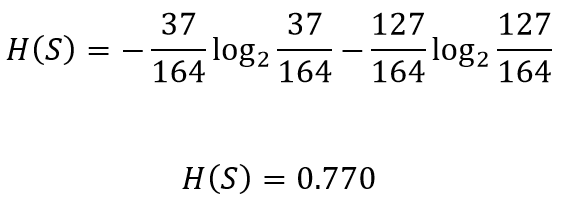


Fig-20.Building an ID3 tree

Consider this part of the problem we discussed above for the CART algorithm. We need to decide which attribute to use from **chest pain** and **blocked arteries** for separating the left node containing 164 patients(37 having heart disease and 127 not having heart disease). We can calculate the entropy before splitting as



Let’s see how well **chest pain** separates the patients

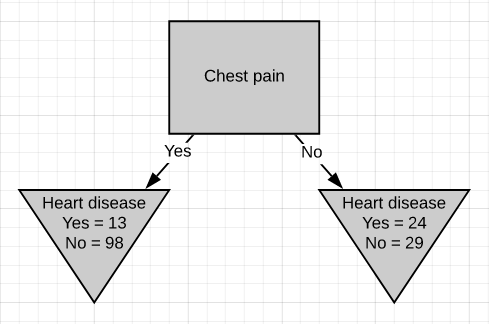
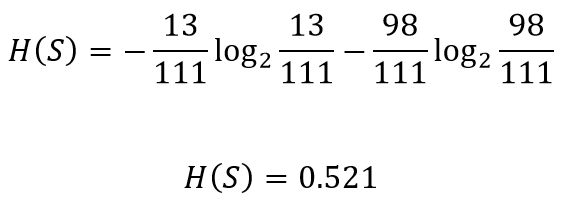
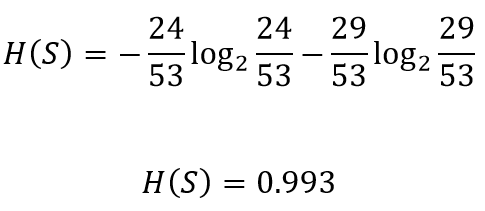


Fig.21- Chest pain separation

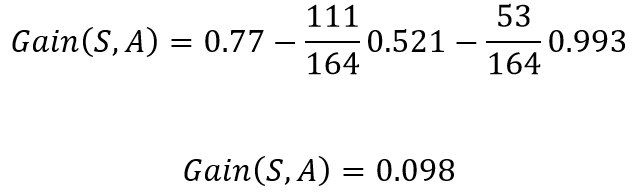
The entropy for the left node can be calculated



Similarly the entropy for the right node



The total gain in entropy after splitting using **chest pain**



This implies that if in the current situation if we were to pick **chest pain** for splitting the patients, we would gain 0.098 bits in certainty on the patient having or not having heart disease. Doing the same for **blocked arteries ,** the gain obtained was 0.117. Since splitting with **blocked arteries** gives us more certainty, it would be picked. We can repeat the same procedure for all the nodes to build a DT based on the ID3 algorithm.

Note: The decision of whether to split a node into 2 or to declare it as a leaf node can be made by imposing a minimum threshold on the gain value required. If the acquired gain is above the threshold value, we can split the node, otherwise, leave it as a leaf node.

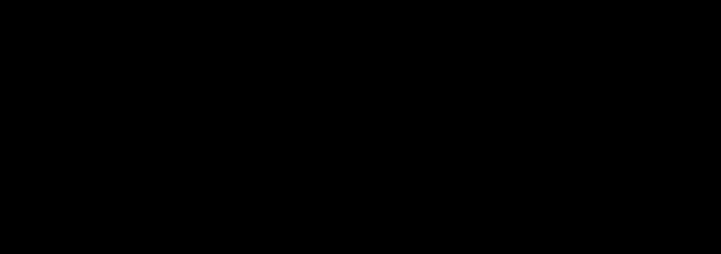
**Training and Visualizing a decision trees in R**

To build your first decision tree in R example, we will proceed as follow in this Decision Tree tutorial:

* [Step 1: Import the data](https://www.guru99.com/r-decision-trees.html#1)
* [Step 2: Clean the dataset](https://www.guru99.com/r-decision-trees.html#2)
* [Step 3: Create train/test set](https://www.guru99.com/r-decision-trees.html#3)
* [Step 4: Build the model](https://www.guru99.com/r-decision-trees.html#4)
* [Step 5: Make prediction](https://www.guru99.com/r-decision-trees.html#5)
* [Step 6: Measure performance](https://www.guru99.com/r-decision-trees.html#6)
* [Step 7: Tune the hyper-parameters](https://www.guru99.com/r-decision-trees.html#7)

**Step 1) Import the data**

If you are curious about the fate of the titanic, you can watch this video on [Youtube](https://www.youtube.com/watch?v=9xoqXVjBEF8). The purpose of this dataset is to predict which people are more likely to survive after the collision with the iceberg. The dataset contains 13 variables and 1309 observations. The dataset is ordered by the variable X.





EXPLORE MORELearn Java Programming with Beginners Tutorial08:32Linux Tutorial for Beginners: Introduction to Linux Operating...01:35What is Integration Testing Software Testing Tutorial03:04What is JVM (Java Virtual Machine) with Architecture JAVA...02:24How to write a TEST CASE Software Testing Tutorial01:08Seven Testing Principles Software Testing05:01Linux File Permissions Commands with Examples13:29How to use Text tool in Photoshop CC Tutorial08:32What is NoSQL Database Tutorial02:00

Important Linux Commands for Beginners Linux Tutorial15:03

set.seed(678)

path <- 'https://raw.githubusercontent.com/guru99-edu/R-Programming/master/titanic\_data.csv'

titanic <-read.csv(path)

head(titanic)

**Output:**

## X pclass survived name sex

## 1 1 1 1 Allen, Miss. Elisabeth Walton female

## 2 2 1 1 Allison, Master. Hudson Trevor male

## 3 3 1 0 Allison, Miss. Helen Loraine female

## 4 4 1 0 Allison, Mr. Hudson Joshua Creighton male

## 5 5 1 0 Allison, Mrs. Hudson J C (Bessie Waldo Daniels) female

## 6 6 1 1 Anderson, Mr. Harry male

## age sibsp parch ticket fare cabin embarked

## 1 29.0000 0 0 24160 211.3375 B5 S

## 2 0.9167 1 2 113781 151.5500 C22 C26 S

## 3 2.0000 1 2 113781 151.5500 C22 C26 S

## 4 30.0000 1 2 113781 151.5500 C22 C26 S

## 5 25.0000 1 2 113781 151.5500 C22 C26 S

## 6 48.0000 0 0 19952 26.5500 E12 S

## home.dest

## 1 St Louis, MO

## 2 Montreal, PQ / Chesterville, ON

## 3 Montreal, PQ / Chesterville, ON

## 4 Montreal, PQ / Chesterville, ON

## 5 Montreal, PQ / Chesterville, ON

## 6 New York, NY

tail(titanic)

**Output:**

## X pclass survived name sex age sibsp

## 1304 1304 3 0 Yousseff, Mr. Gerious male NA 0

## 1305 1305 3 0 Zabour, Miss. Hileni female 14.5 1

## 1306 1306 3 0 Zabour, Miss. Thamine female NA 1

## 1307 1307 3 0 Zakarian, Mr. Mapriededer male 26.5 0

## 1308 1308 3 0 Zakarian, Mr. Ortin male 27.0 0

## 1309 1309 3 0 Zimmerman, Mr. Leo male 29.0 0

## parch ticket fare cabin embarked home.dest

## 1304 0 2627 14.4583 C

## 1305 0 2665 14.4542 C

## 1306 0 2665 14.4542 C

## 1307 0 2656 7.2250 C

## 1308 0 2670 7.2250 C

## 1309 0 315082 7.8750 S

From the head and tail output, you can notice the data is not shuffled. This is a big issue! When you will split your data between a train set and test set, you will select **only** the passenger from class 1 and 2 (No passenger from class 3 are in the top 80 percent of the observations), which means the algorithm will never see the features of passenger of class 3. This mistake will lead to poor prediction.

To overcome this issue, you can use the function sample().

shuffle\_index <- sample(1:nrow(titanic))

head(shuffle\_index)

Decision tree R code Explanation

* sample(1:nrow(titanic)): Generate a random list of index from 1 to 1309 (i.e. the maximum number of rows).

**Output:**

## [1] 288 874 1078 633 887 992

You will use this index to shuffle the titanic dataset.

titanic <- titanic[shuffle\_index, ]

head(titanic)

**Output:**

## X pclass survived

## 288 288 1 0

## 874 874 3 0

## 1078 1078 3 1

## 633 633 3 0

## 887 887 3 1

## 992 992 3 1

## name sex age

## 288 Sutton, Mr. Frederick male 61

## 874 Humblen, Mr. Adolf Mathias Nicolai Olsen male 42

## 1078 O'Driscoll, Miss. Bridget female NA

## 633 Andersson, Mrs. Anders Johan (Alfrida Konstantia Brogren) female 39

## 887 Jermyn, Miss. Annie female NA

## 992 Mamee, Mr. Hanna male NA

## sibsp parch ticket fare cabin embarked home.dest## 288 0 0 36963 32.3208 D50 S Haddenfield, NJ

## 874 0 0 348121 7.6500 F G63 S

## 1078 0 0 14311 7.7500 Q

## 633 1 5 347082 31.2750 S Sweden Winnipeg, MN

## 887 0 0 14313 7.7500 Q

## 992 0 0 2677 7.2292 C

**Step 2) Clean the dataset**

The structure of the data shows some variables have NA’s. Data clean up to be done as follows

* Drop variables home.dest,cabin, name, X and ticket
* Create factor variables for pclass and survived
* Drop the NA

library(dplyr)

# Drop variables

clean\_titanic <- titanic % > %

select(-c(home.dest, cabin, name, X, ticket)) % > %

#Convert to factor level

mutate(pclass = factor(pclass, levels = c(1, 2, 3), labels = c('Upper', 'Middle', 'Lower')),

survived = factor(survived, levels = c(0, 1), labels = c('No', 'Yes'))) % > %

na.omit()

glimpse(clean\_titanic)

Code Explanation

* select(-c(home.dest, cabin, name, X, ticket)): Drop unnecessary variables
* pclass = factor(pclass, levels = c(1,2,3), labels= c(‘Upper’, ‘Middle’, ‘Lower’)): Add label to the variable pclass. 1 becomes Upper, 2 becomes MIddle and 3 becomes lower
* factor(survived, levels = c(0,1), labels = c(‘No’, ‘Yes’)): Add label to the variable survived. 1 Becomes No and 2 becomes Yes
* na.omit(): Remove the NA observations

**Output:**

## Observations: 1,045

## Variables: 8

## $ pclass <fctr> Upper, Lower, Lower, Upper, Middle, Upper, Middle, U...

## $ survived <fctr> No, No, No, Yes, No, Yes, Yes, No, No, No, No, No, Y...

## $ sex <fctr> male, male, female, female, male, male, female, male...

## $ age <dbl> 61.0, 42.0, 39.0, 49.0, 29.0, 37.0, 20.0, 54.0, 2.0, ...

## $ sibsp <int> 0, 0, 1, 0, 0, 1, 0, 0, 4, 0, 0, 1, 1, 0, 0, 0, 1, 1,...

## $ parch <int> 0, 0, 5, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 2, 0, 4, 0,...

## $ fare <dbl> 32.3208, 7.6500, 31.2750, 25.9292, 10.5000, 52.5542, ...

## $ embarked <fctr> S, S, S, S, S, S, S, S, S, C, S, S, S, Q, C, S, S, C...

**Step 3) Create train/test set**

Before you train your model, you need to perform two steps:

* Create a train and test set: You train the model on the train set and test the prediction on the test set (i.e. unseen data)
* Install rpart.plot from the console

The common practice is to split the data 80/20, 80 percent of the data serves to train the model, and 20 percent to make predictions. You need to create two separate data frames. You don’t want to touch the test set until you finish building your model. You can create a function name create\_train\_test() that takes three arguments.

create\_train\_test(df, size = 0.8, train = TRUE)

arguments:

-df: Dataset used to train the model.

-size: Size of the split. By default, 0.8. Numerical value

-train: If set to `TRUE`, the function creates the train set, otherwise the test set. Default value sets to `TRUE`. Boolean value.You need to add a Boolean parameter because R does not allow to return two data frames simultaneously.

create\_train\_test <- function(data, size = 0.8, train = TRUE) {

n\_row = nrow(data)

total\_row = size \* n\_row

train\_sample < - 1: total\_row

if (train == TRUE) {

return (data[train\_sample, ])

} else {

return (data[-train\_sample, ])

}

}

Code Explanation

* function(data, size=0.8, train = TRUE): Add the arguments in the function
* n\_row = nrow(data): Count number of rows in the dataset
* total\_row = size\*n\_row: Return the nth row to construct the train set
* train\_sample <- 1:total\_row: Select the first row to the nth rows
* if (train ==TRUE){ } else { }: If condition sets to true, return the train set, else the test set.

You can test your function and check the dimension.

data\_train <- create\_train\_test(clean\_titanic, 0.8, train = TRUE)

data\_test <- create\_train\_test(clean\_titanic, 0.8, train = FALSE)

dim(data\_train)

**Output:**

## [1] 836 8

dim(data\_test)

**Output:**

## [1] 209 8

The train dataset has 1046 rows while the test dataset has 262 rows.

You use the function prop.table() combined with table() to verify if the randomization process is correct.

prop.table(table(data\_train$survived))

**Output:**

##

## No Yes

## 0.5944976 0.4055024

prop.table(table(data\_test$survived))

**Output:**

##

## No Yes

## 0.5789474 0.4210526

In both dataset, the amount of survivors is the same, about 40 percent.

**Install rpart.plot**

rpart.plot is not available from conda libraries. You can install it from the console:

install.packages("rpart.plot")

**Step 4) Build the model**

You are ready to build the model. The syntax for Rpart decision tree function is:

rpart(formula, data=, method='')

arguments:

- formula: The function to predict

- data: Specifies the data frame- method:

- "class" for a classification tree

- "anova" for a regression tree

You use the class method because you predict a class.

library(rpart)

library(rpart.plot)

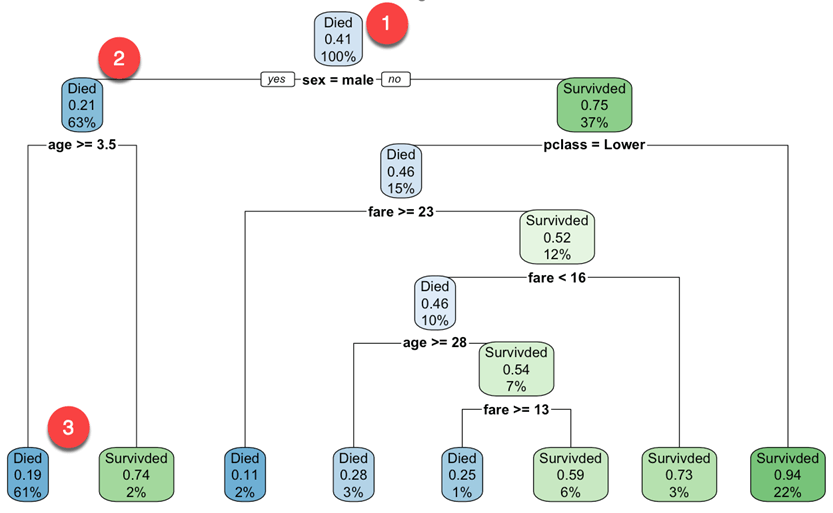
fit <- rpart(survived~., data = data\_train, method = 'class')

rpart.plot(fit, extra = 106

Code Explanation

* rpart(): Function to fit the model. The arguments are:
  + survived ~.: Formula of the Decision Trees
  + data = data\_train: Dataset
  + method = ‘class’: Fit a binary model
* rpart.plot(fit, extra= 106): Plot the tree. The extra features are set to 101 to display the probability of the 2nd class (useful for binary responses). You can refer to the [vignette](https://cran.r-project.org/web/packages/rpart.plot/rpart.plot.pdf) for more information about the other choices.

**Output:**



You start at the root node (depth 0 over 3, the top of the graph):

1. At the top, it is the overall probability of survival. It shows the proportion of passenger that survived the crash. 41 percent of passenger survived.
2. This node asks whether the gender of the passenger is male. If yes, then you go down to the root’s left child node (depth 2). 63 percent are males with a survival probability of 21 percent.
3. In the second node, you ask if the male passenger is above 3.5 years old. If yes, then the chance of survival is 19 percent.
4. You keep on going like that to understand what features impact the likelihood of survival.

Note that, one of the many qualities of Decision Trees is that they require very little data preparation. In particular, they don’t require feature scaling or centering.

By default, rpart() function uses the **Gini** impurity measure to split the note. The higher the Gini coefficient, the more different instances within the node.

**Step 5) Make a prediction**

You can predict your test dataset. To make a prediction, you can use the predict() function. The basic syntax of predict for R decision tree is:

predict(fitted\_model, df, type = 'class')

arguments:

- fitted\_model: This is the object stored after model estimation.

- df: Data frame used to make the prediction

- type: Type of prediction

- 'class': for classification

- 'prob': to compute the probability of each class

- 'vector': Predict the mean response at the node level

You want to predict which passengers are more likely to survive after the collision from the test set. It means, you will know among those 209 passengers, which one will survive or not.

predict\_unseen <-predict(fit, data\_test, type = 'class')

Code Explanation

* predict(fit, data\_test, type = ‘class’): Predict the class (0/1) of the test set

Testing the passenger who didn’t make it and those who did.

table\_mat <- table(data\_test**$**survived, predict\_unseen)

table\_mat

Code Explanation

* table(data\_test$survived, predict\_unseen): Create a table to count how many passengers are classified as survivors and passed away compare to the correct decision tree classification in R

**Output:**

## predict\_unseen

## No Yes

## No 106 15

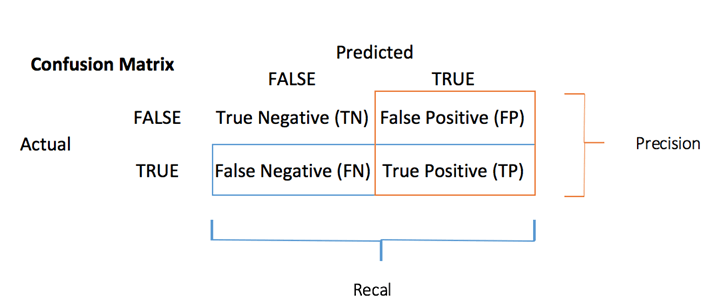
## Yes 30 58

The model correctly predicted 106 dead passengers but classified 15 survivors as dead. By analogy, the model misclassified 30 passengers as survivors while they turned out to be dead.

**Step 6) Measure performance**

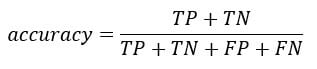
You can compute an accuracy measure for classification task with the [**confusion matrix**](https://www.guru99.com/confusion-matrix-machine-learning-example.html):

The **confusion matrix** is a better choice to evaluate the classification performance. The general idea is to count the number of times True instances are classified are False.



Each row in a confusion matrix represents an actual target, while each column represents a predicted target. The first row of this matrix considers dead passengers (the False class): 106 were correctly classified as dead (**True negative**), while the remaining one was wrongly classified as a survivor (**False positive**). The second row considers the survivors, the positive class were 58 (**True positive**), while the **True negative** was 30.

You can compute the **accuracy test** from the confusion matrix:



It is the proportion of true positive and true negative over the sum of the matrix. With R, you can code as follow:

accuracy\_Test <- sum(diag(table\_mat)) / sum(table\_mat)

Code Explanation

* sum(diag(table\_mat)): Sum of the diagonal
* sum(table\_mat): Sum of the matrix.

You can print the accuracy of the test set:

print(paste('Accuracy for test', accuracy\_Test))

**Output:**

## [1] "Accuracy for test 0.784688995215311"

You have a score of 78 percent for the test set. You can replicate the same exercise with the training dataset.

**Step 7) Tune the hyper-parameters**

Decision tree in R has various parameters that control aspects of the fit. In rpart decision tree library, you can control the parameters using the rpart.control() function. In the following code, you introduce the parameters you will tune. You can refer to the [vignette](https://cran.r-project.org/web/packages/rpart/rpart.pdf) for other parameters.

rpart.control(minsplit = 20, minbucket = round(minsplit/3), maxdepth = 30)

Arguments:

-minsplit: Set the minimum number of observations in the node before the algorithm perform a split

-minbucket: Set the minimum number of observations in the final note i.e. the leaf

-maxdepth: Set the maximum depth of any node of the final tree. The root node is treated a depth 0

We will proceed as follow:

* Construct function to return accuracy
* Tune the maximum depth
* Tune the minimum number of sample a node must have before it can split
* Tune the minimum number of sample a leaf node must have

You can write a function to display the accuracy. You simply wrap the code you used before:

1. predict: predict\_unseen <- predict(fit, data\_test, type = ‘class’)
2. Produce table: table\_mat <- table(data\_test$survived, predict\_unseen)
3. Compute accuracy: accuracy\_Test <- sum(diag(table\_mat))/sum(table\_mat)

accuracy\_tune <- function(fit) {

predict\_unseen <- predict(fit, data\_test, type = 'class')

table\_mat <- table(data\_test$survived, predict\_unseen)

accuracy\_Test <- sum(diag(table\_mat)) / sum(table\_mat)

accuracy\_Test

}

You can try to tune the parameters and see if you can improve the model over the default value. As a reminder, you need to get an accuracy higher than 0.78

control <- rpart.control(minsplit = 4,

minbucket = round(5 / 3),

maxdepth = 3,

cp = 0)

tune\_fit <- rpart(survived~., data = data\_train, method = 'class', control = control)

accuracy\_tune(tune\_fit)

**Output:**

## [1] 0.7990431

With the following parameter:

minsplit = 4

minbucket= round(5/3)

maxdepth = 3cp=0

You get a higher performance than the previous model. Congratulation!

**Summary**

We can summarize the functions to train a decision tree algorithm in R

| **Library** | **Objective** | **Function** | **Class** | **Parameters** | **Details** |
| --- | --- | --- | --- | --- | --- |
| rpart | Train classification tree in R | rpart() | class | formula, df, method |  |
| rpart | Train regression tree | rpart() | anova | formula, df, method |  |
| rpart | Plot the trees | rpart.plot() |  | fitted model |  |
| base | predict | predict() | class | fitted model, type |  |
| base | predict | predict() | prob | fitted model, type |  |
| base | predict | predict() | vector | fitted model, type |  |
| rpart | Control parameters | rpart.control() |  | minsplit | Set the minimum number of observations in the node before the algorithm perform a split |
|  |  |  |  | minbucket | Set the minimum number of observations in the final note i.e. the leaf |
|  |  |  |  | maxdepth | Set the maximum depth of any node of the final tree. The root node is treated a depth 0 |
| rpart | Train model with control parameter | rpart() |  | formula, df, method, control |  |

Note : Train the model on a training data and test the performance on an unseen dataset, i.e. test set.

LOGISTIC REGRESSION

The Logistic Regression is a regression model in which the response variable (dependent variable) has categorical values such as True/False or 0/1. It actually measures the probability of a binary response as the value of response variable based on the mathematical equation relating it with the predictor variables.

The general mathematical equation for logistic regression is −

y = 1/(1+e^-(a+b1x1+b2x2+b3x3+...))

Following is the description of the parameters used −

* **y** is the response variable.
* **x** is the predictor variable.
* **a** and **b** are the coefficients which are numeric constants.

The function used to create the regression model is the **glm()** function.

**Syntax**

The basic syntax for **glm()** function in logistic regression is −

glm(formula,data,family)

Following is the description of the parameters used −

* **formula** is the symbol presenting the relationship between the variables.
* **data** is the data set giving the values of these variables.
* **family** is R object to specify the details of the model. It's value is binomial for logistic regression.

**Example**

The in-built data set "mtcars" describes different models of a car with their various engine specifications. In "mtcars" data set, the transmission mode (automatic or manual) is described by the column am which is a binary value (0 or 1). We can create a logistic regression model between the columns "am" and 3 other columns - hp, wt and cyl.

[Live Demo](http://tpcg.io/c6hBJq)

# Select some columns form mtcars.

input <- mtcars[,c("am","cyl","hp","wt")]

print(head(input))

When we execute the above code, it produces the following result −

am cyl hp wt

Mazda RX4 1 6 110 2.620

Mazda RX4 Wag 1 6 110 2.875

Datsun 710 1 4 93 2.320

Hornet 4 Drive 0 6 110 3.215

Hornet Sportabout 0 8 175 3.440

Valiant 0 6 105 3.460

**Create Regression Model**

We use the **glm()** function to create the regression model and get its summary for analysis.

[Live Demo](http://tpcg.io/VmrzPP)

input <- mtcars[,c("am","cyl","hp","wt")]

am.data = glm(formula = am ~ cyl + hp + wt, data = input, family = binomial)

print(summary(am.data))

When we execute the above code, it produces the following result −

Call:

glm(formula = am ~ cyl + hp + wt, family = binomial, data = input)

Deviance Residuals:

Min 1Q Median 3Q Max

-2.17272 -0.14907 -0.01464 0.14116 1.27641

Coefficients:

Estimate Std. Error z value Pr(>|z|)

(Intercept) 19.70288 8.11637 2.428 0.0152 \*

cyl 0.48760 1.07162 0.455 0.6491

hp 0.03259 0.01886 1.728 0.0840 .

wt -9.14947 4.15332 -2.203 0.0276 \*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 43.2297 on 31 degrees of freedom

Residual deviance: 9.8415 on 28 degrees of freedom

AIC: 17.841

Number of Fisher Scoring iterations: 8

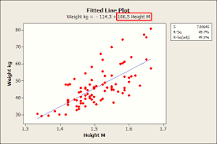
**Conclusion**

In the summary as the p-value in the last column is more than 0.05 for the variables "cyl" and "hp", we consider them to be insignificant in contributing to the value of the variable "am". Only weight (wt) impacts the "am" value in this regression model.

**Interpret the key results for Binary Logistic Regression**

1. Step 1: Determine whether the association between the response and the term is statistically significant.
2. Step 2: Understand the effects of the predictors.
3. Step 3: Determine how well the model fits your data.
4. Step 4: Determine whether the model does not fit the data.

How do you interpret parameters in regression?

[[](https://www.google.com/search?hl=en-US&biw=1440&bih=775&q=How+do+you+interpret+parameters+in+regression?&tbm=isch&source=iu&ictx=1&vet=1&fir=zG51CkvBIE7T6M%2CU3xZ4gjBTPxPoM%2C_&usg=AI4_-kTx0kxqihTKnxOB8OQ4EUpa9J-d3g&sa=X&ved=2ahUKEwiEnMTq_6T5AhV9K0QIHWEZBykQ9QF6BAgTEAE#imgrc=zG51CkvBIE7T6M)](https://www.google.com/search?hl=en-US&biw=1440&bih=775&q=How+do+you+interpret+parameters+in+regression?&tbm=isch&source=iu&ictx=1&vet=1&fir=zG51CkvBIE7T6M%252CU3xZ4gjBTPxPoM%252C_&usg=AI4_-kTx0kxqihTKnxOB8OQ4EUpa9J-d3g&sa=X&ved=2ahUKEwiEnMTq_6T5AhV9K0QIHWEZBykQ9QF6BAgTEAE" \l "imgrc=zG51CkvBIE7T6M)

**The sign of a regression coefficient tells you whether there is a positive or negative correlation between each independent variable and the dependent variable**. A positive coefficient indicates that as the value of the independent variable increases, the mean of the dependent variable also tends to increase.

How do you estimate the parameters of a model?

The two major methods of parameter estimation for process models are **maximum likelihood and least squares**. Both of these methods provide parameter estimators that have many good properties. Both maximum likelihood and least squares are sensitive to the presence of outliers, however.