Machine Learning

2022-12-22

Chapter one : Introduction to Machine Learning

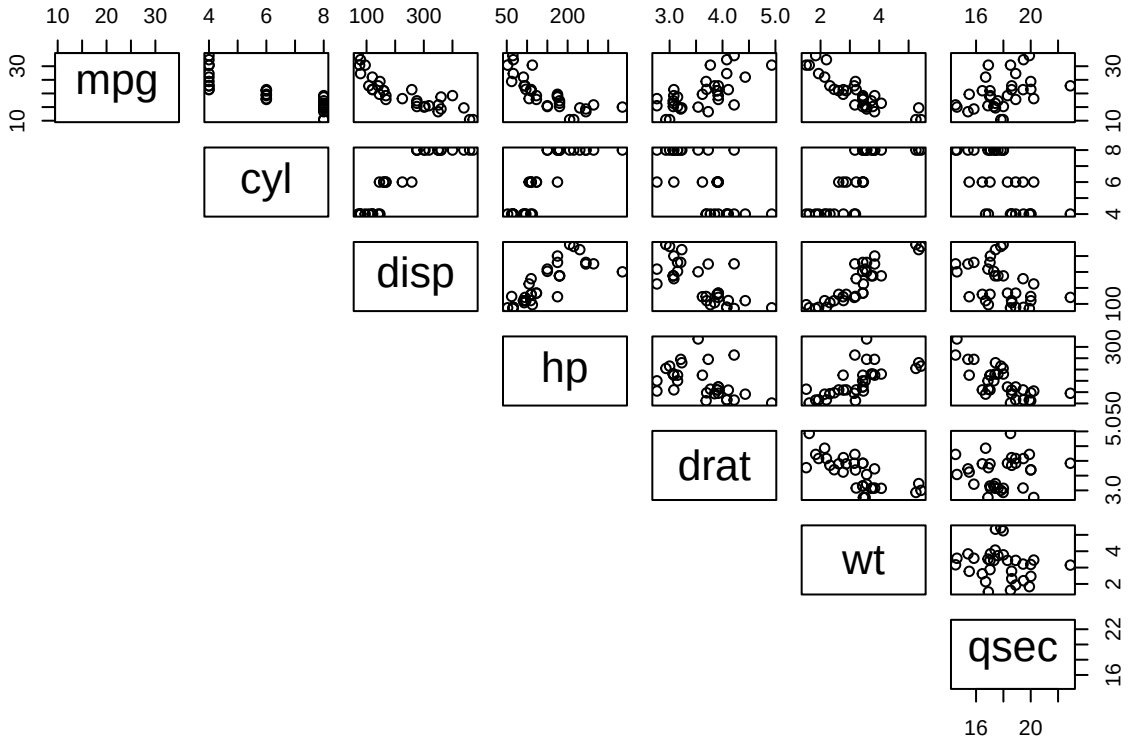
A model is any sort of function that has predictive power. In Machine Learning world, rows of data are sometimes referred to as features. Below is an example using the mtcars in R.

```
head(mtcars)
```

```
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

A pair plot is as follows;

```
pairs(mtcars[1:7], lower.panel = NULL)
```



Each box is its own plot, where the dependent variable is the text box on the bottom of the row and the independent variable is the text box at the beginning of the row. These boxes or plots shows whether there is a linear relationship between features themselves. For example, there is no linear relationship in the *cyl* row.

1.1 Algorithm vs models

An algorithm is a set by set procedure performed in order. For example, in a simple linear regression the algorithm is put two points on a plot and draw a line between them. However, the procedure becomes complicated when there is more than two data points. Essentially, the algorithm is what R function does.

The are three types of models;

1. Regression Models
2. Classification Models
3. Mixed models (involves both regression and classification)

1.2 Modelling Limitations

A statistician George Box said that "All models are wrong but some are useful". What he meant is that all models have an error attributed to them. Different models have different limitations associated with them. Regression models have a specific way of measuring error called the coefficient of determination, often referred to as the "R-squared" value. This is a measure of how close the data fits the model fitted line with values ranging from 0 to 1. The power of a model comes from its usefulness.

1. 3 Data Training

Machine Learning requires us to train the model. Methods of splitting up data for training and testing purposes are known as sampling techniques.

1.4 Cross Validation

In many cases during the training phase, you might want to try other statistical techniques like cross-validation. This is sort of like another mini-step of splitting into training and test sets and running the model, but only on the training data. For example, you take your 50-point total dataset and split 80% into a training set, leaving the rest for your final test phase. We are left with 40 rows with which to train your data model. You can split these 40 rows further into a 32-row training set and an 8-row test set. By doing so and going through a similar training and test procedure, you can get an ensemble of errors out of your model and use those to help refine its tuning even further.Some examples of cross-validation techniques in R include the following:

- Bootstrap cross-validation
- Bootstrap 632 cross-validation
- k-fold cross-validation
- Repeated cross-validation
- Leave-one-out cross-validation
- Leave-group-out cross-validation
- Out-of-bag cross-validation
- Adaptive cross-validation
- Adaptive bootstrap cross-validation
- Adaptive leave-group-out cross-validation

Their usage is highly dependent on the structure of the data itself. The typical gold standard of cross-validation techniques is k-fold cross-validation, wherein you pick $k = 10$ folds against which to validate. This is the best balance between efficient data usage and avoiding splits in the data that might be poor choices.

Chapter Two : Supervised and Unsupervised Machine Learning

Supervised learning models are those in which a machine learning model is scored and tuned against some sort of known quantity. The majority of machine learning algorithms are supervised learners. Unsupervised learning models are those in which the machine learning model derives patterns and information from data while determining the known quantity tuning parameter itself. Supervised machine learning is generally used to classify data or make predictions, whereas unsupervised learning is generally used to understand relationships within data sets.

2.1 Regression

Below is a simple model

```
model <- lm(mtcars$mpg ~ mtcars$disp)
```

You now have a very simple machine learning model! You can use any input for the engine size and get a value out. Before we jump into the other major realm of supervised learning, we need to bring up the topic about training and testing data. As we've seen with simple linear regression modeling thus far, we have a model that we can use to predict future values. Yet, we know nothing about how accurate the model is for the moment. One way to determine model accuracy is to look at the R-squared value from the model:

```
summary(model)
```

```
Call:
lm(formula = mtcars$mpg ~ mtcars$disp)

Residuals:
    Min      1Q  Median      3Q     Max
-4.8922 -2.2022 -0.9631  1.6272  7.2305

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 29.599855   1.229720  24.070  < 2e-16 ***
mtcars$disp -0.041215   0.004712  -8.747 9.38e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.251 on 30 degrees of freedom
Multiple R-squared:  0.7183,    Adjusted R-squared:  0.709
F-statistic: 76.51 on 1 and 30 DF,  p-value: 9.38e-10
```

This value tells us how linearly correlated the data is; the closer the value is to 1, the more likely the model output is governed by data that's almost exactly a straight line with some kind of slope value to it. The reason we are focusing on the adjusted part instead of the multiple is for future scenarios in which we use more features in a model. For low numbers of features the adjusted and multiple R-squared values are basically the same thing. For models that have many features, we want to use multiple R-squared values, instead, because it will give a more accurate assessment of the model error if we have many dependent features instead of just one. But what does this tell us as far as an error estimate for the model? We have standard error values from the output, but there's an issue with the model being trained on all the data, then being tested on the same data. What we want to do, in order to ensure an unbiased amount of error, is to split our starting dataset into a training dataset and test dataset. In the world of statistics, you do this by taking a dataset you have and splitting it into 80% training data and 20% test data. You can tinker with those numbers to your taste, but you always want more training data than test data:

```
split_size = .8

sample_size = floor(split_size * nrow(mtcars))

set.seed(123)
train_indicies <- sample(seq_len(nrow(mtcars)), size = sample_size)
train <- mtcars[train_indicies,]
test <- mtcars[-train_indicies,]
```

floor() function in R returns the largest integer that is smaller than or equal to value passed to it as argument(i.e : rounds downs to the nearest integer). **seq_len()** function in R is used to generate a sequence

from 1 to the specified number. What we want to do now is to build a regression model using only the training data. We then pass the test data values into it to get the model outputs. The key component here is that we have the known data against which we can test the model. That allows us to get a better level of error estimate out:

```
model2 <- lm(mpg ~ disp, data = train)
new.data <- data.frame(disp = test$disp)
test$output <- predict(model2, new.data)
sqrt(sum(test$mpg - test$output)^2/nrow(test))
```

```
[1] 1.56792
```

First, we calculate a new linear model on the training data using lm(). Next, we form a data frame from our test data's disp column. After that, we make predictions on our test set and store that in a new column in our test data. Finally, we compute a root-mean-square error (RMSE) term. We do this by taking the difference between our model output and the known mpg efficiency, squaring it, summing up those squares, and dividing by the total number of entries in the dataset. This gives us the value for the residual standard error. The new value is different from what we've seen before and is an important value for understanding how well our model is performing.

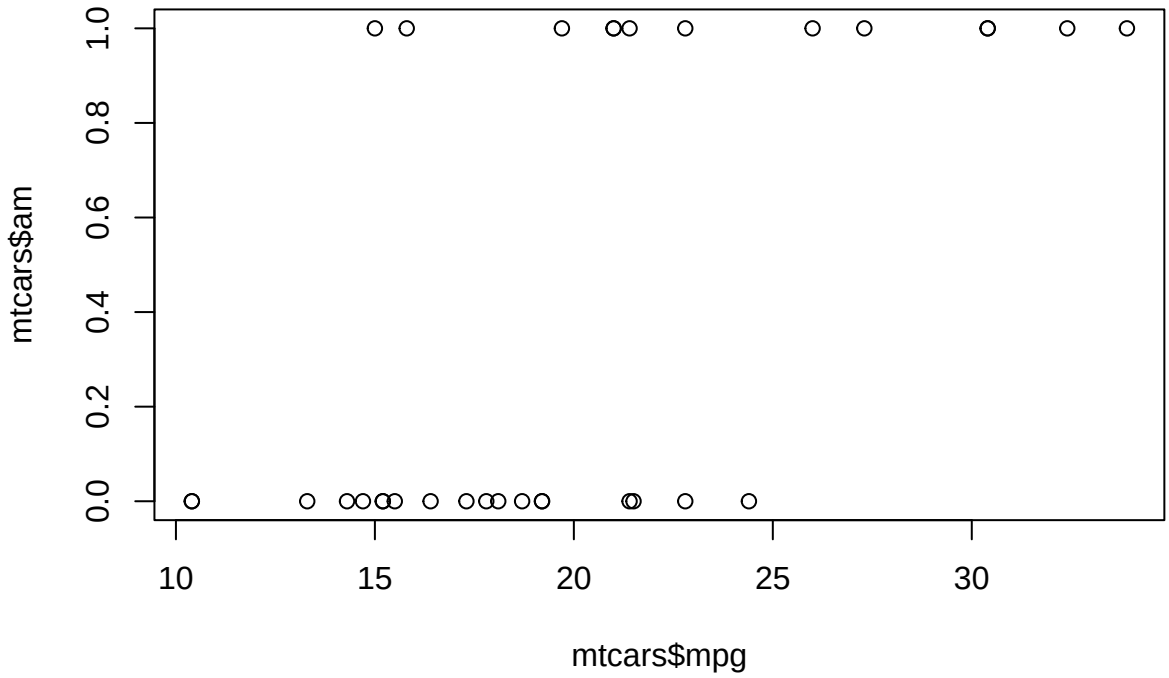
2.2 Classification

In contrast to regression modeling, which you have likely previously done without realizing it, classification is a less frequently encountered part of the machine learning spectrum. Instead of predicting continuous values, like numbers, in classification exercises we'll predict discrete values.

2.2.1 Logistic Regression

In contrast to regression, sometimes you want to see if a given data point is of a categorical nature instead of numeric. Before, we were given a numeric input and calculated a numeric output through a simple regression formula. Figure below presents the same mtcars data set to visually explain the difference:

```
plot(x = mtcars$mpg, y = mtcars$am)
```



The data looks very different compared to what we saw earlier. In the mtcars dataset, each car is given a 0 or a 1 label to determine whether it has an automatic transmission as defined by the column name am. A car with an automatic has a value 1, whereas a manual transmission car has a value of 0. Fitting a linear regression model to this data would not work, because we cannot have half a transmission value. Instead, we need to rely on a logistic regression model to help classify whether new efficiency data belongs to either the automatic or manual transmission groups. We have a slightly different question to answer this time: how is the fuel efficiency related to a car's transmission type? We can't rely on the regression modeling procedure here, unfortunately. We could try to fit a regression line to the data, but the results would be very misleading. Instead, we need to use a classification algorithm. In this case, we will use a logistic regression algorithm. Logistic regression is different than linear regression in that we get discrete outputs instead of continuous ones. Before, we could get any number as a result of our regression model, but with our logistic model, we should expect a binary outcome for the transmission type; it either is an automatic transmission, or it isn't. The approach here is different, as well. First, you need to load the caTools library: This library contains many functions, but, most important, it has a function for logistic regression: LogitBoost. First, you need to give the model the label against which we want to predict as well as the data that you want to use for training the model:

```
library(caTools)
label.train <- train[,9]
Data.train <- train[,-9]
```

You can read the syntax of train[,-9] as follows: "The data we want is the mtcars dataset that we split into a training set earlier, except column number 9." That happens to be the am column we used earlier. This is a more compact way of subsetting the data instead of listing out each column individually for input:

```
model <- LogitBoost(Data.train, label.train)
Data.test = test
Lab = predict(model, Data.test, type = "raw")
data.frame(row.names(test), test$mpg, test$am, Lab)
```

```
      row.names.test. test.mpg test.am        X0           X1
1       Mazda RX4 Wag     21.0       1 0.9996646 3.353501e-04
2             Valiant     18.1       0 0.9999546 4.539787e-05
3          Merc 450SE     16.4       0 0.9996646 3.353501e-04
4          Merc 450SL     17.3       0 0.9996646 3.353501e-04
5 Lincoln Continental     10.4       0 0.9996646 3.353501e-04
6       Toyota Corona     21.5       0 0.9820138 1.798621e-02
7    Pontiac Firebird     19.2       0 0.9996646 3.353501e-04
```

In Python it is done as follows;

```
import statsmodels.formula.api as smf
import statsmodels.graphics.api as smg
import pandas as pd
import numpy as np
import random

data = pd.read_csv("E:\Desktop\Data Sets\mtcars.csv")
data = pd.DataFrame(data)
train = data.sample(n = 25, random_state = 123)
test = data.drop(train.index)
test = pd.DataFrame(test)
model2 = smf.logit("am ~ mpg", train)
results2 = model2.fit()
```

```
Optimization terminated successfully.
         Current function value: 0.294181
         Iterations 8
```

```
results2.summary()
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                           Logit Regression Results
==============================================================================
Dep. Variable:                     am   No. Observations:                   25
Model:                          Logit   Df Residuals:                       23
Method:                           MLE   Df Model:                            1
Date:                Sun, 23 Apr 2023   Pseudo R-squ.:                  0.5307
Time:                        16:48:05   Log-Likelihood:                -7.3545
converged:                       True   LL-Null:                       -15.672
Covariance Type:            nonrobust   LLR p-value:                 4.532e-05
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    -12.4581      5.204     -2.394      0.017     -22.658      -2.259
mpg            0.5452      0.239      2.280      0.023       0.076       1.014
```

```
==============================================================================
"""
```

```
pred = results2.predict(test)
test = test.loc[:,['mpg']]
test["Predicted"] = pred
test
```

```
     mpg  Predicted
2   22.8   0.493272
6   14.3   0.009365
13  15.2   0.015208
17  32.4   0.994553
28  15.8   0.020969
29  19.7   0.152243
30  15.0   0.013658
```

2.3 Supervised Clustering Methods

Clustering is when you have a set of data and want to define classes based on how closely they are grouped. Sometimes, groupings of data might not be immediately obvious, and a clustering algorithm can help you find patterns where they might otherwise be difficult to see explicitly. Clustering is a good example of an ecosystem of algorithms that can be used both in a supervised and unsupervised case. It's one of the most popular forms of classification, and one of the most popular clustering models is the kmeans algorithm. Cluster analysis is the grouping of objects such that objects in the same cluster are more similar to each other than they are to objects in another cluster. The classification into clusters is done using criteria such as smallest distances, density of data points, graphs, or various statistical distributions. Let's examine the iris dataset by looking at the plot of petal width as a function of petal length

```
plot(x = iris$Petal.Length, y = iris$Petal.Width, xlab = "Petal Length",
ylab = "Petal Width")
```

What if we wanted to try to find three distinct groups in which to classify this dataset? The human brain is remarkably good at finding patterns and structure, so the clumping of data in the lower-left corner of Figure 2-3 stands out as one obvious cluster of data. But what about the rest? How do we go about breaking the data in the upper-right part of the plot into two more groups? One clustering algorithm that can accomplish this is the kmeans() approach to clustering. This algorithm works by first placing a number of random test points in our data

This algorithm works by first placing a number of random test points in our data —in this case, two. Each of our real data points is measured as a distance from these test points, and then the test points are moved in a way to minimize that distance, as shown in Figure

```
data = data.frame(iris$Petal.Length, iris$Sepal.Width)

iris.kmeans = kmeans(data, 2)

plot(x = iris$Petal.Length, y = iris$Petal.Width, pch = iris.kmeans$cluster)
```

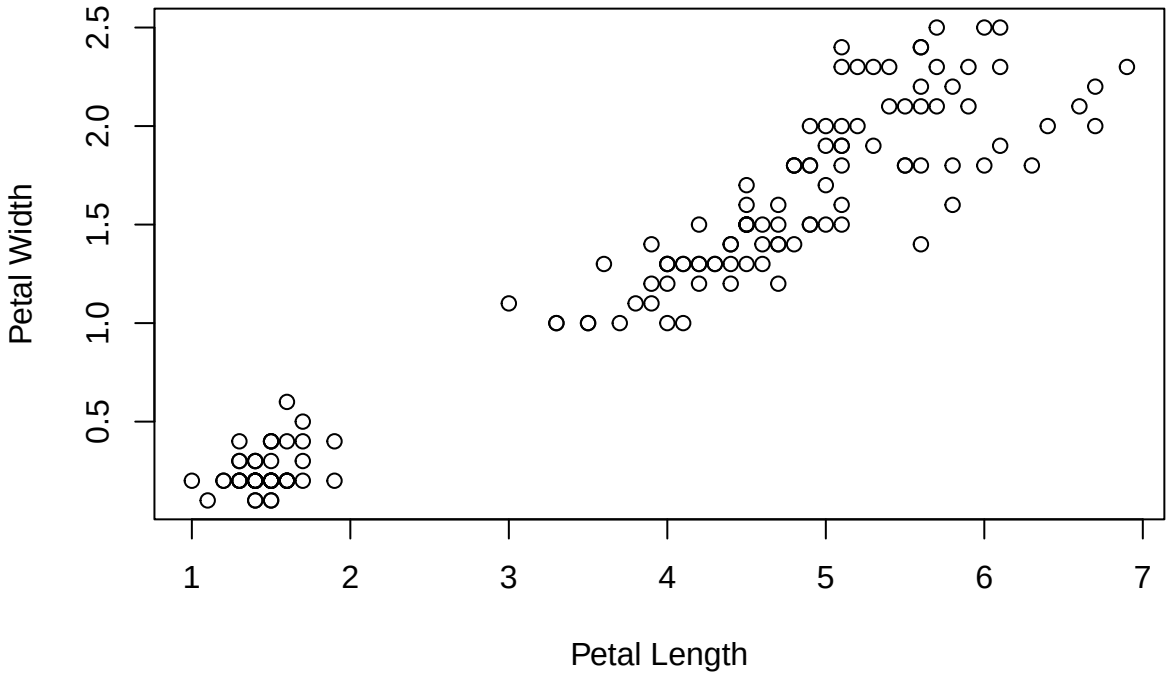
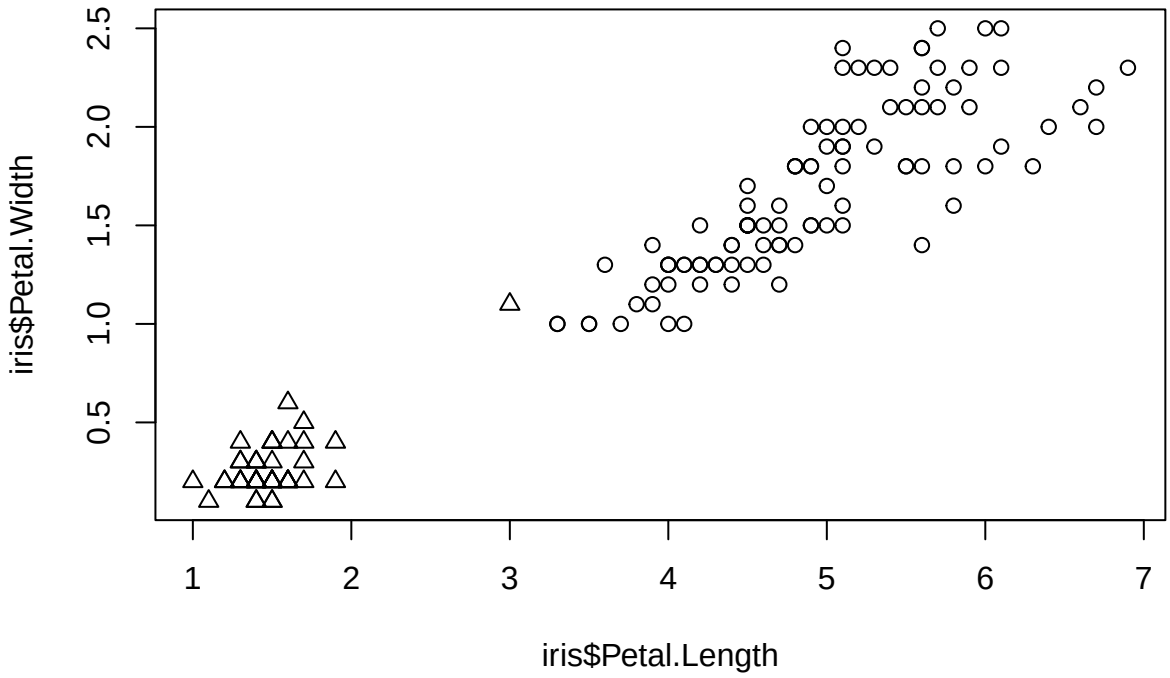
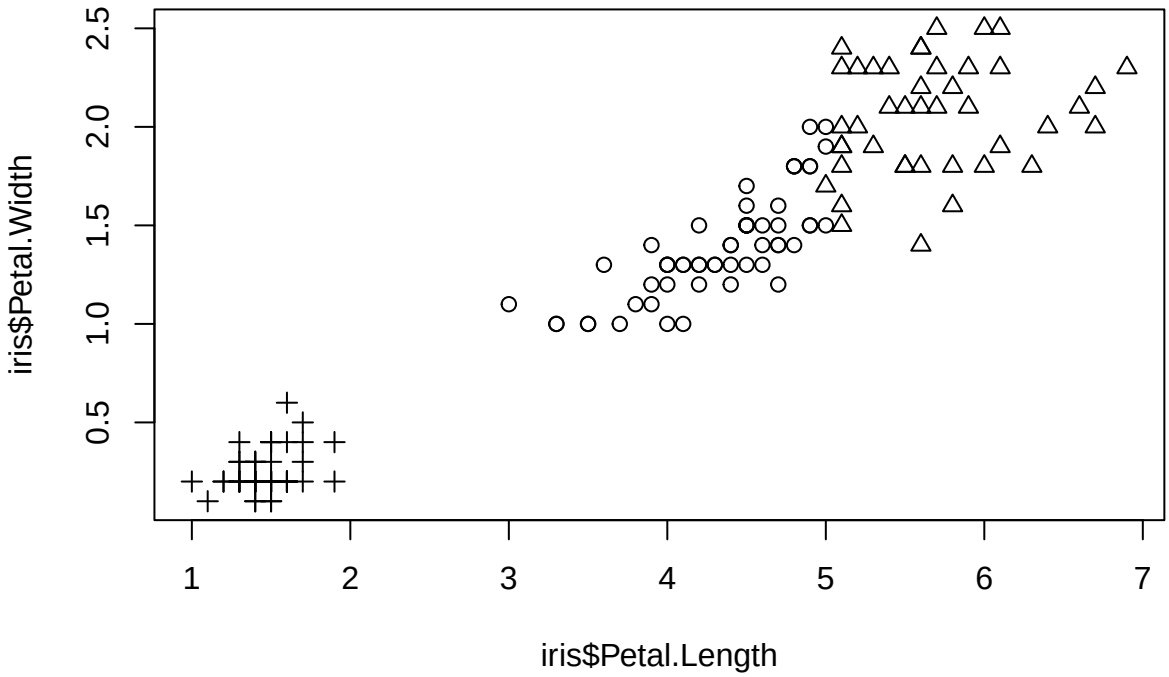


Figure 1: Plot of petal width



We can proceed and;

```
iris.kmeans2 <- kmeans(data,3)

plot(x = iris$Petal.Length, y = iris$Petal.Width, pch = iris.kmeans2$cluster)
```



Data scientists and others use clustering to gain important insights from data by observing what groups (or clusters) the data points fall into when they apply a clustering algorithm to the data. By definition, unsupervised learning is a type of machine learning that searches for patterns in a data set with no pre-existing labels and a minimum of human intervention. Clustering can also be used for anomaly detection to find data points that are not part of any cluster, or outliers.

Clustering is used to identify groups of similar objects in datasets with two or more variable quantities. In practice, this data may be collected from marketing, biomedical, or geospatial databases, among many other places. Ideally, a clustering algorithm creates clusters where intra-cluster similarity is very high, meaning the data inside the cluster is very similar to one another. Also, the algorithm should create clusters where the inter-cluster similarity is much less, meaning each cluster contains information that's as dissimilar to other clusters as possible. An algorithm built and designed for a specific type of cluster model will usually fail when set to work on a data set containing a very different kind of cluster model.

2.4 Mixed Methods

2.4.1 Tree - Based Models

So far, we've seen a linear regression and logistic regression example. Part of the universe of machine learning models includes tree-based methods. Simply put, a tree is a structure that has nodes and edges. For a decision tree, at each node we might have a value against which we split in order to gain some insight from the data. A decision tree is a type of supervised machine learning used to categorize or make predictions based on how a previous set of questions were answered. The model is a form of supervised learning, meaning that the model is trained and tested on a set of data that contains the desired categorization. Before we dive into how a decision tree works External link:open_in_new, let's define some key terms of a decision tree.

- Root node: The base of the decision tree.
- Splitting: The process of dividing a node into multiple sub-nodes.
- Decision node: When a sub-node is further split into additional sub-nodes.
- Leaf node: When a sub-node does not further split into additional sub-nodes; represents possible outcomes.
- Pruning: The process of removing sub-nodes of a decision tree.
- Branch: A subsection of the decision tree consisting of multiple nodes. While you are a consistent golfer, your score is dependent on a few sets of input variables. Wind speed, cloud cover and temperature all play a role. In addition, your score tends to deviate depending on whether or not you walk or ride a cart. And it deviates if you are golfing with friends or strangers.

In this example, there are two leaf nodes: below par or over par. Each of the input variables will determine decision nodes. Was it windy? Cold? Did you golf with friends? Did you walk or take a cart? With enough data on your golfing habits (and assuming you are a consistent golfer), a decision tree could help predict how you will do on the course on any given day.

In the golf example, each outcome is independent in that it does not depend on what happened in the previous coin toss. Dependent variables, on the other hand, are those that are influenced by events before them.

Building a decision tree involves construction, in which you select the attributes and conditions that will produce the tree. Then, the tree is pruned to remove irrelevant branches that could inhibit accuracy. Pruning involves spotting outliers, data points far outside the norm, that could throw off the calculations by giving too much weight to rare occurrences in the data.

Maybe temperature is not important when it comes to your golf score or there was a day when you scored really poorly that's throwing off your decision tree. As you're exploring the data for your decision tree, you can prune specific outliers like your one bad day on the course. You can also prune entire decision nodes, like temperature, that may be irrelevant to classifying your data.

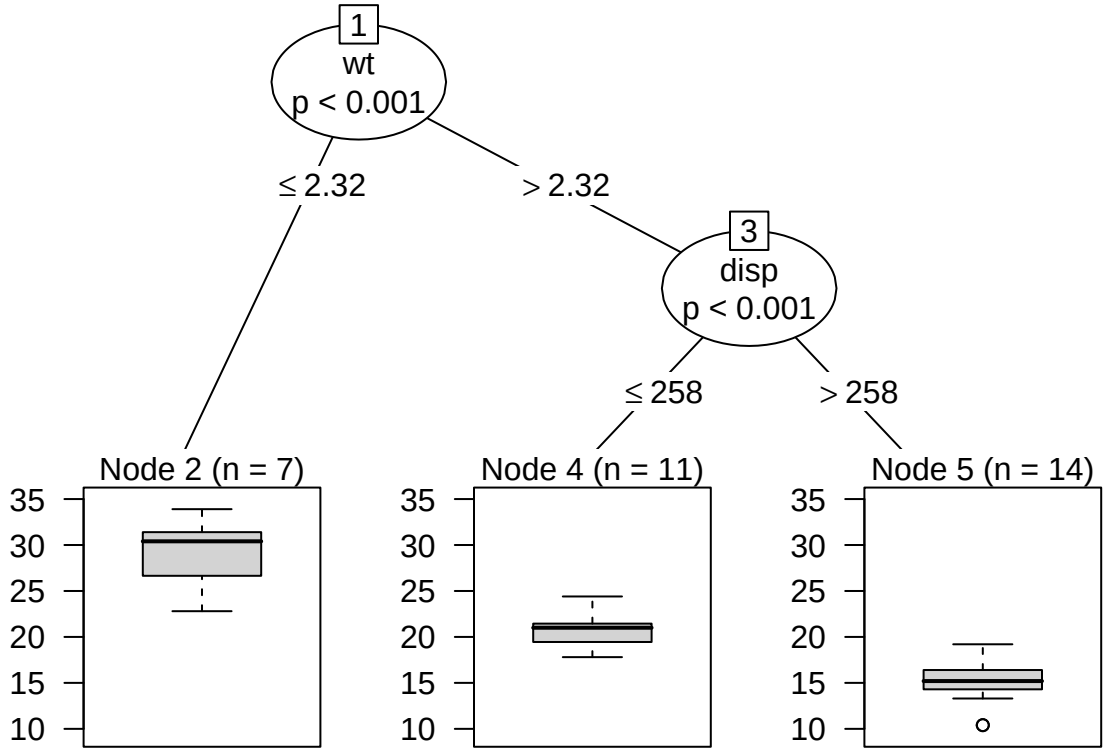
Well-designed decision trees present data with few nodes and branches. You can draw a simple decision tree by hand on a piece of paper or a whiteboard. More complex problems, however, require the use of decision tree software.

This is best explained visually by looking at Figure

```
library(party)
tree <- ctree(mpg ~ ., data = mtcars)
tree
```

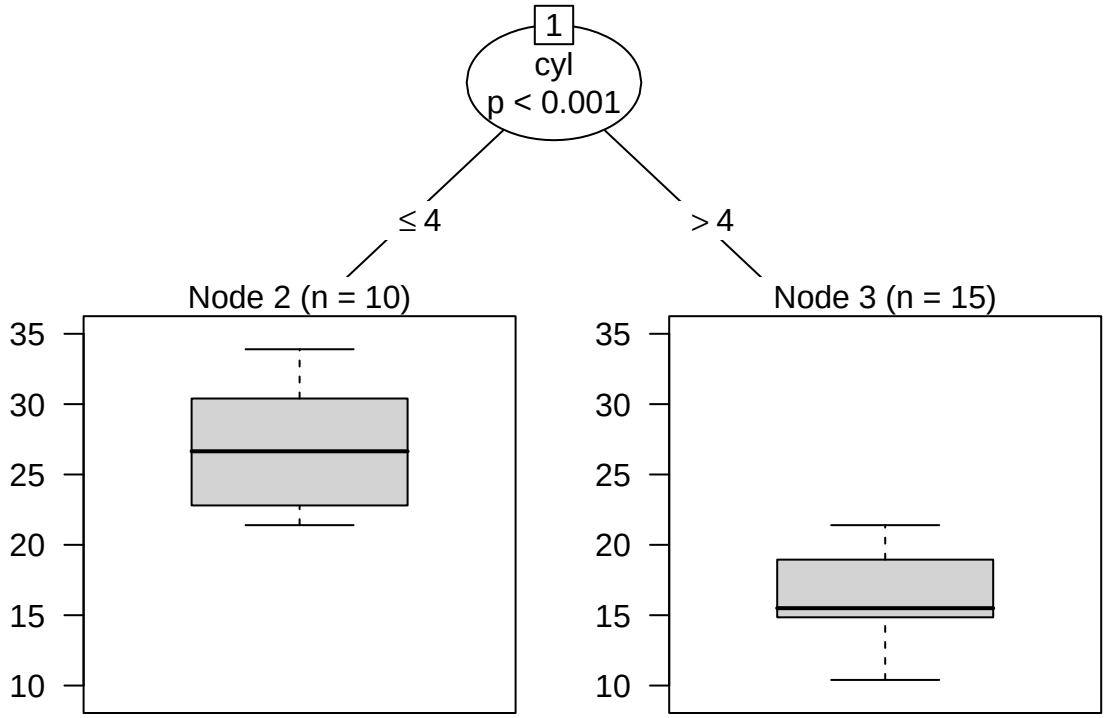
```
     Conditional inference tree with 3 terminal nodes

Response:  mpg
Inputs:  cyl, disp, hp, drat, wt, qsec, vs, am, gear, carb
Number of observations:  32

1) wt <= 2.32; criterion = 1, statistic = 23.338
  2)*  weights = 7
1) wt > 2.32
  3) disp <= 258; criterion = 0.999, statistic = 16.217
    4)*  weights = 11
  3) disp > 258
    5)*  weights = 14
```

```
plot(tree)
```



The figure above demonstrates a plotted **conditional inference tree** (This is also known as regression tree). We are plotting engine fuel efficiency (mpg), but we're using all features in the dataset to build the model instead of just one; hence, the mpg ~ . call in the ctree() function. The output is a distribution (in the form of a box-and-whisker plot) of the fuel efficency as a function of the major features that influence it. The ctree function calls on certain methods to figure these out; this way, you don't have a bunch of branches in the tree that don't amount to anything other than to clog up the view. In this case, the features that are most important to mpg are disp (the engine displacement) and wt (the car's weight). You read this chart from top to bottom. At node 1, there is a split for cars that weigh less than 2.32 tons and those that weigh more. For the cars that weigh more, we split further on the engine displacement. For engine displacements that are less than 258 cubic inches in volume, we go to node 4. For engine displacements that have more than 258 cubic inches, we go to node 5. Notice that for each feature there is a statistical pvalue, which determines how statistically relevant it is. The closer the p-value is to 0.05 or greater, the less useful or relevant it is. In this case, a p-value of almost exactly 0 is very good. Likewise, you can see how many data points make up each class at the bottom of the tree. Let's consider a car that has a weight of four tons, and a small engine size of 100 cubic inches. At node 1, we go along the righthand path to node 3 (because the weight is greater than 2.32 tons) and then go left to node 4 based on the theoretical data we just made up. We should expect the fuel efficiency of this car to be somewhere between 13 and 25 miles per gallon. What if you try to use this new data structure for prediction? The first thing that should pop up is that you are looking at the entire dataset instead of just the training data. Figure 2-8 shows the tree structure for the training data first:

```
tree.train <- ctree(mpg ~ ., train)|>plot()
```



By looking at just the training data, you have a slightly different picture in that the tree depends only on the car's weight. In the following example, there are only two classes instead of the tree as before:

```
library(party)
tree.train <- ctree(mpg ~ ., data = train)
test$mpg.tree <- predict(tree.train, test)
test$class <- predict(tree.train, test, type = "node")
data.frame(row.names(test), test$mpg, test$mpg.tree, test$class)
```

```
      row.names.test. test.mpg   mpg test.class
1       Mazda RX4 Wag     21.0 16.48          3
2             Valiant     18.1 16.48          3
3          Merc 450SE     16.4 16.48          3
4          Merc 450SL     17.3 16.48          3
5 Lincoln Continental     10.4 16.48          3
6       Toyota Corona     21.5 27.18          2
7    Pontiac Firebird     19.2 16.48          3
```

This chunk of code does both a regression and a classification test in two easy lines of code. First, it takes the familiar predict() function and applies it to the entirety of the test data and then stores it as a column in the test data. Then, it performs the same procedure, but adds the type="node" option to the predict() function to get a class out. It then sticks them all together in a single data frame. What you can see from the end result is that it doesn't take a lot of work for some algorithms to provide both a continuous, numeric output (regression) as well as a discrete class output (classification) for the same input data.

2.4.2 Types of Decision Trees

There are two main types of decision trees. The divisions are based on the type of outcome variables used.

2.4.2.1 Categorical Variable Decision Tree In a categorical variable decision tree, the answer neatly fits into one category or another. Was the coin toss heads or tails? Is the animal a reptile or mammal? In this type of decision tree, data is placed into a single category based on the decisions at the nodes throughout the tree.

2.4.2.2 Continuous Variable Decision Tree A continuous variable decision tree is one where there is not a simple yes or no answer. It's also known as a regression tree because the decision or outcome variable depends on other decisions farther up the tree or the type of choice involved in the decision.

The benefit of a continuous variable decision tree is that the outcome can be predicted based on multiple variables rather than on a single variable as in a categorical variable decision tree. Continuous variable decision trees are used to create predictions. The system can be used for both linear and non-linear relationships if the correct algorithm is selected.

The application of decision trees are as follows;

1. Customer Recommendation Engines

Customers buying certain products or categories of products might be inclined to buy something similar to what they're looking for. That's where recommendation engines come in. They can be used to push the sale of snow gloves with a purchase of skis or recommending another holiday movie after you've just finished watching one.

Recommendation engines can be structured using decision trees, taking the decisions made by consumers over time and creating nodes based off of those decisions.

2. Identifying Risk Factors for Depression

A study conducted in 2009 in Australia tracked a cohort of over 6,000 people and whether or not they had a major depressive disorder over a four-year period. The researchers took inputs like tobacco use, alcohol use, employment status and more to create a decision tree that could be used to predict the risk of a major depressive disorder.

Medical decisions and diagnoses rely on multiple inputs to understand a patient and what may be the best way to treat them. A decision tree application like this can be a valuable tool to health care providers when assessing patients.

It is worthy to note that the decision tree does not take or support categorical vaiables as features.[1]

2.5 Working of a Decision Tree

The root node feature(variable) is selected based on the results from the **Attribute Selection Measure (ASM)** The ASM is repated until a leaf node or a terminal node cannot be split into sub-nodes.

The ASM is a technique used in data mining process for data reduction. Data reduction is necessary to make better analysis and prediction of the target variable. There are two main techniques of ASM

1. Gini index or Gini impurity

The Gini index is a measure of the impurity of a set of examples or data points. In the context of decision tree algorithms, the Gini index is used to select the best split or root node. The Gini index measures the

[1] A decision tree is a non-parametric supervised learning algorithm, which is utilized for both classification and regression tasks. It has a hierarchical, tree structure, which consists of a root node, branches, internal nodes and leaf nodes.

probability of a randomly chosen example from a set being incorrectly classified according to the distribution of the classes in the set.. It is given as;

$$Gini = \sum_{i=1}^{n} \pi_i$$

- where π_i is the probability of an object being classified into a particular group

To choose the best root node, we calculate the Gini index for each feature or variable in the dataset. The feature that results in the lowest Gini index is selected as the best split. This means that the feature that results in the most pure splits (i.e., the splits with the least impurity) is chosen as the root node.

Suppose we have the following data set and we want to determine which feature is to be used as the root node;

Age	Height	Weight	Obesity
25	170	80	Yes
30	175	85	Yes
40	180	90	Yes
50	165	70	No
55	185	95	Yes
60	190	100	Yes
35	172	75	No
45	177	88	Yes
28	168	72	No
42	182	92	Yes

We begin with the feature age, where, age<= 40 is;

Age	Height	Weight	Obesity
25	170	80	Yes
30	175	85	Yes
35	172	75	No
28	168	72	No

and age $>$ 40 is;

Age	Height	Weight	Obesity
40	180	90	Yes
50	165	70	No
55	185	95	Yes
60	190	100	Yes
45	177	88	Yes
42	182	92	Yes

The gini index for age <= 40 will be given by;

$$Gini = 1 - (\mathbb{P}(Obesity = Yes)^2 + \mathbb{P}(Obesity = No)^2)$$

Thus;

$$Gini1 - (0.5^2 + 0.5^2 = 0.5$$

Then for age > 40 will be;

$$Gini1 - (0.833^2 + 0.167^2 = 0.278$$

Then the weighted Gini index for the feature age is;

Weighted Average Gini Index for Age = (4/10 * 0.5) + (6/10 * 0.278) = 0.354

The same is repeated for all the variables.

When using the Gini index as the criterion for the algortihm to choose the feature for the root node, the feature with the least gini index is choosen.

2. Information Gain (ID3) or Entropy

Entropy is the main concept of this algorithm which helps to determine which feature or attribute gives the maximum information about a class. By using this method, we can reduce the level of entropy form the root node to the leaf node. The mathematical formula is;

$$E(S) = \sum_{i=1}^{n} -\pi_i \log_2 \pi_i$$

π_i, denotes the probability of E(S), which denotes the entropy. The feature or attribute with the highest ID3 gain is used as the root for the splitting.

Information Gain is another metric used to evaluate the quality of a split in decision trees. It measures the reduction in entropy[2] (or the increase in information) in the target variable due to the split on a particular feature.

Here's how Information Gain is calculated:

First, we calculate the entropy of the target variable before the split. Entropy is a measure of the randomness or unpredictability of the target variable's values. It is calculated as:

$$Entropy(S) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

Where:

S is the data set or subset of examples n is the number of classes in the data set p_i is the proportion of examples in class i in the data set $\log_2$ is the base-2 logarithm.[3]

In the exmple above, there are 6 "Yes" and 4 "No" values for the target variable "Obesity", so the entropy before the split is:

$$Entropy(S) = -\left(\frac{6}{10}\log_2\left(\frac{6}{10}\right)\right) - \left(\frac{4}{10}\log_2\left(\frac{4}{10}\right)\right) = 0.971$$

Next, we calculate the entropy of the target variable after the split based on a particular feature. We calculate this for each possible split based on the feature and then take a weighted average of the entropies based on the proportion of observations in each split.

[2]Entropy is a measure of the degree of randomness or uncertainty in a system

[3]p_i is the proportion of the target variable's values that are equal to i (i.e., the probability of i occurring). In the case of binary classification (e.g., "Yes" or "No" for obesity), there are two possible values for the target variable, so n = 2.

For example, if we split the dataset based on the "Age" feature as in your previous question, we get two subsets: one with 4 observations where Age <= 40 and another with 6 observations where Age > 40. We calculate the entropy of the target variable for each of these subsets as follows:

Subset 1 (Age <= 40): Entropy(S1) = -(2/4 * log2(2/4)) - (2/4 * log2(2/4)) = 1.0 Subset 2 (Age > 40): Entropy(S2) = -(3/6 * log2(3/6)) - (3/6 * log2(3/6)) = 1.0 Then, we take the weighted average of these entropies based on the proportion of observations in each subset:

Entropy(Age) = (4/10 * Entropy(S1)) + (6/10 * Entropy(S2)) = 1.0

Finally, we calculate the Information Gain as the reduction in entropy before and after the split based on a particular feature:

Information Gain(Age) = Entropy(S) - Entropy(Age)

In the example you provided, the entropy before the split is 0.971 and the entropy after the split based on "Age" is 1.0, so the Information Gain is:

Information Gain(Age) = 0.971 - 1.0 = -0.029 [4]

A higher Information Gain value indicates a better split because it results in a larger reduction in entropy and thus a greater increase in information. So in this case, since the Information Gain for "Age" is negative, it's not a very good feature to use for the split. We calculate the Information gain for each variable in the data set and the one which yields to the highest information gain is used as the root node.

This happens at each step. That is after choosing the root node, the algorithm further calculates the Gini Index and Information Gain so as to determine how to split the child node. The algorithm yields to a leaf node [5] when;

1. All examples at the node belong to the same class, in which case the node is assigned that class as its prediction. (for example If all the examples at a node belong to the same class (in this case, all are "Yes" for obesity), then the node is assigned that class as its prediction, and the algorithm stops splitting the tree further.)
2. The number of examples reaching the node falls below a pre-defined threshold or minimum number of examples required to split a node. This threshold is typically set based on the size of the dataset and the complexity of the decision tree.
3. The depth of the tree exceeds a predefined maximum depth.
4. The number of examples at the node falls below a predefined minimum number. In the context of decision trees, "all examples" refer to the set of data points that reach a particular node during the construction of the tree.

For example, consider the decision tree constructed for the given example of age, height, weight, and obesity. Suppose we have split the root node based on the age attribute, resulting in two child nodes: age <= 40 and age > 40.

Suppose further that we apply the same process of calculating the information gain or Gini index to split each child node further. For example, we might split the node age <= 40 based on the weight attribute, resulting in two child nodes: weight <= 75 and weight > 75.

At this point, "all examples" that reach the child node weight <= 75 refer to the subset of data points in our original dataset that satisfy the conditions age <= 40 and weight <= 75. Similarly, "all examples" that reach the child node weight > 75 refer to the subset of data points that satisfy the conditions age <= 40 and weight > 75.

[4] A positive information gain means that the split at the node is reducing the uncertainty or randomness in the target variable. In other words, it is helping to separate the classes in a more distinct way.

[5] A leaf node is a final node in a decision tree that represents a decision or a classification.

The decision tree construction algorithm continues to split each node in this manner, based on the best attribute and threshold values that maximize the information gain or minimize the Gini index. The process continues recursively until the stopping criteria are met and a leaf node is created.

Consider a problem where we wish to predict the loan eligibility process from the given data(this data is found in kaggle). I used the rpart R package that implements CART (classification and regression trees). CART is implemented in many programming languages, including Python. Arguably, CART is a pretty old and somewhat outdated algorithm and there are some interesting new algorithms for fitting trees.

```
# Importing the data

library(readxl)
library(dplyr)
library(tidyverse)
Loandata <- read_excel("E:/Desktop/Machine Learning/loan-data.xlsx")

Loandata <- Loandata |>
  drop_na()
```

A glimpse of the data is as follows;

```
head(Loandata)
```

```
# A tibble: 6 x 13
  Loan_ID  Gender Married Dependents Education    Self_Employed ApplicantIncome
  <chr>    <chr>  <chr>   <chr>      <chr>        <chr>                  <dbl>
1 LP001003 Male   Yes     1          Graduate     No                      4583
2 LP001005 Male   Yes     0          Graduate     Yes                     3000
3 LP001006 Male   Yes     0          Not Graduate No                      2583
4 LP001008 Male   No      0          Graduate     No                      6000
5 LP001011 Male   Yes     2          Graduate     Yes                     5417
6 LP001013 Male   Yes     0          Not Graduate No                      2333
# i 6 more variables: CoapplicantIncome <dbl>, LoanAmount <dbl>,
#   Loan_Amount_Term <dbl>, Credit_History <dbl>, Property_Area <chr>,
#   Loan_Status <chr>
```

We are going to use the features, Gender, LoanAmount, Educational level, employment and Married, thus:

```
Loandata <- Loandata[,c(2,3,5,6,9,13)]
Loandata
```

```
# A tibble: 480 x 6
   Gender Married Education    Self_Employed LoanAmount Loan_Status
   <chr>  <chr>   <chr>        <chr>              <dbl> <chr>
 1 Male   Yes     Graduate     No                   128 N
 2 Male   Yes     Graduate     Yes                   66 Y
 3 Male   Yes     Not Graduate No                   120 Y
 4 Male   No      Graduate     No                   141 Y
 5 Male   Yes     Graduate     Yes                  267 Y
 6 Male   Yes     Not Graduate No                    95 Y
 7 Male   Yes     Graduate     No                   158 N
 8 Male   Yes     Graduate     No                   168 Y
 9 Male   Yes     Graduate     No                   349 N
10 Male   Yes     Graduate     No                    70 Y
# i 470 more rows
```

Let us now use coding;

```
Loandata <- Loandata|>
  mutate(Gender = ifelse(Gender == "Male", 1, 0))|>
  mutate(Married = ifelse(Married == "Yes", 1,0))|>
  mutate(Education = ifelse(Education == "Graduate", 1,0))|>
  mutate(Self_Employed = ifelse(Self_Employed == "Yes",1,0))|>
  mutate(Loan_Status = ifelse(Loan_Status == "Y",1,0))
head(Loandata)
```

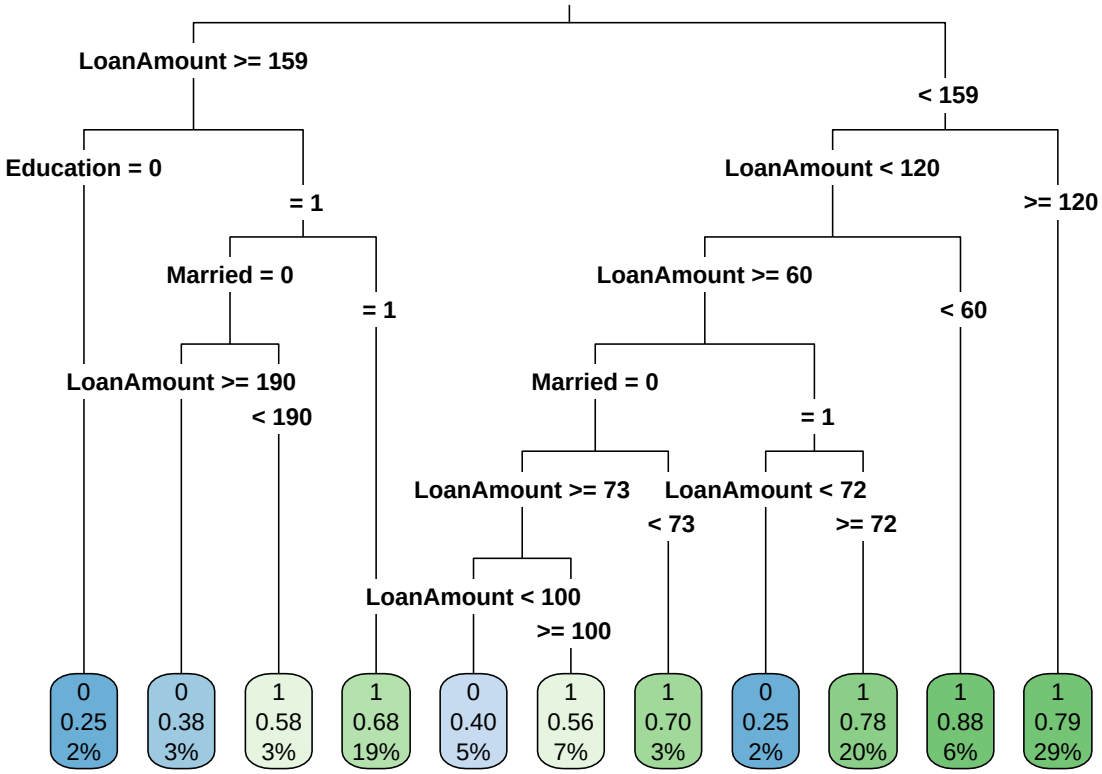
```
# A tibble: 6 x 6
  Gender Married Education Self_Employed LoanAmount Loan_Status
   <dbl>   <dbl>     <dbl>         <dbl>      <dbl>       <dbl>
1      1       1         1             0        128           0
2      1       1         1             1         66           1
3      1       1         0             0        120           1
4      1       0         1             0        141           1
5      1       1         1             1        267           1
6      1       1         0             0         95           1
```

We can now therefore proceed to split the data into training and testing data as follows

```
split_size2 <- .8
sample_size2 <- floor(split_size2 * nrow(Loandata))

set.seed(1234)

traindata_indicies2 <- sample(seq_len(nrow(Loandata)), size = sample_size2)

traindata <- Loandata[traindata_indicies2,]
testdata <- Loandata[-traindata_indicies2,]
```

Thus the decision tree is done as follows;

```
library(rpart);library(rpart.plot)
decesion.tree <- rpart(Loan_Status ~ ., data = traindata, method = 'class')
rpart.plot(decesion.tree, type = 3)
```



6

[6]Remember there are two types of Decision tree, the classification and Regression tree (CART), and the Regression tree