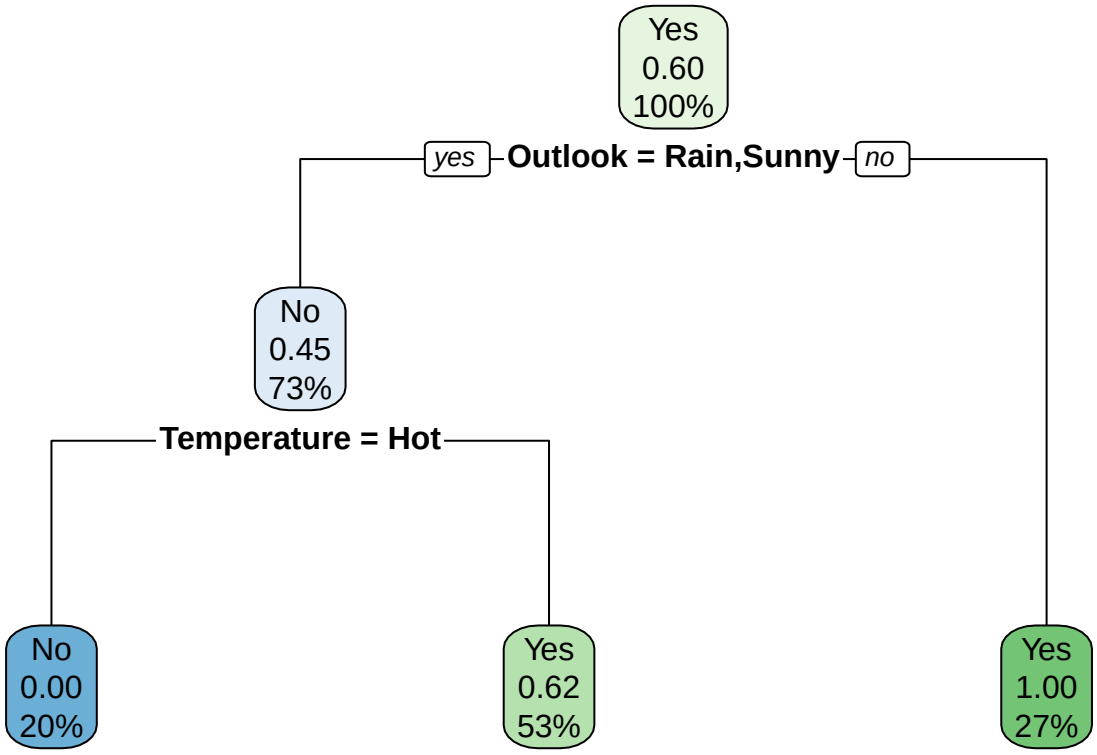
Report_Part1

2024-11-15

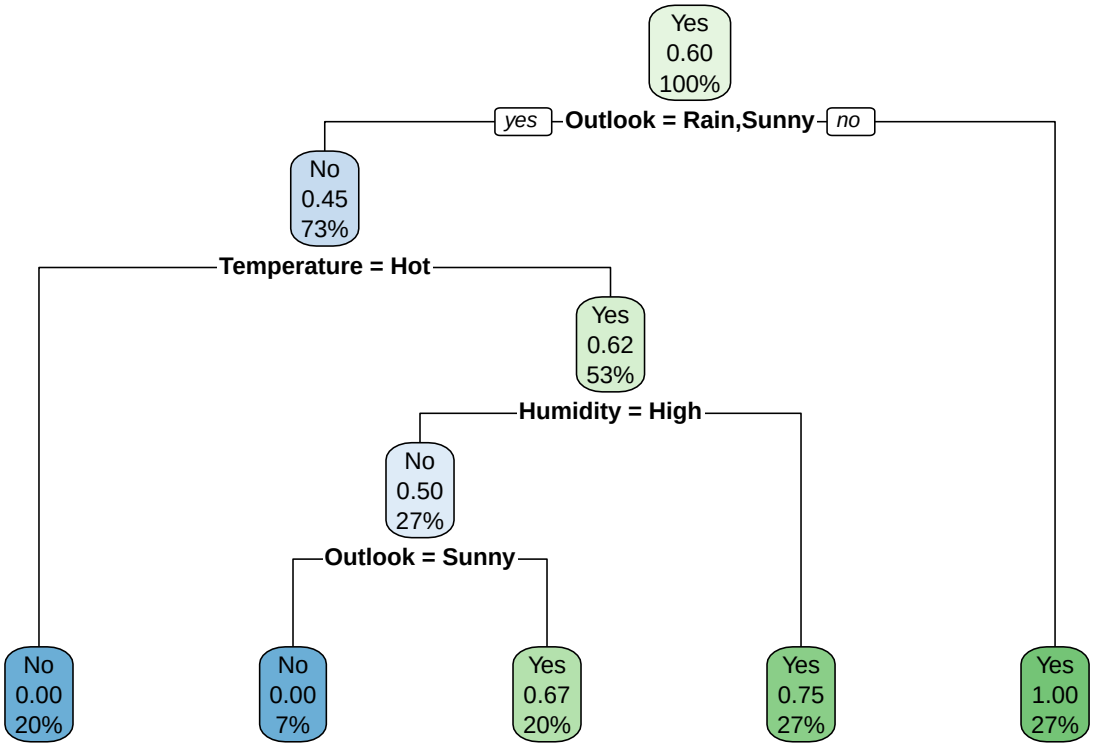
2 and 3

The first is to create a function that only changes the max depth of the decision tree. This will allow to assess the effects of changing this parameter. The first tree is the original with depth 2 and we will be checking for 4, 5, 6 and 10.

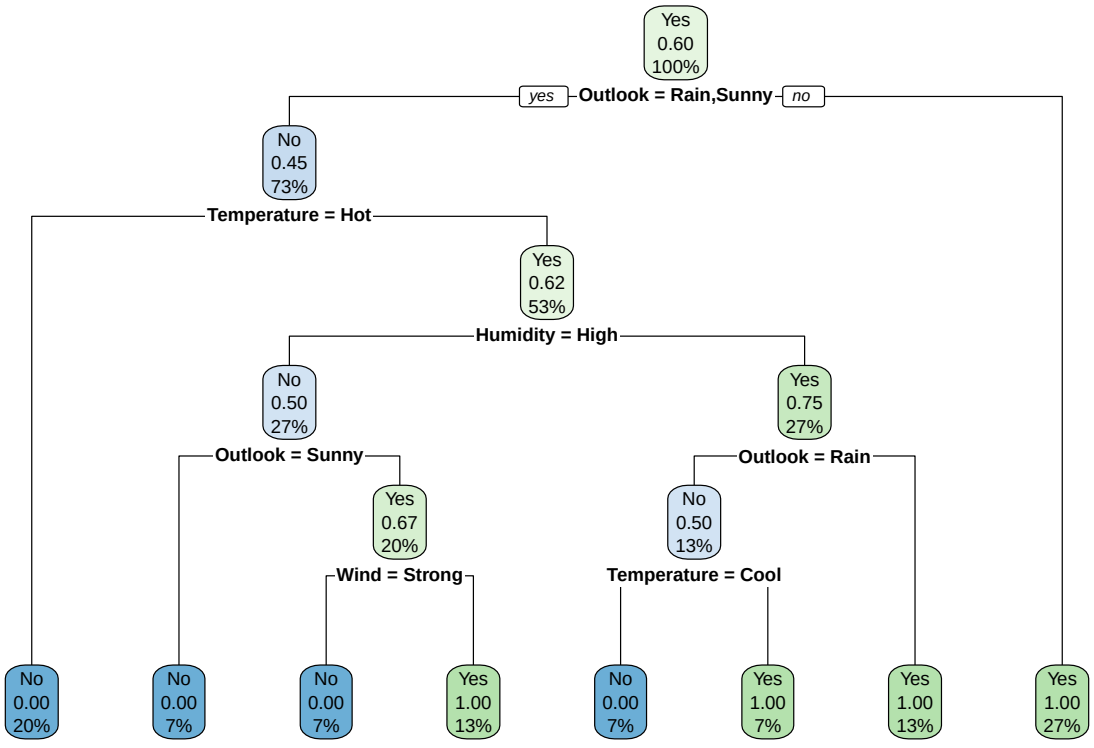
```
options(width = 80)
change_depth = function(maxde){
  model <- rpart(PlayTennis ~ ., play_tennis, control = rpart.control(cp = 0, maxdepth = maxde, minsplit
                                                                      minbucket = 1))
  return (rpart.plot(model))
}
change_depth(2)
```




```
change_depth(4)
```




```
change_depth(5)
```




```
change_depth(6)
change_depth(10)
```

As you can see when you have a smaller value for max_depth, the tree is simpler. As you increase the value of max_depth, the tree becomes more complex, eventually reaching a point where the nodes perfectly separate the categories (i.e. the categories become 100% pure).For this tree the complexity is maximized at max_depth = 5. Meaning that you get the same tree when you use any value greater than or equal to 5.

In general, having a perfect division in the data can lead to problems like overfitting, and this is why it is typical to prune the trees.

4 and 5

```
gini_impurity_R <- function(left,right){
  len_right= length(right) #length of the right
  len_left = length(left) #length of the right
  n_rows = len_left+len_right #length of the total rows
  gin_left = 1 - ((sum(left=='Yes')/len_left)^2+(sum(left=='No')/len_left)^2) #gini calculation for the
  gin_right = 1 - ((sum(right=='Yes')/len_right)^2+(sum(right=='No')/len_right)^2) #gini calculation fo
  weight_gini = len_left/n_rows * gin_left + len_right/n_rows*gin_right #weighted
  return(weight_gini)
}

best_split_R <- function(X, y) { #X is your predictors and y is the response variable
  best_gini <- 1.0 # the worst possible gini will be 1 so you leave it as a max possible value
  best_feature <- -1 #feature auxiliar only to be change for another one.
  n_features <- ncol(X) #number of columns to check

  for (feature in 1:n_features) { # runs through all the columns
    values <- unique(X[, feature]) #gives all the unique values in the column i(feature) if you have ca

    # The for value in values divide the X in 2. The ones that are equal to the i value in values
    # and the ones that are different from the value.
    for (value in values) {
      left_indices <- X[, feature] == value #vector value of the values that are equal to the value i
      right_indices <- X[, feature] != value

      left <- y[left_indices] #subset of element of y where left_indice is true
      right <- y[right_indices] #same but right_indice is true

      gini <- gini_impurity_R(left, right) #how much does the value (i) divide the dataset
      if (gini < best_gini) { #check if gini get improves (get smaller) and if this happen change the g
        best_gini <- gini
        best_feature <- feature # Store the best feature index
        best_value <- value #which division of the values you use
      }
    }
  }

  output <- list( #give the save values for the split
    best_feature = best_feature,
    best_value = best_value,
    best_gini = best_gini
  )
```

```
  output
}
```

The inputs to the function are the predictors (X) and the response variable (y), which is the one you want to predict. The function begins by setting default values: the Gini impurity is initialized to 1, as this represents the worst possible value, and the feature is set to -1, which is a placeholder since -1 is a possible value in the columns.

Next, the function iterates through the columns of the predictors. For each column, it first retrieves the unique values in that column. Then, for each unique value, it creates two Boolean vectors: one that is TRUE when the current value in the column matches the value being considered, and another that is TRUE when the value does not match. These two Boolean vectors are used to create two subsets of the response variable (y): one for the left subset (where the value is present) and one for the right subset (where the value is absent).

Finally, the Gini impurity is calculated between the two subsets, and if the Gini value is smaller than the current one, the function updates the best Gini value, the best feature, and the corresponding value that resulted in the current best split.

6 with R

```
best_split_R(play_tennis[0:3],play_tennis$PlayTennis)
```

```
## $best_feature
## [1] 1
##
## $best_value
## [1] "Overcast"
##
## $best_gini
## [1] 0.3636364
```

6 wich C++

```
library(Rcpp)
sourceCpp("C:/Users/flore/Desktop/git/advance_programming/uc3m_advanced_programming/advporgrammingpart1/
```

```
best_split_c(play_tennis[0:3],features = list("Outlook", "Temperature", "Humidity", "Wind"),play_tennis$
```

```
## Processing feature: 0
## Processing feature: 1
## Processing feature: 2

## $best_gini
## [1] 0.3636364
##
## $best_feature
## [1] "Outlook"
##
## $best_value
## [1] "Overcast"
```

As you can the both code give the same result. ### 7 Next we will be adding a variable call "Energy_Level" that have High, Low or Neutral.

```
play_tennis$Energy_level =c("Low","Neutral","High","High","Low","Low","Low","Low","High","High","High",
                            "High","High","High","Low")
play_tennis
```

```
##     Outlook Temperature Humidity   Wind PlayTennis Energy_level
## 1     Sunny         Hot     High   Weak         No          Low
## 2     Sunny         Hot     High Strong         No      Neutral
## 3  Overcast         Hot     High   Weak        Yes         High
## 4      Rain        Mild     High   Weak        Yes         High
## 5      Rain        Cool     High   Weak        Yes          Low
## 6      Rain        Cool   Normal   Weak         No          Low
## 7  Overcast        Cool   Normal Strong        Yes          Low
## 8     Sunny        Mild     High   Weak         No          Low
## 9     Sunny        Cool   Normal   Weak        Yes         High
## 10     Rain        Mild   Normal   Weak        Yes         High
## 11    Sunny        Mild   Normal Strong        Yes         High
## 12 Overcast        Mild     High Strong        Yes         High
## 13 Overcast         Hot   Normal   Weak        Yes         High
## 14     Rain        Mild     High Strong         No         High
## 15     Rain         Hot     High Strong         No          Low
```

This variable should be the new best_feature and the gini impurity should be theorically 0.3071429.

$$Gini(Low, Neutral) = 1 - \left(\left(\frac{2}{7}\right)^2 + \left(\frac{5}{7}\right)^2\right)$$

$$Gini(High) = 1 - \left(\left(\frac{7}{8}\right)^2 + \left(\frac{1}{8}\right)^2\right)$$

$$WeightGini = \frac{7}{15} \cdot Gini(Low, Neutral) + \frac{8}{15} \cdot Gini(High) = 0.3071429$$

```
best_split_R(play_tennis[c("Outlook","Temperature","Humidity","Wind","Energy_level")],play_tennis$PlayT
```

```
## $best_feature
## [1] 5
##
## $best_value
## [1] "High"
##
## $best_gini
## [1] 0.3071429
```

```
best_split_c(play_tennis[c("Outlook","Temperature","Humidity","Wind","Energy_level")],features = list("
```

```
## Processing feature: 0
## Processing feature: 1
## Processing feature: 2
## Processing feature: 3
## Processing feature: 4
```

```
## $best_gini
## [1] 0.3071429
##
## $best_feature
## [1] "Energy_level"
##
## $best_value
## [1] "High"
```

As you can see the obtain values in both of our functions are the same and it's the expected result.

8

IT is best c++ function than R

9

```
source("C:/Users/flore/Desktop/git/advance_programming/uc3m_advanced_programming/advporgrammingpart1/R/
fit_decision_stump_R_v2(play_tennis[c("Outlook","Temperature","Humidity","Wind")],play_tennis$PlayTennis
```

```
## $best_feature
## [1] 1
##
## $best_value
## [1] "Overcast"
##
## $best_gini
## [1] 0.3636364
##
## $left
## [1] "Yes" "Yes" "Yes" "Yes"
##
## $right
##  [1] "No"  "No"  "Yes" "Yes" "No"  "No"  "Yes" "Yes" "Yes" "No"  "No"
##
## $mayority_left
## [1] "Yes"
##
## $mayority_right
## [1] "No"
```

```
library(Rcpp)
sourceCpp("C:/Users/flore/Desktop/git/advance_programming/uc3m_advanced_programming/advporgrammingpart1/
```

```
best_split_c_v2(play_tennis[c("Outlook","Temperature","Humidity","Wind")],features = list("Outlook","Te
```

```
## Processing feature: 0
## Processing feature: 1
## Processing feature: 2
## Processing feature: 3
```

```
## $best_gini
## [1] 0.3636364
##
## $best_feature
## [1] "Outlook"
##
## $best_value
## [1] "Overcast"
##
## $best_left
## [1] "Yes" "Yes" "Yes" "Yes"
##
## $best_right
##  [1] "No"  "No"  "Yes" "Yes" "No"  "No"  "Yes" "Yes" "Yes" "No"  "No"
##
## $Mayority_left
## [1] "Yes"
##
## $Mayority_right
## [1] "No"
```

Both function create the same branch for the tree and also have the same mayority category.