Report_Part1

Florencia Luque and Simon Schmetz

2024-11-15

Introduction

The following RMarkdown document contains the documentation of the first assignment in the course of advanced programming for the Master of Statistics in Data Science at Universidad Carlos III de Madrid. The Chapters are related to the exercises as defined in the assignment sheet.

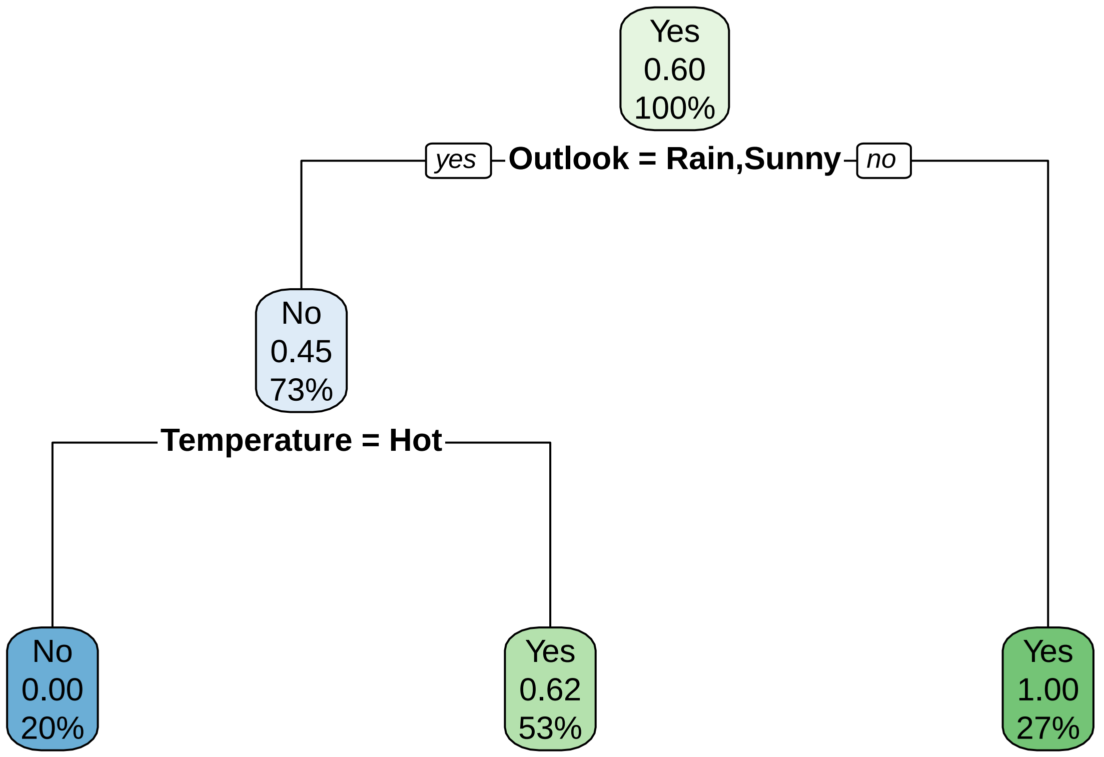
Excercise 1

The data file was successfully stored to the data directory as can be seen in the project folder

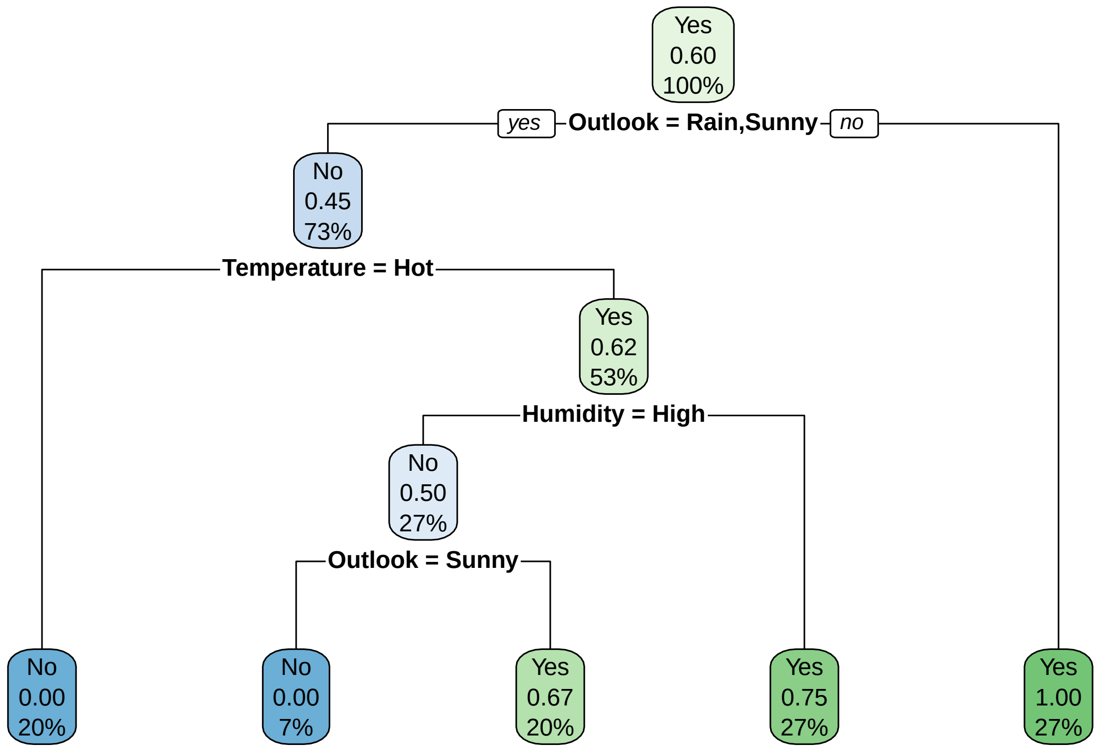
Excercises 2 and 3

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

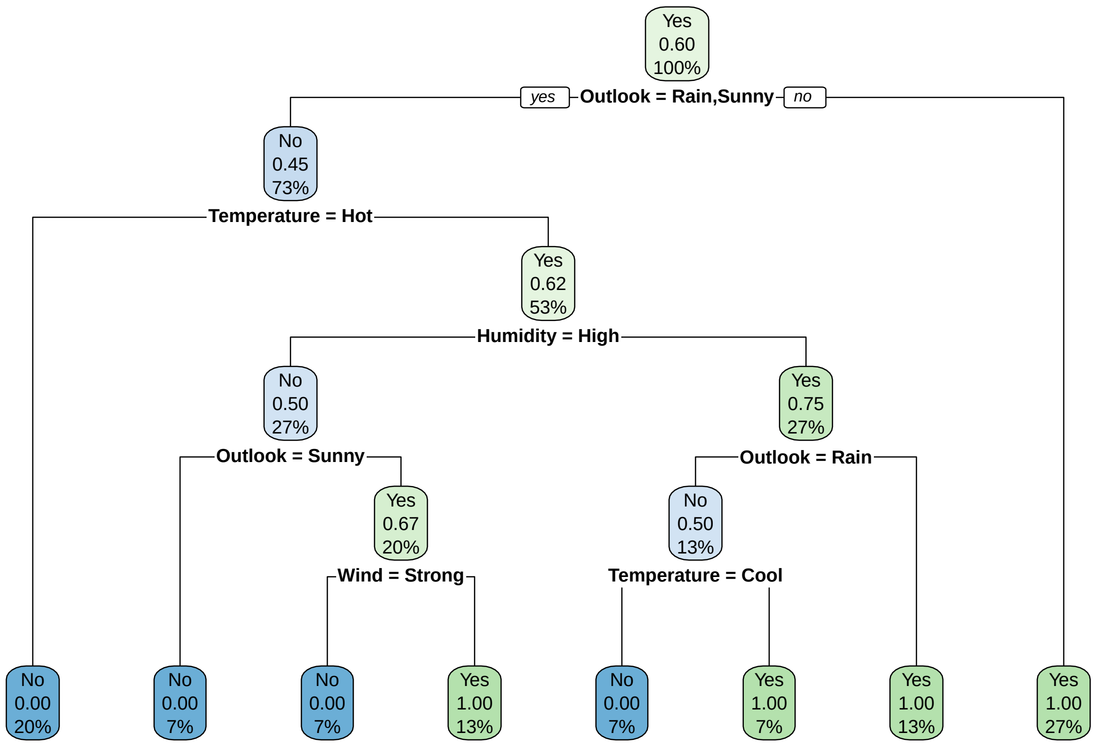
```r
options(width = 80)
change_depth = function(maxde){
  model <- rpart(PlayTennis ~ ., play_tennis, control = rpart.control(cp = 0, maxdepth = maxde, minspli
                                                                    minbucket = 1))
  return (rpart.plot(model))
}
change_depth(2)
```



```
change_depth(4)
```



```
change_depth(5)
```



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

As can be seen, when the value for max_depth is smaller, the tree is simpler. With increasing 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 the resulting tree is the same for any value greater than or equal to 5.

In general, having a perfect division in the data can lead to problems like overfitting, which is it is normal to "prune" trees, or in other words, removing nodes that generate perfect divisions.

Excercises 4 and 5

The inputs to the function are the predictors (X) and the response variable (y), which is what is suppsoed to be predicted. 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.

```r
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
}
```

Excercise 6

The function written in R in the prvious excercise now is translated into C++ code stored in “src/decision_stump_c.cpp”. For the most part, the same structure is maintained, with the only difference being an additional function to count entries in a CharacterVector. This function is required to, instead of transforming the string entires to factors, be able to use the string entries directly.

Excercise 7 Compare results from R and C++ code

The Results are initially verified by comparing the results from the R code:

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

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

With the results from the C++ code by calling the Rcpp function as shown below

```
library(Rcpp)
sourceCpp("advporgrammingpart1/src/decision_stump_c.cpp")
```

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

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

Where both functions give the same result. Additionally, a variable called "Energy_Level" that has High, Low or Neutral is added.

```
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 is designed be the new best_feature and the gini impurity should be 0.3071429 as shown in the calculation below

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

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

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

As observable below, the obtained values in both of the functions (R/C++) are the same and as expected. Both functions thus seem to work well.

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

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

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

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

Excercise 8

Having shown that both the R and the C++ functions work well, a benchmark test is run to evaluate the efficiency of both functions using the package “microbenchmark”. Doing so, gives out a table with values in microseconds for runtime. Both functions are fun 100 times to create values like mean, median, min and max. Running the benchmark of both functions, as expected the C function outperforms the R function significantly. The speed advantage is about 2.5 times, e.g. the R function requiring about 2.5 more time on average. This is in line with the reasoning of writing functions used in R separately as C++ functions, to leverage the speed advantage code written in C++ has.

```r
library(microbenchmark)

# Benchmarking
benchmarking_results <- microbenchmark(
    best_split_c(play_tennis[0:3],
                 features = list("Outlook", "Temperature", "Humidity", "Wind"),
                 play_tennis$PlayTennis),
    best_split_R(play_tennis[0:3],play_tennis$PlayTennis),
    times = 100
)

# Results
print(benchmarking_results)
```

```
## Unit: microseconds
##
##  best_split_c(play_tennis[0:3], features = list("Outlook", "Temperature",      "Humidity", "Wind"), p
##                                                                               best_split_R(play_tennis[0:3], p
##    min     lq    mean median     uq   max neval
##   41.0  43.55  47.928  45.45  49.85  88.4   100
##  118.3 124.45 138.386 129.75 139.35 256.1   100
```

Excercise 9

In the last Excercise, the majority categories for left and right branch both in the R ("R/r_version_of_ds_v2.R") and the C++ function ("src/r_version_of_ds_v2.R"). Using both function, it is apparent that both create the same branch for the tree and also have the same majority category.

```r
source("advporgrammingpart1/R/r_version_of_ds_v2.R")
fit_decision_stump_R_v2(play_tennis[c("Outlook","Temperature","Humidity","Wind")],play_tennis$PlayTenni
```

```
## $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"
```

```r
library(Rcpp)
sourceCpp("advporgrammingpart1/src/decision_stump_c_v2.cpp")
```

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

```
## $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"
```

Use of ChatGPT

ChatGPT was used to find bugs in this project.