Handmade Decision Tree

Álvaro Orgaz Expósito

THEORY

The aim of the *Decision Tree* algorithm is to predict the variable y (categorical for *classification* problem or continuous for *regression problem*) knowing the explanatory variables $x = \{x_1, \ldots, x_p\}$.

Firstly, you need to understand the concept of *tree-based models.* These are non-parametric (which means that no statistical distributions are assumed in the models) methods with the objective of predicting y based on decision rules derived from the data. In this way, data observations are divided into several homogeneous groups with respect to y to discriminate well this variable. In short, prediction via stratification or splitting of the feature space to create internal nodes from the *root node* to the *leaf nodes.*

Note: The *root node* is the first node which contains the whole dataset without splits, and the *leaf nodes* are the last split nodes of the tree without more splits from them.

Secondly, you need to understand the concept of *recursive binary splitting.* As we mentioned, the goal is finding the J more homogeneous non-overlapping regions of the feature space $R = \{R_1, R_2, \ldots, R_J\}$ with respect to y, and then, every observation that falls into the region R_j will have the same prediction. But unfortunately, it is computationally infeasible to consider every possible partition of x into J regions. For this reason, a *recursive binary splitting* is used starting at the top of the tree with the *root node* (all observations in x belong to a single region) and then successively splitting the regions into two new regions or nodes. It is greedy because at each step of the tree-building process the best split is made at that particular step, rather than picking a split that will lead to a better tree in some future step.

Note: There are many types of splitting methods for *Decision Tree*, but this paper covers the *recursive binary splitting.*

Thirdly, let's see the main steps of the *Decision Tree* algorithm.

1. Define the inputs $x_{[NxP]}$ and $y_{[Nx1]}$.
2. Define the split criteria for choosing the optimal data splits in a node t. It is the *impurity* of the resulting nodes, which measures the *uncertainty* of the data with respect to y.

   2.1 In a regression problem

   $$Impurity(t) = Variance(y^t) = \frac{\sum_{i=1}^{N_t} (\hat{y}_i^t - y_i^t)^2}{N_t}$$

   where N_t is the number of observation in node t data, y^t is the value of observation i in the variable to predict in node t data, $\hat{y}_i^t$ is the predicted value of observation i in the node t data.

   2.2 In a classification problem there are mainly two options:

   - $Impurity(t) = Gini(y^t) = \sum_{k=1}^{K} P(y^t = k)(1 - P(y^t = k)) = 1 - \sum_{k=1}^{K} P(y^t = k)^2$
   - $Impurity(t) = Entropy(y^t) = \sum_{k=1}^{K} P(y^t = k) log_2\left(\frac{1}{P(y^t=k)}\right) = -\sum_{k=1}^{K} P(y^t = k) log_2 P(y^t = k)$

   where K is the number of unique categories in the y, and $P(y^t = k)$ represents the probability of belonging to class k of the response variable in the node t, basically the % of observations with $y = k$ in the node t.

   Note: Basically, the entropy is minimum when one category has probability 1, and maximum when all categories have the same probability $1/K$.

3. Perform the first *binary split* in the *root node* ($t = 1$). For each explanatory feature $x^t = \{x^t_1, \ldots, x^t_p\}$:

   3.1 If x_j is quantitative, consider all its observations in the node data $x^t_j = x^1_j, ..., x^{N_t}_j$ as cut-off c and split the node data into 2 regions:

   - $t_{left} = \{x^t | x^t_j \leq c\}$
   - $t_{right} = \{x | x_j > c\}$

   3.2 If x_j is categorical, consider all its unique C categories in the node data $unique(x^t_j) = x^1_j, ..., x^C_j$ as cut-off s and split the node data into 2 regions:

   - $t_{left} = \{x^t | x^t_j = s\}$
   - $t_{right} = \{x^t | x^t_j \neq s\}$.

4. Find the optimal split in the *root node*, it means computing the *new impurity* for all created splits in point 3 and select the split with lower value. The formula of the *new impurity* for a split in node t is:

$$New\_Impurity(t) = \frac{N_{t_{left}} Impurity(t_{left}) + N_{t_{right}} Impurity(t_{right})}{N_t}$$

   where t_{lelft} and t_{right} are the 2 split regions from node t.

5. Next, repeat the steps 3 and 4 to minimize the impurity of the previous split regions or *leafs nodes*. However, this time, instead of splitting the entire x, only split the data in each of these regions.

6. Again, repeat the step 5 until a *stopping criteria* is reached. For example, in this paper, the *binary splitting* is stopped at a node t when one of the following criteria is violated:

   - The tree depth is higher than *maximum depth* hyperparameter.
   - The *new impurity* is not lower than the node t *impurity* for all possible splits.
   - Both split regions from node t have at least as observations as *minimum instances per node* hyperparameter.

7. Once we have the tree built, we can make the predictions $\hat{y}$ in each *leaf node* t as:

   - In a regression problem, the mean of y^t values in the *leaf node* t data.
   - In a clasification problem, the % of observations with $y^t = k$ in the *leaf node* t data for each K unique categories in y, and the category with higher % will be the prediction.

8. Now, we can predict a x^{test} data by passing each observation through the tree and assigning the prediction of the *leaf node* in which it falls. Then, every observation that falls into the same *leaf node* will have the same prediction.

Fourthly, this paper covers the *Decision Tree* with *stopping criteria*. However, another common option is to build a *full tree* which means that the tree grows without *stopping criteria* until the *leaf nodes impurity* cannot be improved by a new split. And then, it is typical applying *pruning* to the *full tree* but this papers will not cover this topic.

Note: Apart from the *stopping criterias* included in this paper, there are other types like: minimum reduction of impurity to consider a split, maximum number of bins to discretize a continuos explanatory feature and not trying as much cut-offs as observations, or weights of observations to penalize more some observations.

CODE

The data used is the popular dataset *iris*. Let's train a *Decision Tree* to predict the categorical variable *Species* (then it is a *classification* problem with 3 categories) with the explanatory features: *Petal.Width* and *Sepal.Width*. Let's define the *train* data and the name of the target variable in the data.

```r
train <- iris[,c("Petal.Width","Sepal.Width","Species")]
target_name <- "Species"
```

Create the *optimal node split* function which provides the optimal split for a given node t: new impurity, optimal feature, optimal cut-off, data in each split regions (t_{left} and t_{right}), and the prediction for each region.

```r
OptimalNodeSplit <- function(data_node,features_names,target_name,impurity_node){
  N_node <- nrow(data_node)
  # Iterate all features
  for(j in features_names){
    feature <- data_node[,j]
    # Distinct if the feature is numerical or categorical
    if(is.numeric(feature)){
      # If the feature is numerical, iterate all its observation as cut-offs
      for(i in feature){
        # Create 2 regions from node data with cut-off i
        data_left <- data_node[feature<=i,]
        data_right <- data_node[feature>i,]
        # Compute the new impurity after the split
        impurity_left <- nrow(data_left)*impurity(data_left[,target_name])
        impurity_right <- nrow(data_right)*impurity(data_right[,target_name])
        new_impurity <- (impurity_left+impurity_right)/N_node
        # If the new impurity is lower than actual impurity save this optimal split
        if(new_impurity<impurity_node){
          impurity_node <- new_impurity
          optimal_feature <- j
          optimal_cutoff <- i
          regions <- list(data_left=data_left,data_right=data_right)
          predict_left <- names(which.max(table(data_left[,target_name])))
          predict_right <- names(which.max(table(data_right[,target_name])))
        }
      }
    }else{
      # If the feature is categorical, iterate all its unique categories as cut-offs
      for(i in unique(feature)){
        # Create 2 regions from node data with cut-off i
        data_left <- data_node[feature==i,]
        data_right <- data_node[feature!=i,]
        # Compute the new impurity after the split
        impurity_left <- nrow(data_left)*impurity(data_left[,target_name])
        impurity_right <- nrow(data_right)*impurity(data_right[,target_name])
        new_impurity <- (impurity_left+impurity_right)/N_node
        # If the new impurity is lower than actual impurity save this optimal split
        if(new_impurity<impurity_node){
          impurity_node <- new_impurity
          optimal_feature <- j
          optimal_cutoff <- i
          regions <- list(data_left,data_right)
          predict_left <- names(which.max(table(data_left[,target_name])))
          predict_right <- names(which.max(table(data_right[,target_name])))
        }
      }
    }
  }
  return(list(impurity_node,optimal_feature,optimal_cutoff,regions,predict_left,predict_right))
}
```

Create the *impurity* function for this classification problem. We will use the *entropy* based option.

```
impurity <- function(target){
  entropy <- 0
  for(category in unique(target)){
    probability <- mean(target==category)
    entropy <- entropy-probability *log(probability,2)
  }
  return(entropy)
}
```

Let's create the *Decision Tree* algorithm.

```
DecisionTree <- function(train,target_name,max_depth,min_instances){
  tree <- c()
  features_names <- names(train)[!(names(train) %in% target_name)]
  # Create the list to store the leaf nodes data of every depth
  leafs <- list(Root=train)
  # Iterate every depth
  for(depth in 1:max_depth){
    # Create an empty list to store the leaf nodes data in next depth
    leafs_next_depth <- list()
    # Iterate every leaf node in the depth iteration
    leafs_number <- length(leafs)
    leafs_names <- names(leafs)
    for(leaf in 1:leafs_number){
      data_node <- leafs[[leaf]]
      # Compute the node impurity before splitting
      impurity_node <- impurity(data_node[,target_name])
      # Find the optimal split for the node (considering the stopping criteria)
      if(impurity_node==0){
        # Add the node split information to the tree
        predict <- names(which.max(table(data_node[,target_name])))
        tree <- c(tree,paste0(leafs_names[leaf]," has impurity 0 with prediction '",predict))
      }else{
        ons <- OptimalNodeSplit(data_node,features_names,target_name,impurity_node)
        if(min_instances<=nrow(ons[[4]][[1]]) & min_instances<=nrow(ons[[4]][[2]])){
          # Update the list of leaf nodes data in next depth with split regions
          leafs_next_depth[[leaf]] <- ons[[4]]
          # Add the node split information to the tree
          tree <- c(tree,paste0(leafs_names[leaf]," by '",ons[[2]],"' in '",ons[[3]],
                               "' with predictions '",ons[[5]],"'<->'",ons[[6]]))
        }
      }
    }
    # Update the list of leaf nodes data for the next depth
    leafs <- NULL
    for(leaf in 1:leafs_number){
      leafs[[paste0(leafs_names[leaf],"_Left")]] <- leafs_next_depth[[leaf]][[1]]
      leafs[[paste0(leafs_names[leaf],"_Right")]] <- leafs_next_depth[[leaf]][[2]]
    }
  }
  return(tree)
}
```

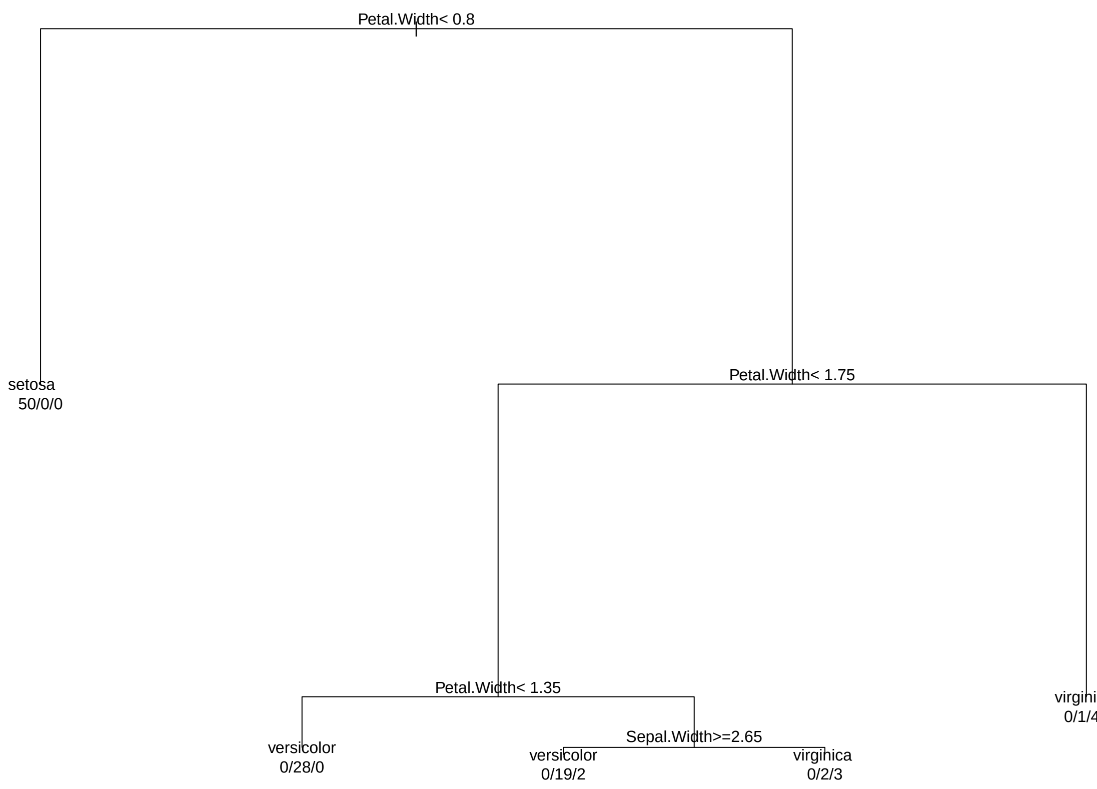
Let's train the *Decision Tree* and see the output.

```r
max_depth <- 3
min_instances <- 5
DecisionTree(train,target_name,max_depth,min_instances)
```

```
## [1] "Root by 'Petal.Width' in '0.6' with predictions 'setosa'<->'versicolor"
## [2] "Root_Left has impurity 0 with prediction 'setosa"
## [3] "Root_Right by 'Petal.Width' in '1.7' with predictions 'versicolor'<->'virginica"
## [4] "Root_Right_Left by 'Petal.Width' in '1.3' with predictions 'versicolor'<->'versicolor"
## [5] "Root_Right_Right by 'Petal.Width' in '1.8' with predictions 'virginica'<->'virginica"
```

Let's compare our tree with the implemented function *rpart* for *Decision Trees* in the R package *rpart*, setting the same hyperparameters and splitting method.

```r
library(rpart)
decision_tree <- rpart(Species~.,data=train,method="class",parms=list(split='information'),
                       control=rpart.control(minsplit=min_instances,maxdepth=max_depth+1,cp=0))
plot(decision_tree)
text(decision_tree,use.n=TRUE)
```



In conclusion, following the splits of our tree and the plot of the implemented function in R, we can see that they match very well. They use the same optimal feature and almost the same cut-off on each node split, and the predictions are the same.