

# Statistical Machine Learning

## Tree-Based Methods

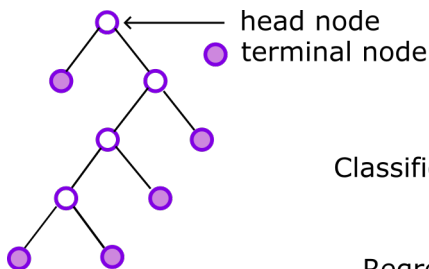
Horacio Gómez-Acevedo  
Department of Biomedical Informatics  
University of Arkansas for Medical Sciences

February 21, 2025





# Tree structures

A typical tree is depicted with the root being the top node, and growing down. Decisions are being made at each node until a terminal node or *leaf* is reached. Each non-terminal node contains a question on which a split is based. Each leaf contains the label (classification) or the predicted mean value (regression).



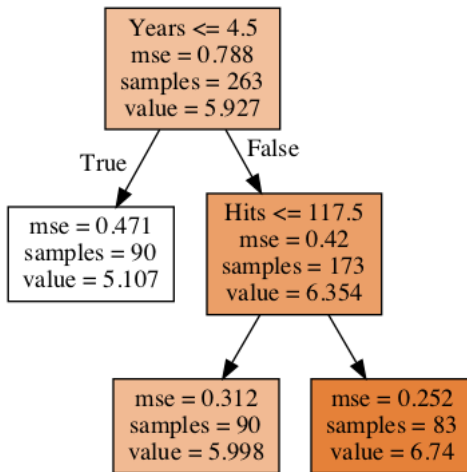
Examples:

Classification  High

Regression  3.14

# Regression Trees

We will use the **Hitters** data set as an example for regression. After running the code we obtain the following figure

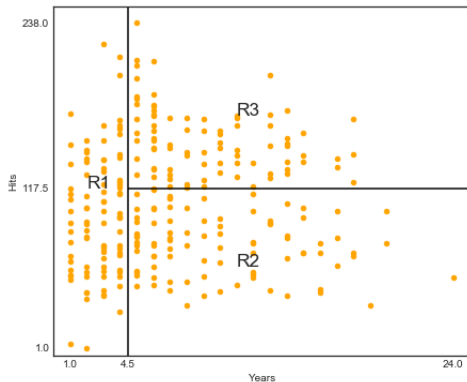


## Regression Trees (cont)

How to read this information? The first feature for this set is *years*, from which the total samples are split into two depending on whether their time spent in the baseball league has been less than 4.5 years. The case on the left (True) is already given a value of 5.107, which means that the average salary is  $\exp(5.107) \approx 165.17$  thousand dollars a year. Players with more than 4.5 years will be further divided into the variable *hits*, and someone above 117.5 will earn on average  $\exp(6.74) = 845.56$  thousand dollars!

# Regression Trees (cont)

We can see that the graphical representation of a tree is as follows



The regions  $R_i$  are the representing the tree leaves.

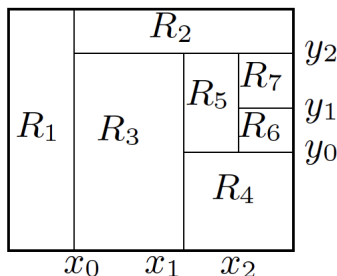
# Prediction in Regression Trees

There are two main steps involved in building a tree

- Divide the predictor space  $X_1 \times X_2 \times \cdots \times X_p$  into  $J$  distinct and non-overlapping regions  $R_1, \dots, R_J$ .
- For every observation that falls into the region  $R_j$ , we make the same prediction, which is the mean of the response values for the training observations in  $R_j$ .

## 2D Example

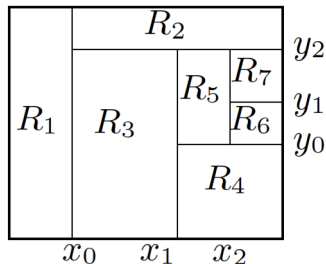
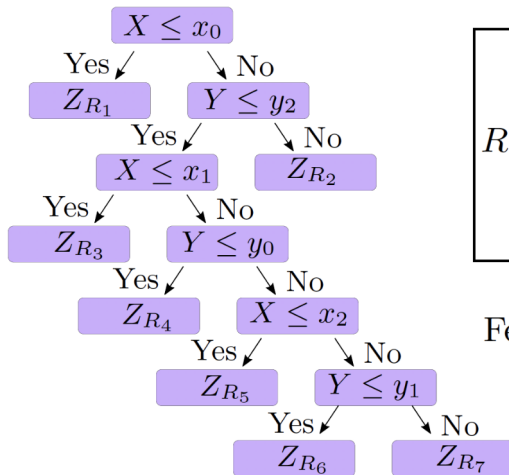
Suppose we have the outcome  $Z$  depending on two predictors  $X$  and  $Y$  defined in certain rectangular area  $A \subset X \times Y$  of the plane. One feasible partition of  $A$  is depicted below



Feasible Partition

## 2D Example (cont)

The previous partition can be described as a tree structure



Feasible Partition



## 2D Example (cont)

The corresponding prediction for  $\hat{Z}(x, y)$  based on given partition of  $A$  is defined as

$$\hat{Z}(x, y) = \text{Average}\{Z(x_r, y_r) : (x_r, y_r) \in R_j\}$$

where  $(x, y) \in A$ .

# Dividing the Predictor Space

Ideally, one must find regions  $R_1, \dots, R_J$  (also called **boxes**) that minimize the  $RSS$  given by

$$\sum_{j=1}^J \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

where  $\hat{y}_{R_j}$  is the mean response for the training observations within the  $j$ th box. Since it is not computationally feasible to test every single box combination, we use a *greedy* approach that is known as *recursive binary splitting*. This process is top down since we start with the whole region and after a number of successive splits into two branches further down on the tree. At each step we select the *best* split at the given time without looking at other alternatives further down.

# Recursive Binary Splitting

We begin by selecting a predictor  $X_j$  and the cut point  $s$  such a that splitting the predictor space into the regions  $\{X|X_j < s\}$  and  $\{X|X_j \geq s\}$  leads to the creates possible reduction in RSS. Thus, for any  $j$  and  $s$ , we define a pair of half-planes

$$R_1(j, s) = \{X|X_j < s\} \quad \text{and} \quad R_2(j, s) = \{X|X_j \geq s\} \quad (1)$$

and we look for the value  $j$  and  $s$  that minimize the equation

$$\sum_{i: x_i \in R_1(j, s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i: x_i \in R_2(j, s)} (y_i - \hat{y}_{R_2})^2, \quad (2)$$

We repeat the process looking for the best predictor and best cut point in order to split the data further so as to minimize the RSS within each of the resulting regions but this time we we don't split the entire space but each of the two regions found. This process continues until a certain criteria is reach (e.g., until no region contains fewer than a certain number of elements within).

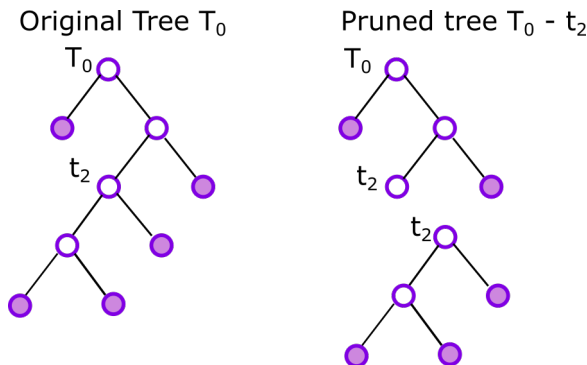
## Recursive Binary Splitting (cont)

Once the regions are found, the predicted response for a given test observation is the mean of the training observations in the region to which that test observation belongs.

The recursive binary splitting may produce good predictions on the training set, but it follows too close the data and more likely produce overfitting. Thus, a smaller tree with fewer splits might lead to lower variance and better interpretation at the cost of bias.

# Tree Pruning

One way to address the overfitting problem is to grow a very large tree  $T_0$ , and then prune it back to obtain a sub-tree. Ideally, we select a sub-tree that leads to the lowest test error rate using cross-validation.



Once again we are facing a computationally intensive problem since the number of possible sub-trees is very large.

## Tree Pruning (cont)

A process known as *weakest link pruning* considers a sequence of trees indexed by a non-negative tuning parameter  $\alpha$ . For each value of  $\alpha$ , we find a sub-tree  $T \subset T_0$  such that the following quantity is as small as possible

$$\sum_{m=1}^{|T|} \sum_{i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|, \quad (3)$$

where  $|T|$  denotes the number of leaves of the tree  $T$ .

# Regression Tree Algorithm

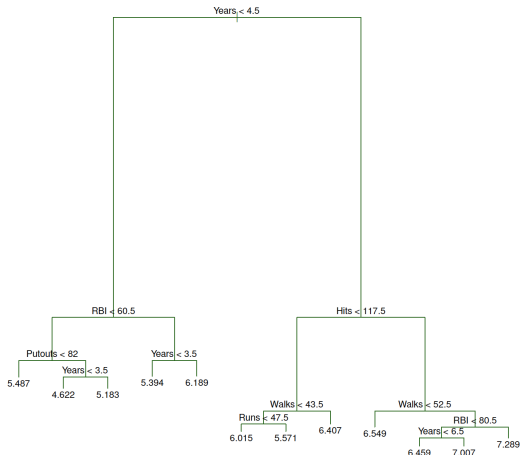
- ➊ Use recursive binary splitting to grow a large tree on the training data. Use a stopping rule such as the minimum number of observations on each leaf.
- ➋ Apply the weakest link pruning to the large tree in order to obtain a sequence of best sub-trees as a function of a parameter  $\alpha$ .
- ➌ Use  $K$ -fold cross-validation to choose  $\alpha$ . More precisely, divide the training observations into  $K$  folds, and for  $k \in \{1, \dots, K\}$  do:
  - ➊ Repeat steps 1 and 2 on all but the  $k$ th fold of the training data.
  - ➋ Evaluate the mean squared prediction error on the data in the left-out  $k$ th fold, as a function of  $\alpha$

Average the results for each value of  $\alpha$ , and pick  $\alpha$  that minimize the average error.

- ➍ Return the sub-tree from step 2 that corresponds to the chosen value of  $\alpha$ .

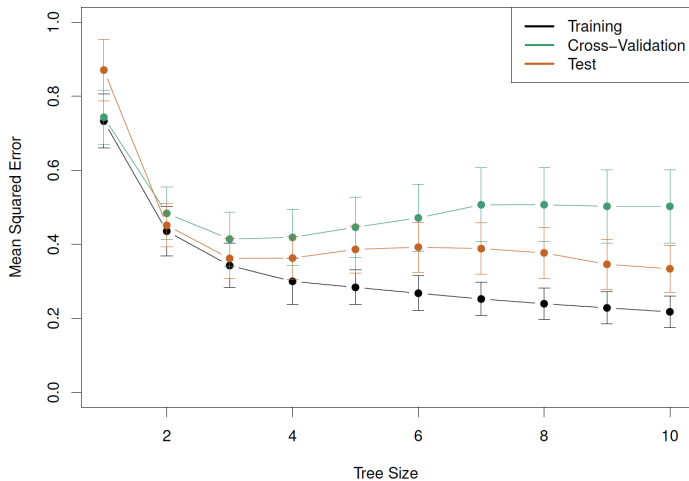
## Example.

The unpruned tree that results from greedy splitting on the training data from the **Hitters** data set is depicted below





## Example (cont)



# Classification Trees

The idea of the classification trees is very similar to the regression trees. However, in the classification setting we cannot use RSS as an splitting criteria for the binary selection. Instead, we will use the **classification error rate**, which is the fraction of the training observations in a region that do not belong to the most common class.

$$E_m = 1 - \max_k(\hat{p}_{mk})$$

where  $\hat{p}_{mk}$  represent the proportion of training observations in the  $m$ th region that are from the  $k$ th class.

# Purity metrics

A node is **pure** if all training instances belong to the same class. Some metrics to assess the impurity are:

- **Gini index.** Defined by

$$G_m = \sum_{k=1}^K \hat{p}_{mk}(1 - \hat{p}_{mk}),$$

is a measure of total variance across the  $K$  classes.

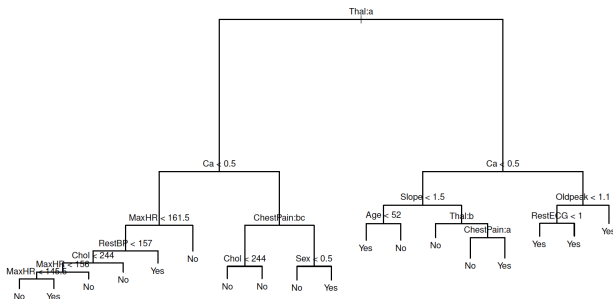
- **Cross-entropy.** Defined by

$$D_m = - \sum_{k=1}^K \hat{p}_{mk} \log \hat{p}_{mk}.$$

Note that both metrics will take on a small value if either  $\hat{p}_{mk} \sim 1$  or  $\hat{p}_{mk} \sim 0$  and that means that the node is pure. Both the Gini index and the cross-entropy are quite similar numerically.

# Example

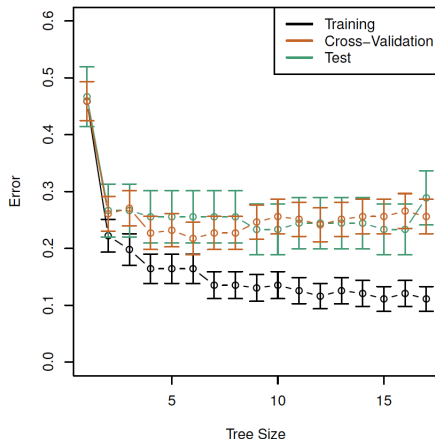
Using the **Heart** data, the classification tree (unpruned) is shown below.



Note the leaf  $\text{RestECG} < 1$ . The response is Yes so why do we need to partition further? Then answer is that it does not reduce the classification error but it improves the Gini index and cross-entropy.

## Example (cont)

Cross-validation error, training and test error



# CART Algorithm

Géron's book gives some details of the *Classification and Regression Tree* algorithm. Originally described in the book by Breiman et al. (1984) the idea is similar to (1) and (2), and the cost function that we want to minimize (for classification)

$$J(s, t_s) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} G_{right}, \quad (4)$$

where  $G_{left}$  is the measure of impurity for the *left* subset, and  $m_{left}$  represents the number of instances.

The same algorithm has the following cost function for regression

$$J(s, t_s) = \frac{m_{left}}{m} RSS_{left} + \frac{m_{right}}{m} RSS_{right},$$

# CART algorithm pruning

This process consists in searching for the best pruned sub-tree of the maximal tree. It should be a good intermediate tree between the two extremes

- The maximal tree that has a high variance and low bias, and
- The tree containing only the root (stump) that has low variance but high bias.

Despite the fact that there are a finite number of sub trees, it is computationally expensive to calculate all. Instead, we look from a sequence of nested sub-trees  $T_1, \dots, T_K$  all pruned from  $T_{max}$ , where  $T_k$  minimizes a penalized criterion where the penalty is proportional to the number of leaves of the tree.

## CART algorithm pruning (cont)

Let's consider the regression setup,  $t$  denotes a node, and data is represented as pairs  $(x_i, y_i)_{i=1}^n$  of explanatory and response variables, respectively.

$$\overline{\text{err}}(T) = \frac{1}{n} \sum_{t \text{ leaf of } T} \sum_{(x_i, y_i) \in t} (y_i - \bar{y}_t)^2$$

and the penalized least squares criterion:

$$\text{crit}_\alpha = \overline{\text{err}}(T) + \alpha |T|$$

where  $|T|$  represents the number of leaves of the tree  $T$ .

In (Brieman et al., 1984) is proven that there is a strictly increasing sequence of parameters  $(\alpha_i)_{i=1}^K$  such that  $\alpha_1 = 0$  and an associated sequence  $T_1 \succ T_2 \succ \dots \succ T_K$  of nested trees with  $T_1 = T_{\max}$  and

$$\forall \alpha \in [\alpha_k, \alpha_{k+1}[ , T_k = \underset{T \text{ subtree of } T_{\max}}{\text{argmin}} \text{crit}_\alpha(T) = \underset{T \text{ subtree of } T_{\max}}{\text{argmin}} \text{crit}_{\alpha_k}(T).$$



# CART algorithm pruning (cont)

- Input: Maximal tree  $T_{max}$ .
- Initialize:
  - 1  $\alpha_1 = 0$ ,  $T_1 = \underset{T \text{ pruned from } T_{max}}{\operatorname{argmin}} \bar{\text{err}}(T)$ ;
  - 2  $T \leftarrow T_1$ ;  $k \leftarrow 1$ .
- Loop while  $|T| > 1$ 
  - 1 Calculate  $\alpha_{k+1} = \min_{t \text{ node of } T} \frac{\bar{\text{err}}(t) - \bar{\text{err}}(T)}{|T_t| - 1}$
  - 2 Prune all  $T_t$  branches of  $T$  such that

$$\bar{\text{err}}(T_t) + \alpha_{k+1}|T_t| = \bar{\text{err}}(t) + \alpha_{k+1}$$

- 3  $T_k$  will be the pruned tree obtained.
  - 4  $T \leftarrow T_k$ ;  $k \leftarrow k + 1$ .
- Output:
    - 1 Trees:  $T_1 \succ T_2 \succ \dots \succ T_K = \{t_1\}$
    - 2 Parameters:  $\alpha_1 = 0, \alpha_2, \dots, \alpha_K$ .

# CART details

Intuitively, the minimal cost-complexity pruning works like this: It starts with  $T_1$  and finds the weakest-link branch  $T_{\bar{t}_1}$  and prunes it to get  $T_2$  when  $\alpha$  reaches  $\alpha_2$ , and so on.

The choice of the optimal tree can be made directly by minimizing the error obtained by cross-validation or by applying the "1 standard error rule". This rule aims at selecting in the sequence a more compact tree reaching statistically the same error. Thus, it chooses the most compact tree reaching an error lower than the value of the previous minimum augmented by the estimated standard error of this error.

Decision Trees have more robust libraries in R. Namely, **rpart** package is particularly useful.

# Cross-validation in CART

The cross-validation (V-fold cross-validation) in **rpart** goes as follows: Denote the whole data set by  $\mathcal{L}_n$  ( $n$  is the number of samples). We obtain the sequences  $(T_k)_{k=1}^K$  and  $(\alpha_k)_{k=1}^K$  as before. As usual in CV, we divide  $\mathcal{L}_n = E_1 \cup E_2 \cup \dots \cup E_V$ . For each  $v \in 1, \dots, V$ , we build the sequence of subtrees  $(T_k^v)_{k=1}^K$  with  $\mathcal{L}_n - E_v$  as a learning sample. We calculate the validation errors of the sequence as

$$R^{cv}(T_k) = \frac{1}{V} \sum_{v=1}^V \sum_{(x_i, y_i) \in E_v} (y_i - T_k^v(x_i))^2,$$

where  $T_k^v$  minimizes the penalized criterion  $\text{crit}_{\alpha'_k} = \sqrt{\alpha_k \alpha_{k+1}}$ . Finally, we chose the model  $T_{\hat{k}}$  where  $\hat{k} = \text{argmin}_{1 \leq k \leq K} R^{cv}(T_k)$

# Normalized Penalty CP parameter

For the **rpart** library the so-called CP (normalized complexity-penalty parameter), the value by default is  $cp = 0.01$ , the tree provided corresponds to the one obtained by selecting the one corresponding to  $\alpha = 0.01 * \overline{\text{err}}(T_n)$

# Surrogate Splits CART

A **split**  $s$  on a continuous variable  $x$  is of the form  $\{\text{is } x \leq c?\}$ , then the complementary split is  $\{\text{is } x > c?\}$ . For a categorical variable  $x$  a split  $s$  is of the form  $\{\text{is } x \in \{C_1, \dots, C_k\}?\}$  and its complementary split  $\{\text{is } x \notin \{C_1, \dots, C_k\}?\}$ .

For any given node  $t$ , let  $s^*$  be the best split of the node into  $t_L$  (left node) and  $t_R$  (right node), for instance the best univariate split.

Take any variable  $x_m$  and let  $S_m$  be the set of all splits on  $x_m$ , and  $\bar{S}^m$  the set of splits complementary to  $S_m$ . For a given split  $s_m \in S_m \cup \bar{S}^m$ , let  $p(s^*, s_m)$  the probability that  $s_m$  predicts  $s^*$  correctly.

A split  $\tilde{s}_m \in S_m \cup \bar{S}^m$  is called a **surrogate split** on  $x_m$  for  $s^*$  if

$$p(s^*, \tilde{s}_m) = \max_{s_m} p(s^*, s_m) \quad s_m \in S_m \cup \bar{S}^m$$

The intuitive idea of a surrogate split  $\tilde{s}_m$  on  $x_m$  is the most accurate split that predicts the action of  $s^*$ .

## Surrogate Splits CART (cont)

Suppose  $s^*$  sends the cases in  $t$  left with relative probability  $p_L$  and right with relative probability  $p_R$  ( $p_R + p_L = 1$ ).

The **predictive measure of association**

$$\lambda(s^*|\tilde{s}_m) = \frac{\min(p_L, p_R) - (1 - p(s^*, \tilde{s}_m))}{\min(p_L, p_R)}$$

This measure is the relative reduction in error gotten by using  $\tilde{s}_m$  to predict  $s^*$  as compared with the  $\max(p_L, p_R)$ . If  $\lambda(s^*|\tilde{s}_m) \leq 0$ ,  $\tilde{s}_m$  is no help in predicting  $s^*$  and is discarded as a surrogate split.

Why do we need surrogate splits? For missing data and to determine the variables of importance!

# Missing Data Algorithm

Suppose that we have the best split  $s^*$  on a node is being found. If there are missing values, the best split  $s_m^*$  on  $x_m$  is computed using all cases containing a valid of  $x_m$  and then  $s^*$  selected as that split  $s^*_m$  which maximizes decrease on impurity.

If a case has missing values so that  $s^*$  is not defined for that case. Among all non-missing variables in the case, find that one, say  $x_m$  with  $\tilde{s}_m$  having the highest measure of predictive association with  $s^*$ . The split the case using  $\tilde{s}_m$ .

This procedure is similar to replacing a missing value in a linear model by regression on the non-missing value most highly correlated with it.

# Variable Ranking CART

Let  $T$  be an optimal subtree selected by the cross-validation or test sample procedure. If the Gini splitting rule has been used, then at each node compute  $\Delta I(\tilde{s}_m, t) = I(t) - I(t_L) - I(t_R)$ .

The **measure of importance of variable**  $x_m$  is defined as

$$M(x_m) = \sum_{t \in T} \Delta I(\tilde{s}_m, t)$$

The measure of importance is the sum over all nodes of the decrease in impurity produced by the surrogate split on  $x_m$  at each node. Since only the relative magnitudes of the  $M(x_m)$  are the interesting, the actual measures of importance we use are the normalized quantities  $100M(x_m)/\max_m M(x_m)$ . The most important variable has measure 100, and the other range 0 to 100.



# Trees vs. Linear Models

The classical linear regression assumes a relationship of the form

$$f(X) = \theta_0 + \sum_{j=1}^p \theta_j X_j$$

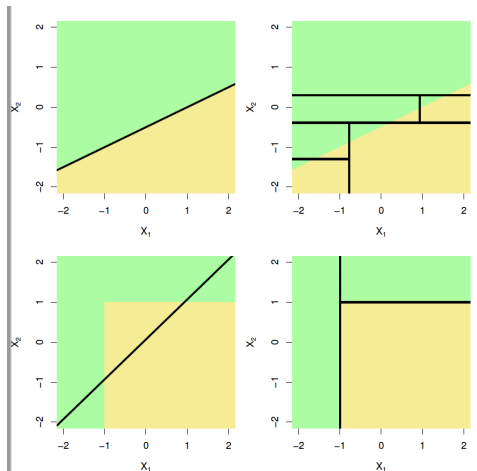
whereas the regression trees assume that the model has the form

$$f(X) = \sum_{i=1}^M c_m \cdot 1_{X \in R_m}$$

where  $R_1, \dots, R_M$  represent a partition of feature space and  $1_A$  represents the indicator function (1 if  $x \in A$  and 0 elsewhere)

# Trees vs. Linear Models (cont)

As we know, there is not a single model that will work all the time for any data set.



# Trees vs. Linear Models

Advantages of trees are:





- They are very easy to explain even for non-expert people.
- Trees can easily handle a mix of predictors (quantitative, continuous) without the need of *dummy* variables.
- Trees are extremely robust with respect to outliers and misclassified points.

But

- Trees do not have the same level of predictive accuracy.
- Trees are unstable, small changes in data can cause significant changes in the final model.

# References

Materials and some of the pictures are from (Gareth et al., 2015), (Genuer and Poggi, 2020), (Brieman et al., 1984), and (Géron, 2019).

-  Brieman, L. et al. (1984). *Classification and Regression Trees*. Wadsworth Inc. ISBN: 0-534-98054-6.
-  Gareth, J. et al. (2015). *An Introduction to Statistical Learning*. 1st edition. Springer. ISBN: 978-1-4614-7137-0.
-  Genuer, R. and Jean-Michel Poggi (2020). *Random Forests with R. Use R!* ISBN: 978-3-030-56484-1.
-  Géron, A. (2019). *Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow. Concepts, Tools, and Techniques to Build Intelligent Systems*. 2nd. Edition. O'Reilly. ISBN: 978-1-492-03264-9.

I have used some of the graphs by hacking TiKz code from StakExchange, Inkscape for more aesthetic plots and other old tricks of T<sub>E</sub>X