Statistical Machine Learning
Part 7
Tree-Based Methods

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

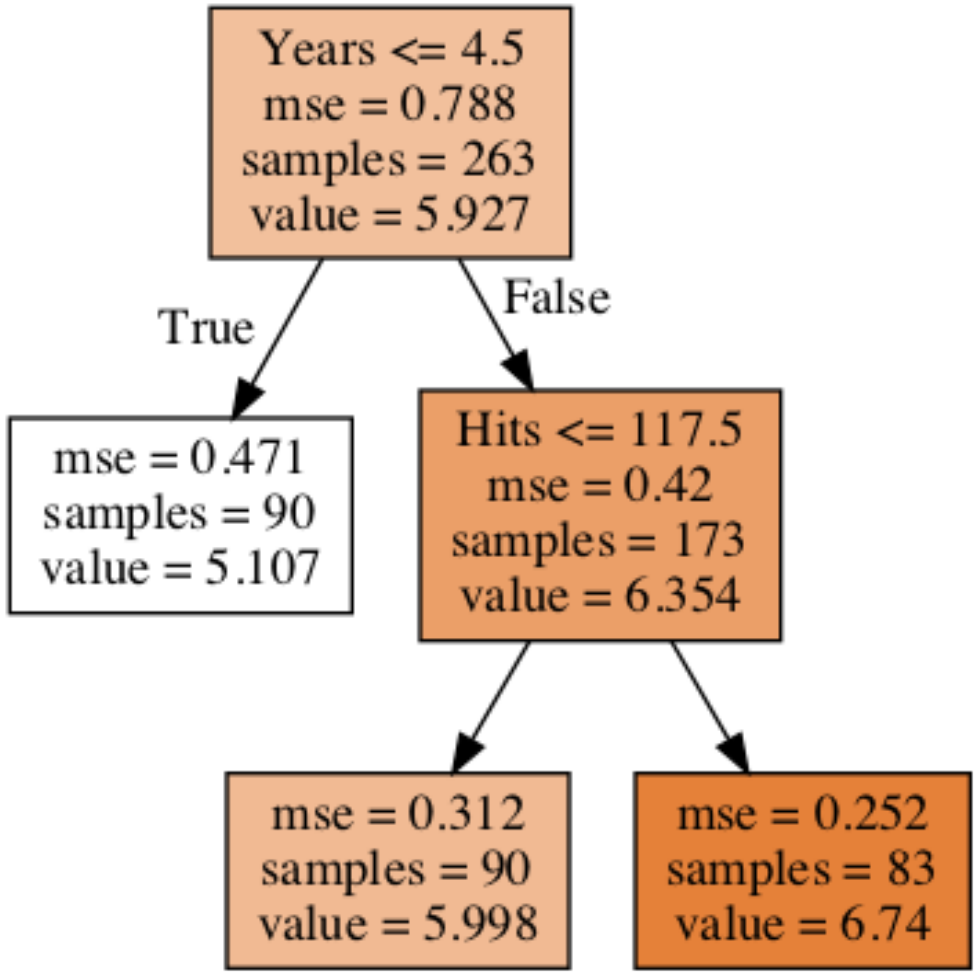
March 9, 2021

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).

Regression Trees

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

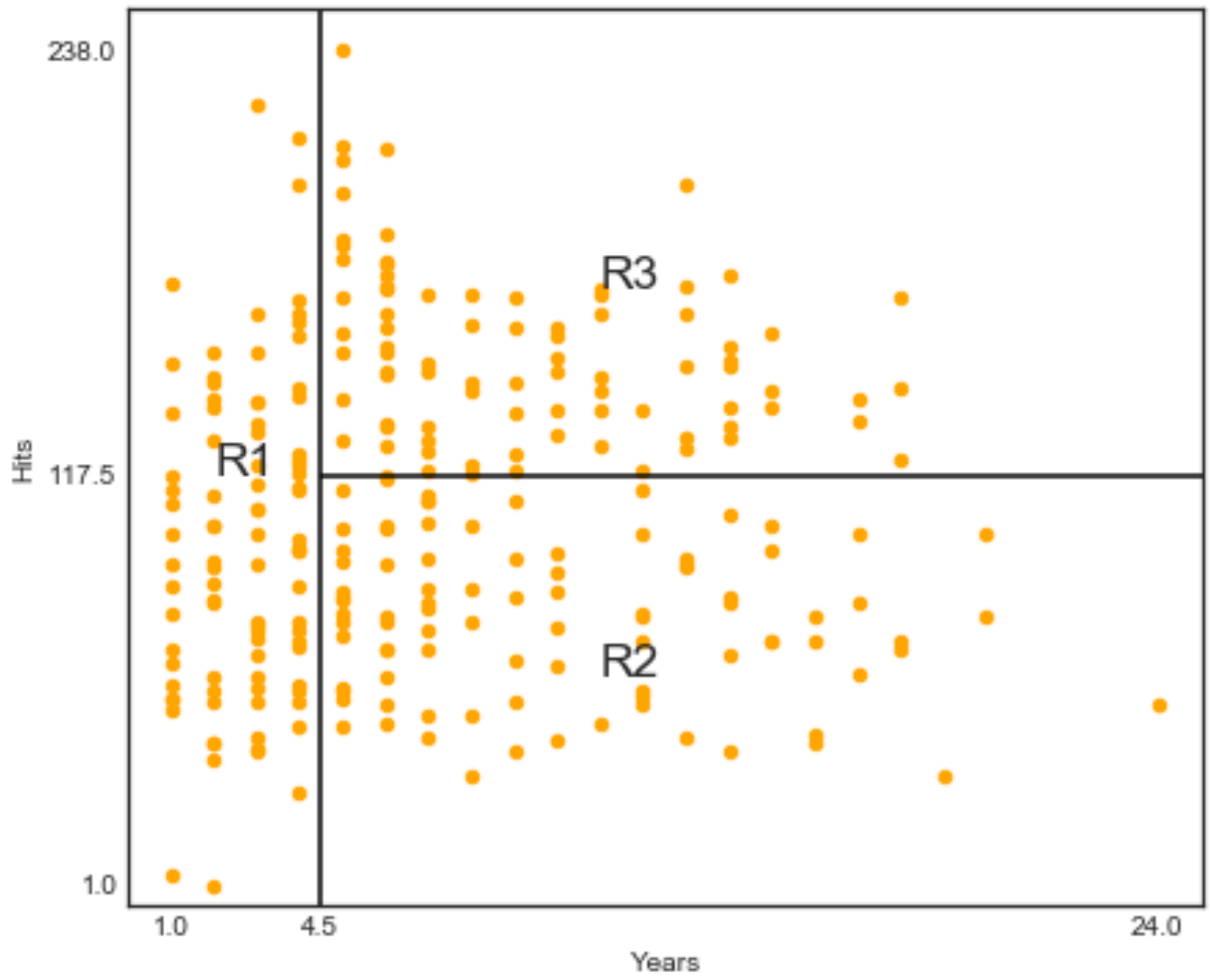


Regression Trees (cont)

How to read this information? The main 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 leaves of the tree.

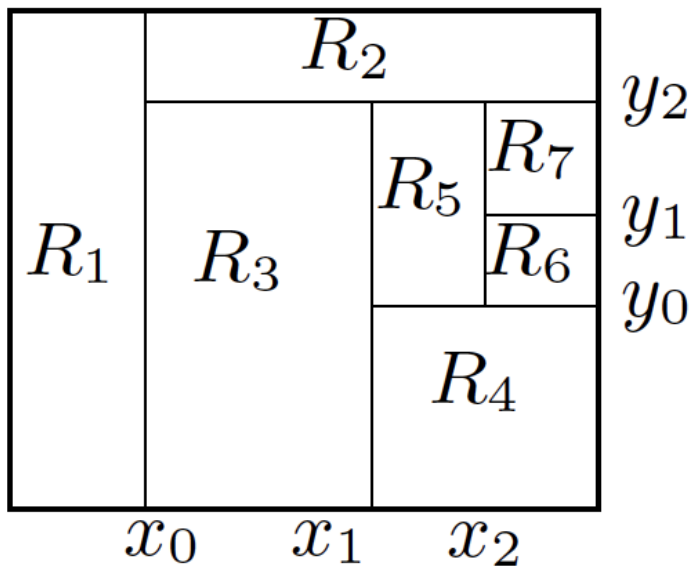
Prediction in Regression Trees

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 $R_1, \ldots, R_J$.
- 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

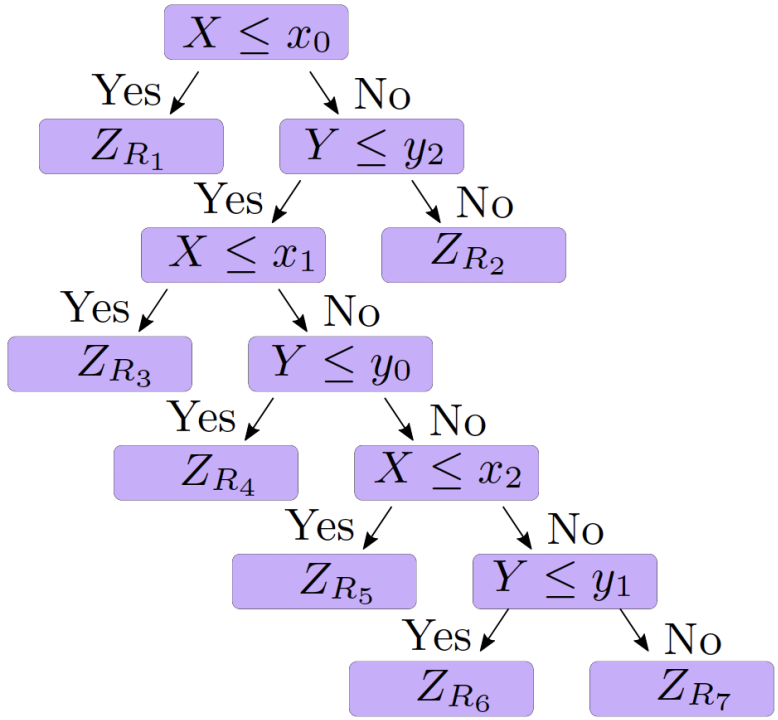
Suppose we have the outcome Z depending on two predictors X and Y defined in certain rectangular area $A \subset X \times Y$ of the plane. One feasible partition of A is depicted below



Feasible Partition

2D Example (cont)

The previous partition can be described as a tree structure



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

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

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

Dividing the Predictor Space

Ideally, one must find regions $R_1, \ldots, R_J$ (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 jth 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 reagion 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 cutpoint s sucha that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greates possible reduction in RSS. Thus, for any j and s, we define a pair of half-planes

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

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

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

We repeat the process looking for the best predictor and best cutpoint 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 to close the data and more likely produce overfit. 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 overfiting 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 say cross-validation.
Once again we faced the problem that the number of possible sub-trees is computationally intensive.
A process known as *weakest link pruning* considers a sequence of trees indexed by a non-negative tuning parameter α. For each value of α there corresponds a sub-tree $T \subset T_0$ such that the following quantity is as small as possible

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

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

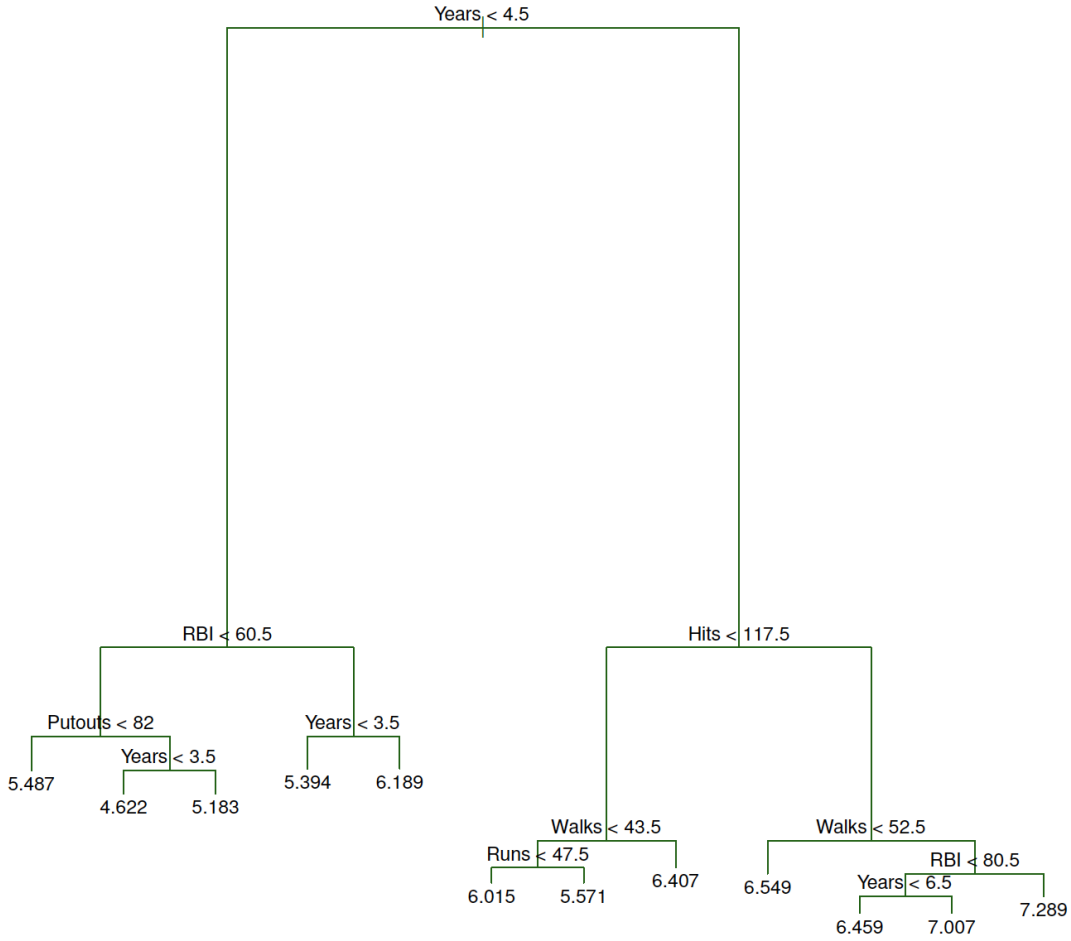
Regression Tree Algorithm

1. 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.
2. 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 α.
3. Use K-fold cross-validation to choose α. More precisely, divide the training observations into K folds, and for $k \in \{1, \ldots, K\}$ do:
   - 3.1 Repeat steps 1 and 2 on all but the kth fold of the training data.
   - 3.2 Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α

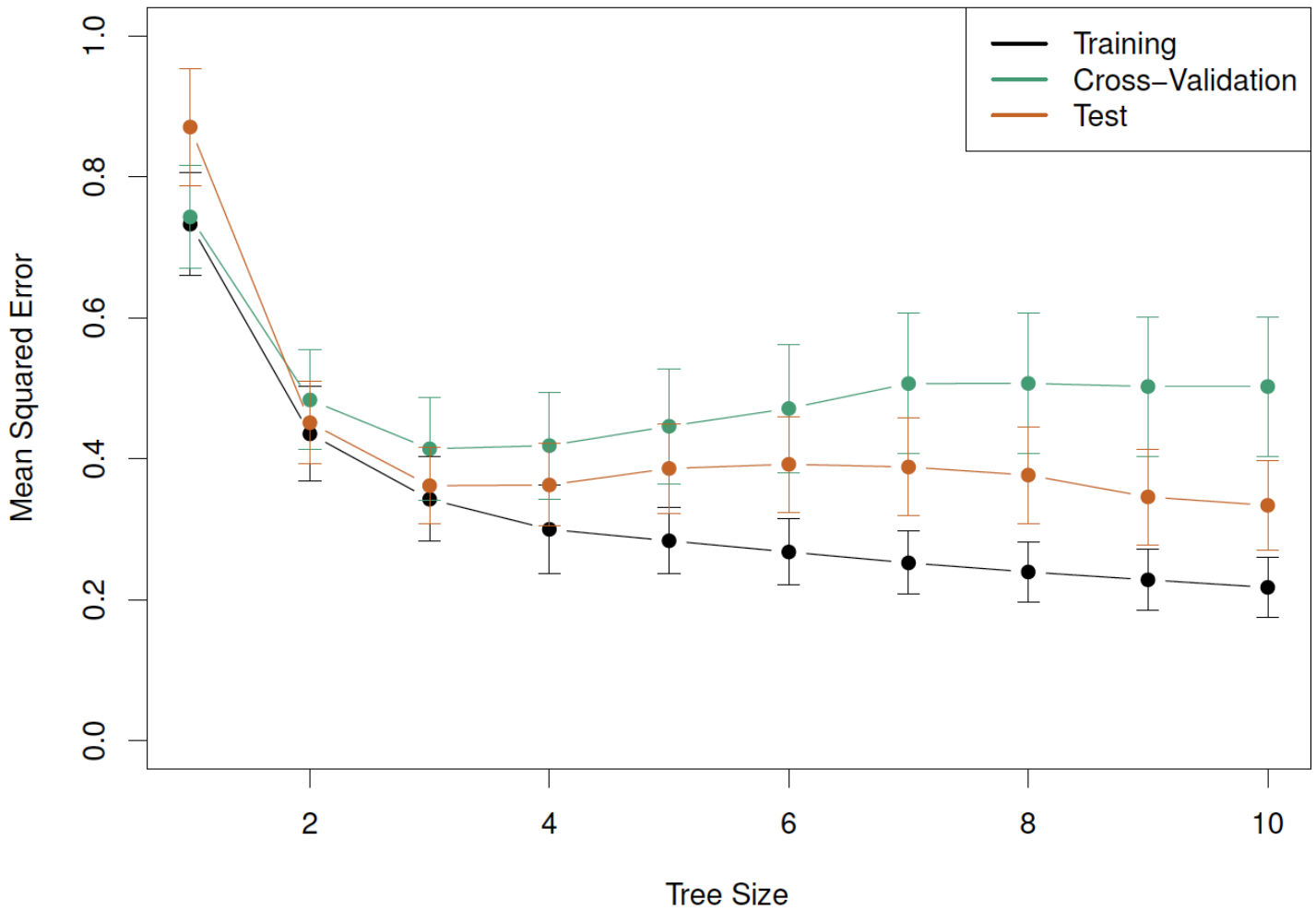
   Average the results for each value of α, and pick α that minimize the average error.
4. Return the sub-tree from step 2 that corresponds to the chosen value of α.

Example.

The unpruned tree that results from greedy splitting on the training data from the **Hitters** dataset 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.

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

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

Purity metrics

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

- **Gini index.** Defined by

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

  is a measure of total variance across the K classes.

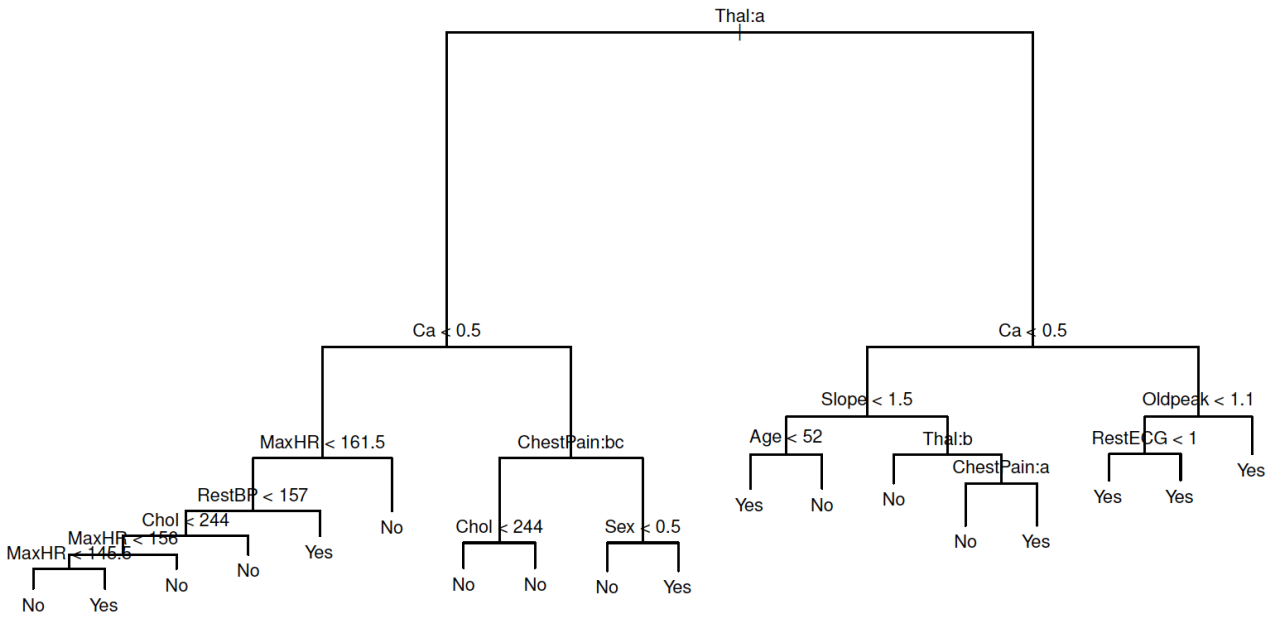
- **Cross-entropy.** Defined by

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

  Note that both metrics will take on a small value if the mth node is pure. Both the Gini index and the cross-entropy are quite similar numerically.

Example

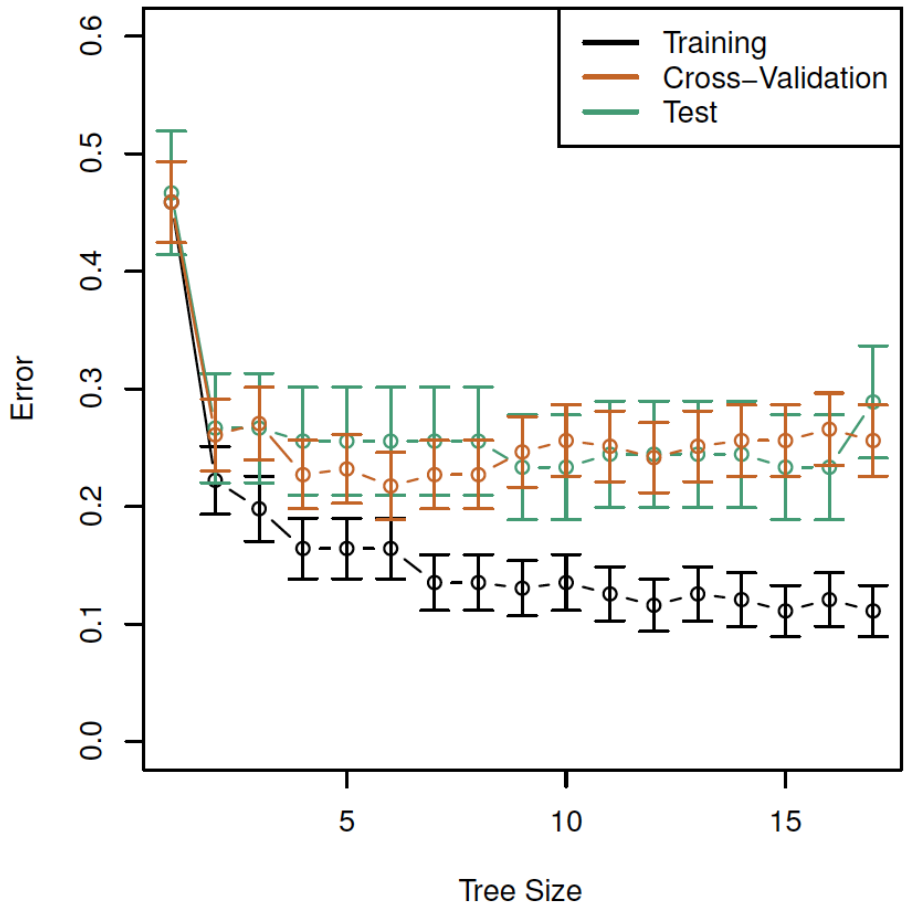
Using the **Heart** data, the classification tree (unpruned) is shown below.



Note the leave `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 describes 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)

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

where G_{left} is the measure of impurity for the *left* subset.
The same algorithm has the following cost function for regression

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

Trees vs. Linear Models

The classical linear regression assumes a relationship of the form

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

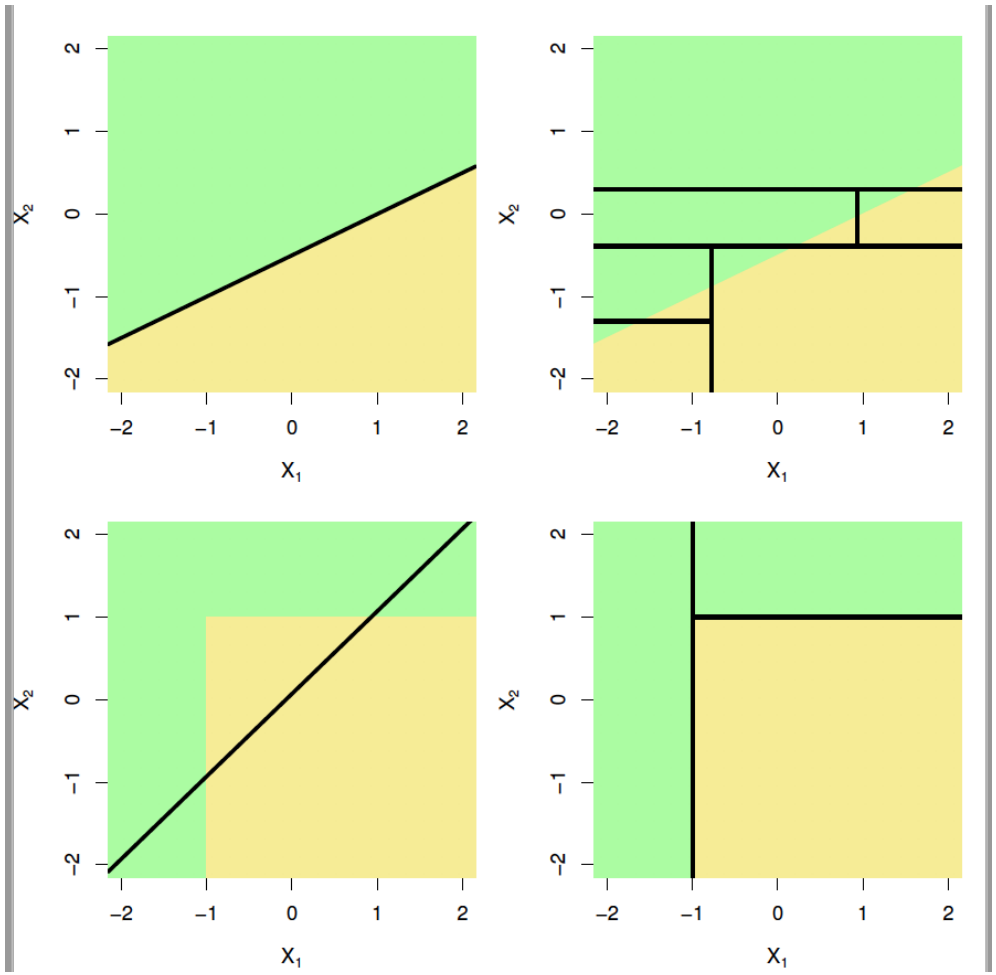
whereas the regression trees assume that the model has the form

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

where $R_1, \ldots, R_M$ 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 dataset.



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.

But

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

References

Materials and some of the pictures are from (1),(2), and (3).

1. Gareth James et al. *An Introduction to Statistical Learning with applications in R*. Springer (2015)
2. Brian D. Ripley *Pattern Recognition and Neural Networks*. Cambridge University Press (1996).
3. Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn & TensorFlow* O'Relly (2017)
4. Wiebe R. Pestman *Mathematical Statistics* de Gruyter (1998)
5. L. Brieman et al. *Classification and Regression Trees*. Wadworth and Brooks/Coke (1994)

I have used some of the graphs by hacking TiKz code from StakExchange, Inkscape for more aesthetic plots and other old tricks of TEX