Lex Analytics

Tool: Classification and Regression Trees (CART)

The Analytics Edge: Classification and regression trees (CART) is a simple yet powerful approach to prediction in machine learning. Unlike linear and logistic regression, CART does not develop a prediction equation. Instead, data are partitioned along the predictor axes into subsets where the most "important" variable in the particular dataset is determined, to help researchers craft a potent explanatory model.

1 The Supreme Court Forecasting Problem

1.1 Background

The US Supreme Court consists of nine lifetime-tenure justices, appointed by the US President. The court handles around 80 cases per year. A decision happens when the majority agrees on an outcome.

Generally, the categories for case selection includes 1) cases of national importance, 2) lower court invalidates federal law, 3) resolve split decision.

Note that in this problem, the responses are discrete. This leads us to think about classification problems.

1.2 The Supreme Court Forecasting Project

This is a study published by Martin et al. (2004), who:

- Used data spanning the period 1994-2001 (longest period with the same justices) → training dataset
- Compared predictions (for the year 2002) made by legal experts and statistical models → testing dataset or validation dataset
- Found very interesting results:
  - Accuracy on the entire court decision: models, 75%; experts, 59.1%
  - Accuracy at the individual justice level: models, 66.7%; experts, 67.9%

The results of the project indicate the power of analytics tools in forecasting the entire court decision.

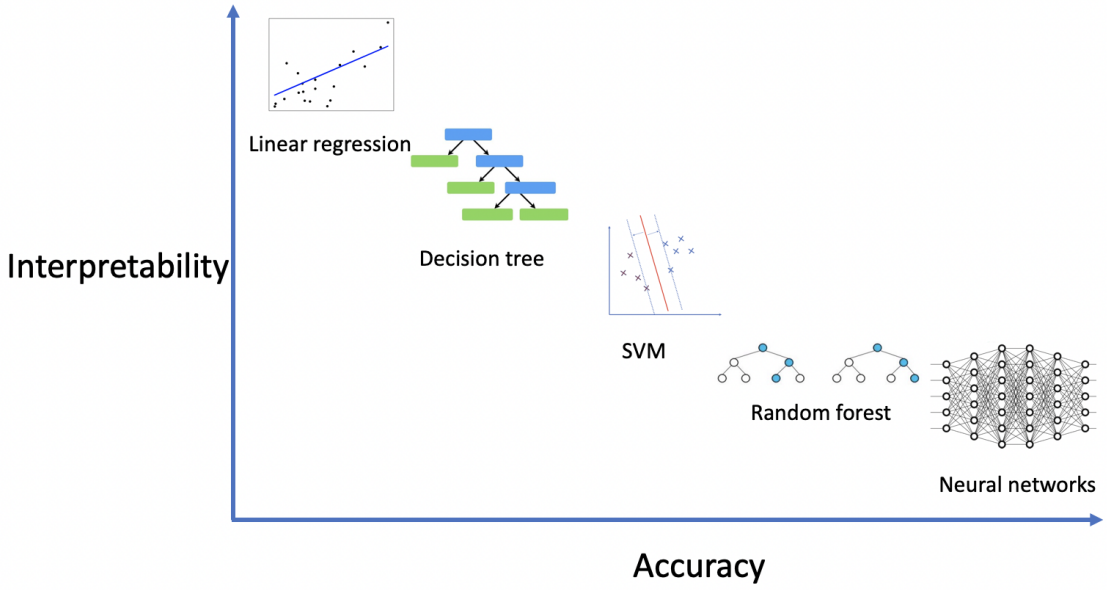
1.3 Description of the dataset

- 623 observations (about 80 cases per year), 20 variables
- Output variable, or predictand: `result`, which takes value 0 (liberal) or 1 (conservative). Liberal: reverse; conservative: affirm
- Input variables, or predictors:
  - `petit`: petitioner type (e.g., US, employer, injured person)
  - `respon`: type of respondent
  - `circuit`: circuit of origin of the case

- `unconst`: whether the petitioner argued the constitutionality of a law of practice
- `lctdir`: ideaological direction of the lower court (liberal or conservative)
- `issue`: issue area of the case

2 Classification And Regression Trees

Decision Trees are an important type of algorithm in supervised learning, where CART is one famous class of decision trees. The following figure shows the interpretability and accuracy of some commonly-used supervised learning algorithms.

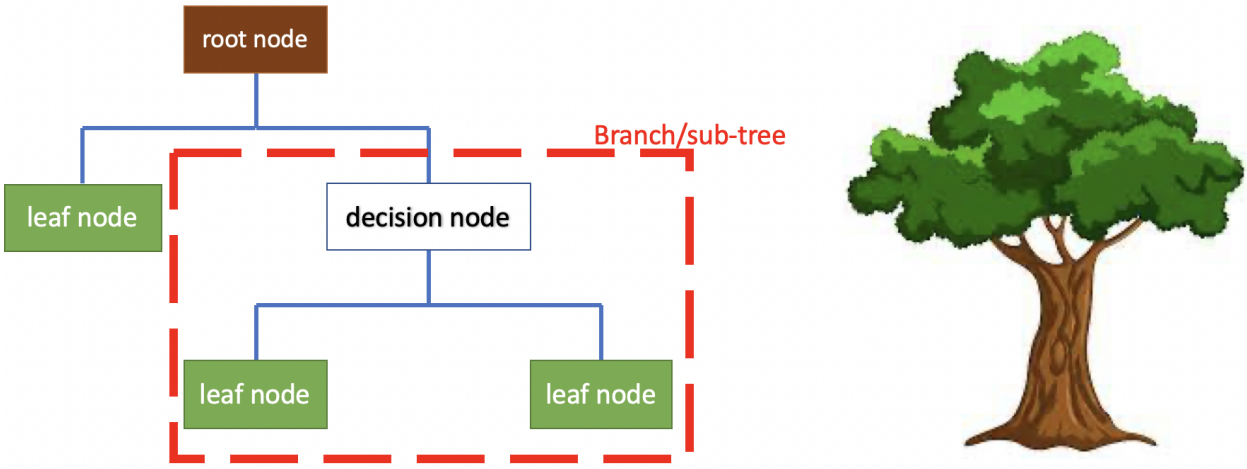


Before going into details of CART, we first introduce some basic knowledge of decision trees.

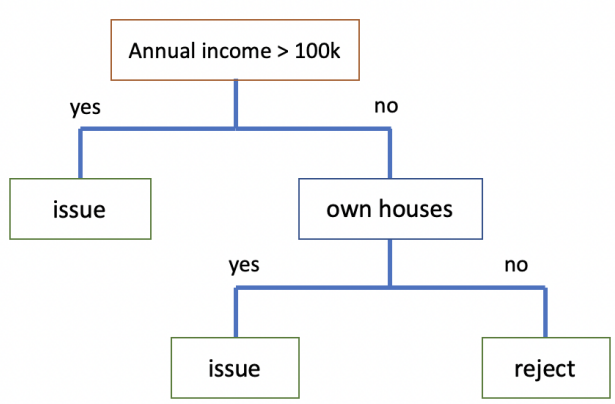
2.1 Decision trees

In the following illustration of a decision tree, there are three kinds of nodes:

- The root node: the node that starts the graph, including all data in the training set
- Decision node: nodes where variables are evaluated but which are not the final nodes where predictions are made
- Leaf nodes: final nodes of the tree, where the predictions are made.



We also give a simple example of decision trees to answer the question "how a bank determine whether to issue loans to customers or not". A possible strategy is shown as follows.



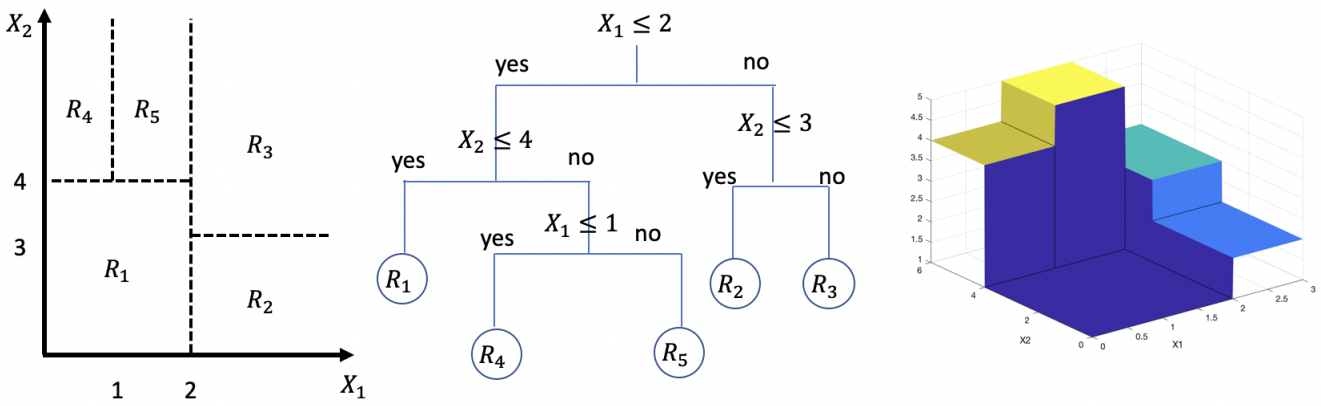
Decision Trees can be applied to both regression and classification problems. The term "Classification And Regression Tree (CART)" is used to refer to procedures that learn a Classification or Regression Tree. We begin by considering Regression Trees

2.2 Regression Trees

2.2.1 Intuition

Suppose we are working on a regression problem with response variable Y and predictors X_1 and X_2. The underlying idea of Regression Trees is to divide, or partition, the predictor space into a number of regions, where we then apply a simple model.

Example:



2.2.2 Problem setup

Given a dataset $\{(X_1, y_1), \cdots, (X_n, y_n)\}$, with variable $X_i \in \mathbb{R}^p$, response y_i. The goal of a regression tree is to construct a function $f(\cdot)$ to minimize RSS:

$$\min \sum_{i=1}^{n} (f(X_i) - y_i)^2.$$

2.2.3 Main steps to learn the regression tree

There are two main steps.

Step 1. Partition the predictor space into J distinct and non-overlapping regions $(R_1, R_2, \ldots, R_J)$.

Step 2. For every observation that falls into the j-th region R_j, we make the same prediction c_j.

Then the estimated function $\hat{f}(\cdot)$ can be given in the form:

$$\hat{f}(X) = \sum_{j=1}^{J} c_j 1_{R_j}(X),$$

where $I_A(\cdot)$ is the indicator function of the set A defined as

$$I_A(z) = \begin{cases} 1 & z \in A \\ 0 & \text{otherwise} \end{cases}.$$

We give some discussions on how to implement the two steps in practice. In **Step 2**, we need to solve

$$\min_{c_1,\ldots,c_J} \sum_{j=1}^{J} \sum_{i:X_i \in R_j} (y_i - c_j)^2.$$

Based on the criterion minimization of the sum of squares, c_j takes the mean of the response values for the observations in R_j, that is

$$c_j = \text{average}(y_i | X_i \in R_j).$$

In **Step 1**, the problem of partitioning the predictor space into J regions can be formulated as follows:

$$\min_{R_1,\ldots,R_J} \sum_{j=1}^{J} \sum_{i:X_i \in R_j} (y_i - c_j)^2.$$

In general, the problem is computationally unfeasible. To solve it, we use a top-down, greedy approach known as **recursive binary splitting**, which is a heuristic algorithm to find $R_1, \cdots, R_J$.

2.2.4 Recursive binary splitting

- Start with all variables in one region
- Consider all predictors $X^{(1)}, \ldots, X^{(p)}$ and all possible values of the cut (or split) point s, and choose the predictor and cut point s.t. the resulting partition has the lowest RSS.

  Given the k-th predictor and the cut point s, we define the following half-planes

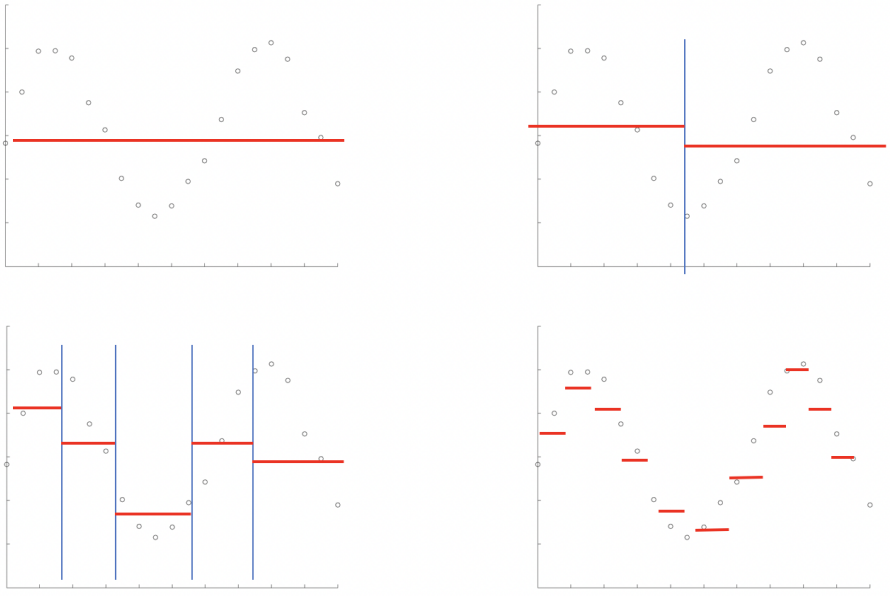
  $$R_1(k,s) = \{X | X^{(k)} < s\} \text{ and } R_2(k,s) = \{X | X^{(k)} \geq s\},$$

  And we seek the value of k and s that minimizes

  $$\underbrace{\sum_{i:X_i \in R_1(k,s)} (y_i - c_1)^2}_{\text{error in } R_1} + \underbrace{\sum_{i:X_i \in R_2(k,s)} (y_i - c_2)^2}_{\text{error in } R_2}.$$

  Specifically, we first fix k, find the best s; then we get p different policies, choose the one with the lowest RSS.
- We repeat the process, splitting one of the two previously identified regions. The process continues until an exit condition is met (e.g., minimum number of points in each region)

2.2.5 Illustration of regression tree



2.3 Classification Trees

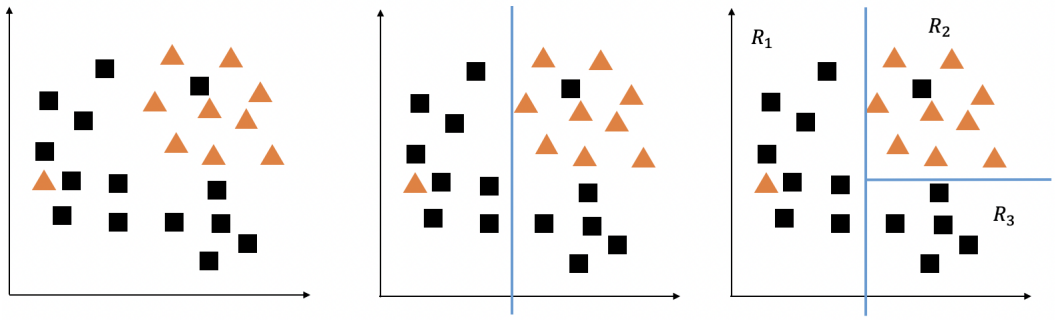
Similarities with Regression Trees:

- The representation for the CART model is a binary tree, where each node (except leaf nodes) has two child nodes.
- The predictions in one leaf node are the same.

Two differences w.r.t. Regression Trees:

- For each region, the prediction c_j is the most commonly occurring class
- When learning a tree, we cannot use the RSS. Instead, we use a **measure of impurity** (a split is pure if, for all branches, all the instances choosing a branch fall within the same class)

Here is a simple example of classification trees.



2.3.1 Measures of impurity

1. Classification error rate:

$$E = 1 - \max_k(p_{mk})$$

where p_{mk} is the proportion of training observations in the m-th region that are from the k-th class.

   - $N_m = \#$ instances in the region R_m
   - $N_{mk} = \#$ instances in the region R_m belonging to class k
   - $p_{mk} = \frac{N_{mk}}{N_m}$

2. Gini index:

$$G = \sum_{k=1}^{K} p_{mk}(1 - p_{mk})$$

where K is the total number of classes, and G varies between 0 and 0.5.

Example: $K = 2$ classes, 10 instances in the region R_m:

Case 1: (Best case) all instances are from class 1,

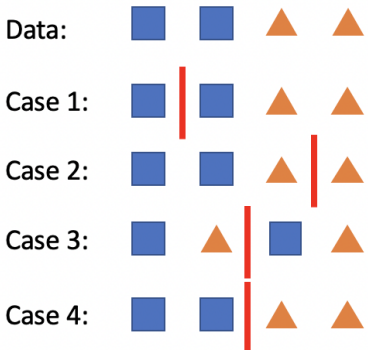
$$p_{m1} = 1,\ p_{m2} = 0,\ G = 1(1-1) + 0(1-0) = 0$$

Case 2: (Worst case) 5 instances are from class 1, 5 are from class 2,

$$p_{m1} = \frac{1}{2},\ p_{m2} = \frac{1}{2},\ G = \frac{1}{2}(1-\frac{1}{2}) + \frac{1}{2}(1-\frac{1}{2}) = \frac{1}{2}$$

If we partition the points into 2 regions, the total Gini index is

$$G = \frac{\#instances\ in\ Region\ 1}{\#total\ instances} \times G(\text{Region 1}) + \frac{\#instances\ in\ Region\ 2}{\#total\ instances} \times G(\text{Region 2})$$



Case 1: G(Region 1) = 0, G(Region 2) = $\frac{4}{9}$, G = $\frac{1}{4}\times0 + \frac{3}{4}\times\frac{4}{9} = \frac{1}{3}$

Case 2: G(Region 1) = $\frac{4}{9}$, G(Region 2) = 0, G = $\frac{3}{4}\times\frac{4}{9} + \frac{1}{4}\times0 = \frac{1}{3}$

Case 3: G(Region 1) = $\frac{1}{2}$, G(Region 2) = $\frac{1}{2}$, G = $\frac{1}{2}\times\frac{1}{2} + \frac{1}{2}\times\frac{1}{2} = \frac{1}{2}$ Worst case

Case 4: G(Region 1) = 0, G(Region 2) = 0, G = $\frac{1}{2}\times0 + \frac{1}{2}\times0 = 0$ Best case

3. Entropy:

$$D = -\sum_{k=1}^{K}(p_{mk}\log_2(p_{mk})).$$

Since $0 \le p_{mk} \le 1$, and D varies between 0 and 1. (Note: let $0\log_2 0 = 0$.)

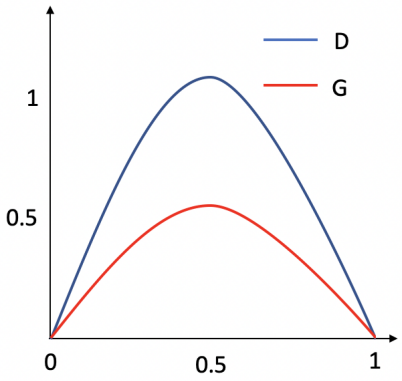
Example: $K = 2$ classes, 10 instances in the region R_m:

Case 1: (Best case) all instances are from class 1,

$$p_{m1} = 1,\ p_{m2} = 0,\ D = -1\log_2 1 - 0\log_2 0 = 0$$

Case 2: (Worst case) 5 instances are from class 1, 5 are from class 2,

$$p_{m1} = \frac{1}{2},\ p_{m2} = \frac{1}{2},\ D = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1$$



2.4 Pruning

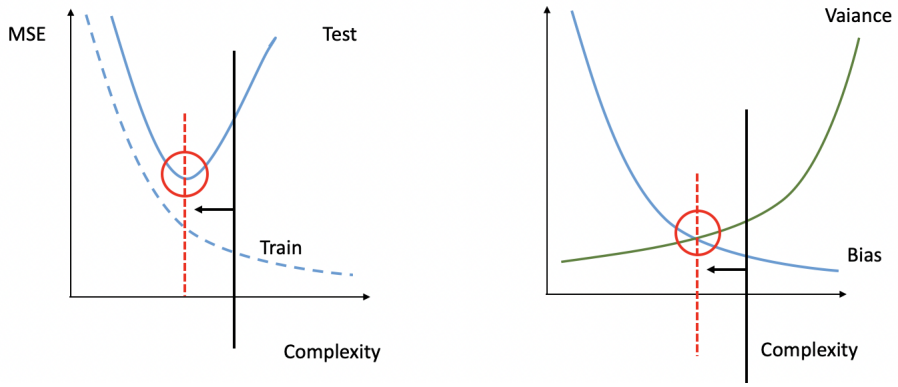
The CART's learning algorithm is likely to build complex trees that overfit the training data. A smaller tree (with fewer regions $R_1, \cdots, R_J$) may lead to lower variance at the cost of a little bias.

2.4.1 Bias-Variance tradeoff

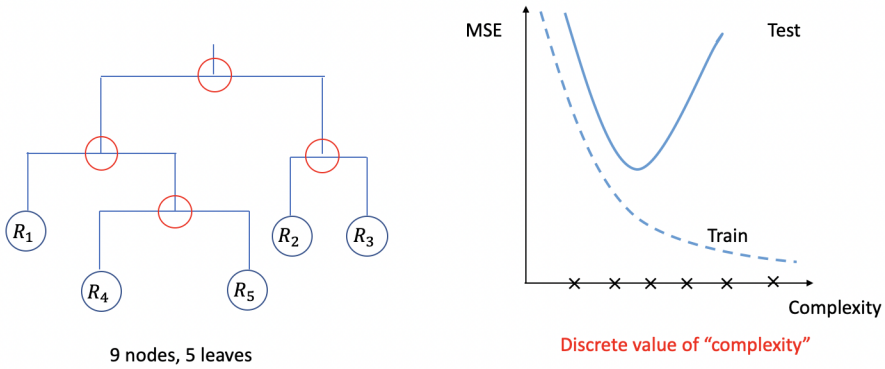
Suppose we are going to fit a function $f(\cdot)$ such that $y = f(x) + \varepsilon$, where the random variable ε satisfies $\mathbb{E}[\varepsilon] = 0$, $\text{Var}[\varepsilon] = \sigma^2$. Let $\hat{y} = \hat{f}(x)$ be a model fitted by solving $\min(y - \hat{f}(x))^2$. Then for any fixed x_0, we have

$$\begin{aligned}\mathbb{E}[(y_0 - \hat{f}(x_0))^2] &= \Big(\mathbb{E}[\hat{f}(x_0)] - f(x_0)\Big)^2 + \text{Var}[\hat{f}(x_0)] + \sigma^2 \\ &= \text{Bias}^2 + \text{Variance} + \text{Random Error}\end{aligned}$$

- Bias: the difference between the average prediction of our model and the correct value which we are trying to predict
  – the fitting performance of model
- Variance: captures how much your performance changes if you train on a different training set
  – the effect of data perturbation



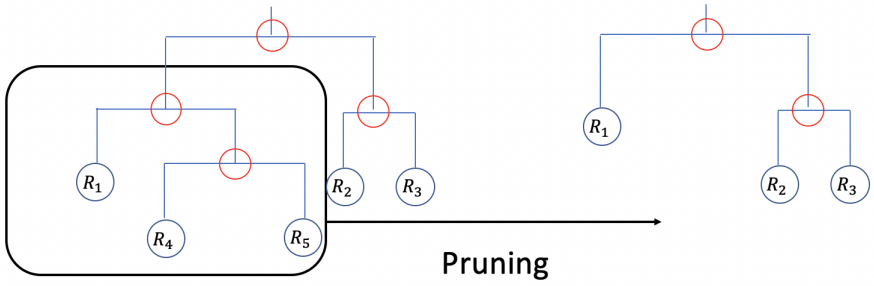
As for the CARTs, we can treat the complexity of a CART as the "Depth" or "Width" of the tree , or the number of leaves in the tree. Therefore, the complexity of CARTs usually takes discrete values.



In order to reduce the complexity of CARTs, we need an important technique called pruning.

2.4.2 Pruning

The idea of pruning can be seen in the following figure.



There are two different types of pruning: early stopping and pruning. Sometimes these are referred to simplistically as pre-pruning and post-pruning.

- Early stopping or pre-pruning:

  Try and stop the tree-building process early, before it produces leaves with very small number of observations.

  - At each stage of splitting the tree, we check the cross-validation error.
  - If the error does not decrease significantly enough, then we stop.
  - Early stopping may underfit by stopping too early
    – The current split may be of little benefit, but having made it, subsequent splits may significantly reduce the error.

- Pruning or post-pruning: cutting back the tree
  – usually lower risk of underfitting, higher test accuracy, more branches, longer computational time

In this course, we mainly focus on the second type. The idea of pruning is first grow a large tree T_0 and then prune it back to obtain a subtree. To determine how to prune the tree, we can use the cross-validation error. Not that we cannot calculate the cross-validation error for all trees, because there are too many subtrees and it would take too long!

A commonly used method is called cost complexity pruning (or weakest link pruning). We explain the idea by regression trees. For each value of a nonnegative tuning parameter α, there corresponds a subtree $T \subset T_0$ s.t. the value

$$\underbrace{\sum_{m=1}^{|T|} \sum_{i:X_i \in R_m} (y_i - c_m)^2}_{\text{RSS: measure accuracy}} + \quad \alpha \quad \underbrace{|T|}_{\text{Complexity}}$$

is as small as possible. Here $|T|$ means the number of leaves in T. We can see that if $\alpha = 0$, $T = T_0$, and if α increases, $|T|$ decreases (we pay a price for building a tree with many leaves). Note that the above expression is a reminiscent of the LASSO, which controls the complexity of a linear model (Week 5, Lecture 2).

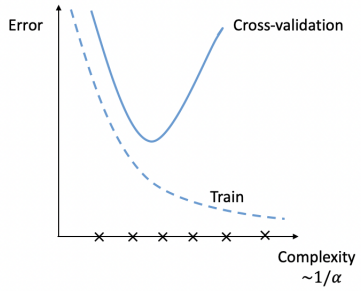
Here is the (Full) algorithm for building a regression tree.

Step 1 Use recursive binary splitting to grow a large tree on the training data

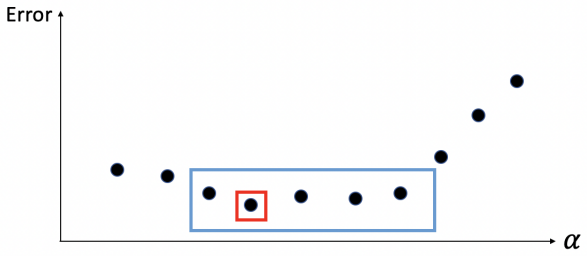
Step 2 Use cost complexity pruning to obtain a sequence of subtrees as a function of α

Step 3 Use k-fold cross-validation to choose the best value of α

Step 4 Return the subtree from Step 2 that corresponds to the chosen α



Then we need to know how to choose the best value of α. A commonly used method is to choose the value of α with the smallest k-fold cross-validation error. While in many cases, we might see the situation in the following figure.



It is obvious that the dot in the red box has the smallest k-fold cross-validation error. Note that the other dots in the blue box have similar cross-validation errors, while some of them lead to trees with smaller complexity. In some cases, we may want to pay a price of some increase in cross-validation error to get trees with smaller complexity. Thus there is another method called "One-standard error" rule, to choose the largest α whose cross-validation error is still within a SE of the minimum possible cross-validation error.

When pruning a Classification Tree, we follow the same procedure, keeping in mind that the model error is calculated with a measure of impurity. This leads to the following expression

$$\underbrace{\sum_{m=1}^{|T|} E_m}_{\text{Error: measure accuracy}} + \alpha \underbrace{|T|}_{Complexity}$$

which we still want to minimize.

2.5 Advantages and Disadvantages of CARTs

Pros:

- Interpretability
- Can be displayed graphically
- Can handle qualitative predictors (that take no continuous values)
- No assumptions on the relationship between input and output variables

Cons:

- They are not very accurate
- Not robust

3 Back to R!

To learn CARTs, we will use the function `rpart`, implemented in the package ... `rpart`:

`model <- rpart(formula, data, method, ...)` – fit CART

`print(model)` or `summary(model)` – print details of CART

`prp(model,type)` or `plot(model);text(model)` – visualize CART

`predict(model,newdata,method)` – prediction via CART

`printcp(model)` or `plotcp(model)` – the table or plot of α

`cp <- model$cptable[which.min(model$cptable[,"xerror"]),"CP"]` – optimal α (by Method 1)

`model2 <- prune(model,cp)` – prune CART with given α