Non-Linear Regression algorithms

Here we are discussing Non-Linear regression algorithms which are used particularly when there is **a high non-linearity & complex relationship between regressors(X) & predictors(Y)** then these models will outperform a classical regression method.

Here we are interested in predicting the outcome Y using the raw regressors Z, which are k-dimensional or k-regressors. So the best prediction rule is,

$$g(Z) = \mathrm{E}(Y|Z)$$

So far we have used linear methods to approximate g(Z). Now we are going to consider nonlinear prediction rules to approximate g(Z).

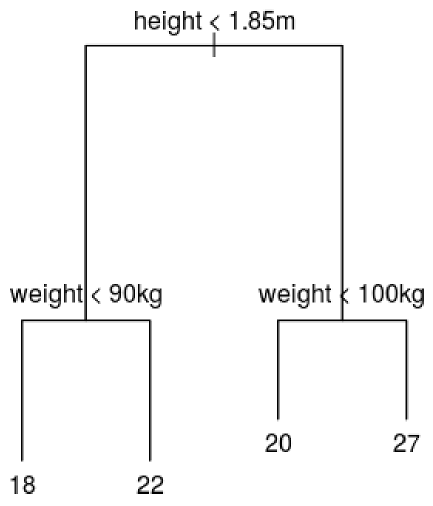
Let's start with tree-based algorithms,

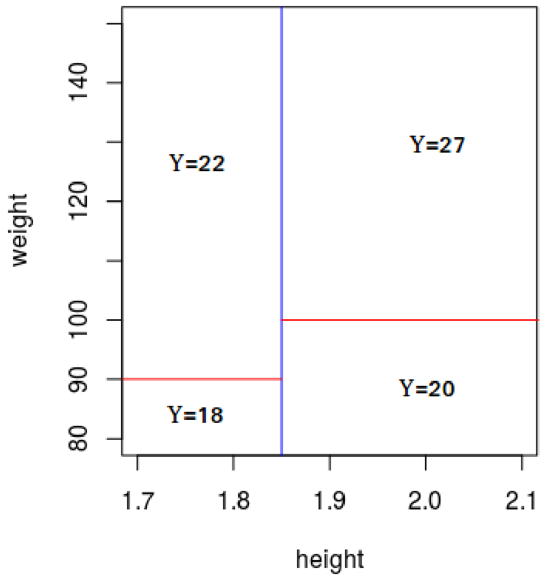
Regression Trees :

The idea of Regression Trees is to partition the regressor space, where Z takes values, into a set of rectangles, and then for each rectangle provide a predicted value.

Let's understand the definition with an example,
Suppose, for example, that we have the number of goals scored by a set of basketball players and we want to relate it to the player's weight and height. The Regression Tree will simply split the height-weight space and assign a number of points to each partition.

The figure below shows two different representations of a small tree. On the left, we have the tree itself and on the right how the space is partitioned (the blue line shows the first partition and the red lines the following partitions). The numbers at the end of the tree (and in the partitions) represent the value of the response variable. Therefore, if a basketball player is less than 1.85 meters and weighs more than 90kg it is expected to score 22 points.





In the example above we have 4 rectangles or the data has been split into four parts.

Formally,
Suppose we have n observations (Z_i, Y_i) for i ranging from 1 to n. Given a partition of the space into M rectangles R_1 through R_m, which will be determined by data, the regression tree is a prediction rule of the form:

$$\hat{g}(Z) = \sum_{m=1}^{M} \hat{\beta}_m 1(Z \in R_m)$$

Each instance or each observation falls into exactly one leaf node or one rectangle R_m. $1(Z \in R_m)$ returns 1 if Z is in the subset R_m and 0 otherwise. $\hat{\beta}_m$ is the predicted value for the region m.

For example, if we need to find the prediction with height=1.75 and weight= 75 then

$$\hat{g}(Z) = (18 \times 1) + (22 \times 0) + (20 \times 0) + (27 \times 0)$$
$$= 18$$

The predicted values $b = (b_1, b_2 ...... b_m)$ are obtained by minimizing the sample Mean squared error (MSE),

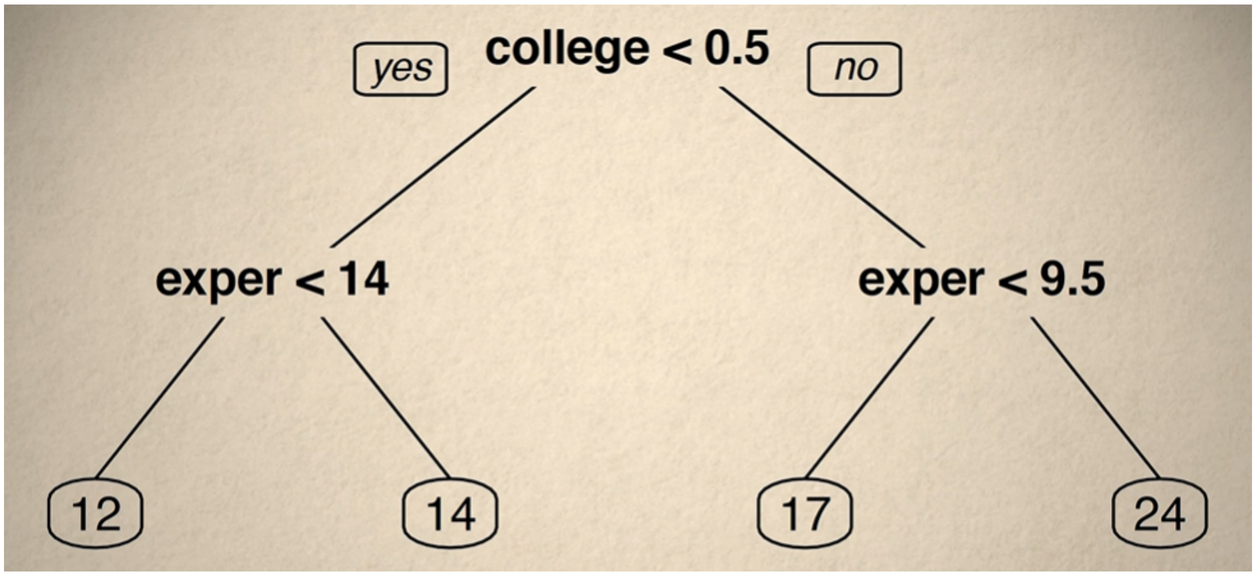
$$\hat{\beta} = \arg\min_{b_1,\ldots,b_M} \sum_{i=1}^{n} \left(Y_i - \sum_{m=1}^{M} b_m 1(Z_i \in R_m) \right)^2$$

From the formula, we can conclude that $\widehat{\beta}_m$ is set equal to the average of Y_i's with Z_i's falling in the rectangle Rm. Whenever we have multiple observations falling under the same rectangle then the predicted value is the average of all the values in the rectangle. The rectangles or regions Rm are called nodes, and each node has a predicted value $\widehat{\beta}_m$ associated with it.

A nice thing about regression trees is that you get to draw cool pictures and their outputs are easy to read and interpret without requiring statistical knowledge.

Example :

Let's take another example and understand more clearly,



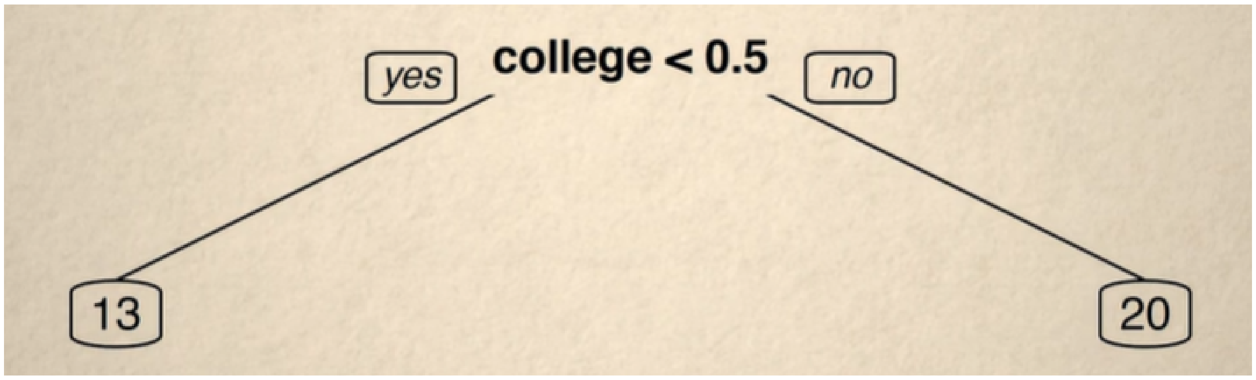
Here we show a tree-based prediction rule for our Wage Example,

where Y is wage, and Z is experience, geographic, and educational characteristics. As you can see from the looking at the terminal nodes of the tree, in this tree the predicted hourly wage for college graduates with more than 9.5 years of experience is 24 dollars, and otherwise, it is 17; the predicted wage for non-college graduates with more than 14 years of experience is 14 dollars and otherwise, it is 12 dollars.

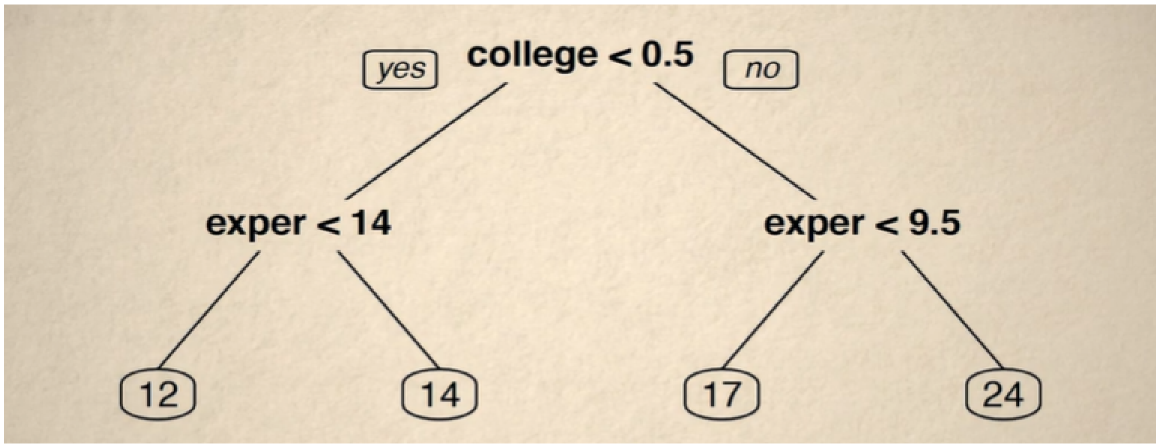
Growing the tree:

To make computation easy, we repeatedly split into two outcomes until all regressors are completed which is called recursive binary partitioning.

First, we cut the regressor space into two regions by choosing a regressor and a split point that achieve the best improvement in sample MSE. This gives us the tree of depth 1. The best variable to split on here is the indicator of a college degree, and it takes values of 0 or 1, so the natural split point is .5. This gives us a starting tree-based prediction rule, which predicts 20 dollars hourly wage for college graduates and 13 for all others.

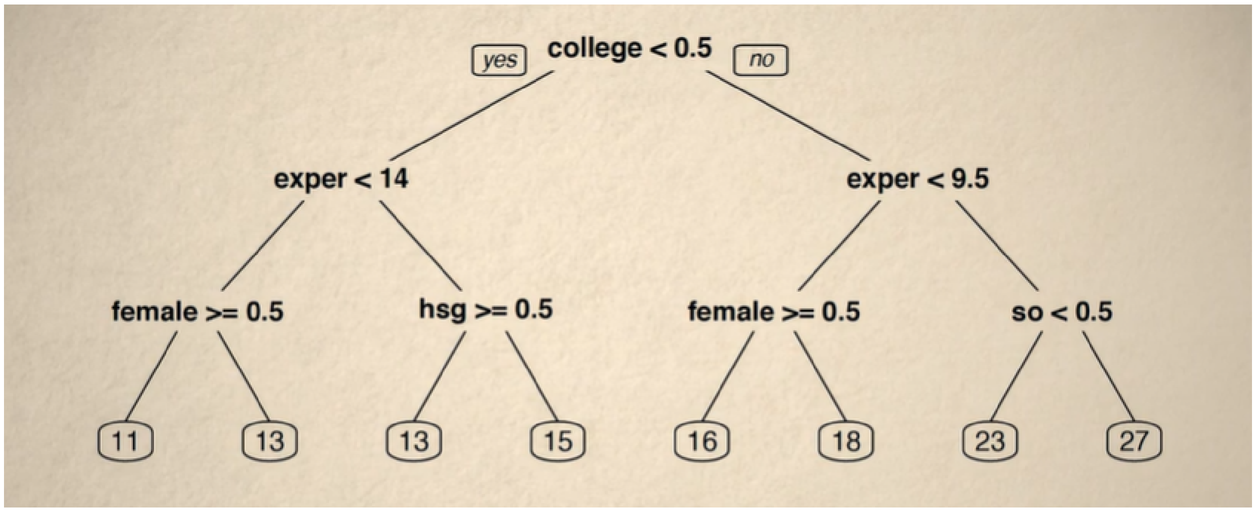


Second, we repeat this procedure over two resulting regions or nodes (college graduates and non-college graduates), obtaining in this step four new nodes that we see in this below figure,



For college graduates, the splitting rule that minimizes MSE is the experience regressor at 9.5 years. This refines the prediction rule for graduates to 24 dollars if the

experience is greater than 9.5 years, and 12 dollars otherwise. For non-graduates, the procedure works similarly.
Third, we repeat this procedure over each of the four nodes. The tree of depth 3 now has 8 nodes. We see that in the final level we are now splitting on gender indicator, high-school graduate indicator, and the “South” indicator.



Finally, we stop growing the tree, when the desired depth of the tree is reached, or, when the minimal number of observations per node, called minimal node size, is reached.

Pruning Trees:
Pruning a decision tree helps to prevent overfitting the training data so that our model generalizes well to unseen data. Pruning a decision tree means removing a subtree that is redundant and not a useful split and replacing it with a leaf node.

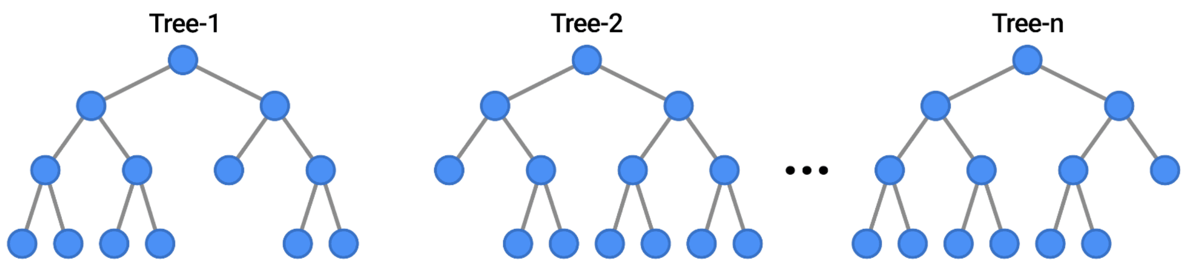
First, the deeper we grow the tree, the better is our approximation to the regression function g(Z). On the other hand, the deeper the tree, the noisier our estimate $\widehat{g}(Z)$ becomes since there are fewer observations per terminal node to estimate the predicted value for this node. From a prediction point of view, we can try to find the right depth or the structure of the tree by cross-validation.

For example, in the wage example, the tree of depth 2 performs better in terms of cross-validated MSE than the trees of depth 3 or 1. The process of cutting down the branches of the tree to improve predictive performance is called “Pruning the Tree”.

However, in practice, pruning the tree often does not give satisfactory performance, because a single pruned tree provides a very crude approximation to the regression function g(Z).

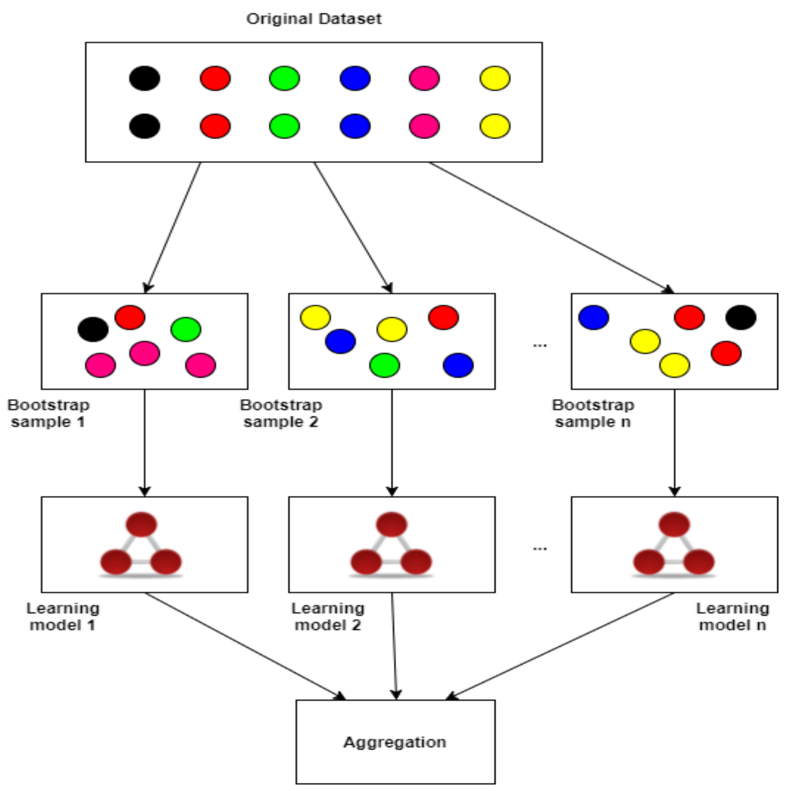
Random Forest:

A much more powerful and widely used approach to improve simple regression trees is to build the Random Forest. A Random Forest operates by constructing several decision trees during training time and outputting the mean of the classes as the prediction of all the trees.



The trees are grown over different artificial data samples by sampling randomly with replacement from the original data. This way of creating artificial samples is called the Bootstrap in statistics. The trees are grown deep to keep the approximation error low, and averaging is meant to reduce the noisiness of individual trees.

Bootstrapping is a statistical resampling technique that involves random sampling of a dataset with replacement.



Formally,
Given a bootstrap sample, indexed by b, we build a tree-based prediction rule. We repeat the procedure B times, and then average the prediction rules that result from each of the bootstrap samples:

$$\hat{g}_{random\ forest}(Z) = \frac{1}{B}\sum_{b=1}^{B}\hat{g}_b(Z)$$

Using bootstrap here is an intuitive idea: if we could have many independent copies of the data, we could average the prediction rules obtained over these copies to reduce the noisiness. Since we don't have such copies, we rely on bootstrap copies of the data.

The key underlying idea here is that the trees are grown deep to keep the approximation error low, and averaging is meant to reduce the noisiness of individual trees. The procedure of averaging noisy prediction rules over the bootstrap samples is called Bootstrap Aggregation or Bagging.

Boosting:

Boosting consists of the idea of filtering or weighting the data that is used to train our team of weak learners, so that each new learner gives more weight or is only trained with observations that have been poorly classified by the previous learners. By doing this our team of models learns to make accurate predictions on all kinds of data, not just on the most common or easy observations. Also, if one of the individual models is very bad at making predictions on some kind of observation, it does not matter, as the other N-1 models will most likely make up for it.

Boosting should not be confused with Bagging, while in bagging the weak learners are trained in parallel using randomness, in boosting the learners are trained sequentially, in order to be able to perform the task of data weighting/filtering described in the previous paragraph.

We estimate a tree-based prediction rule, then we take the residuals and estimate another tree-based prediction rule for these residuals, and so on. Summing up the prediction rules for the residuals gives us the prediction rule for the outcome. Unlike in random forests, we use shallow trees rather than deep trees, to keep the noise low, and each step of boosting aims to reduce the approximation error. In order to avoid overfitting in boosting, we can stop the procedure, once we don’t improve the cross-validated MSE.

Boosting Algorithm :

Formally, the boosting algorithm looks as follows.

Step1 : We initialize the residuals Ri = Yi for i = 1, ..., n

In step 1 first, we take the average of all the values and subtract from the original Yi to get the residuals and now we are going to fit the model for these residuals.

Step2: For j=1,....,. J

1. Fit a tree-based prediction rule $\widehat{g_j}(Z)$ to the data $(Z_i, R_i)_{i=1}^{n}$
2. Update the residuals $R_j \leftarrow R_i - \lambda\widehat{g_j}(Z_i)$

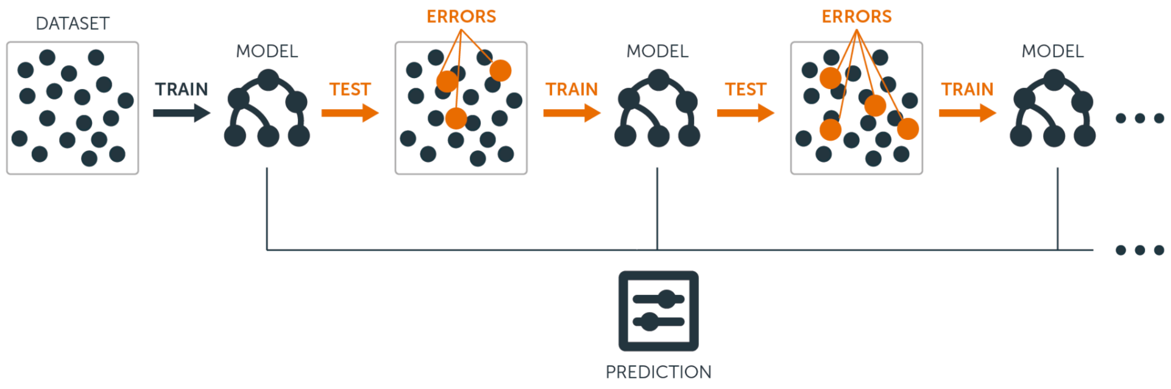
In step 2 we are going to fit a tree-based prediction rule to the data $(Z_i, R_i)_{i=1}^{n}$ and once we get the predictions we are going to update the residuals.

The capital J and λ are the tuning parameters here, representing the number of steps and the degree of updating the residuals. We can choose them by cross-validation.

Step3: Construct a boosted prediction rule,

$$\hat{g}(Z) = \sum_{j=1}^{J} \lambda \hat{g}_j(Z)$$

In step 3 we output the boosted prediction rule by taking a summation of the number of times we have fit the model.



To summarize, Boosting uses recursive fitting of residuals by a sequence of tree-based prediction models. The sum of these prediction rules gives the prediction for the outcome.