BIOST 527 Final project: Regression tree

Yichen Lu

6/2020

1 Introduction

Throughout this quarter, we have studied many non-parametric estimators in class. In this report, another non-parametric method for predicting continuous outcome will be introduced: *regression tree*.

In this section, we will get familiar with the intuition behind the model. A sample of $n = 100$ observations with (X_i, y_i) where $X_i = (x_{i1}, x_{i2})$ is generated as follows:

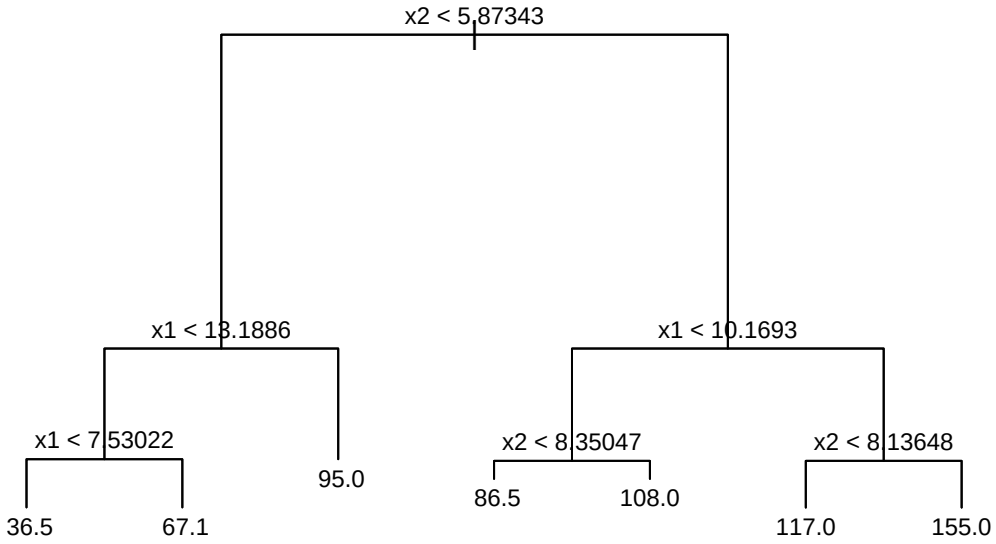
$$x_1 \overset{i.i.d.}{\sim} N(10, 5)$$
$$x_2 \overset{i.i.d.}{\sim} Uniform(0, 10)$$
$$f(x_1, x_2) = 5x_1 + x_2^2$$
$$y = f(x_1, x_2) + \epsilon, \epsilon \overset{i.i.d.}{\sim} Normal(0, 1)$$

A regression tree is fitted to these observations and **Graph 1a** and **Graph 1b** present two ways of visualizing the result. Without much instruction, the audience can easily figure out how to model y for an observation.

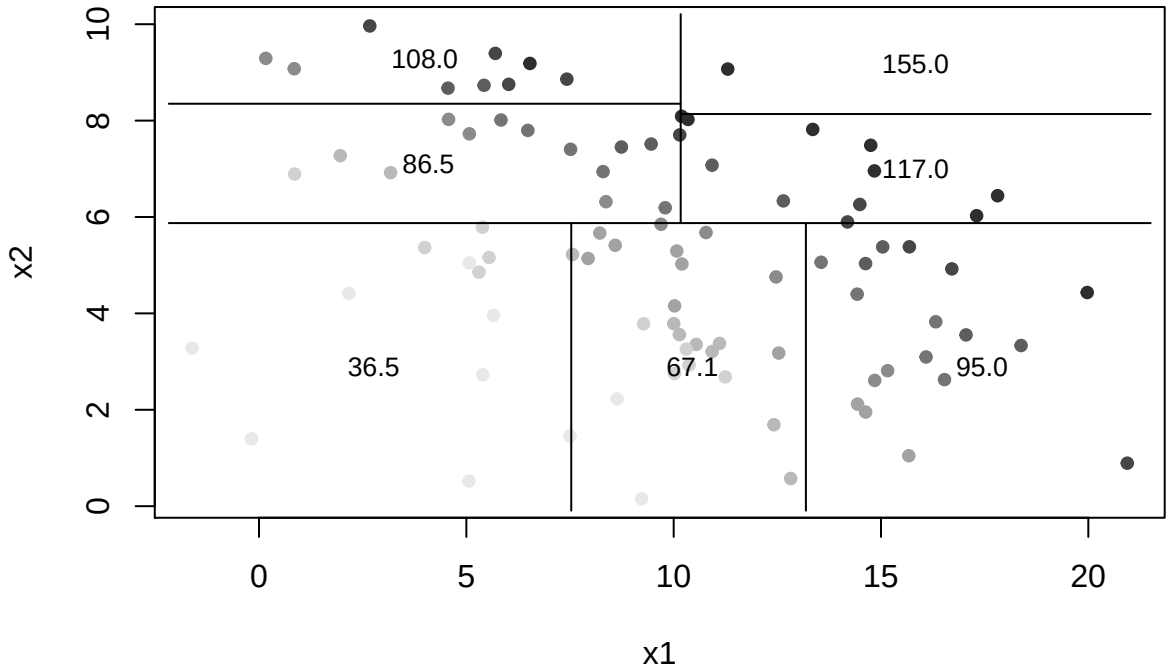
Graph 1a closely mimics the structure of a tree. The single node at the top is the *root* of the tree and the 7 nodes at the bottom level are usually referred to as *terminal nodes* or *leaves*. To find the fitted y value for observation i, we simply start from the root and compare x_{i2} to 5.87. We follow the left branch if $x_{i2} < 5.87$ or proceed to the right if $x_{i2} \geqslant 5.87$. Then we continue comparing x_{i1} or x_{i2} to the splitting criterion at each *internal node* until we reach the leaves with the fitted values.

Graph 1b projects the 100 observations to a 2-dimensional graph where x_1 and x_2 are the x-axis and y-axis respectively. The shade of grey of the dots indicates the size of the corresponding y value . We first find the the horizontal line that divides the entire sample into two parts at $x_2 = 5.87$. This corresponds to the first splitting criterion after the root in **Graph 1a**. Upon deciding on which rectangle to move into, we keep finding the lines that splits the current space into half, and stop when we land in a space with no more split (terminal nodes). Compared to **Graph 1a**, visualization illustrated by **Graph 1b** is less flexible since it can not accommodate higher-dimensional problems.

Graph 1a: Visualization of regression tree model in tree shape



Graph 1b: Visualization of regression tree model using rectangles



The simplicity and straightforwardness of **Graph 1a** and **Graph 1b** explains why regression tree has become a popular method: the model is easy-to-understand and it is fast to find the estimated value of y. One can clearly explain the structure of the model to the audience regardless of their statistical background.

2 Mathematical notation

In this section, we formally describe the regression tree model in statistical terms.

Assume we have n observations (X_i, y_i) such that $X_i \in \mathbb{R}_p$ and $X_i = (x_{i1}, x_{i2}, ...x_{ip})$. The regression tree partitions them into M non-overlapping regions $R_1, R_2, ...R_M$ as illustrated in **Graph 1b**. These regions are equivalent to the *terminal notes*. The number of observations that get partitioned into region R_m can be summarized with $N_m = \sum_{i=1}^{n} I(X_i \in R_m)$.

Similar to other estimators, the goal of the regression tree is also to minimize the difference between the fitted y values and the observed y values, which can be measured by residual sum of square RSS:

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

This is similar to minimizing the *training error* $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ that we have learned in class, which is an empirical version for $MSE = E[(f(x) - \hat{f}(x))^2]$ because we don't have the entire data distribution to calculate the expectation in the *MSE*.

We can consider every possible tree with different number of nodes and different ways of branching, and find the one that gives the lowest *RSS*. However, this approach is computationally taxing and can quickly become infeasible if we increase sample size or move into high-dimensional space for X. Therefore, regression trees adopts a binary splitting, top-down greedy approach. We begin with the root of the tree and divide the data into *two* regions R_1 and R_2, which correspond to the *internal nodes* in the tree structure.

To determine the splitting criterion, we consider all x_1 to x_p and possible cut point s, such that we find the pair of j and s that gives us the smallest RSS:

$$RSS = \sum_{x_i \in R_1} (y_i - \hat{y}_i)^2 + \sum_{x_i \in R_2} (y_i - \hat{y}_i)^2$$

We then apply the same strategy to R_1 and R_2 and continue splitting our observations in these two nodes into four nodes. We do so *recursively* until we reach the terminal nodes that gives us the *RSS* we are satisfied with.

We can always build a regression tree with only one data point in each terminal nodes (number of terminal nodes = sample size). This method has its drawback and will be discussed later. By being greedy and requesting the local minimal *RSS* at each split, building the tree becomes much faster with the compromise that the tree may not land at the global minimal *RSS*.

One remaining component of the formula is to find the fitted value $\hat{y}_i$. For observations $X_i \in R_m$, the regression tree models their y with a constant number c_m. Regression tree then models the entire sample with M constant values: $c_1, c_2, ...c_M$. The model is then a piecewise constant model, which makes it fast to model the response variable y_i for observation i:

$$\hat{y}_i = \sum_{m=1}^{M} c_m I(x_i \in R_m)$$

Given this property, we can rewrite the *RSS* at each binary split into two new nodes R_1 and R_2 as:

$$RSS = \sum_{x_i \in R_1} (y_i - c_1)^2 + \sum_{x_i \in R_2} (y_i - c_2)^2$$

and the overall RSS for the regression tree will be:

$$RSS = \sum_{m=1}^{M} \sum_{x_i \in R_m} (y_i - c_m)^2$$

To minimize *RSS* with constant c_m, it makes the most sense to have c_m take the average value of y among observations $X_i \in R_m$. Thus, we have:

$$\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i$$

Some extensions of the regression tree model may allow for using a linear combination of x_i in the form of $\sum \beta_j x_j < s$ as the splitting criterion (e.g. $x_{i1} + 2x_{i2} < 5.87$). The model can also use other modeling technique inside the terminal nodes instead of a constant number to model the y value. For this report, we focus on the simple version of binary regression as described earlier in this section.

2.1 Simulation in 1-dimensional space

To evaluate the performance of regression tree, a simulation study is conducted following the regular set-up in the homework. Regression tree is compared to other parametric or non-parametric estimators that we have learned in class.

The first simulation example uses $x \in \mathbb{R}_1$. $n = 100$ observations are generated with $x \overset{i.i.d.}{\sim} Uniform(-1, 1)$, and

$$y_i = f(x_i) + \epsilon_i$$

for $i = 1, 2, ..., n$ where $\epsilon_i \sim N(0, 1)$.

In different scenarios, $f(x)$ is constructed as follows:

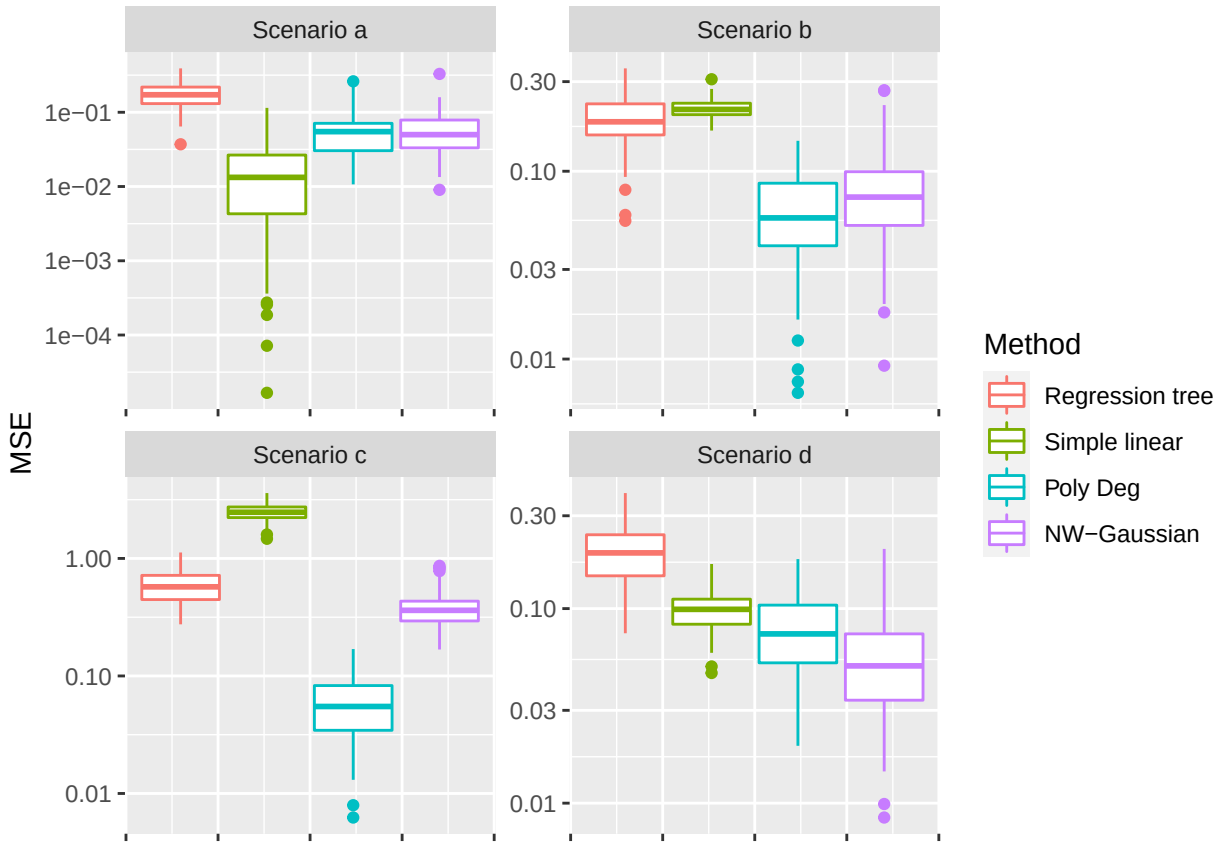
$$(a)\ f(x) = 2x$$
$$(b)\ f(x) = sin(x * \pi)$$
$$(c)\ f(x) = 2x + x^3 - 6x^4$$
$$(d)\ f(x) = \frac{1}{1 + (5x)^2}$$

Estimators for comparison include simple linear regression, parametric polynomial regression model with degree of 5, and Nadaraya-Watson with a 'gaussian' kernel with bandwidth 0.4. **Graph 2a** uses boxplots to show the performance of each estimator in terms of the empirical *MSE*:

$$\hat{MSE} = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - \hat{y}_i)^2$$

The simulation study is replicated 100 times to get a more stable estimate of $\hat{MSE}$ as well as to check the variance of the estimator. In all four scenarios, the red boxplot represents the regression tree estimator and we see that it does not have a satisfying performance. It is understandable that it fails to outperform the two parametric estimators: linear regression or polynomial regression in the first three scenarios *a* to *c* because the underlying relationship $f(x)$ is either linear or polynomial. In secnario *d*, $f(x)$ is rather complicated that it can not be easily approximated by the parametric models while the non-parametric estimator Nadaraya-Watson with a 'gaussian' kernel is able to capture such wiggliness. However, not only does regression tree perform worse compared to Nadaraya-Watson, its $\hat{MSE}$ is higher than that of parametric methods.

Graph 2a: regression tree vs other estimators in 1-dimensional space



2.2 Simulation in multi-dimensional space

The second simulation example is based on observations with $x \in \mathbb{R}_3$. Again, $n = 100$ observations are generated with $x_1 \overset{i.i.d.}{\sim} Uniform(0,1)$, $x_2 \overset{i.i.d.}{\sim} Uniform(0,1)$, and $x_3 \overset{i.i.d.}{\sim} Uniform(0,1)$.

$$y_i = f(X_i) + \epsilon_i$$

where $i = 1, 2, ..., n$ and $\epsilon_i \sim N(0,1)$.

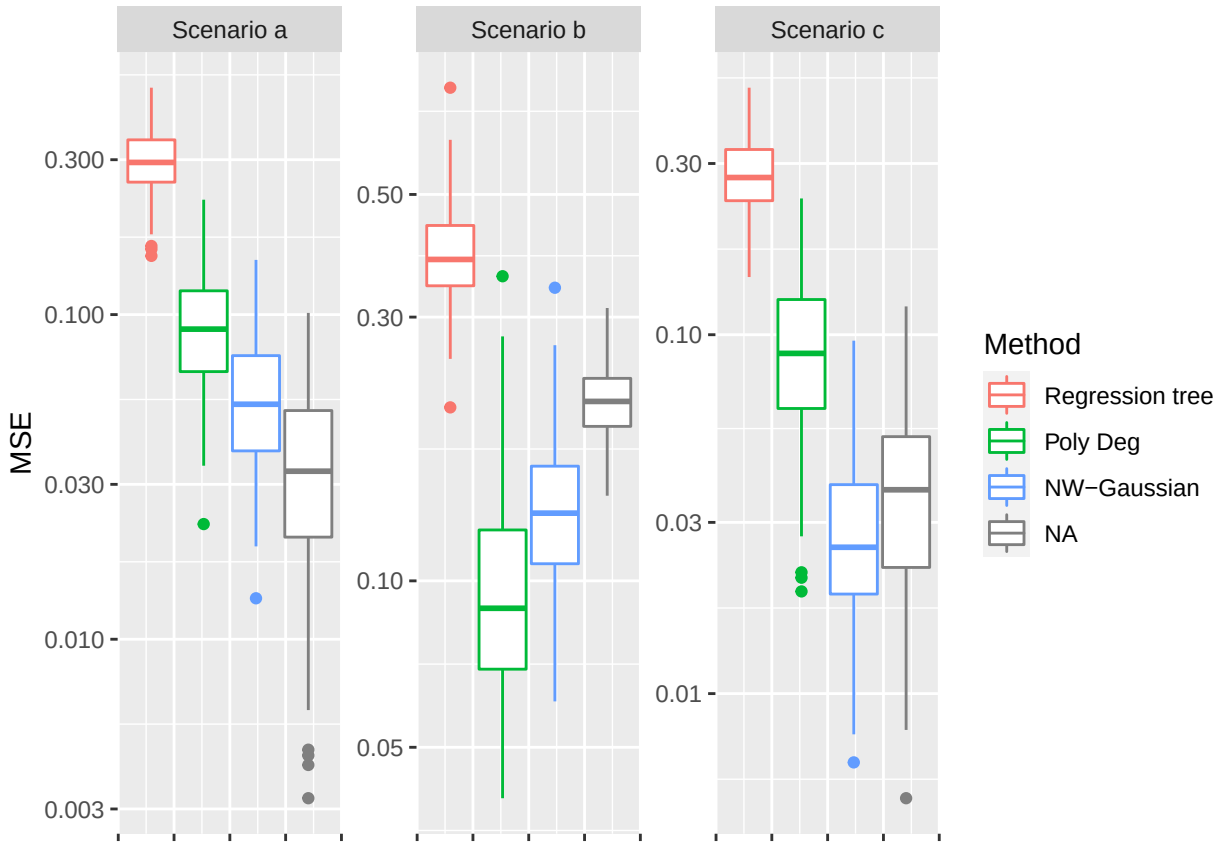
The functions include:

$$(a)\ f(x_1, x_2, x_3) = x_1 + x_2 + x_3$$
$$(b)\ f(x) = sin(x * \pi)$$
$$(c)\ f(x_1, x_2, x_3) = (x_1 x_2 x_3)^{1/3}$$

Estimators include regression tree, multivariate linear regression, additive model with polynomial regression and Nadaraya-Watson with a 'gaussian' kernel. The additive polynomial regression raises x_1, x_2 and x_3 simultaneously to degree of 2 to 5. The optimal polynomial degress is selected using cross-validation. The optimal bandwidth for the Nadaraya-Watson estimator is also chosen using cross-validation.

Graph 2b shows the regression tree is dominated by the other estimators when the underlying function is linear (scenario a), additive (sceanrio b) or non-linear (scenario c). It constantly yields the largest empirical MSE compared to the other 3 estimators.

Graph 2b: regression tree vs other estimators in 3-dimensional space



2.3 Simulation in high-dimensional space

The third simulation example generates $n = 100$ observations with $X \in \mathbb{R}_{100}$. $x_p \overset{i.i.d.}{\sim} Uniform(0,1)$, $p = 1, 2, ...100$. Functions are listed below:

$$(a)\ f(x_1, x_2, ..., x_{100}) = \sum_{p=1}^{100} x_p$$

$$(b)\ f(x_1, x_2, ..., x_{100}) = \left(\prod_{p=1}^{100} x_p \right)^{\frac{1}{100}}$$

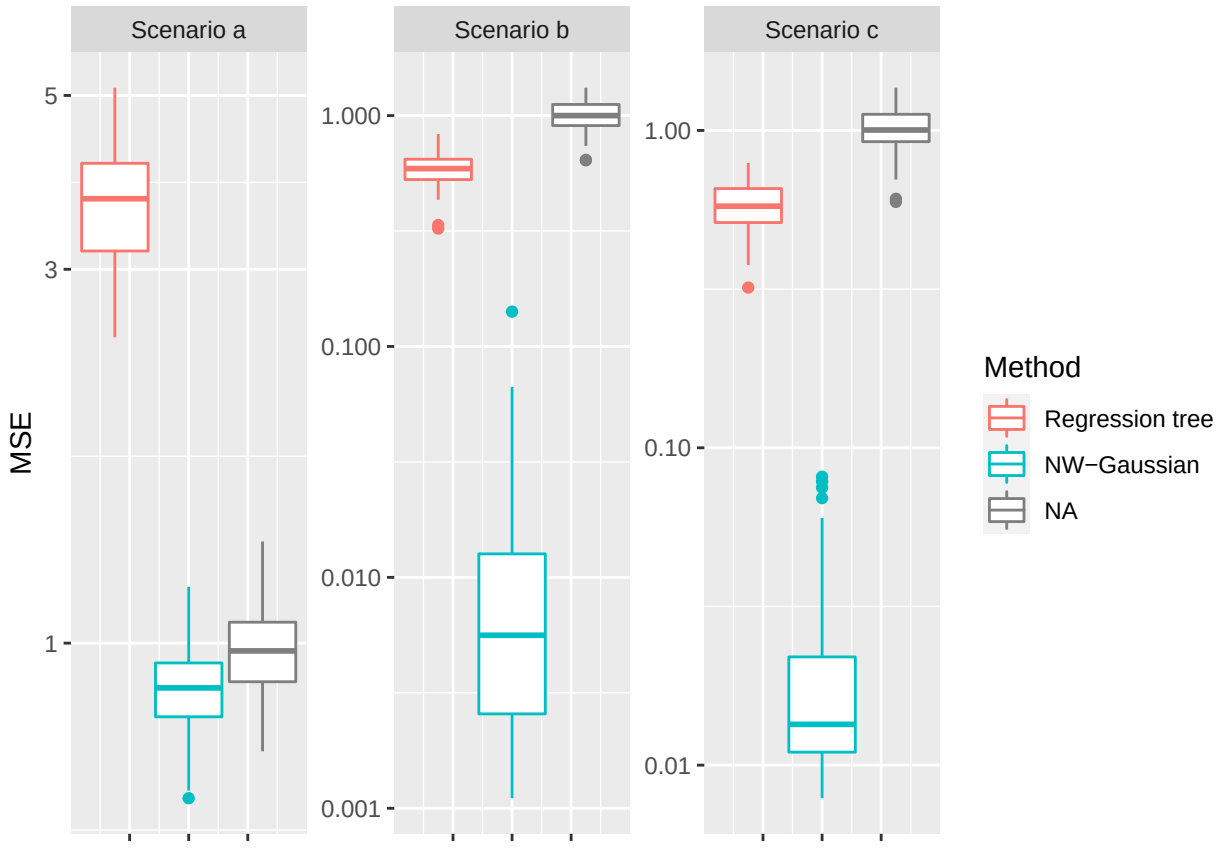
$$(c)\ f(x_1, x_2, ..., x_{100}) \overset{i.i.d.}{\sim} N(0.5, 0.1)$$

In this round, regression tree is compared to multivariate linear regression and Nadaraya-Watson with a 'gaussian' kernel (bandwidth chosen with cross-validation).

Graph 2c shows in sceanrio a that when the generating mechanism for y is linear, regression tree is surpassed by the other two estimators. When $f(x_1, x_2, ..., x_{100})$ is more complicated as in scenario b and c, while regression tree is still defeated by Nadaraya-Watson with a 'gaussian' kernel, it at least outperforms linear regression for the first time.

In the opening section, we state that the regression tree model is simple and intuitive to follow. However, from the three simulation examples, we observe the disappointing performance of regression tree. It seems like the model is too simple to work nicely: compared to other estimators, it models y less accurately.

Graph 2c: regression tree vs other estimators in high-dimensional space



3 Pruning

In this section, we evaluate regression tree model from a prediction problem perspective and introduce an important component of the model: pruning.

As mentioned in the second section of this report, we can theoratically build a *full* regression tree such that each terminal node only contains very few observations. Although this tree gives us an unbiased estimateof the current sample, it is overfitting the data. The regression tree built under this fashion also has large variance: it can change drastically when modeled on different samples from the same population.

At the cost of increasing bias of the estimator but to avoid overfitting and to decrease variance of the tree model, we want a more flexible tree. There are multiple ways to do so: we can stop splitting nodes when the number of data points in a node falls below 10 percent of the entire sample. We can continue branching only if doing so will decrease *RSS* by a certain number. This method is not ideal because a node that does not pass the threshold to be further splitted in the early stage coul lead to large decrease in *RSS* later on.

Another way is to do the opposite: we grow a full tree first, and then prune it back to a smaller subtree by collapsing some nodes, hoping for a low *test error*. However, this method can be rather clumsy if we have a large full tree to begin with and we will need to consider every possible subtrees. To address this issue, we commonly adopt the *cost-complexity pruning* method which narrows down the range of subtree for consideration by introducing a tuning parameter: α. The general idea is to penalize complicated trees with large number of terminal nodes.

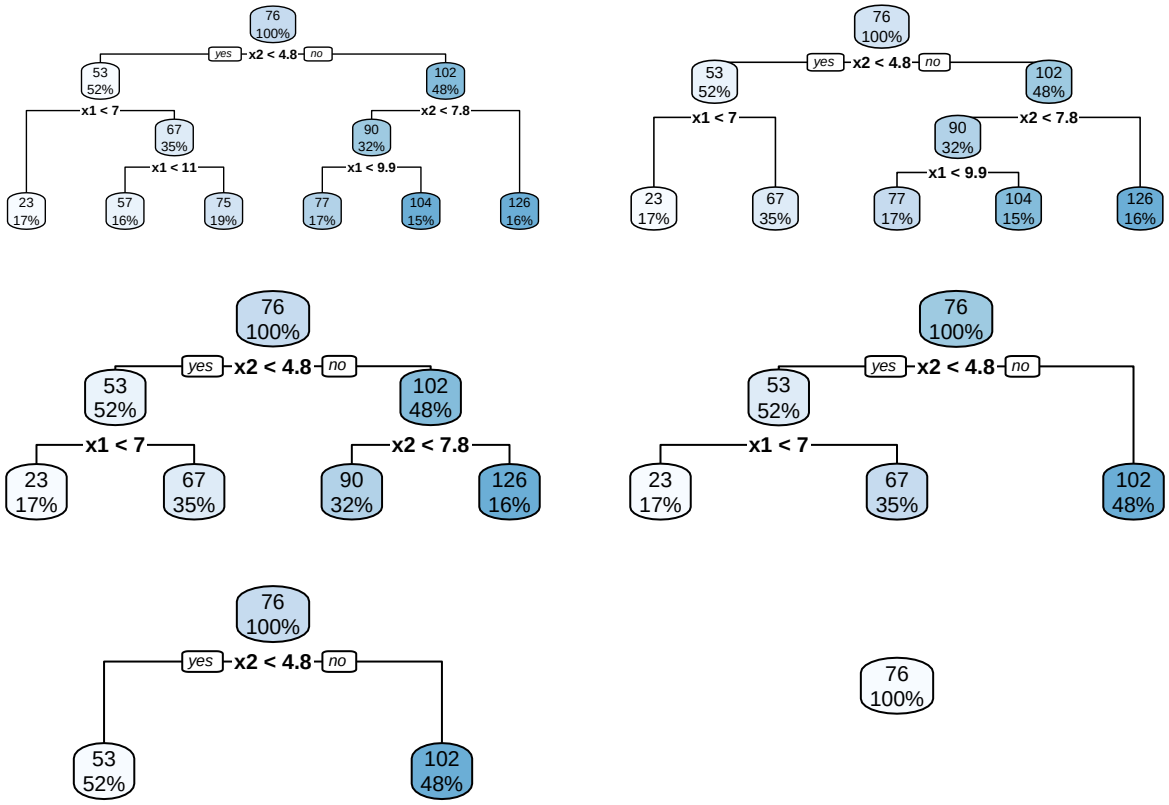
We first consider a range of α values to test. For a given α, the goal is to find a subtree with $\tilde{M}_\alpha$ terminal

nodes, such that this tree has the *minimal cost-complexity value*:

$$C_\alpha(\tilde{M}_\alpha) = \sum_{m=1}^{\tilde{M}_\alpha} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2 + \alpha \tilde{M}_\alpha$$

It turns out that the corresponding best pruned subtree for increasing α will be nested and predictable as illustrated in **Graph 3**. From left to right and top to bottom, we increase α and see a sequence of nested pruned trees. The first row in each node indicates the predicted value $\hat{c}_m$ and the second row shows the percentage of the sample that falls into the node. When $\alpha = 0$ (the top-left graph), the best tree with the *minimal cost-complexity value* is the original full tree with 6 terminal nodes. When α increases, we collapse the terminal nodes from the full tree with the smallest increase in $\sum_m \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2$ to get a subtree. As we continue increasing α, we collapse the new terminal nodes in the subtree (originally internal nodes) to a even smaller subtree. The best pruned tree selected for each α with the minimal $C_\alpha(\tilde{M}_\alpha)$ will then have fewer terminal nodes and is a subtree of the previous pruned subtree. Although the pruned tree has worse fit on the sample, it performs better at predicting future data set. When α gets really large, we are left with one *single* node (the bottom-right one) which is the root from the original full tree.

Graph 3: A sequence of nested pruned trees



After we grow a sequence of subtree indexed by a range of α values, the best α is picked through cross-validation, similar to choosing the optimal bandwitdth for Nadaraya-Watson with a "gaussian" kernel. The data will be splitted into K folds and a range of α values will be considered for the pruning. For $k = 1, 2, ..., K$, use data from every other $k-1$ folds except for the kth fold as the training set and the kth fold is be held out as the cross-validation fold. Among the *training set*, we build a sequence of trees corresponding to the sequence of α values and use these trees to predict on the *cross-validation set* and calculated their *RSS*. Each α value wil then be associated with K regression trees and consequently K RSS results. We then choose the α associated with the smallest average *RSS* as the tuning parameter for our model.

3.1 Simulation to compare training and test error

To confirm if pruning helps regression tree do a better job at predicting, the study includes another simulation with $X \in \mathbb{R}_{100}$. $n = 1000$ observations are generated and are radomly splitted into a *training set* and *test set* evenly. The scenarios tested here correspond to the same three functions as in the second section:

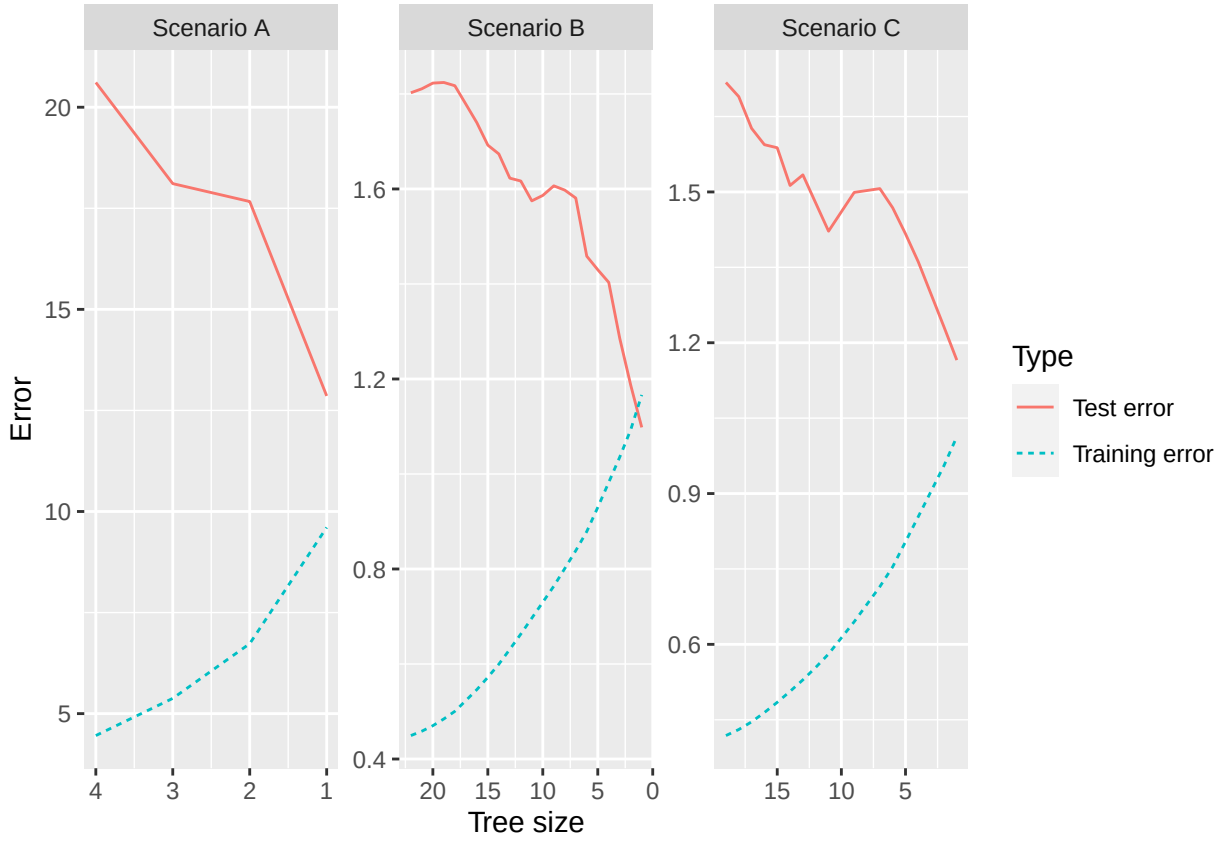
$$(a)\ f(x_1, x_2, ..., x_{100}) = \sum_{p=1}^{100} x_p$$

$$(b)\ f(x_1, x_2, ..., x_{100}) = \left(\prod_{p=1}^{100} x_p \right)^{\frac{1}{100}}$$

$$(c)\ f(x_1, x_2, ..., x_{100}) \sim Normal(0.5, 0.1)$$

In **Graph 4**, x-axis corresponds to the number of terminal nodes in the regression trees. We observe that in any scenario, with bigger and more complicated trees, the training error is small but the test error is really large. If we prune the tree to be smaller, we observe a drop in test error as we expect.

Graph 4: Test error vs training error

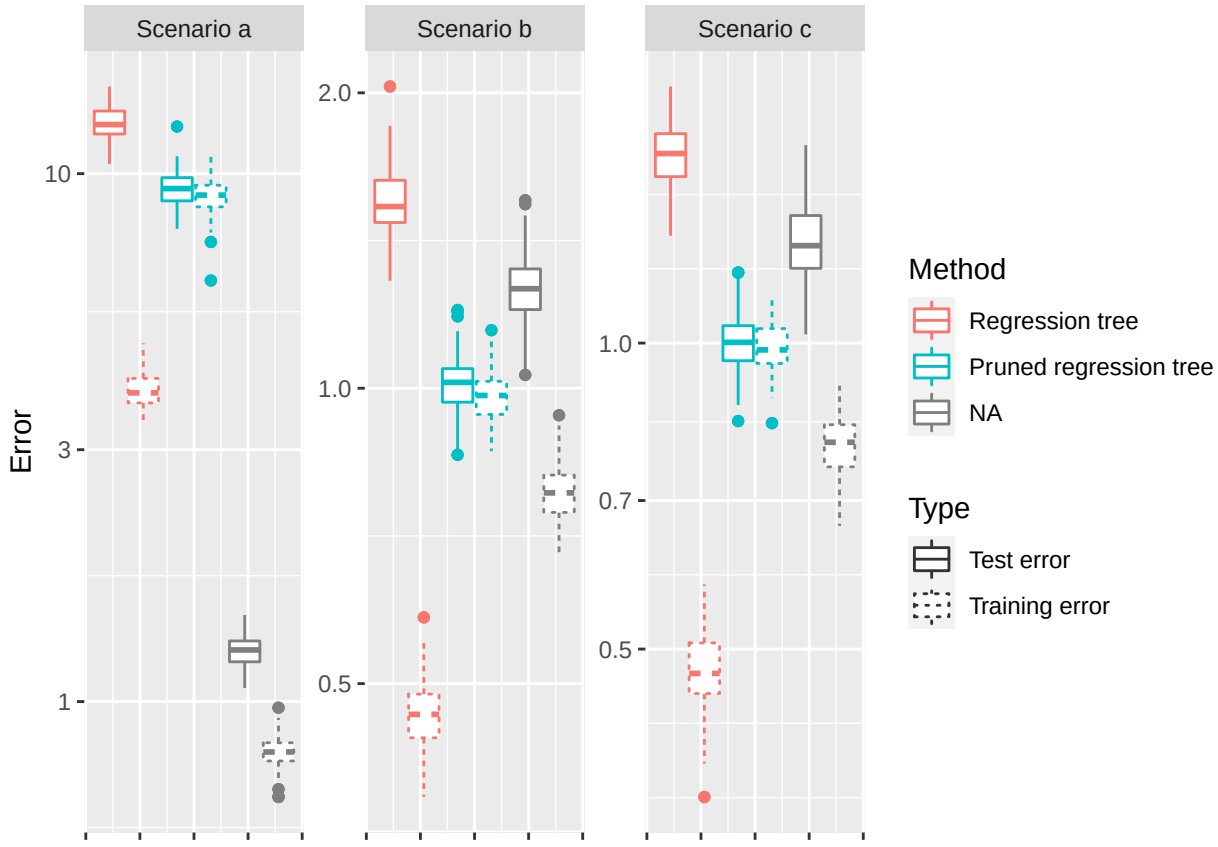


3.2 Simulation to evaluate pruned regression tree

To evaluate the performance of pruned tree against other estimators, the study uses another simulation example to compare the performance of a fully grown tree, a pruned tree and simple linear estimator in terms of *test error*. The optimal α for the pruned tree is determined by cross-validation on the training set as

described before. It is confirmed in **Graph 5** that the pruned tree consistently outperforms full tree in terms of test error though it has larger training error. While full tree fails to outperform linear regression, pruned tree does a better job than multivariate linear model at predicting test data in scenario b and c when the undelying function is not linear.

Graph 5: Pruned regression tree vs other estimators in high-dimensional space



4 Conclusion

In this study, we learn about the basic regression tree model using a few simulation examples. Although a regression tree is easy-to-interpret, its accuracy in modeling y is not as good when compared to some other estimators such as the Nadaraya-Watson with a 'gaussian' kernel. We can build a large regression tree with fewer observations in each terminal node to *increase accuracy* and *reduce bias*, but that approach risks the tree being over-complicated and overfitting the data. Instead, we can make the tree more flexible through *cost-complexity pruning* method to boost its performance in predicting future events.

5 References

[1] James, Gareth, et al. *An introduction to statistical learning.* Vol. 112. New York: springer, 2013.

[2] Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning.* Vol. 1. No. 10. New York: Springer series in statistics, 2001.

[3] CMU statistics. *Classification and Regression Trees.* 2009. https://www.stat.cmu.edu/~cshalizi/350/lectures/22/lecture-22.pdf (accessed May 20, 2020)

[4] Milborrow, Stephen. *Plotting rpart trees with the rpart.plot package.* 2018. http://www.milbo.org/rpart-plot/prp.pdf (accessed May 23, 2020)