Lecture 18: MATH 342W: Introduction to Data Science and Machine Learning

Sergio E. Garcia Tapia*

April 10th, 2025 (last updated April 11, 2025)

R Demo

See `QC_MATH_342W_Spring_2025/practice_lectures/lec18.Rmd`.

Tree Models

Suppose we have a data set $\mathbb{D}$ with a numeric response (output space is $\mathcal{Y} = \mathbb{R}$) and one raw feature ($p = 1$), as depicted in Figure 1. If we wish to make predictions about the phenomenon from which the observations in $\mathbb{D}$ were measured, we might first try OLS with a linear fit:

$$\mathcal{H}_1 = \{ w_0 + w_1 x \mid \boldsymbol{w} \in \mathbb{R}^2 \}$$

An approximate OLS linear fit $g_1 \in \mathcal{H}_1$ is shown in Figure 2. By looking at the data set, we can infer that g_1 incurs a lot of misspecification error since the trend in $\mathbb{D}$ does not appear to be linear. Another attempt might be to create a transformed feature x^2 and to use OLS with a quadratic fit:

$$\mathcal{H}_1 = \{ w_0 + w_1 x + w_2 x^2 \mid \boldsymbol{w} \in \mathbb{R}^3 \}$$

An approximate OLS quadratic fit $g_2 \in \mathcal{H}_2$ is shown in Figure 3. The quadratic fit may

*Based on lectures of Dr. Adam Kapelner at Queens College. See also the course GitHub page.

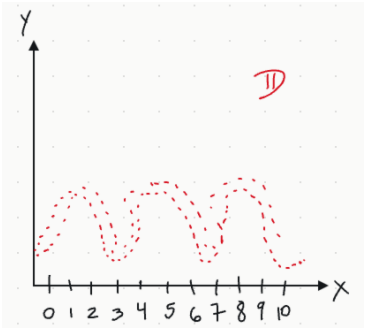

Figure 1: Data set $\mathbb{D}$ with numeric response space $\mathcal{Y} = \mathbb{R}$ and 1 feature.

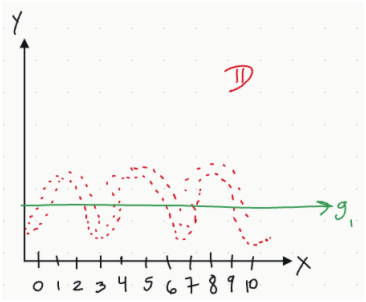

Figure 2: A linear fit for $\mathbb{D}$ as from Figure 1.

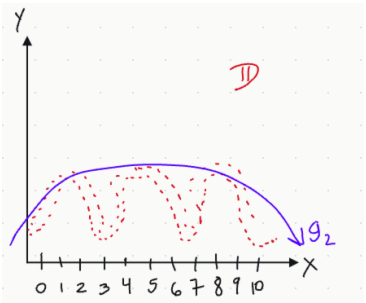

Figure 3: A quadratic fit for $\mathbb{D}$ as from Figure 1.t-numeric-response-1-feature-quadratic

or may not do any better than the linear fit. A third attempt would be to consider a set of candidate functions with sinusoidal functions:

$$\mathcal{H}_3 = \{ w_0 + w_1 \sin(w_2(x - w_3)) \mid \boldsymbol{w} \in \mathbb{R}^4 \}$$

An approximate least-squares sinusoidal fit $g_3 \in \mathcal{H}_3$ is shown in Figure 4. We see g_3 likely incurs lower misspecification than $\mathcal{H}_1$ and $\mathcal{H}_2$. Note that the least squares fit for $\mathcal{H}_3$ would not be "ordinary" (hence we do not call it OLS), since we cannot create a design matrix X and compute $(X^\top X)^{-1} X^\top$ to get the least squares best values $\boldsymbol{w} \in \mathbb{R}^4$. We would need to use an optimizer to minimize the objective function

$$\sum_{i=1}^{n} (y_i - w_0 + w_1 \sin(w_2(x_i - w_3)))^2$$

The point of this exercise is to emphasize that when $p = 1$, it is fairly easy to offer M candidate modeling procedures, and to have at least one of them perform well. However,

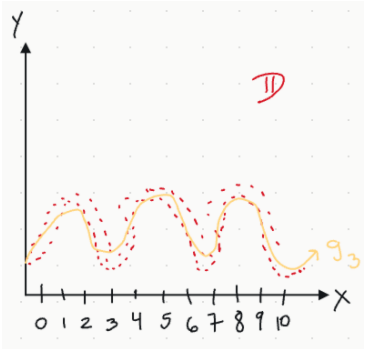

Figure 4: A sinusoidal fit for $\mathbb{D}$ as from Figure 1.

if $p \gg 1$, how do we offer M such models? We lose the ability to visualize, say, 37 dimensions. The problem is that *there is no easy way to a priori (before the fact) specify all potentially fruitful non-linearities and interactions among features.* This is where non-parametric machine learning (hereby abbreviated as ML) comes in.

What do we mean by non-parametric? Note, for example, that $\mathcal{H}_1$ has 2 parameters, $\mathcal{H}_2$ has 3 parameters, and $\mathcal{H}_3$ has 4 parameters. By non-parametric, we mean that the number of parameters, and the parameter definitions themselves, are *not* pre-specified, but they are fit automatically. It is "as if" the candidate set expands to what is needed. We will study **tree models**. In particular

- **Regression trees** for when $\mathcal{Y} =\subseteq \mathbb{R}$ (numerical).
- **Classification trees** for when $\mathcal{Y} = \{C_1, \ldots, C_L\}$ (categorical).

Both approaches were published in 1984 by Breiman. We are interested in the **Classification and Regression Tree Algorithm (CART)**.

Regression Trees

For the sake of our discussion, we will continue to consider $\mathcal{Y} = \mathbb{R}$ and $p = 1$ as in Figure 1. Note that $x \in \mathcal{X}$, where the input (covariate) space $\mathcal{X} = \mathbb{R}$. We can subdivide (partition) it as

$$\mathcal{X} = \mathbb{R} = \overbrace{(-\infty, 0)}^{A_1} \cup \overbrace{[0, 1)}^{A_2} \cup \overbrace{[1, 2)}^{A_3} \cup \cdots \cup \overbrace{[9, 10)}^{A_{11}} \cup \overbrace{[10, \infty)}^{A_{12}}$$

Let's transform x from a one-dimensional interval feature to a $L = 12$-dimensional categorical variable, where each category denotes membership in one of the sets A_ℓ. If X_{raw} is our raw design matrix using the raw interval feature x, then after the transformation we get an $n \times L$ matrix X(without intercept):

$$X_{\text{raw}} \implies X = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 1 \\ 0 & 0 & \cdots & 1 \end{bmatrix} = \begin{bmatrix} \mathbb{I}_{x \in A_1} & \mathbb{I}_{x \in A_2} & \cdots & \mathbb{I}_{x \in A_{12}} \end{bmatrix}$$

Note that if we are assuming that $n \gg p$ and that we have at least one data point in each interval A_ℓ, X will be full rank. If we fit using OLS, and x_* is a new observation (in $\mathcal{X} = \mathbb{R}$), our prediction function would assign the predicted response $\hat{y}_* = \overline{y}_\ell$, corresponding to the mean response of the observations in $\mathbb{D}$ for which $x \in A_\ell$:

$$\hat{\boldsymbol{y}} = \begin{bmatrix} \overline{y}_1 \\ \overline{y}_2 \\ \vdots \\ \overline{y}_{12} \end{bmatrix}$$

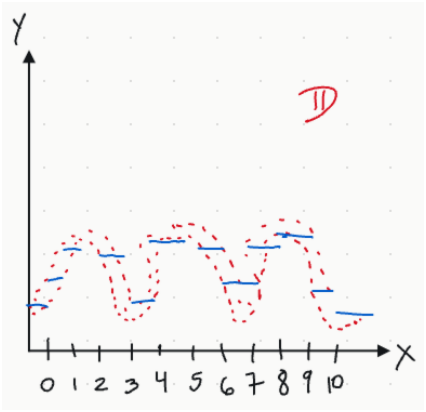

Figure 5: Using step functions to fit the data in Figure 1.

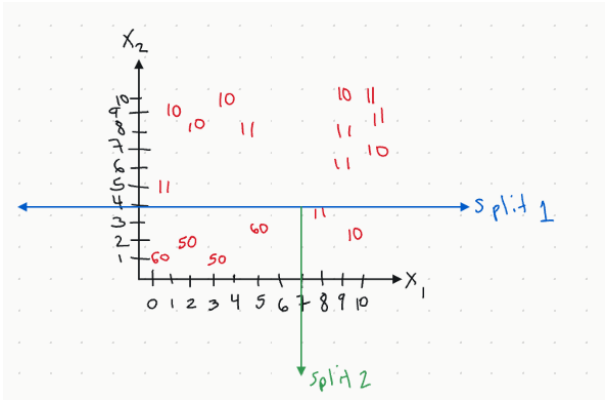

Figure 6: Dynamically splitting data set with $p = 2$ features.

Effectively, we cut the input space into pieces, and we use step functions, as depicted in Figure 5. There is a hyperparameter in this procedure, namely, the size of each interval, or equivalently, the number of bins (here 12). This model might not beat $\mathcal{H}_3$, but it might do better than $\mathcal{H}_1$ and $\mathcal{H}_2$. To select the number of bins, we can use model procedure setting III that we discussed last lecture. Note that if you use 1 bin, you have the null model g_0, and if you use n bins (where n is the number of data points), you overfit.

When $p = 1$, there is no much downside to using this approach. What if p is large? The Boston Housing data set, for example, has $n = 506$ and $p = 13$. If there are 5 bins per feature, then the total number of bins explode exponentially to $5^{13} \approx 1.22$ billion. We will have many empty bins in this case. This is called **the curse of dimensionality**, i.e., the number of parameters grows exponentially with p, which is $\gg n$. We will have approximately 1.22 billion columns, most consisting only of zeros. In summary, this procedure does not extend well to multiple features.

What do we need to do to fix this problem? In our original approach, we carved the input space from the get-go into $A_1, A_2, \ldots, A_{12}$. Do we need to do that? How about an algorithm $\mathcal{A}$ that dynamically chooses custom bins that improve performance? We don't need bins of equal size, or an equal number of bins. Figure 6 shows an example where $p = 2$. Let's restrict bins to be orthogonal to the coordinate axes.

- **Step 1**: Find the best two-bin model by splitting all of $\mathbb{R}^2$. In Figure 6, this was achieved with the line $x_2 = 4$.

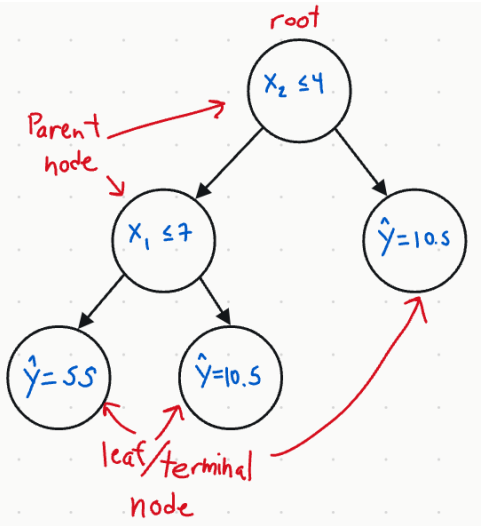

Figure 7: Binary decision tree built from the partitioning algorithm applied in Figure 6.

- **Step 2**: Find the best three-bin model by splitting a previous bin into two bins. In Figure 6, this was achieved with the line $x_1 = 7$.
- **Step 3**: Find the best four-bin model by splitting a previous bin into two bins (not depicted in Figure 6).
- Eventually, stop.

Now we can construct a **rooted binary decision tree** to make predictions, shown in Figure 7. First, some nomenclature:

- In a rooted tree, there is a node designated as the **root**, which is the ancestor of all other nodes. In Figure 7, it corresponds to the node with $x_2 \leq 4$.
- A node may have **descendants**. A node that has an immediate descendant is called a **parent** node, and its immediate descendants are called **children** nodes.
- A node in a tree without any children is called a **leaf**. The leaves in our example correspond to the nodes with $\hat{y} = 10.5$, $\hat{y} = 55$, and $\hat{y} = 10.5$.
- In a **binary tree**, each node may have at most 2 children.

Since we have three bins in our example, there are three values we can predict (and hence three leaves in the resulting binary tree). For example, if we have an observation with $\boldsymbol{x}_* = [x_1 \ x_2] = [15, 21]$, then we first note $x_2 = 21$ is larger than 4, so we immediately predict $\hat{y}_* = 10.5$. If instead $\boldsymbol{x}_* = [x_1 \ x_2] = [62]$, then $x_2 \leq 4$ and $x_1 \leq 4$, so we predict $\hat{y}_* = 55$.

The usage of orthogonal axes in Figure 6 is depicted by the fact that the lines used to partition the plane are parallel to the coordinate axes. Alternatives have been investigated, but it is hard to beat random forests and it can be hard to interpret.

Let's state the regression algorithm more fully.

Algorithm 1 (Regression Tree Algorithm)**.** Start with $\mathbb{D}$. Let $x_{(i)}$ denote the ith smallest value in a vector $\boldsymbol{u}$.

(1) Let $\boldsymbol{x}_k$ denote the kth vector, for $1 \leq k \leq p$, containing all n possible values for the kth feature. Let's consider every possible orthogonal-to-axis split, i.e:

$$\begin{aligned} x_1 \leq \boldsymbol{x}_{\cdot 1(1)}, x_1 \leq \boldsymbol{x}_{\cdot 1(2)}, \ldots, x_1 \leq \boldsymbol{x}_{\cdot 1(n-1)} \\ x_2 \leq \boldsymbol{x}_{\cdot 2(1)}, x_2 \leq \boldsymbol{x}_{\cdot 2(2)}, \ldots, x_2 \leq \boldsymbol{x}_{\cdot 2(n-1)} \\ x_p \leq \boldsymbol{x}_{\cdot p(1)}, x_p \leq \boldsymbol{x}_{\cdot p(2)}, \ldots, x_p \leq \boldsymbol{x}_{\cdot p(n-1)} \end{aligned}$$

This means we have $n \cdot (p-1)$ possible splits. Note we do not consider, for example, $x_1 \leq \boldsymbol{x}_{\cdot 1(n)}$, or $x_2 \leq \boldsymbol{x}_{\cdot 2(n)}$, and so on because such a split would have *all* data on one side of the split, and hence there is no split at all.

(2) Pick the best split, i.e., the one with the lowest error, where error is defined as:

$$SSE_{\text{weighted}} := \frac{n_L SSE_L + n_R SSE_R}{n_L + n_R}$$

where n_L is the number of observations in the left child, n_R is the number of observations in the right child, SSE_L is the SSE in the left child, and SSR_R is the SSE in the right child. Note that since we are fitting a local optimization by picking the *best* split each time, this algorithm $\mathcal{A}$ is *greedy*.

(3) Repeat steps (1) and (2) on the subset of $\mathbb{D}$ consisting of all the daughter nodes. Note that we can make at most n splits, otherwise we will have n dummies and we would surely overfit. Therefore, stop when the number of points in a child node is below some threshold N_0 (default $N_0 = 5$).

Classification Trees

Let $\mathcal{Y} = \{C_1, \ldots, C_L\}$ (categorical response space). The classification tree algorithm is similar to Algorithm 1, but differs a bit in steps 2 and 3.

Algorithm 2 (Classification Tree Algorithm)**.** Start with $\mathbb{D}$. Let $x_{(i)}$ denote the ith smallest value in a vector $\boldsymbol{u}$.

(1) Let $\boldsymbol{x}_k$ denote the kth vector, for $1 \leq k \leq p$, containing all n possible values for the kth feature. Let's consider every possible orthogonal-to-axis split, i.e:

$$\begin{aligned} x_1 \leq \boldsymbol{x}_{\cdot 1(1)}, x_1 \leq \boldsymbol{x}_{\cdot 1(2)}, \ldots, x_1 \leq \boldsymbol{x}_{\cdot 1(n-1)} \\ x_2 \leq \boldsymbol{x}_{\cdot 2(1)}, x_2 \leq \boldsymbol{x}_{\cdot 2(2)}, \ldots, x_2 \leq \boldsymbol{x}_{\cdot 2(n-1)} \\ x_p \leq \boldsymbol{x}_{\cdot p(1)}, x_p \leq \boldsymbol{x}_{\cdot p(2)}, \ldots, x_p \leq \boldsymbol{x}_{\cdot p(n-1)} \end{aligned}$$

This means we have $n \cdot (p-1)$ possible splits. Note we do not consider, for

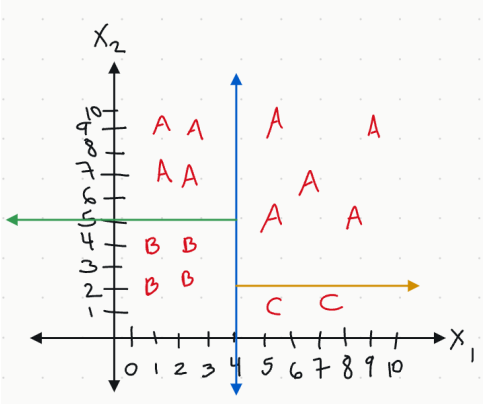

Figure 8: Classification tree algorithm with a categorical response space.

example, $x_1 \leq \boldsymbol{x}_{\cdot 1(n)}$, or $x_2 \leq \boldsymbol{x}_{\cdot 2(n)}$, and so on because such a split would have *all* data on one side of the split, and hence there is no split at all.

(2) Pick the best split, i.e., the one with the lowest error. We need a new error metric because SSE is not appropriate for a categorical response. Define the **Gini** error metric:

$$G_{\text{neighbor, avg}} = \frac{n_L G_L + n_R G_R}{n_L + n_R}$$

where n_L is the number of observations in the left child, n_R is the number of observations in the right child. We get something similar to SSE by defining

$$G := \sum_{l=1}^{L} \hat{p}_\ell (1 - \hat{p}_\ell)$$

where $\hat{p}_\ell$ is the proportion of observations of whose response is category ℓ in a given node:

$$\hat{p}_\ell := \frac{\text{number of observations with category } \ell \text{ in node}}{\text{total number of observations in node}}$$

Note that since we fitting a local optimization by picking the *best* split each time, this algorithm $\mathcal{A}$ is *greedy*.

(3) Repeat steps (1) and (2) on the subset of $\mathbb{D}$ consisting of all the daughter nodes. Note that we can make at most n splits, otherwise we will have n dummies and we would surely overfit. Therefore, stop when the number of points in a child node is below some threshold (default $N_0 = 1$, possibly because there are only L levels).

See Figure 8 and Figure 9 for an example of applying Algorithm 2. The larger G is, the more heterogeneous the node is. For example, if the node has all A's, then $\hat{p}_A = \hat{p}_B = \hat{p}_C = 0$, and hence $G = 0$ for that node.

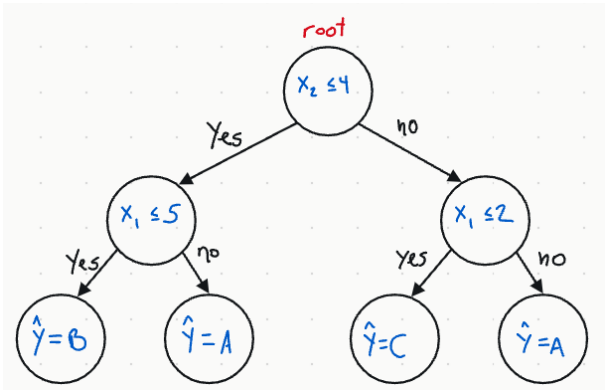

Figure 9: Classification tree corresponding to Figure 8.