Lecture 04: MATH 342W: Introduction to Data Science and Machine Learning

Sergio E. Garcia Tapia*

February 6, 2025 (last updated March 7, 2025)

Recap

Let's begin by recapping some concepts. When studying a phenomenon in real life, there is a response variable y related to other quantities of interest:

$$\begin{aligned} y &= t(z_1, \ldots, z_t) \\ &= f(x_1, \ldots, x_p) + \delta \\ &= h^*(x_1, \ldots, x_p) + \epsilon \\ &= g(x_1, \ldots, x_p) + e \end{aligned}$$

- $z_1, \ldots, z_t$: The "true" drivers or causal information for the phenomenon of interest.
- $x_1, \ldots, x_p$: Features (also known as predictors, independent variables, covariates, etc) that are proxies to the z's.
- f: "best" function that maps the features to the response y.
- h^*: "best" candidate function in our hypothesis set $\mathcal{H}$.
- $\mathcal{H}$: Set of candidate functions of some functional form that we choose to approximate f.

There are three types of errors:

(i) *Ignorance error* $\delta = t - f$: The error incurred because the predictors $x_1, \ldots, x_p$ cannot possibly capture all the information implied by the true drivers $z_1, \ldots, z_t$. We can decrease δ by increasing the number of features we use (increase p).

(ii) *Misspecification error* $\epsilon - \delta = f - h^*$: Error incurred by choosing a hypothesis set $\mathcal{H}$ that may not correctly capture the functional behavior of f. We can decrease it by choosing a more expansive $\mathcal{H}$.

(iii) *Estimation error* $h^* - g$: Error incurred by not having enough data (i.e., enough examples). We can decrease it by improving $\mathcal{A}$ (the algorithm) or by increasing n (using more data).

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

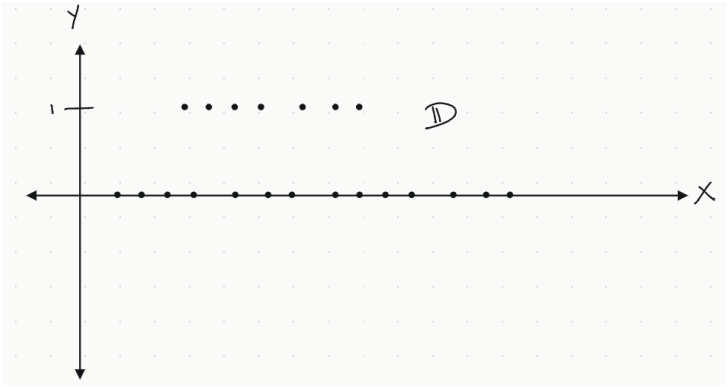

Figure 1: A plot of historical data $\mathbb{D}$, where $\mathcal{X} = \mathbb{R}$ and $\mathcal{Y} = \{0, 1\}$.

The Null Model, g_0

What if we only have outputs, and have no features, so that the tuples in $\mathbb{D}$ consist only of the values of $\mathbf{y}$? This is called g_0, **the null model**. If $\mathcal{Y} = \{0, 1\}$, then we can define

$$g_0 := \text{Mode}[\mathbf{y}]$$

Here g_0 is a constant function from $\mathcal{X}$ to $\mathcal{Y}$. For example, if the most frequent response is 0, the g_0 will always predict 0 regardless of its input.

Note that g_0 is the best we can do if we do not have any features. However, the concept of a null model is useful even if we do have features because we can use it as our reference for performance. For example, suppose that $\mathcal{Y} = \{0, 1\}$, and that the responses in our historical data $\mathbb{D}$ have 24% 1's and 76% 0's. Then the mode is 0, and the null model g_0 will always predict zero. Therefore, g_0 will misclassify 24% of the data in the training sample. Note that g_0 does not take any predictors into account, so if our predictors $x_1, \ldots, x_p$ are chosen carefully, we should be able to beat g_0 (that is, the **misclassification error** should be smaller). If our misclassification error is ever higher than g_0, then either our algorithm $\mathcal{A}$ is bad, or our features are poor proxies to the z's.

Extending the Threshold Model

Recall the threshold model. Given $f : \mathcal{X} \to \mathcal{Y}$ that best approximates a true phenomenon t, where $\mathcal{X} = \mathbb{R}$ and $\mathcal{Y} = \{0, 1\}$, we have historical data $\mathbb{D}$ as in Figure 1. Continuing with the binary response space $\mathcal{Y} = \{0, 1\}$, suppose now that we have 2 numeric features, so that $\mathcal{X} = \mathbb{R}^2$, as in Figure 2. How can we extend the threshold model to two dimensions? Specifically, we need to come up with a hypothesis set $\mathcal{H}$ of functions that map each coordinate pair to 0 or 1. Recalling that $\mathcal{H}$ was made up of indicator functions with a parameter δ in the case of the threshold model, one idea could be to have $\mathcal{H}$ consist of functions that are each a sum of two indicator functions, such as:

$$\mathcal{H} = \{\mathbb{I}_{x \geq \theta_2} + \mathbb{I}_{x \geq \theta_2} : \theta_1, \theta_2 \in \mathbb{R}\}$$

However, such functions don't map to $\mathcal{Y} = \{0, 1\}$, since they could have an output of 2. Another idea is to use a product of indicator functions:

$$\mathcal{H} = \{\mathbb{I}_{x \geq \theta_1} \cdot \mathbb{I}_{x \geq \theta_2} : \theta_1, \theta_2 \in \mathbb{R}\}$$

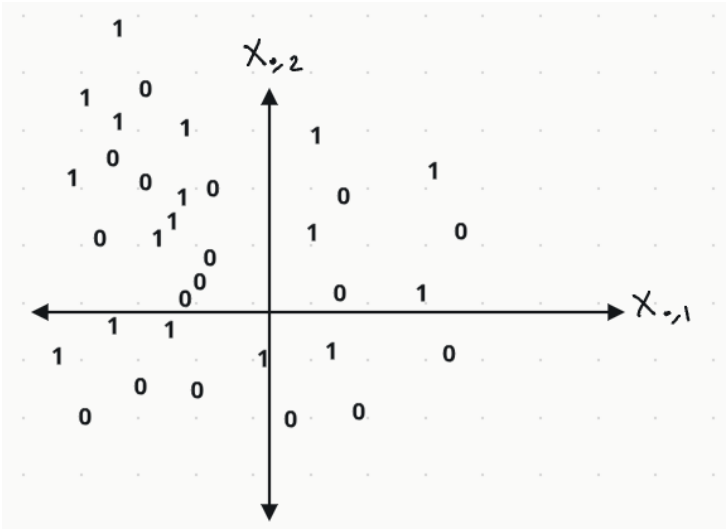

Figure 2: A plot of historical data $\mathbb{D}$, where $\mathcal{X} = \mathbb{R}^2$ and $\mathcal{Y} = \{0, 1\}$.

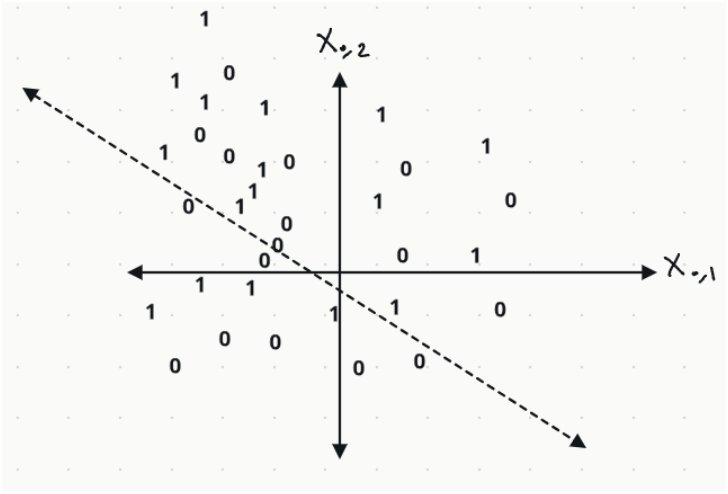

Figure 3: A linear model for the data in Figure 2.

While the functions in this $\mathcal{H}$ do map into $\mathcal{Y}$, we pay the price in high misspecfication error. By the way, θ_1 and θ_2 define what is referred to as a **parameter space**, and the number of parameters (in this case 2) is referred to as the **degrees of freedom**.

A third idea is the following: we can draw a line through the data, as in Figure 3. A set $\mathcal{H}$ that captures this idea is

$$\mathcal{H} = \{\mathbb{I}_{x_2 \geq \theta_1 + \theta_2 x_1} : \theta_1, \theta_2 \in \mathbb{R}\}$$

This is called the **linear model**. Here, we say that if a point lies on or above the line, then the response should be 1; otherwise, 0.

Perceptron

The **perceptron model** follows naturally from the linear model idea that we just described. We'll introduce it with a slight change of notation, using w instead of θ for the parameters:

$$\mathcal{H} = \{\mathbb{I}_{w_0 + w_1 x_1 + w_2 x_2 \geq 0} : w_0, w_1, w_2 \in \mathbb{R}\}$$

The w_0 term is the **bias** or **intercept**, and the w_1 and w_2 terms are the **feature weights**. Next, recall that we typically refer to X as an $n \times p$ matrix corresponding to $\mathbb{D}$:

$$X = \begin{bmatrix} \mathbf{x}_{\cdot,1} & \cdots & \mathbf{x}_{\cdot,p} \end{bmatrix}$$

where $\mathbf{x}_{\cdot,i}$ is our way of denoting the ith column vector of X, which consists of all possible values for the ith feature coming from $\mathbb{D}$. For the perceptron, we'll add another column to X, so we'll refer to it as

$$X = \begin{bmatrix} \vec{\mathbf{1}}_n & \mathbf{x}_{\cdot,1} & \cdots & \mathbf{x}_{\cdot,p} \end{bmatrix}, \quad \text{where } \vec{\mathbf{1}} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

Here we use $\vec{\mathbf{1}}_n$ to denote a column of 1's. Hence, X is now an $n \times (p+1)$ matrix. This extra column is for the *intercepts*, so it corresponds to the intercept term w_0. Now, the **perceptron learning algorithm** $\mathcal{A}$ tries to do the following:

$$\mathcal{A} : \mathbf{w}_* = \underset{\tilde{\mathbf{w}} \in \mathbb{R}^3}{\operatorname{argmin}} \left\{ \sum_{i=1}^{n} \mathbb{I}_{\hat{y}_i \neq y_i} \right\}, \qquad \text{where } \hat{y}_i = \mathbb{I}_{\mathbf{w} \cdot \mathbf{x} \geq 0}$$

where the sum is the *misclassification error*. In other words, the algorithm attempts to find the parameters for a line that minimize the misclassification error. There is no analytic solution to this.

Definition (Perfect linear separability)**.** Given historical data $\mathbb{D}$, there is perfect linear separability if there exists $\mathbf{w}$ such that

$$h(\mathbf{x}) = \mathbb{I}_{\mathbf{w} \cdot \mathbf{x} \geq 0}$$

has no misclassification errors for all $\mathbf{x} \in \mathbb{D}$.

See Figure 4. Assuming a perfectly linear separable $\mathbb{D}$, the "perceptron" learning algorithm is guaranteed to converge to a $\mathbf{w}$ with no error. The algorithm works as follows

(1) Initialize $\mathbf{w}^{t=0} = \vec{\mathbf{0}}_{p+1}$, or to a vector of $p+1$ random values. Here, t is the iteration variable, and $t = 0$ means first iteration.

(2) Fix i. Compute $\hat{y}_i = \mathbb{I}_{\mathbf{w}^t \cdot \mathbf{x}_i} \geq 0$. In other words, we use our current vector of feature weights $\mathbf{w}^t$ and the features $\mathbf{x}_i$ to compute $\mathbf{w}^t \cdot \mathbf{x}_i$. We assign the prediction $\hat{y}_i = 1$ if the result is ≥ 0, and we assign $\hat{y}_i = 0$ otherwise.

(3) Now we update our feature weight $\mathbf{w}^t$. For $j = 0, 1, \ldots, p$, set

$$\begin{aligned} w_0^{t+1} &= w_0^t + (y_i - \hat{y}_i) \cdot 1 \\ w_1^{t+1} &= w_1^t + (y_i - \hat{y}_i) \cdot x_{i,1} \\ &\vdots = \vdots \\ w_p^{t+1} &= w_p^t + (y_i - \hat{y}_i) \cdot x_{i,p} \end{aligned}$$

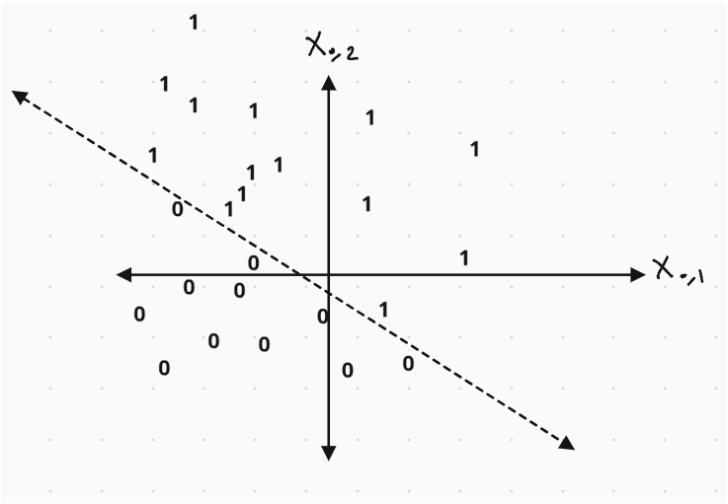

Figure 4: An example of a perfectly linearly separable $\mathbb{D}$

In vector notation, this is:

$$\mathbf{w}^{t+1} = \mathbf{w}^t + (y_i - \hat{y}_i)\mathbf{x}_{\cdot,i}$$

Here, the vector $\mathbf{x}_i$ has been extended to length $n+1$ by prepending 1 (placing an extra entry 1 as its first entry). This step attempts to improve the predictions obtained from the feature weight vector $\mathbf{w}^t$ computed on iteration t.

(4) Repeat steps 2 and 3 for all $i = 1, \ldots, n$. That is, repeat it for each sample point $\mathbf{x}_i$ in $\mathbb{D}$

(5) Repeat steps 2, 3, 4 until misclassification error is zero, meaning $y_i = \hat{y}_i$ for all i, or until B iterations, where B is large.