AI/ML Overview

2 General Classes of problems:

1. <u>Supervised</u> (i.e. Predictive) <u>Learning</u>
2. <u>Unsupervised Learning</u>

<u>Supervised</u>: <u>Goal</u> is To learn a $\boxed{\text{Mapping}}$ from inputs: $X \xrightarrow{\text{To}} \underset{\sim}{Y}$ outputs (labels)

With supervised learning we are given $\boxed{\text{labels}}$.

$$D = \{(x_i, y_i)\}_{i=1}^{n} \quad \text{where} \quad x_i \in \mathbb{R}^d$$

$\mathbb{X}$ usually denotes The $\boxed{\text{design Matrix}}$

where $\mathbb{X}_{n \times d}$.

$$\begin{array}{cc} n & \begin{bmatrix} \mathbb{X} \end{bmatrix} \\ & d \end{array}$$

When $\underbrace{y_i}_{\text{labels}} \in \mathbb{R}$, the problem archetype is usually referred to as $\boxed{\text{Regression}}$

When $y_i \in \underbrace{\{1, \ldots, K\}}_{\text{Discrete classes}}$ the problem is referred to as $\boxed{\text{Classification}}$.

E.g. Predict expected income from education level (Regression)

E.g. Predict whether digital image contains a pedestrian (Binary Classification).

Goal: $\underbrace{Y = f(X)}_{\text{"True" mapping}} + \underbrace{\varepsilon}_{\text{error}}$

Why estimate f?

① Prediction

② Inference

$\hat{Y} = \hat{f}(x)$ (Our estimate)

Flow Chart for a standard AI/ML Algorithm

WANT: $\underbrace{f}_{\text{"True" relationship b/w } X \text{ \& } Y} \approx \hat{f}$ (our estimate of f)

How To Quantify The $\underbrace{\text{proximity}}$ of f To $\hat{f}$?

Use a $\boxed{\text{Loss Function}}$.

Examples of Loss Functions:

① $\boxed{\text{0-1 Loss}}$ (for Classification)

$$L(f(x), \hat{f}(x)) = \begin{cases} 0 & \text{if } f(x) = \hat{f}(x) \\ 1 & \text{if } f(x) \neq \hat{f}(x) \end{cases}$$

② $\boxed{\text{Quadratic Loss}}$

$$L(f(x), \hat{f}(x)) = \{ \, (f(x) - \hat{f}(x))^2$$

Flow Chart

$$\begin{tikzcd}
\boxed{\text{I. Collect Data: } D=\{(x_i,y_i)\}_{i=1}^n} \arrow[r, bend left, "\text{Partition into Training \& Test sets}"] \arrow[rd, bend right] & \boxed{D_1 \text{: Training Set}, \; D_1 \subset D} \\
& \boxed{D_2 \text{: Test Set}, \; D_2 \subset D}
\end{tikzcd}$$

$(D_1 \cap D_2 = \emptyset)$

Note!

II. Train a Model: $\hat{f}$ using D_1
(Regression, NN, SVM)

III. Evaluate Model with Loss function, using D_2

(*) Big Ideas: The smaller (Total) Loss on the Test set, the better the model (ideally).

We use the results on the Test Set to ~~generalization~~ approximate how well the model will generalize to new data.

With $\boxed{\text{unsupervised learning}}$ we are just given data <u>without labels</u>.

In this case we are interested in discovering "interesting structure" in the data; this is sometimes called $\boxed{\text{knowledge discovery}}$ or $\boxed{\text{cluster analysis}}$.

<u>Note</u>: $\boxed{\text{Reinforcement Learning}}$ offers a 3rd problem class in AI/ML – where an "agent" learns how to act or behave when given occasional reward or punishment signals (e.g. Alpha GO (2016), Atari w/ Deep Q-learning (2014))

$\boxed{\text{Parametric Models vs. Non-Parametric Models}}$

Parametric Models consist of a <u>finite</u> (a fixed #) of parameters: $\bar{\Theta} = \langle \Theta_1, \Theta_2, \ldots, \Theta_N \rangle$.

Idea: Using The training data, we "learn" values for These parameters.

Ex. **Polynomial Regression:** Fit a polynomial curve To a data set (e.g. using OLS, etc.)

Linear Regression

$$\hat{f}(x) = \theta_0 + \theta_1 x$$

"Learning" This model entails finding plausible values for θ_0, θ_1

Quadratic Regression

$$\hat{f}(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

Model Parameters: $\vec{\theta} = \langle \theta_0, \theta_1, \theta_2 \rangle$.

Polynomial Regression

$$\hat{f}(x) = \sum_{i=0}^{d} \theta_i x^i \rightarrow (d+1 \text{ parameters})$$

Note: If we use a model with a **small number** of parameters it is usually **easier To train** (requires less Time & data).

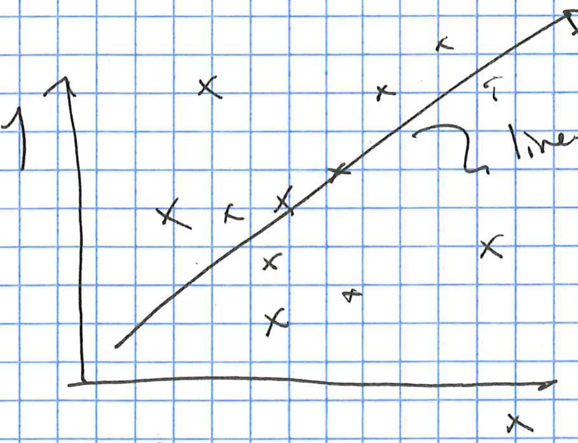
However, a <u>low dimensional Model</u> might not be sufficiently "complex" to capture all The interesting & useful patterns in our data!

(This is called $\boxed{\text{underfitting}}$)

On The other hand, a large dimension / complex model requires more computation & Time on average ; Moreover, an excessively complex model will be "overly tuned" To The training data

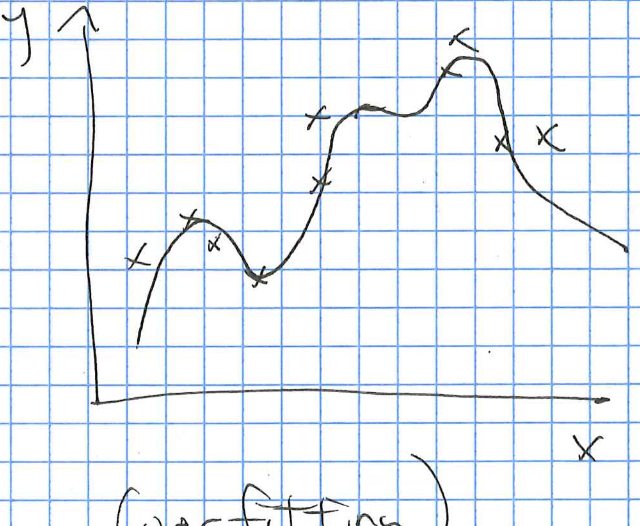
(This is called $\boxed{\text{overfitting}}$)

<u>Conclusion</u>: There is <u>No "free lunch"</u> in data science!



(underfitting)

low complexity Model



(overfitting)

high complexity model

How do I know when I get it right?

This is the "<u>art</u>" of AI/ML/Data Science!

(In other words there is no simple answer)

In general, however, remember that we can assess our model accuracy with a Loss function:

$\boxed{\text{Quad Loss}}$: $$\underbrace{MSE}_{\text{Mean-squared error}} = \frac{1}{n} \underbrace{\sum_{i=1}^{n} (y_i - \hat{f}(x_i))^2}_{\text{over test data}}$$

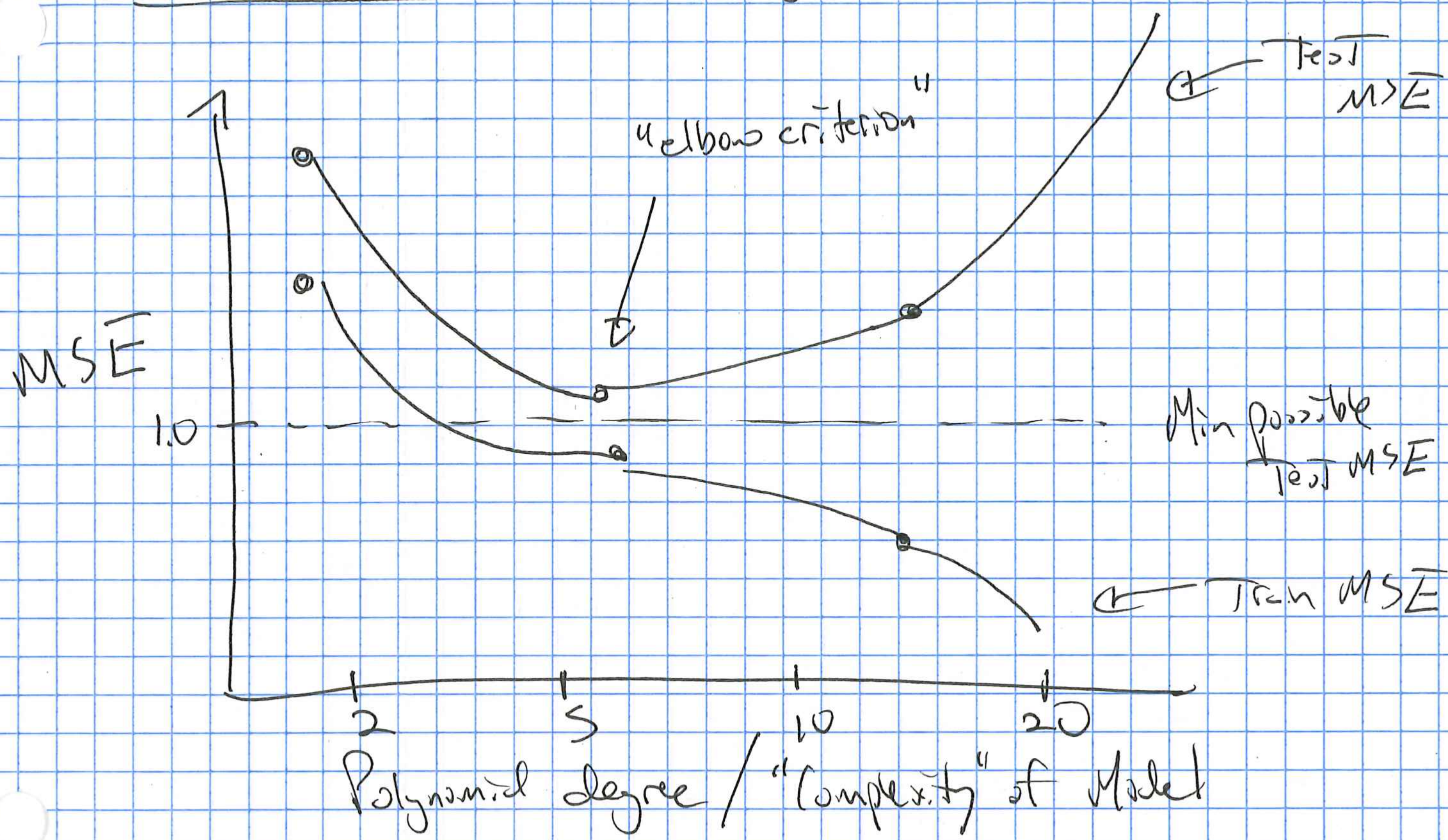
$\boxed{\text{0-1 Loss}}$: $$\underbrace{Ave}_{\text{Take Average}} = \frac{1}{n} \underbrace{\sum_{i=1}^{n} L(y_i, \hat{f}(x_i))}_{\text{counts \# of "mistakes"}}$$

<u>Note:</u> Unfortunately, having a <u>low training error</u> (e.g. MSE) <u>does not guarantee</u> low Test error in general.

One common remedy for <u>parametric Models</u>:

Train several models of varying complexity (e.g. linear regression, quad, cubic regression), compute MSE for each Test set, choose Model w/ lowest MSE.

Common "U-shape" error graph"



Bias-Variance Tradeoff

The "U-shape" phenomenon in the Test MSE is indicative of two competing properties of learned models: Bias & Variance.

low-dimensional (simple models): high Bias & low variance

high-dimensional (complex/flexible models): low Bias & High Variance

More concretely,

The expected $\overset{\text{Test}}{\wedge}$ MSE for a given value x_0,

can always be decomposed into the sum

of $\textcircled{3}$ fundamental quantities:

$$\underbrace{E\left[\left(y_0-\hat{f}(x_0)\right)^2\right]}_{\substack{\text{expected Test}\\ \text{MSE}}} = \operatorname{Var}\left[\hat{f}(x_0)\right] + \left[\operatorname{Bias}\left(\hat{f}(x_0)\right)\right]^2$$

$$+ \underbrace{\operatorname{Var}(\varepsilon)}_{\text{(irreducible error)}}$$

From above, we see that the ideal model will simultaneously achieve $\boxed{\text{low variance \& low bias.}}$

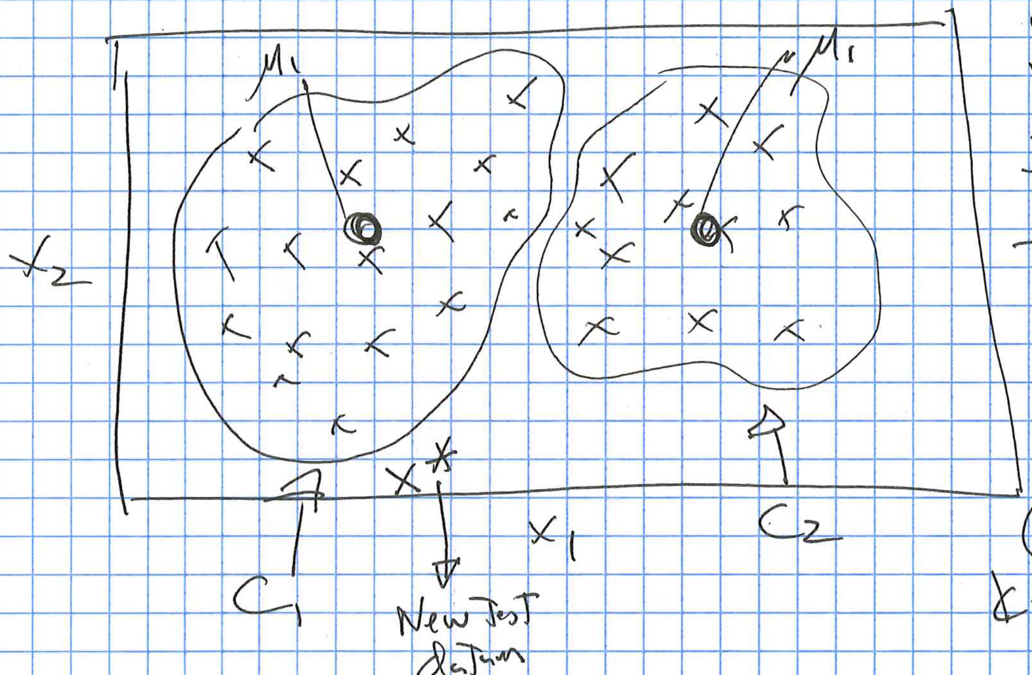
$\boxed{\text{Ex.}}$ Unsupervised Learning

Suppose we have $D = \left\{\left(x_1^{(i)}, x_2^{(i)}\right)\right\}_{i=1}^{n}$ (superscript indicates datum #)

with no class labels (i.e. no y values). $\underbrace{}_{\text{(2-d data)}}$

We will use a <u>clustering method</u> to first cluster the data (let k = # of clusters) & then classify a new datum based on a <u>nearest centroid</u> <u>criterion</u> — this is usually called $\boxed{\text{k-means}}$.

<u>Idea:</u> Suppose we obtain biometric data from men & women (so $k=2$), where $\vec{x} \in \mathbb{R}^2$.



We use the <u>k-means algorithm</u> to identify each training datum as belonging to either cluster 1 or cluster 2. (Don't worry about k-means details now)

$$\mu_1 = \frac{1}{|C_1|} \sum_{C_1} (x_1, x_2) \rightarrow \boxed{\text{centroid of } C1}$$

$$\mu_2 = \frac{1}{|C_2|} \sum_{C_2} (x_1, x_2) \rightarrow \boxed{\text{centroid of } C2}$$

How to classify new test datum (x^*)?

Class for $\boxed{x^* = \arg\min_i \|x^* - \mu_i\|_2}$ → i.e. classify x^* based on nearest cluster center.

Note that the previous examples are considered <u>parametric models</u>, because the # of parameters is fixed & once we "learn" these parameter values, the training data can be "discarded" when we perform test prediction/classification.

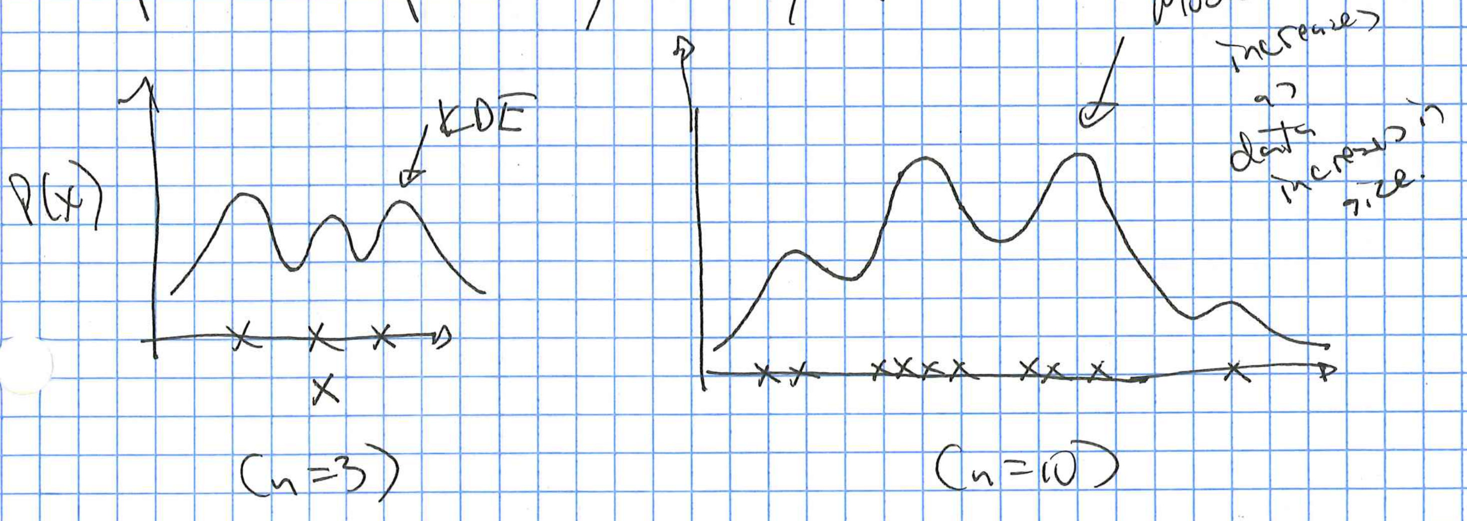
What would a non-parametric model look like?

<u>Two conditions</u>: $\textcircled{1}$ The # of parameters is <u>not</u> <u>fixed</u>, and typically grows as the training set gets larger;

$\textcircled{2}$ The training data is retained for prediction/classific. (necessarily)

$\boxed{\text{Ex.}}$ $\boxed{\text{Kernel Density estimation}}$

<u>Idea</u>: fit a smooth/flexible "histogram" to data - produce a probability density function.



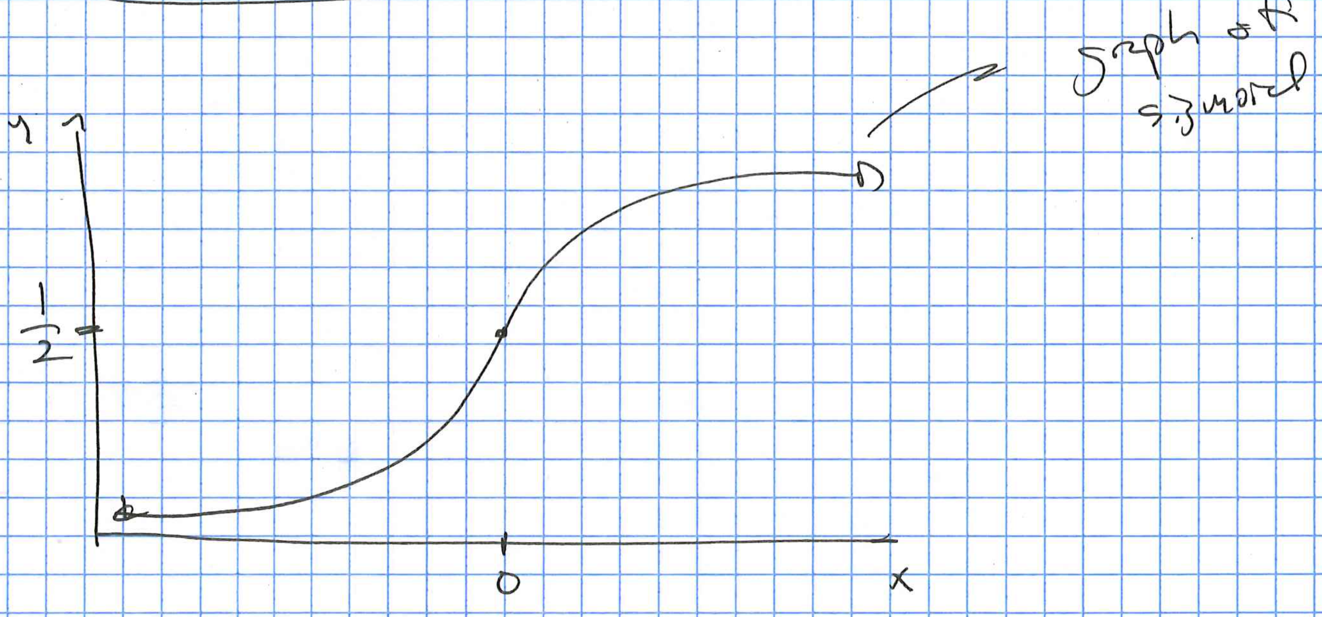
One more useful & common Model

E7. Logistic Regression : A parametric (Binary) Classification Model.

Def. The sigmoid function :

$$\boxed{\operatorname{sigm}(x) = \frac{1}{1+e^{-x}}}$$

used for logistic regression & as "activation" function in NNs.



Note: $\lim_{x \to +\infty} \operatorname{sigm}(x) = 1$

$\lim_{x \to -\infty} \operatorname{sigm}(x) = 0$

$\boxed{\text{Idea:}}$ Parameter Models $\vec{\theta} = \langle \theta_0, \dots, \theta_N \rangle$

$$\underbrace{p}_{\text{}}(\underset{\substack{\text{Binary}\\\text{class}}}{y} \mid \underset{\substack{\text{datum}\\\text{for}\\\text{classification}}}{\vec{x}}, \underset{\substack{\text{Model}\\\text{parameters}}}{\theta}) = \underbrace{\text{Ber}}_{\text{Bernoulli}}\left(y \mid \text{sigm}\left(\theta^T \vec{x}\right)\right)$$

$y \in \{0,1\}$

Steps: ① With training, learn Model parameters $\vec{\theta}$.

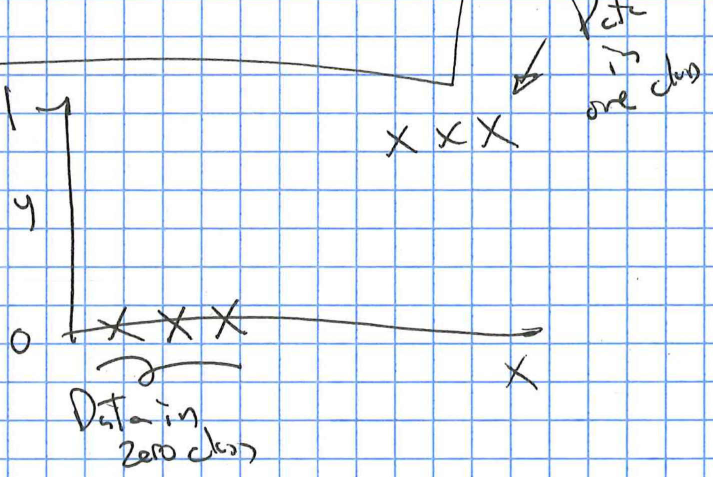
② evaluate : $\text{sigm}(\theta^T \vec{x}) = \dfrac{1}{1 + e^{-\theta^T \vec{x}}}$

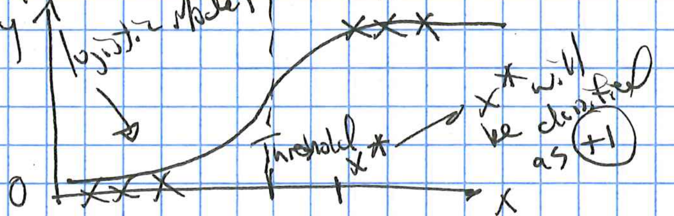
③ Apply a $\boxed{\text{decision rule}}$ (Theshold)

e.g. $\boxed{\hat{y}(x) = 1 \iff p(y=1 \mid x) > .5}$

$\boxed{\text{Toy example}}$: Training Data → Recall : $y \in \{0,1\}$

↓ Model Fit


y
X
Data in one class
Data in zero class


y
x
logistic Model
Threshold
x*
x* will be classified as (+1)