Linear Regression

Mahdi Roozbahani
Georgia Tech

These slides are inspired based on slides from Le Song, Chao Zhang, Yaser Abu-Mostafa, Andrew Zisserman.

Outline

- Supervised Learning
- Linear Regression
- Extension

$X_{n \times d}$ $\quad Y_{n \times 1} \rightarrow$ actual

Supervised Learning: Overview

Functions $\mathcal{F}$

$\color{red}{f} : \mathcal{X} \rightarrow \mathcal{Y}$

Training data

$\{\color{red}{(x_i, y_i)} \in \mathcal{X} \times \mathcal{Y}\}$

LEARNING

$$\boxed{\begin{array}{l} \text{find } \color{red}{\hat{f}} \in \mathcal{F} \\ \text{s.t. } y_i \approx \hat{f}(x_i) = \hat{y} \end{array}}$$

predicted

Learning machine

New data

$\boxed{x}$

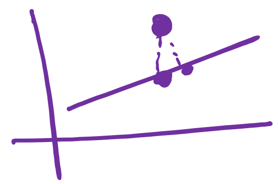
PREDICTION

$$\boxed{\color{red}{y} = \hat{f}(x)}$$

Supervised Learning: Two Types of Tasks

Given: training data $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$

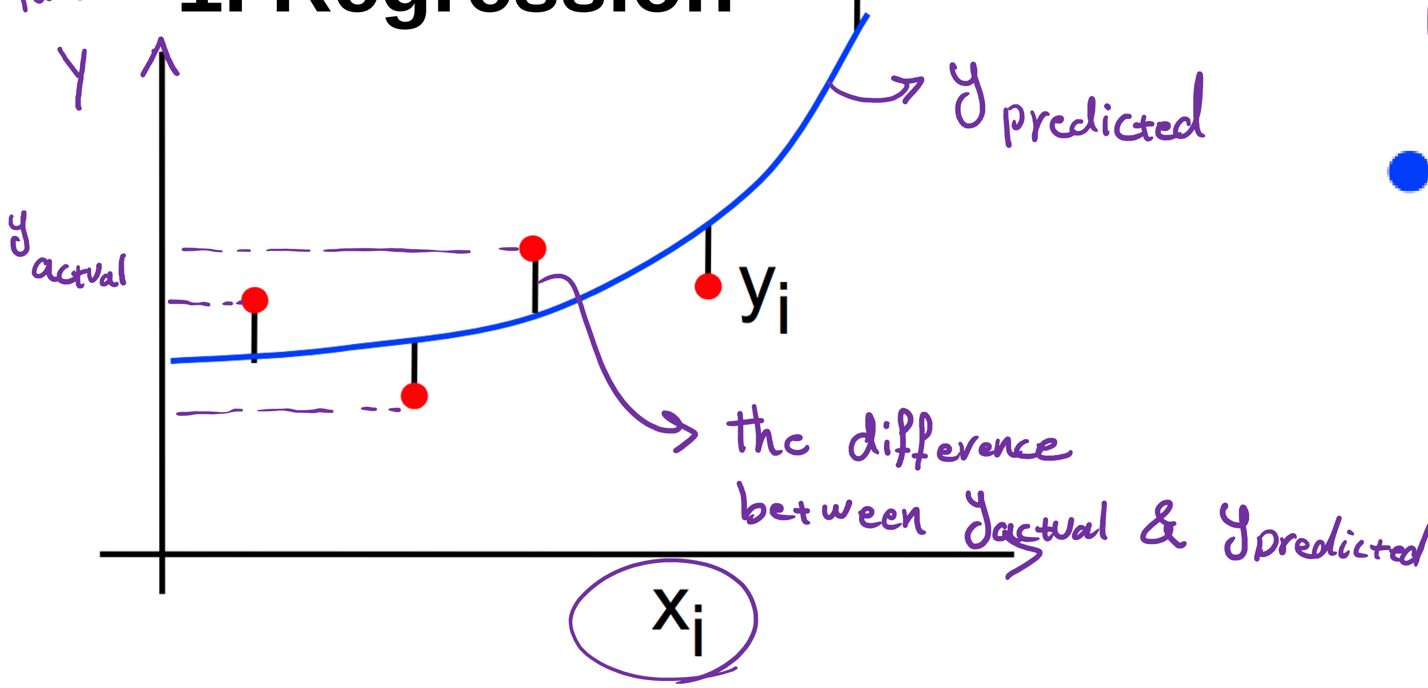
Learn: a function $f(\mathbf{x}) : y = f(\mathbf{x})$



When y is continuous:

Targeted values = labels

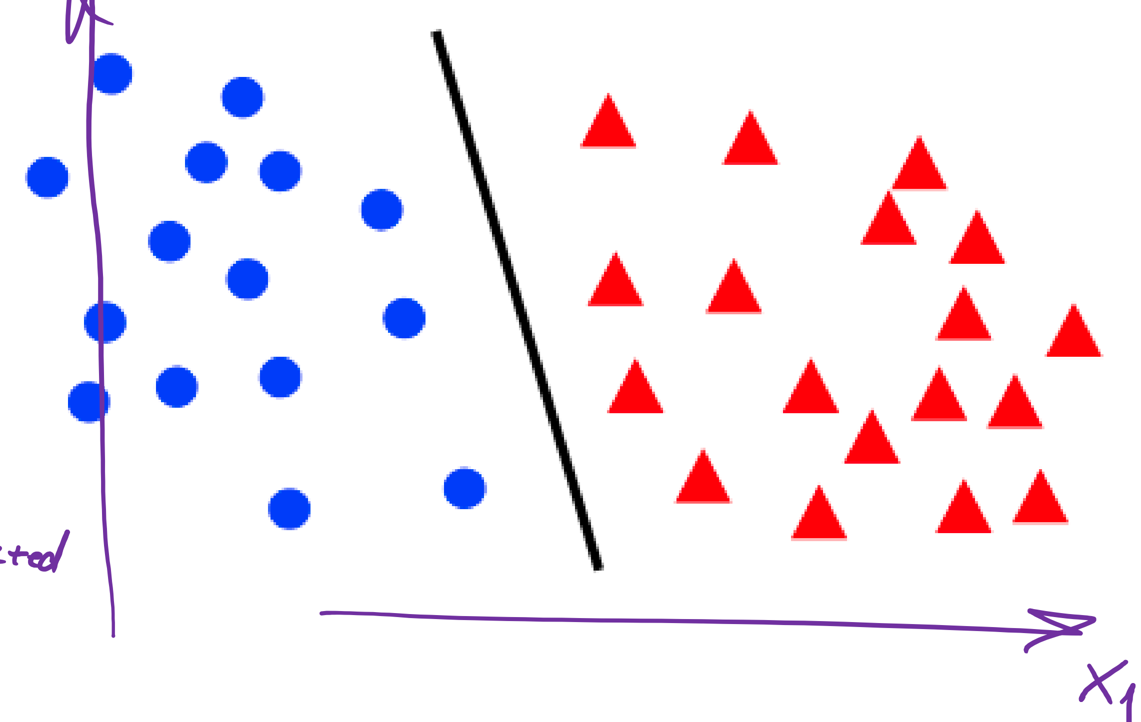
1. Regression



Curve fitting

When y is discrete:

2. Classification

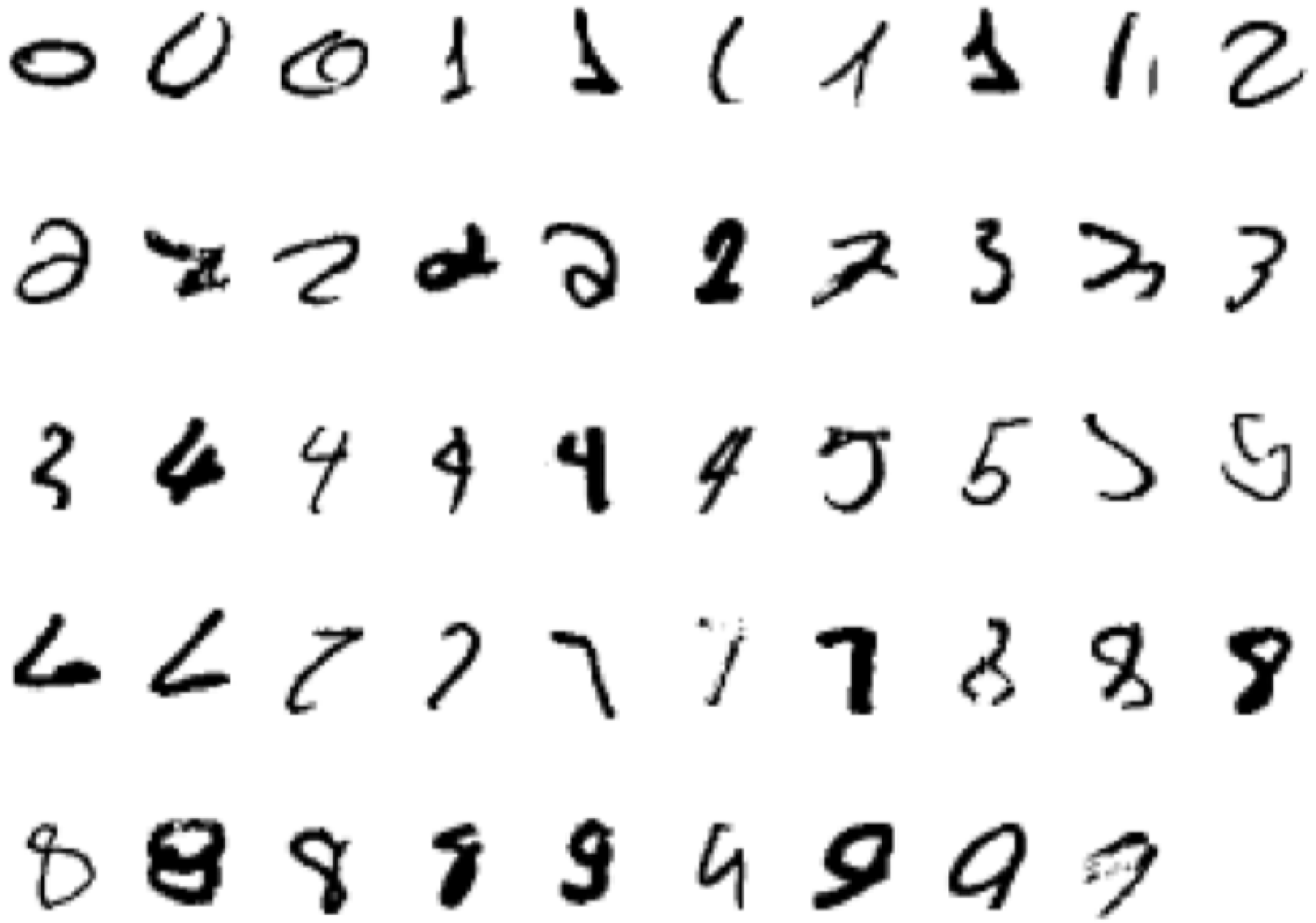


Class estimation

Classification Example 1: Handwritten digit recognition

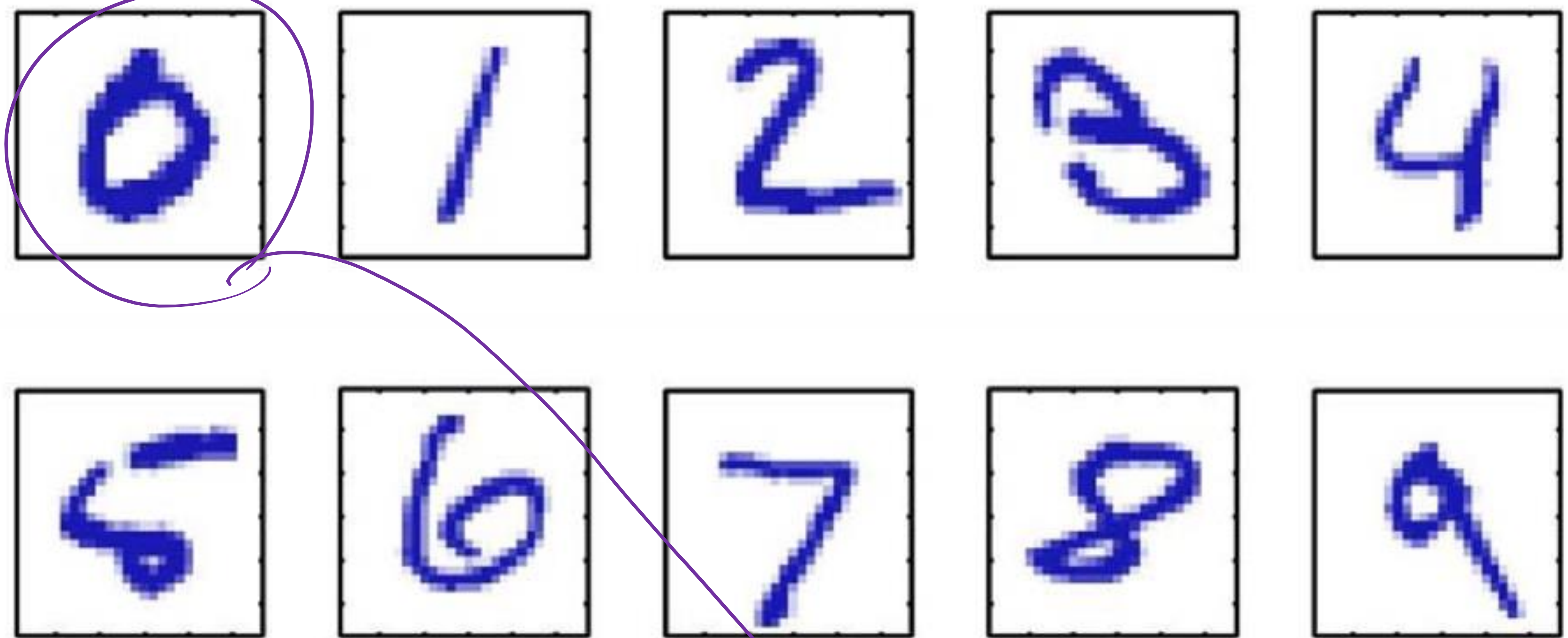
As a supervised classification problem

Start with training data, e.g. 6000 examples of each digit



- Can achieve testing error of 0.4%
- One of first commercial and widely used ML systems (for zip codes & checks)

Classification Example 1: Hand-Written Digit Recognition



Images are 28 x 28 pixels

A classification problem

Represent input image as a vector $\mathbf{x} \in \mathbb{R}^{784}$

Learn a classifier $f(\mathbf{x})$ such that,

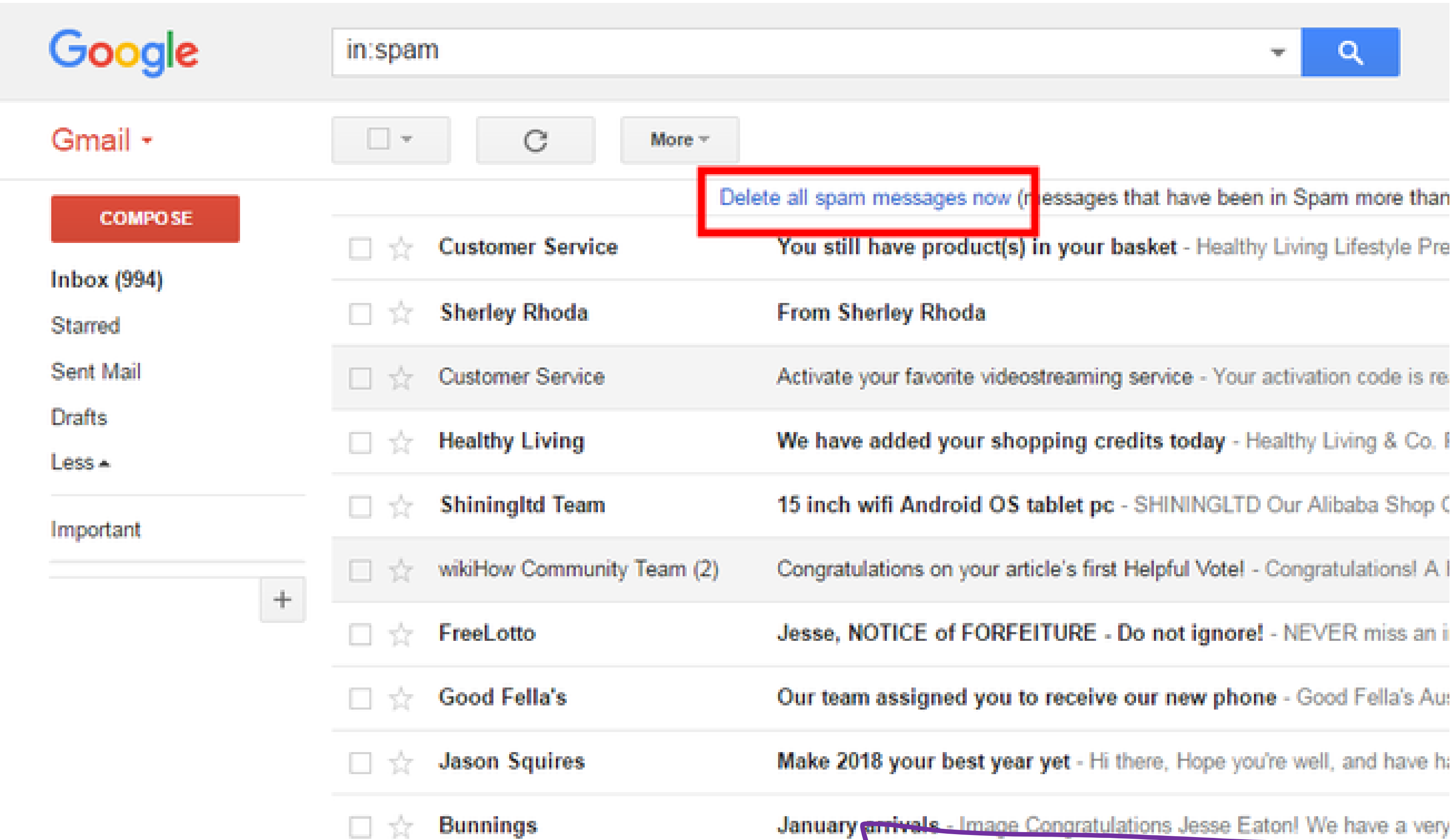
$$f : \mathbf{x} \rightarrow \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

$x =$

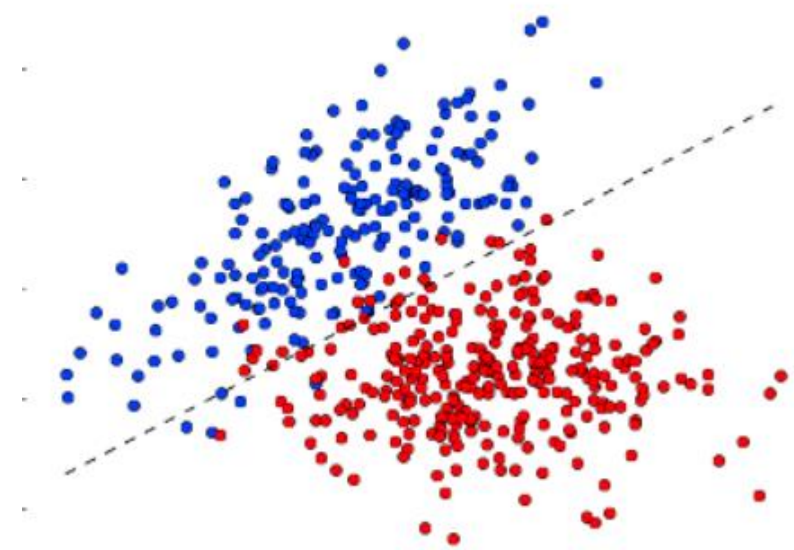
$10 \times d$

Classification Example 2: Spam Detection

TF-IDF



NOT SPAM



SPAM

"Bag of words encoding"

$$
X = \begin{array}{c|ccccc}
 & \text{all} & \text{great} & \text{the} & \text{bad} & \cdots \\
\text{Email 1} & 2 & 10 & 0 & 1 & 0 \\
\text{Email 2} & 0 & 0 & 2 & 3 & 4 \\
\vdots & & & & & \\
\text{Email}_n & & & & &
\end{array}
$$

A classification problem

- This is a classification problem
- Task is to classify email into spam/non-spam
- Data x_i is word count
- Requires a learning system as "enemy" keeps innovating

Regression Example 1: Apartment Rent Prediction

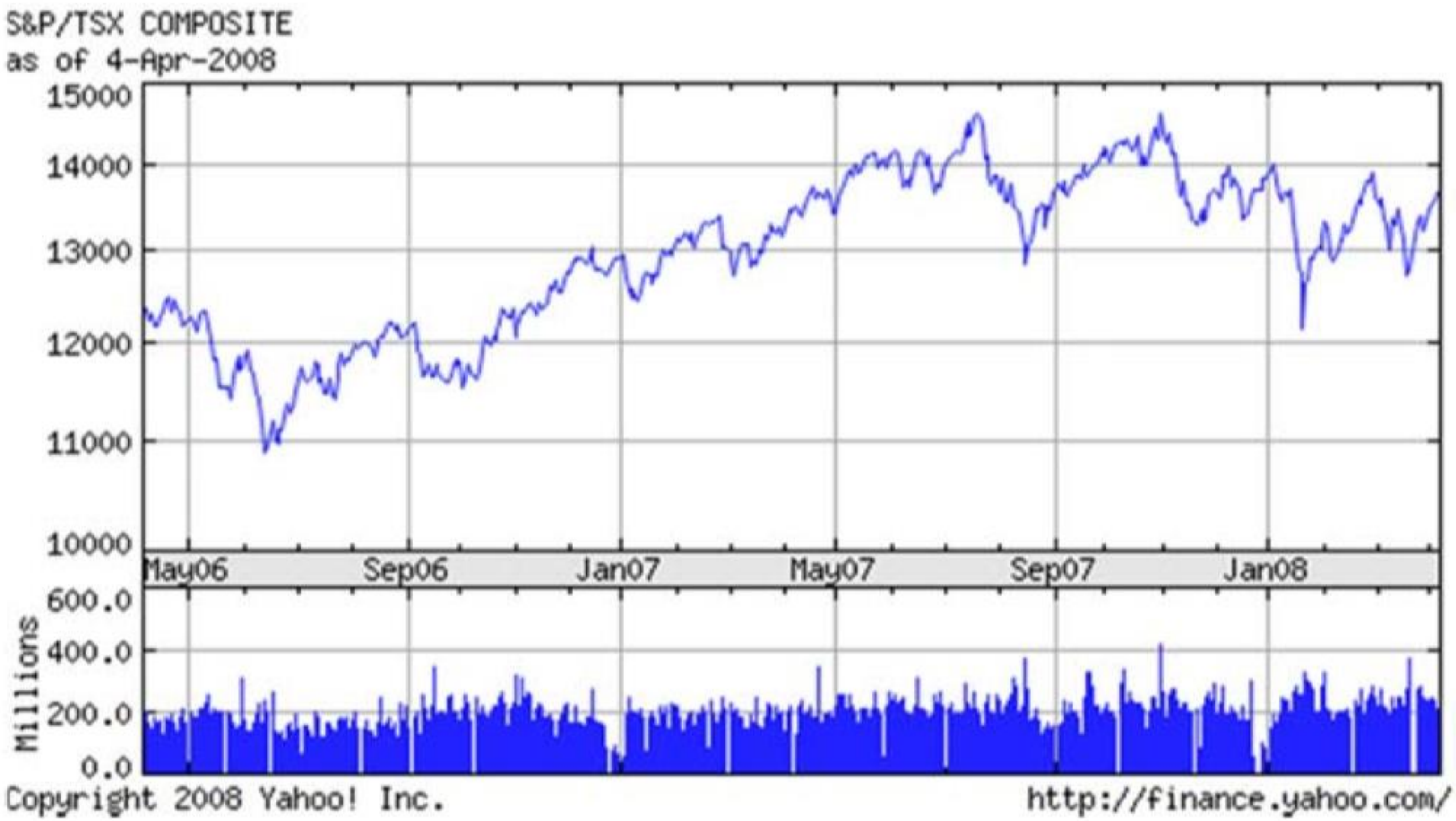
- Suppose you are to move to Atlanta
- And you want to find the **most reasonably priced** apartment satisfying your **needs:**

  square-ft., # of bedroom, distance to campus …

A regression problem

Living area (ft^2)	# bedroom	Rent ($)
230	1	600
506	2	1000
433	2	1100
109	1	500
…		
150	1	?
270	1.5	?

Regression Example 2: Stock Price Prediction



- Task is to predict stock price at future date

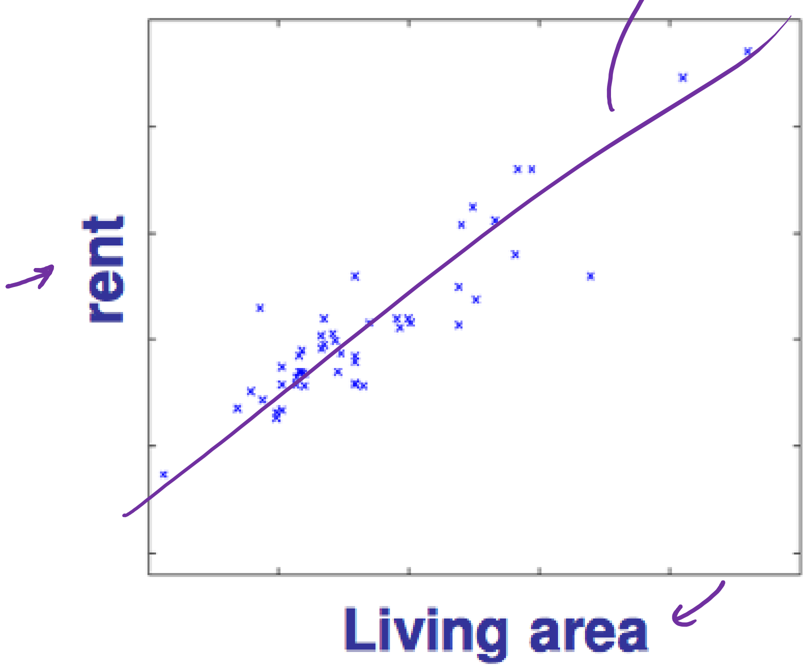
A regression problem

slope

$\hat{y} = mx + b$ Intercept

Living area (feature)

$\hat{y} = \underbrace{\theta_0}_{\text{bias term}} + \theta_1 X_1$



- Features:
  - Living area, distance to campus, # bedroom …
  - Denote as $x = (x_1, x_2, \ldots, x_d)$

Linear combination of FEATURES

$\hat{y} \in \mathbb{R}$

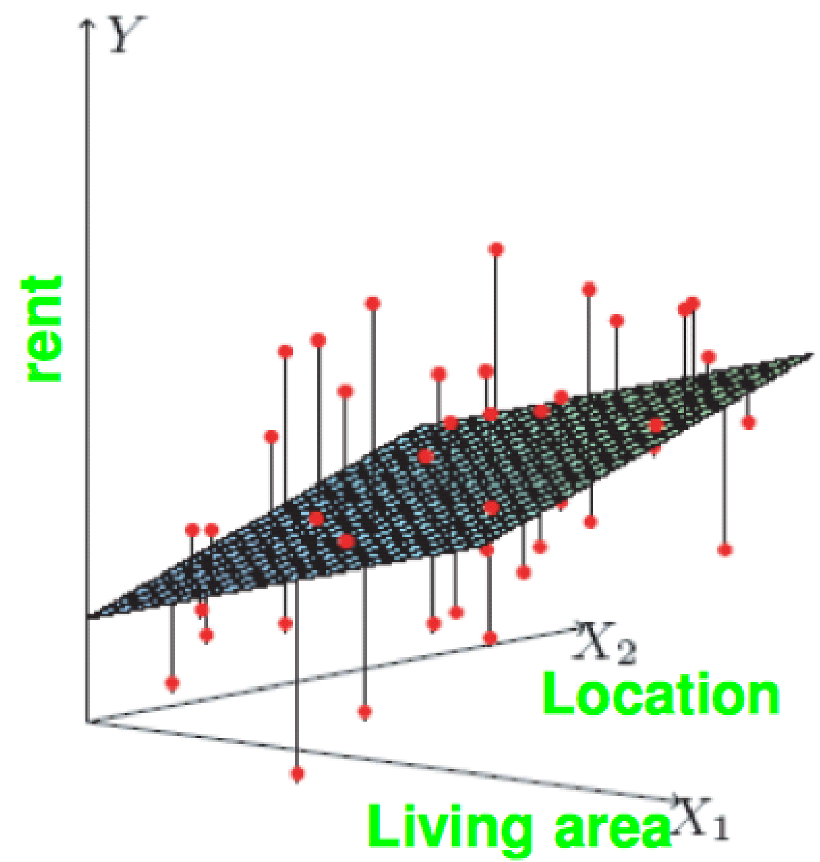
$\hat{y} = m_1 x_{Loc} + m_2 x_{Liv} + b$

- Target:
  - Rent
  - Denoted as y

$\hat{y} = \theta_0 + \theta_1 X_1 + \theta_2 X_2 = \boxed{X\theta}$

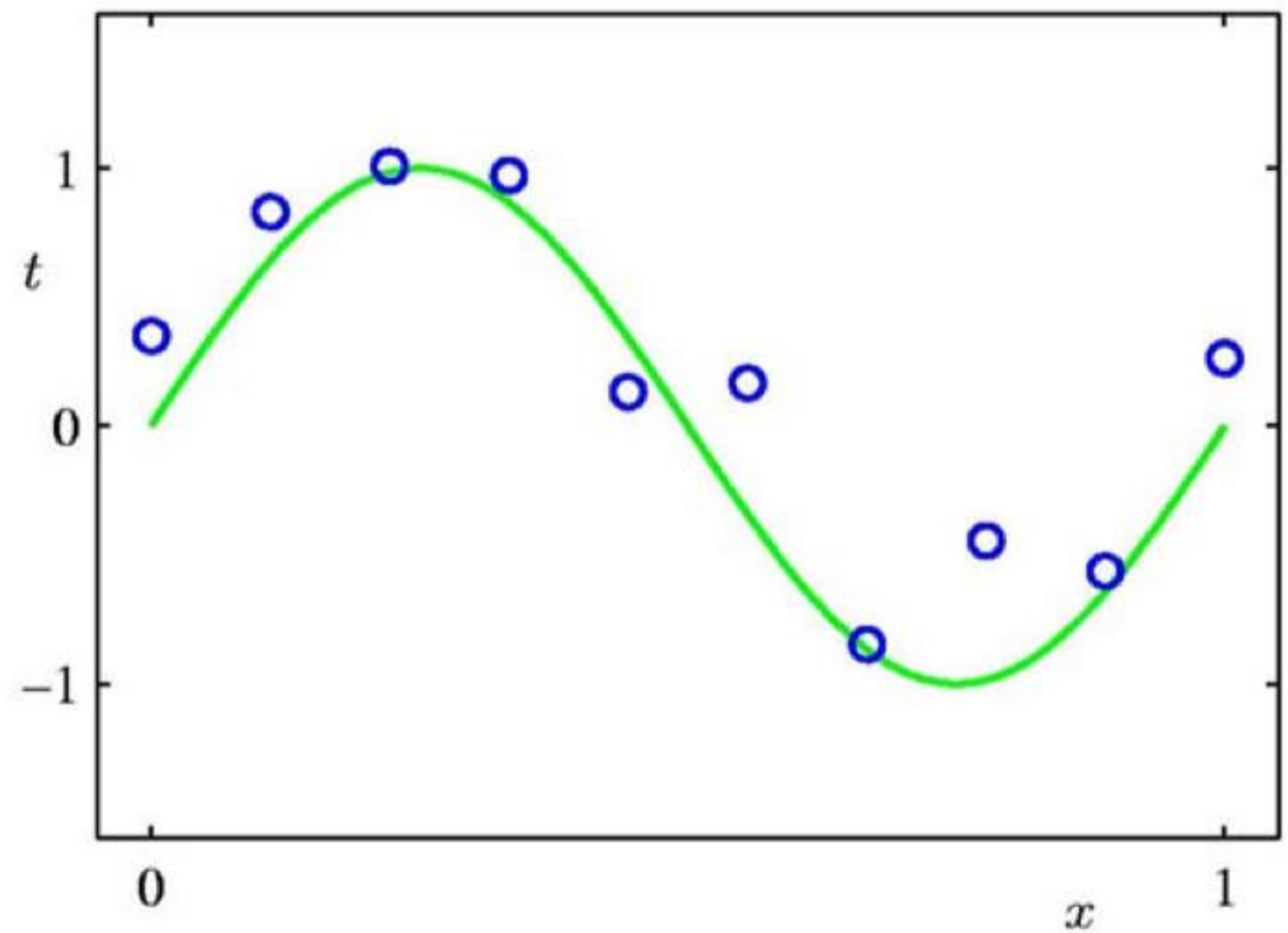
$\underset{1\times(d+1)}{X_i} = [1 \;\; X_1 \;\; X_2 \; \ldots \; X_d]$

$\underset{(d+1)\times 1}{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix}$



- Training set:
  - $x = \{x_1, x_2, \ldots, x_n\} \in R^d$
  - $y = \{y_1, y_2, \ldots, y_n\}$

Regression: Problem Setup



- Suppose we are given a training set of N observations

$$(x_1, \ldots, x_N) \text{ and } (y_1, \ldots, y_N), x_i, y_i \in \mathbb{R}$$

- Regression problem is to estimate y(x) from this data

Outline

- Supervised Learning
- Linear Regression
- Extension

Linear Regression

- Assume y is a linear function of x (features) plus noise ϵ

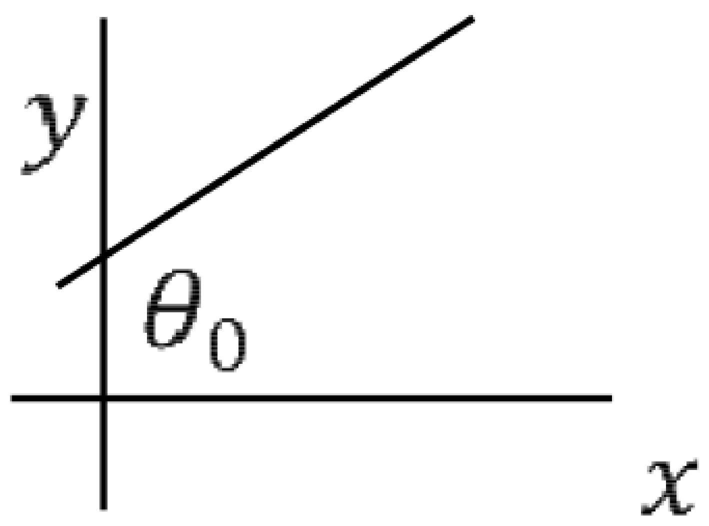
$$y = \theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d + \epsilon$$

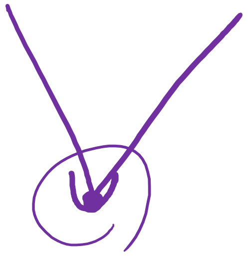
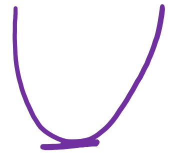
- where ϵ is an error term of unmodeled effects or random noise
- Let $\theta = (\theta_0, \theta_1, \ldots, \theta_d)^T$, and augment data by one dimension

linear combination of features

noise

- Then $\hat{y} = x\theta + \epsilon$



$|x|$  x^2 

$|y_i - x_i\theta|$

Least Mean Square Method

- Given n data points, find θ that minimizes the mean square error

y_{actual} $\quad$ $y_{predicted}$ $\quad$ X_i $1\times(d+1)$ $\quad$ θ $(d+1)\times 1$

Training $\qquad$ $$\hat{\theta} = argmin_{\,\theta}\ \overset{E(\theta)}{L(\theta)} = \frac{1}{n}\sum_{i=1}^{n}(\underset{1\times1}{y_i} - \underset{1\times1}{x_i\theta})^2$$

- Our usual trick: set gradient to 0 and find parameter $\frac{\partial L(\theta)}{\partial\theta} = 0$

$$\frac{\partial L(\theta)}{\partial\theta} = -\frac{2}{n}\sum_{i=1}^{n} x_i^T(y_i - x_i\theta) = 0$$

$$\boxed{\frac{\partial L(\theta)}{\partial\theta} = -\frac{2}{n}\cancel{\sum_{i=1}^{n}} x_i^T y_i + \frac{2}{n}\cancel{\sum_{i=1}^{n}} x_i^T x_i\theta = 0}$$

Matrix form

$$x = \begin{bmatrix} 1 & x_1^{\{1\}} & \cdots & x_1^{\{d\}} \\ 1 & x_2^{\{1\}} & \ddots & x_2^{\{d\}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n^{\{1\}} & \cdots & x_n^{\{d\}} \end{bmatrix}_{n \times (d+1)} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}_{n \times 1} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix}_{(d+1) \times 1}$$

$$MSE(\theta) = argmin_{\,\theta}\, L(\theta) = \frac{1}{n}(\underbrace{y}_{n\times 1} - \overbrace{\underbrace{x}_{(n\times d+1)}\underbrace{\theta}_{(d+1\times 1)}}^{n\times 1})^{\mathrm{T}}(\underbrace{y}_{n\times 1} - \underbrace{x\theta}_{n\times 1})$$

$1\times n$ $\quad$ $n\times 1$ $\quad$ 1×1

$$x\theta = \begin{bmatrix} \theta_0 + \theta_1 x_1^{\{1\}} + \theta_2 x_1^{\{2\}} + \cdots + \theta_d x_1^{\{d\}} \\ \theta_0 + \theta_1 x_2^{\{1\}} + \theta_2 x_2^{\{2\}} + \cdots + \theta_d x_2^{\{d\}} \\ \vdots \\ \theta_0 + \theta_1 x_n^{\{1\}} + \theta_2 x_n^{\{2\}} + \cdots + \theta_d x_n^{\{d\}} \end{bmatrix}_{n \times 1}$$

Matrix Version and Optimization

$$\frac{\partial L(\theta)}{\partial \theta} = -\frac{2}{n}\sum_{i=1}^{n} x_i^T y_i + \frac{2}{n}\sum_{i=1}^{n} x_i^T x_i \theta = 0$$

Let's rewrite it as:

$$\frac{\partial L(\theta)}{\partial \theta} = -\frac{2}{n}\overset{n \times (d+1)}{\boxed{(x_1, \dots, x_n)^T}}(y_1, \dots, y_n) + \frac{2}{n}(x_1, \dots, x_n)^T(x_1, \dots, x_n)\theta = 0$$

Define X $= (x_1, \dots, x_n)$ and y $= (y_1, \dots, y_n)$

$$\frac{\partial L(\theta)}{\partial \theta} = -\frac{2}{n}\overbrace{\overset{(d+1)\times n}{\mathrm{X}^{\mathrm{T}}}\underset{n\times 1}{\mathrm{y}}}^{(d+1)\times 1} + \frac{2}{n}\overbrace{\overset{(d+1)\times(d+1)}{X^T X}\times\overset{(d+1)\times 1}{\theta}}^{(d+1)\times 1} = 0$$

$$\Rightarrow \overset{(d+1)\times 1}{\theta} = \overbrace{(X^T X)^{-1}}^{(d+1)\times(d+1)}\overset{(d+1)\times n}{X^T}\overset{n\times 1}{y} = X^+ y$$

X^+ is the **pseudo-inverse** of X

$$X^T X X^+ = X^T$$

$$\theta = (X^T X)^{-1} X^T y = X^+ y$$

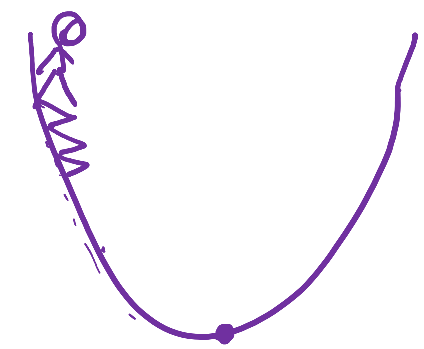
$X_{n \times d}$ $\quad$ $n =$ instances $\quad$ $d =$ dimension

$$X^T X = \begin{bmatrix} d \times n \end{bmatrix} \begin{bmatrix} n \times d \end{bmatrix} = \begin{bmatrix} d \times d \end{bmatrix}$$

Not a big matrix because $n \gg d$This matrix is invertible most of the times. If we are VERY unlucky and columns of $\mathbf{X^T}\boldsymbol{X}$ are not linearly independent (it’s not a full rank matrix), then it is not invertible.

Alternative Way to Optimize

- The matrix inversion in $\theta = (X^T X)^{-1} X^T y$ can be very expensive to compute



- $$\frac{\partial L(\theta)}{\partial \theta} = -\frac{2}{n} \sum_{i=1}^{n} x_i^T (y_i - x_i \theta)$$

$$\theta^{t+1} \leftarrow \theta^{t} - \alpha \frac{\partial L(\theta)}{\partial \theta}$$

learning step

- Gradient descent

$$\hat{\theta}^{t+1} \leftarrow \hat{\theta}^{t} + \frac{\alpha}{n} \sum_{i=1}^{n} x_i^T (y_i - x_i \theta)$$

- Stochastic gradient descent (use one data point at a time)

SGD $\leftarrow$ $$\boxed{\hat{\theta}^{t+1} \leftarrow \hat{\theta}^{t} + \beta_t \times x_i^T (y_i - x_i \theta)}$$

Batch $\leftarrow$ BGD

Training data

$\underset{n \times (d+1)}{X}$ $\quad$ $\underset{n \times 1}{y}$ Actual labels $\quad$ $\underset{(d+1) \times 1}{\Theta}$

$f(x) = \hat{y}$ predicted labels

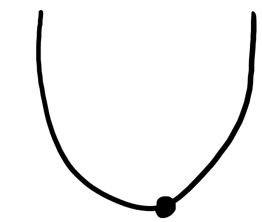
y_p is as close as possible to y_a

$$f(x) = \hat{y} = \Theta_0 + \Theta_1 X_1 + \cdots + \Theta X = X\Theta$$

Linear combination of features

actual $\quad$ predicted

$$L(\Theta) = E(\Theta) = \frac{1}{N} \sum_{i=1}^{N} \left(\underbrace{y_i}_{\text{actual}} - \underbrace{\hat{y}_i}_{\text{predicted}}\right)^2 = \frac{1}{N} \sum_{i=1}^{N} (y_i - x_i\Theta)^2$$

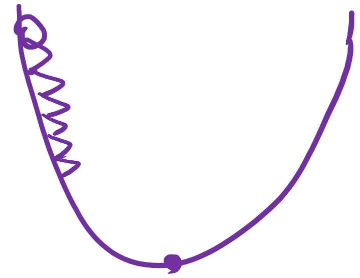


i.e. convex

$$= E\left[(y_i - X_i\Theta)^2\right] = \text{bias}^2 + \text{Variance}$$

$$\underset{(d+1) \times 1}{\Theta} = \underbrace{\underbrace{(X^T X)^{-1}}_{(d+1) \times (d+1)} \underset{(d+1) \times n}{X^T} \underset{n \times 1}{Y}}_{(d+1) \times 1}$$

① GD



i.e. $L(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - x_i\theta)^2$

$t=0$ Initialize θ with random numbers or zero

$$\theta^{t+1} \leftarrow \theta^{t} - \alpha \frac{\partial L(\theta)}{\partial(\theta)}$$

$$\frac{\partial L(\theta)}{\partial \theta} = -\frac{2}{N}\sum_{i=1}^{N} x_i^T (y_i - x_i\theta)$$

$$\theta^{t+1} \leftarrow \theta^{t} + \frac{\alpha}{N}\sum_{i=1}^{N} x_i^T (y_i - x_i\theta)$$

$$\theta^{t} \rightarrow \theta^{t+1}$$

---

② SGD

$$\theta^{t+1} \leftarrow \theta^{t} + \beta\, x_i^T (y_i - x_i\theta)$$

---

③ BGD or Batch GD

Using a Batch of datapoints

In total, I have 100 data points and I use 20 datapoints in each iteration

It takes 5 iteration to reach to one epoch

Recap

- Stochastic gradient update rule

$$\hat{\theta}^{t+1} \leftarrow \hat{\theta}^{t} + \beta_t \ \times x_i^T (y_i - x_i \theta)$$

  - Pros: on-line, low per-step cost
  - Cons: coordinate, maybe slow-converging
- Gradient descent

$$\hat{\theta}^{t+1} \leftarrow \hat{\theta}^{t} + \frac{\alpha}{n} \sum_{i=1}^{n} x_i^T (y_i - x_i \theta)$$

  - Pros: fast-converging, easy to implement
  - Cons: need to read all data
- Solve normal equations

$$\theta = (X^T X)^{-1} X^T y$$

  - Pros: a single-shot algorithm! Easiest to implement.
  - Cons: need to compute inverse $(X^T X)^{-1}$, expensive, numerical issues (e.g., matrix is singular ..)

Linear regression for classification

Raw Input $x = (\overset{1}{\cancel{x_0}}, x_1, \ldots, x_{25\cancel{5}6})$

Linear model $(\theta_0, \theta_1, \ldots, \theta_{25\cancel{5}6})$

16

16

Extract useful information

$intensity\ and\ symmetry\ x = (1, x_1, x_2)$

Sum up all the pixels = intensity
Symmetry = -(difference between flip version)

$$\hat{y} = \Theta_0 + \Theta_1 \underbrace{X_1}_{\text{intensity}} + \Theta_2 \underbrace{X_2}_{\text{symmetry}} \rightsquigarrow \mathbb{R}$$

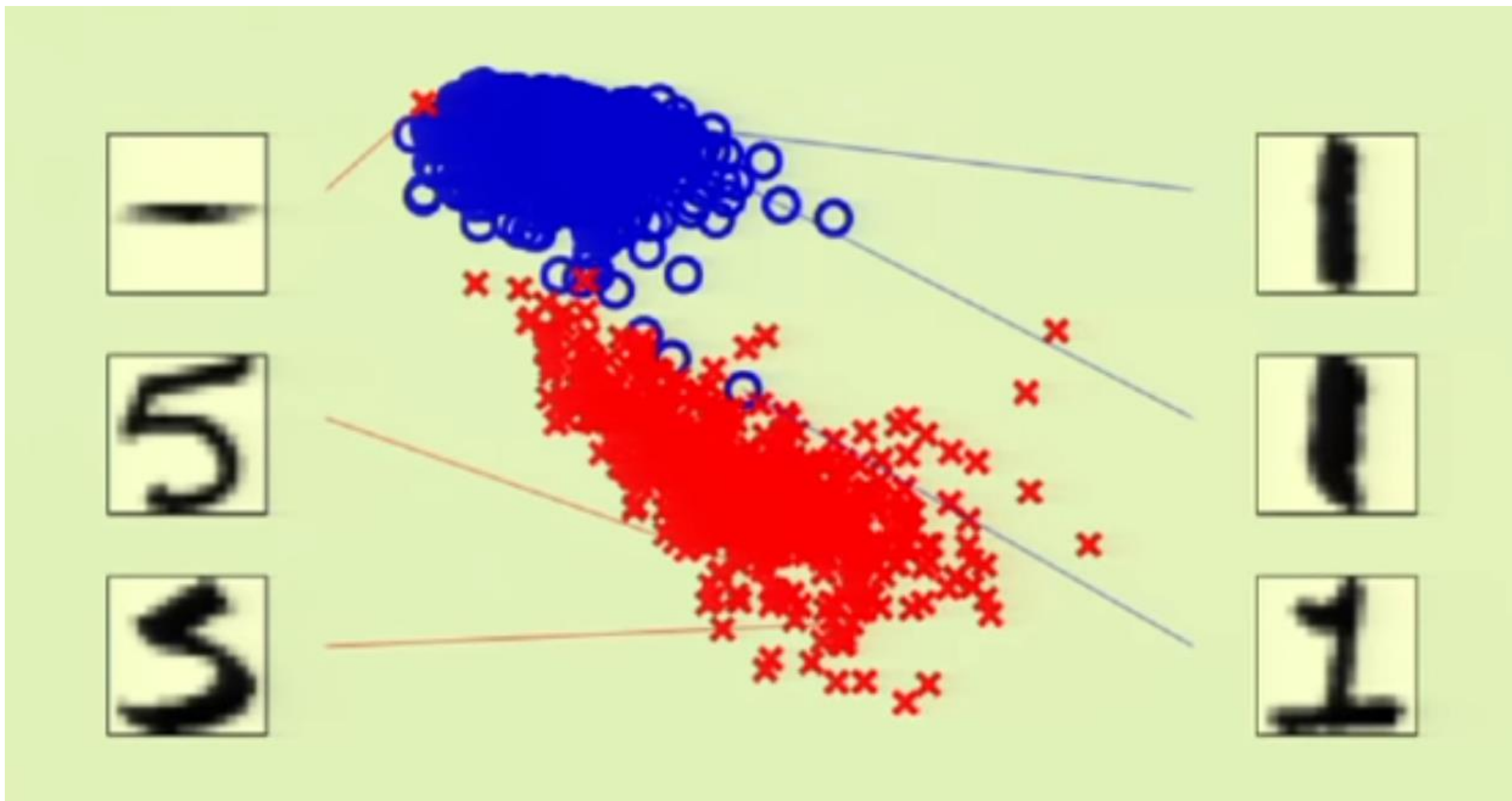
$$x = (1, x_1, x_2)$$

$$x_1 = intensity \; x_2 = symmetry$$

$$\begin{array}{c|cc} & \text{int} & \text{sym} \\ \hline x_1 & - & -- \\ x_2 & & \\ \vdots & & \\ x_n & & \end{array}$$

It is almost linearly separable

symmetry



intensity

Linear regression for classification

Binary-valued functions are also real-valued $\pm 1 \in R$

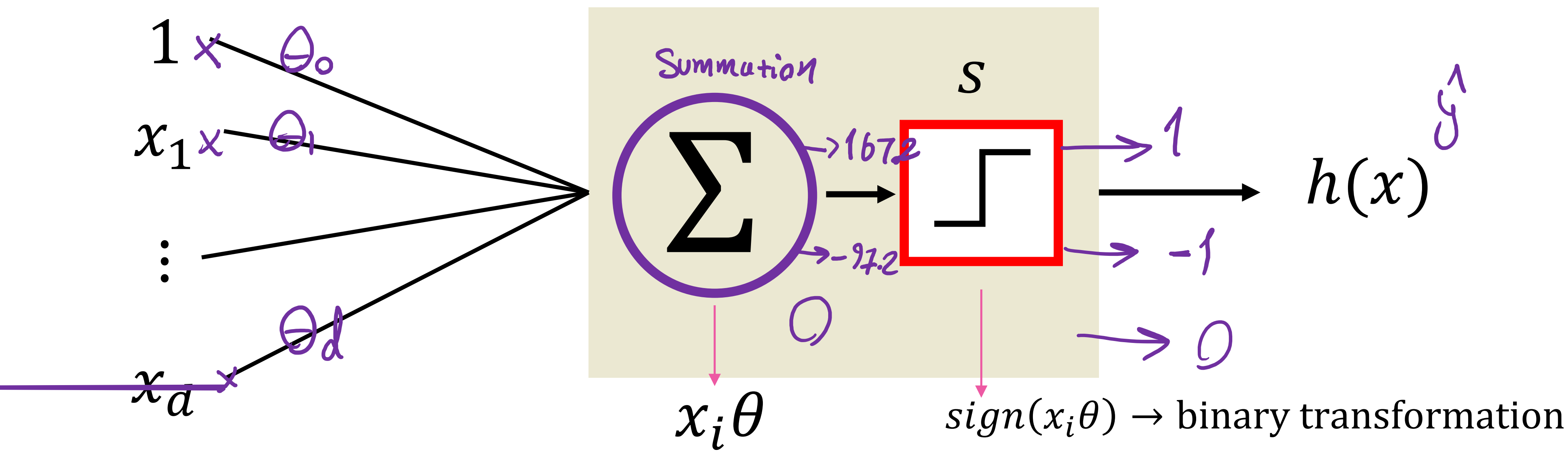
Use linear regression $x_i\theta \approx y_n = \pm 1$ $\quad i =$ index of a data-point

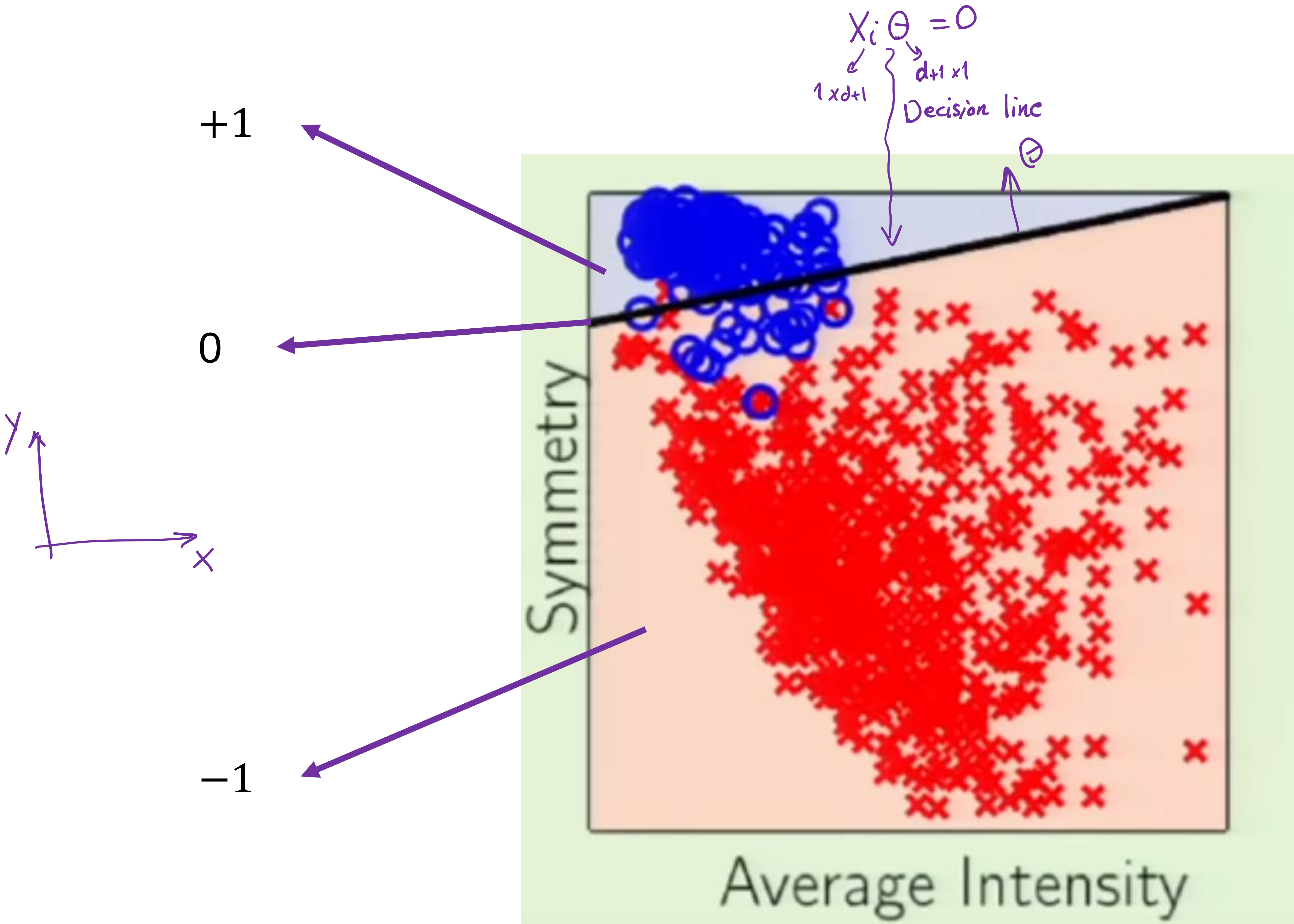
Let's calculate, $sign(x_i\theta) = \begin{cases} -1 & x_i\theta < 0 \\ 0 & x_i\theta = 0 \\ 1 & x_i\theta > 0 \end{cases}$

$\Sigma = \theta_0 + \theta_1 X_1 + \dots + \theta_d X_d$ A linear combination of features

One learning block

For one data point (data-point i) with **d** dimensions (instance):



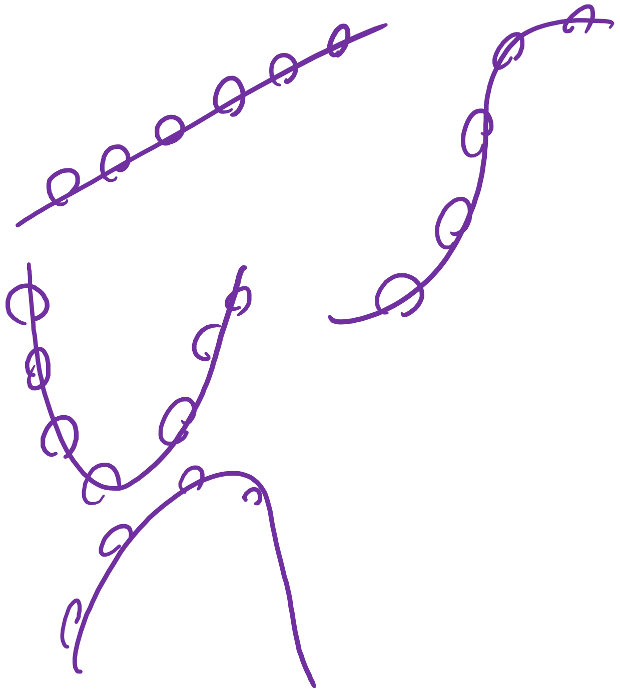


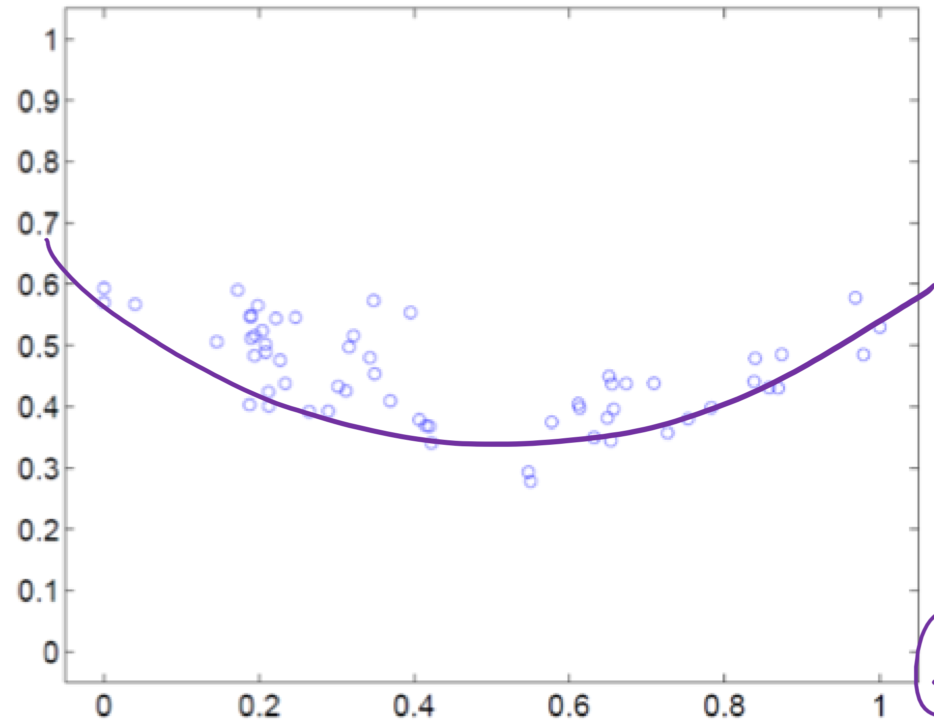
Not really the best for classification, but t’s a good start

Outline

- Supervised Learning
- Linear Regression
- Extension

Extension to Higher-Order Regression





$X = \begin{bmatrix} h \\ \\ \end{bmatrix}$ $\quad Z = \begin{bmatrix} h & h^2 & h^3 & \cdots & h^d \\ \\ \end{bmatrix}$

$X = \begin{bmatrix} h & \omega \\ \\ \end{bmatrix}$ $\quad Z = \begin{bmatrix} h & \omega & h^2 & \omega^2 & h\omega \\ \\ \end{bmatrix}$ $h^2\omega \quad \omega^2 h$

- Want to fit a polynomial regression model

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_d x^d + \epsilon$$

$$\theta_0 + \theta_1 z_1 + \theta_2 z_2 + \cdots + \theta_d z_d$$

- $z = \{1, x, x^2, \ldots, x^d\} \in R^d$ and $\theta = (\theta_0, \theta_1, \theta_2, \ldots, \theta_d)^T$

$$y = z\theta$$

Least Mean Square Still Works the Same

- Given n data points, find θ that minimizes the mean square error

$$\hat{\theta} = argmin_{\theta}\, L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - z_i\theta)^2$$

- Our usual trick: set gradient to 0 and find parameter

$$\frac{\partial L(\theta)}{\partial \theta} = -\frac{2}{n}\sum_{i=1}^{n} z_i^T(\, y_i - z_i\theta) = 0$$

$$\frac{\partial L(\theta)}{\partial \theta} = -\frac{2}{n}\sum_{i=1}^{n} z_i^T y_i + \frac{2}{n}\sum_{i=1}^{n} z_i^T z_i\theta = 0$$

Matrix Version of the Gradient

$$z = \{1, x, x^2, \dots, x^d\} \in R^d \qquad y = \{y_1, y_2, \dots, y_n\}$$

$$\frac{\partial L(\theta)}{\partial \theta} = -\frac{2}{n} z^{\mathrm{T}} \mathrm{y} + \frac{2}{n} z^T z \theta = 0$$

$$\Rightarrow \; \theta = (z^T z)^{-1} z^T y = z^{+} \, y$$

- If we choose a different maximal degree **d** for the polynomial, the solution will be different.

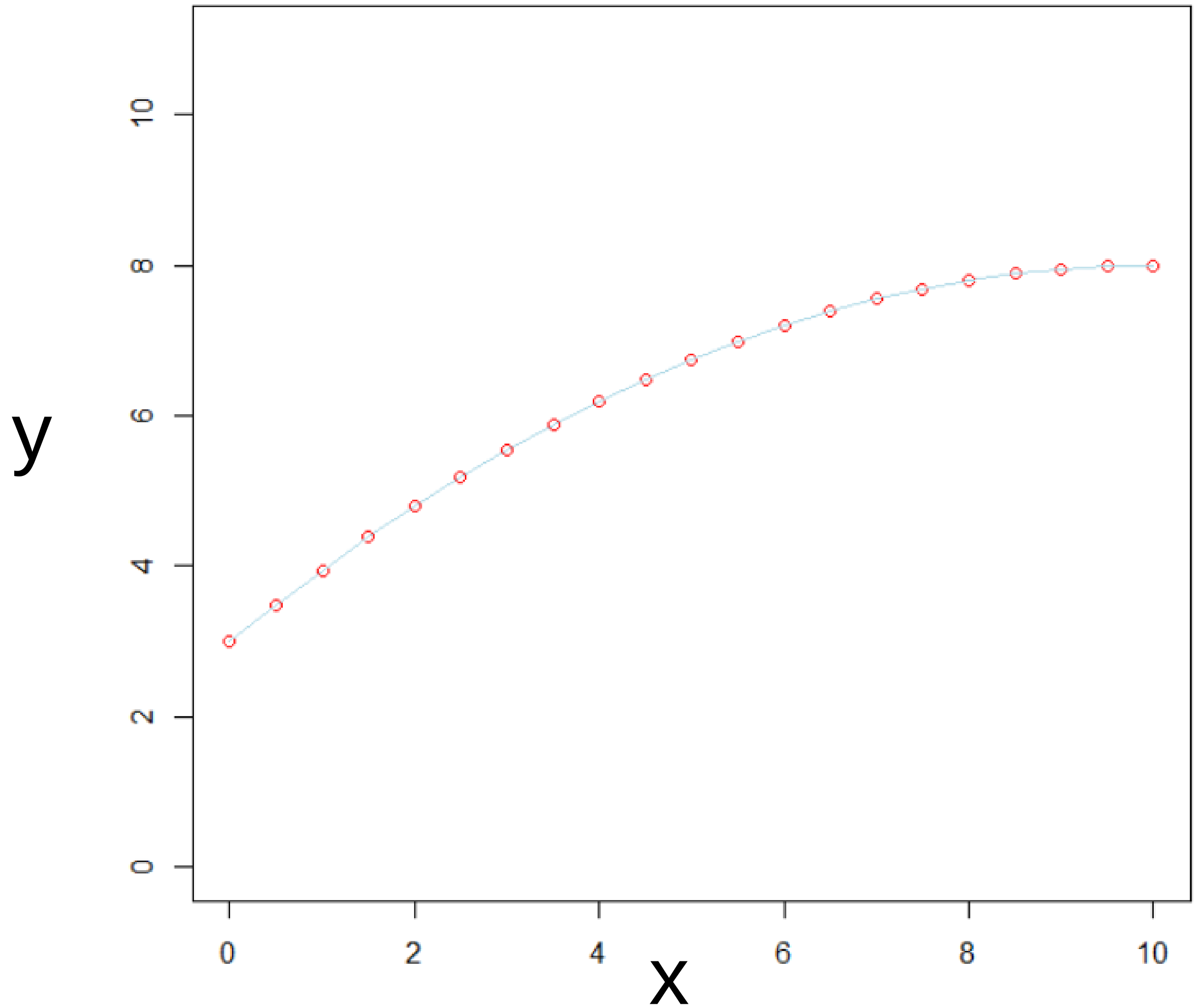
What is happening in polynomial regression?

$$x = [0,0.5,1,\ldots,9.5,10]$$

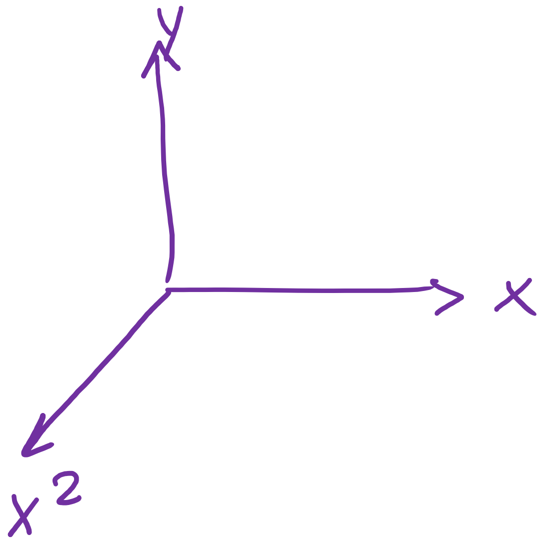
$$y = [3,3.4875,3.95,\ldots,7.98,8]$$

$$f = \theta_0 + \theta_1 x + \theta_2 x^2$$

$$\theta_0 = 3; \theta_1 = 1; \theta_2 = -0.5$$



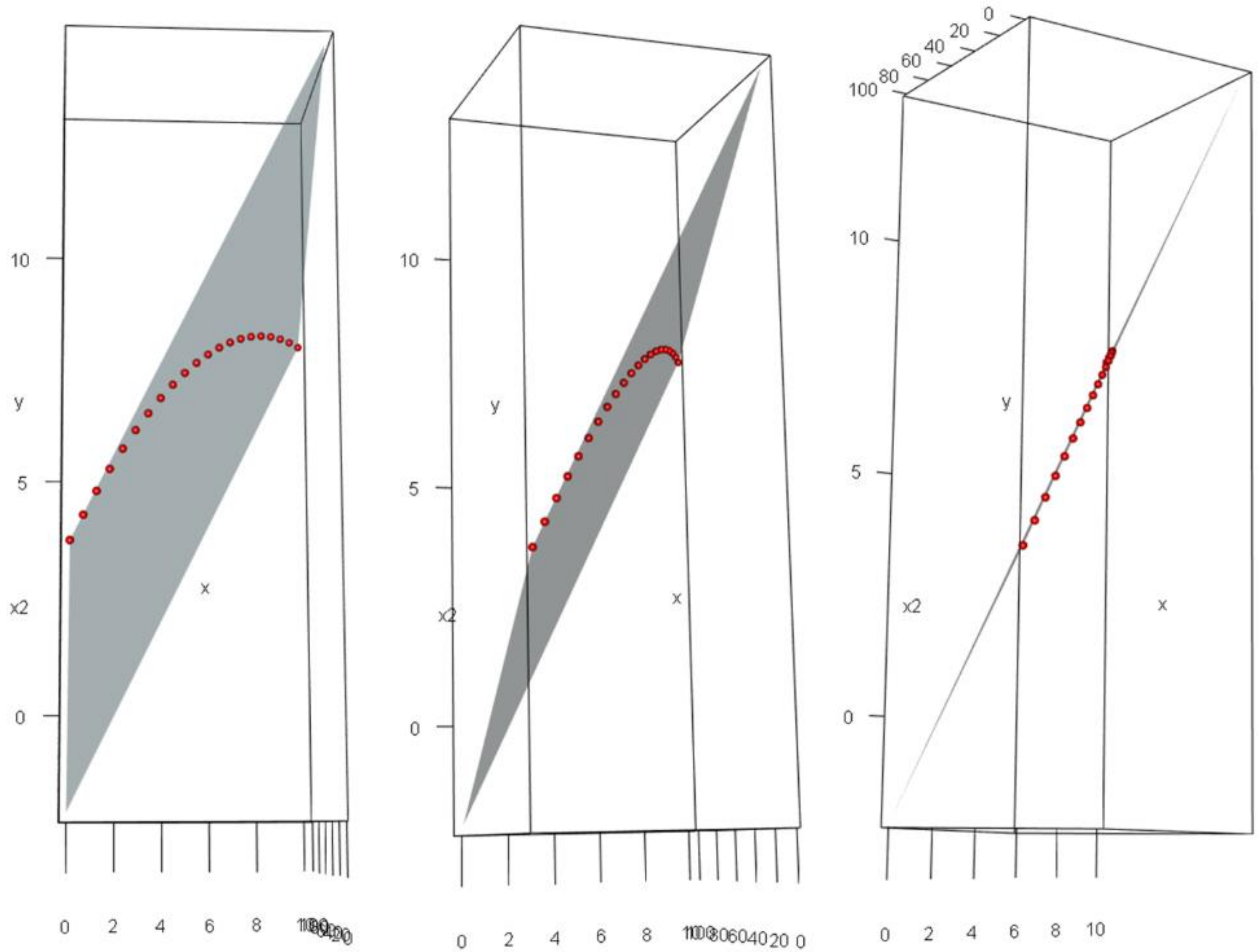
RMSE=0



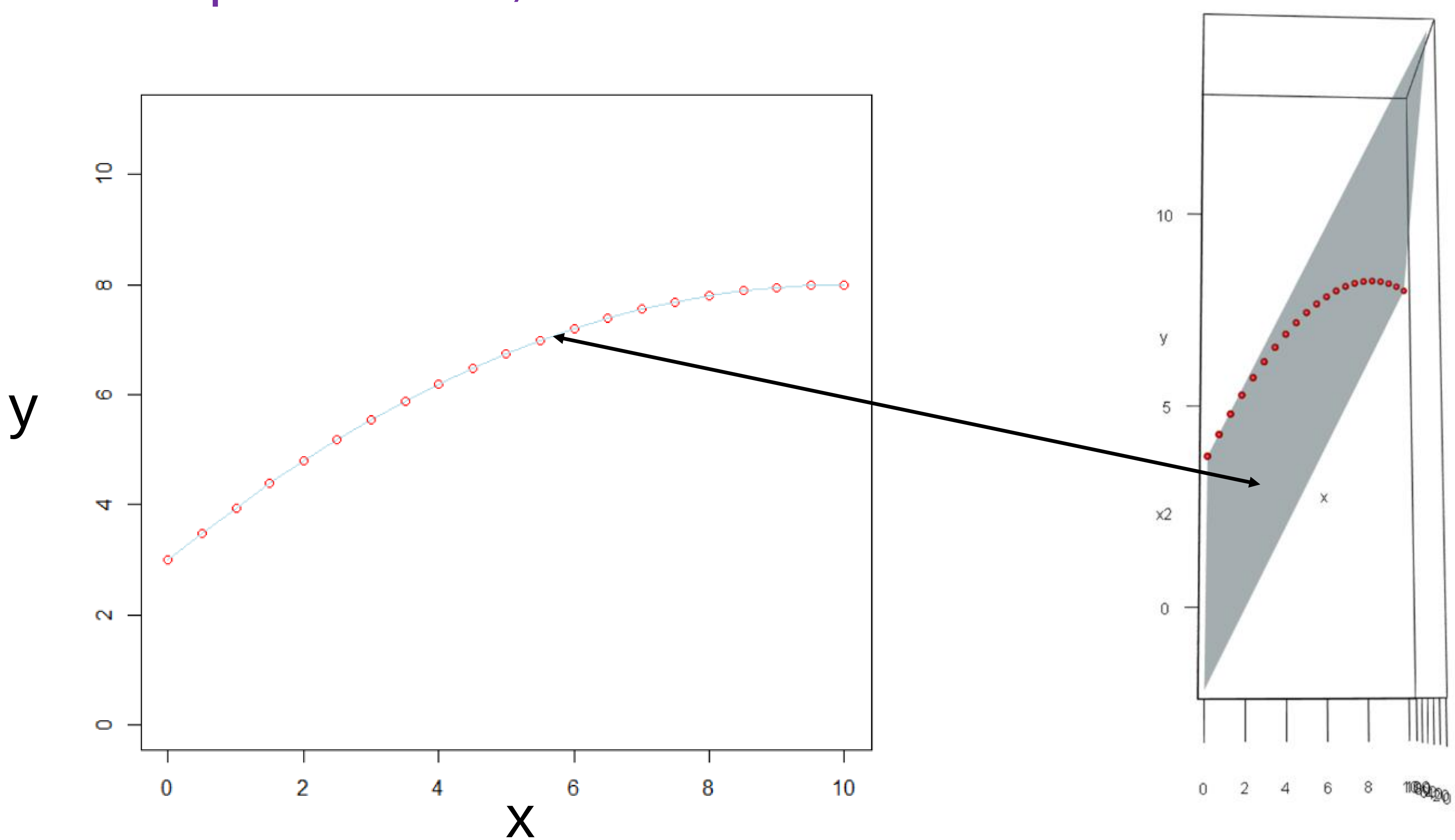
Let's add to the feature space

$x_1 = [0,0.5,1, \ldots, 9.5,10]$ $\quad$ $x_2^2 = [0,0.25,1, \ldots, 90.25,100]$

$y = [3,3.4875,3.95, \ldots, 7.98,8]$

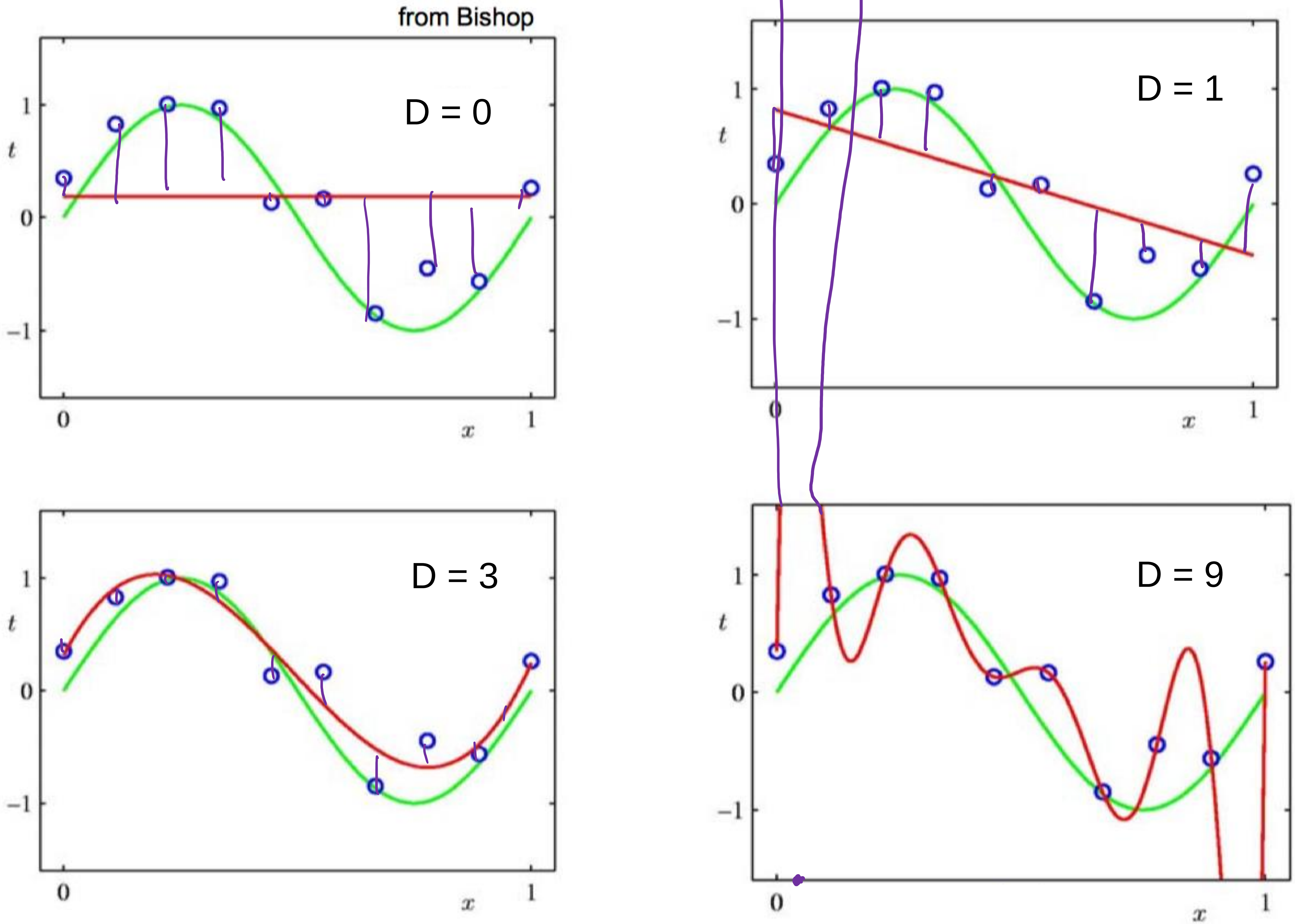


We are fitting a D-dimensional hyperplane in a D+1 dimensional hyperspace (in above example a 2D plane in a 3D space). That hyperplane really is 'flat' / 'linear‘ in 3D. It can be seen a non-linear regression (a curvy line) in our 2D example in fact it is a flat surface in 3D. So the fact that it is mentioned that the model is linear in parameters, it is shown here.



$y = \theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d$

Increasing the Maximal Degree



Bias-Variance Trade off

[Animation]

We will have multiple prediction values (i.e. through Cross validation) $E[y_p]$

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i\theta)^2 = E\left[(\mathrm{y_a} - \mathrm{y_p})^2\right]$$

$$(\mathrm{y_a} - \mathrm{y_p})^2 = \left(y_a - E[y_p] + E[y_p] - y_p\right)^2$$

$E[E[y_p]] = E[y_p]$

$E \qquad E[y_p]$

$$= E_y\left[\left(y_a - E[y_p]\right)^2 + \left(E[y_p] - y_p\right)^2 + 2\left(y_a - E[y_p]\right)\left(\cancel{E[y_p]} - y_p\right)\right]$$

0

$$\mathbf{E}\left[(\boldsymbol{y_a} - \boldsymbol{y_p})^{\mathbf{2}}\right] = \left(y_a - E[y_p]\right)^2 + E\left[\left(E[y_p] - y_p\right)^2\right]$$

$$= [Bias]^2 + Variance$$

$$= [\mathit{true\ value} - \mathit{mean(predictions)}]^2 - \mathit{mean}[(\mathit{mean}(\mathit{prediction}) - \mathit{prediction})^2]$$

Why $E[2(y_a - E[y_p])(E[y_p] - y_p)] = 0$?

$y_a - E[y_p]$ is a scalar, therefore $E\left[y_a - E[y_p]\right] = y_a - E[y_p]$

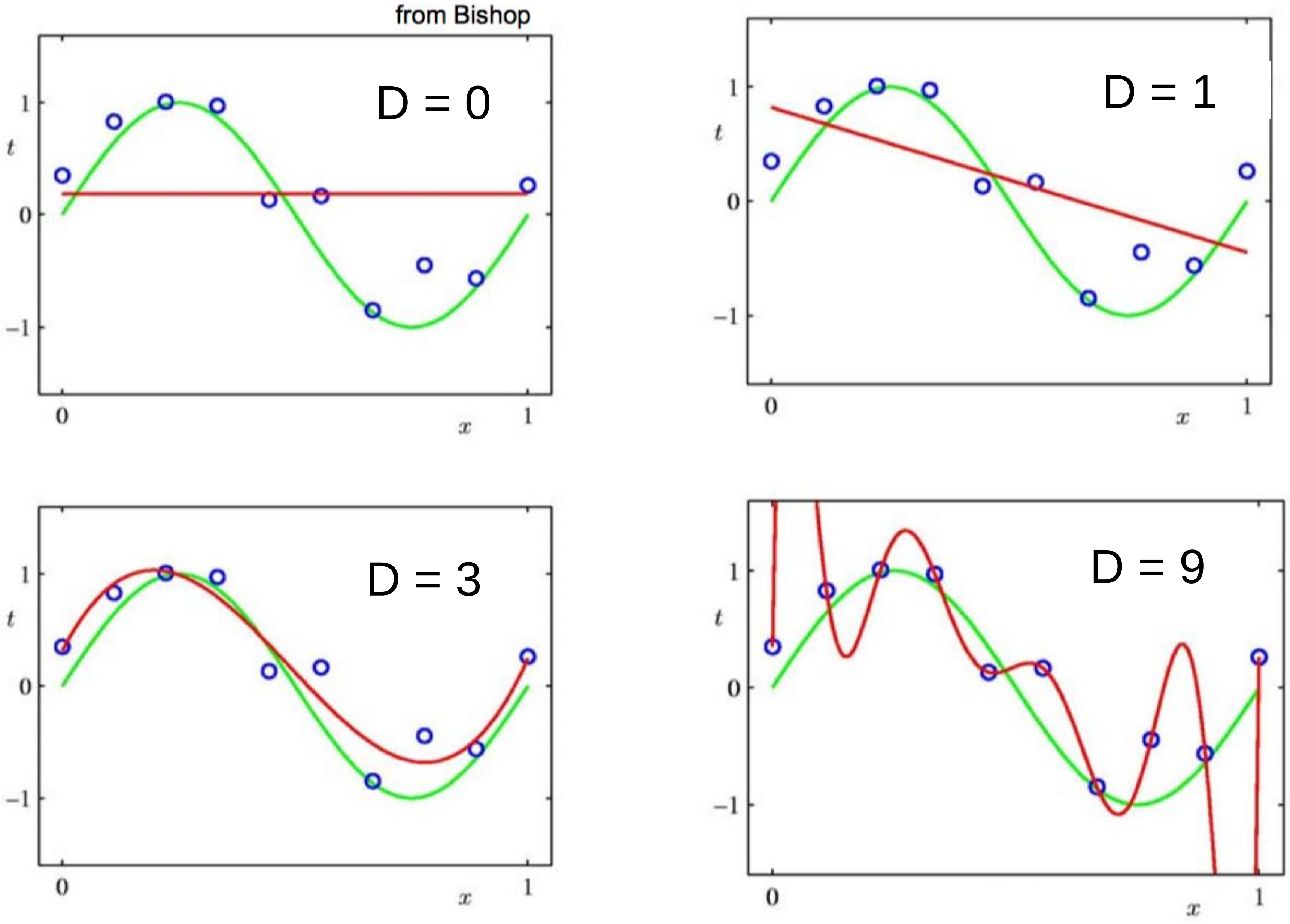
$$E[2(y_a - E[y_p])(E[y_p] - y_p)]$$

$$= 2(y_a - E[y_p])E[E[y_p] - y_p]$$

$$= 2(y_a - E[y_p])\left(E\left[E[y_p]\right] - E[y_p]\right)$$

$$= 2(y_a - E[y_p])(E[y_p] - E[y_p]) = 0$$

Which One is Better?



- Can we increase the maximal polynomial degree to very large, such that the curve passes through all training points?

  - We will know the answer in next lecture.

Take-Home Messages

- Supervised learning paradigm
- Linear regression and least mean square
- Extension to high-order polynomials