# 4. Supervised Learning & Linear Regression

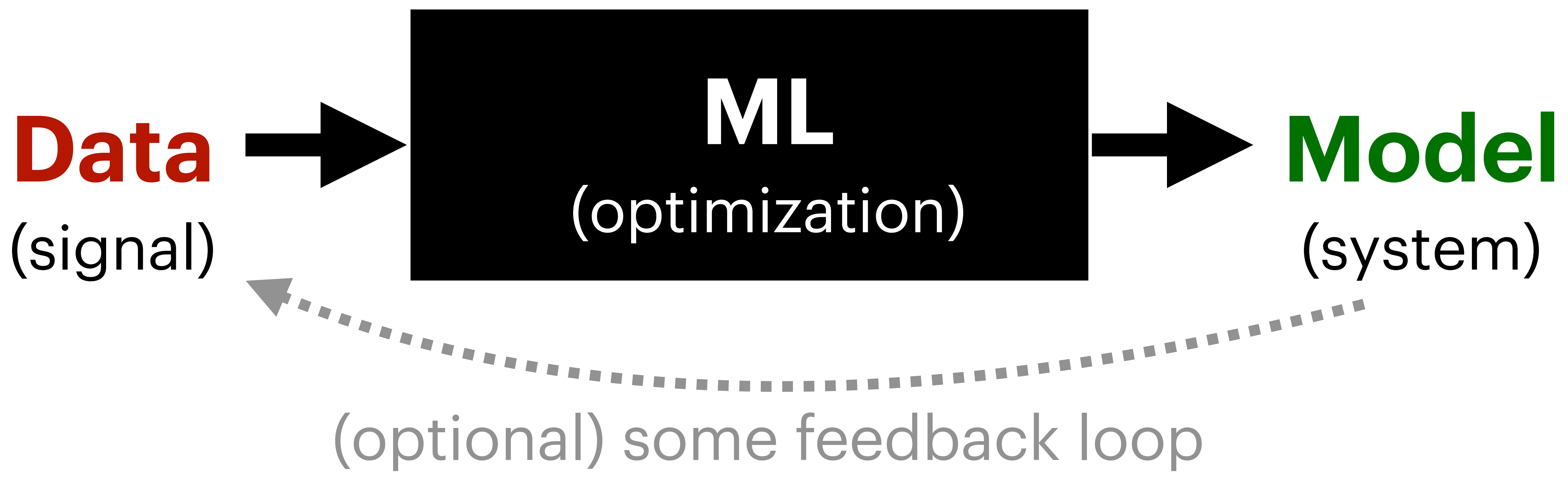
## EEC454 Introduction to Machine Learning Systems

2023 Fall, Jaeho Lee

# Notice

- Get ready for attendance checks & assignments!

# Big Picture




- **Linear Algebra.** Vectors and Matrices formalize both Data and Model
    - **Matrix Calculus.** Needed for optimization of models
- **Probability.** Formalizes *uncertainty* in data and optimization
- **Today.** Start formally studying ML!

# Supervised Learning: The basic framework

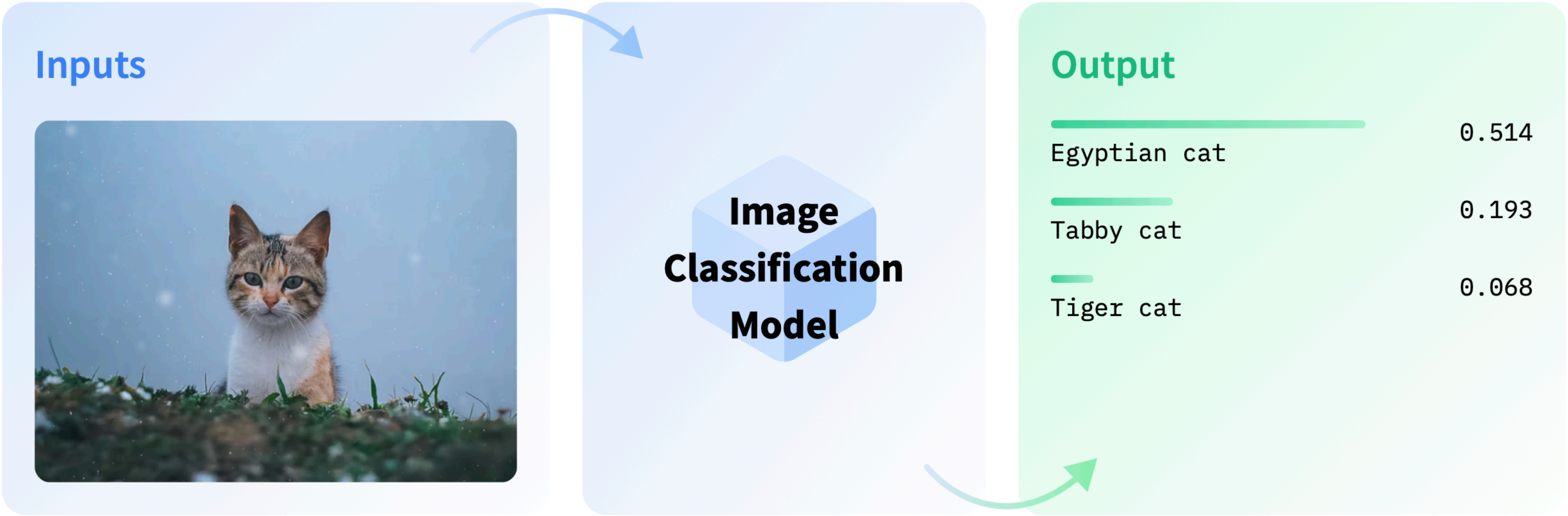
# Setup

- **Goal.** Build a nice ***predictor—***
  - Predict some ***output*** $Y$ given a (jointly distributed) ***input*** $X$.

* image source: HuggingFace

# Setup

- **Goal.** Build a nice ***predictor***—
  - Predict some ***output*** $Y$ given a (jointly distributed) ***input*** $X$.




* image source: HuggingFace

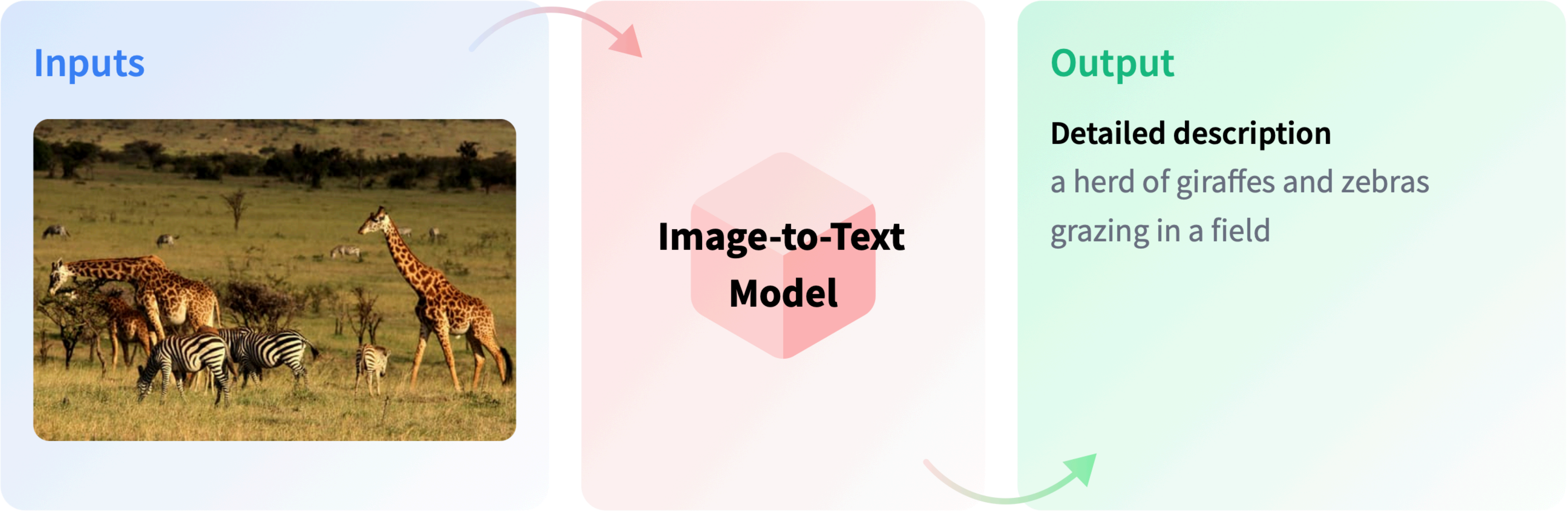
# Setup

- **Goal.** Build a nice *predictor*—
  - Predict some ***output*** $Y$ given a (jointly distributed) ***input*** $X$.


* image source: HuggingFace

# Setup

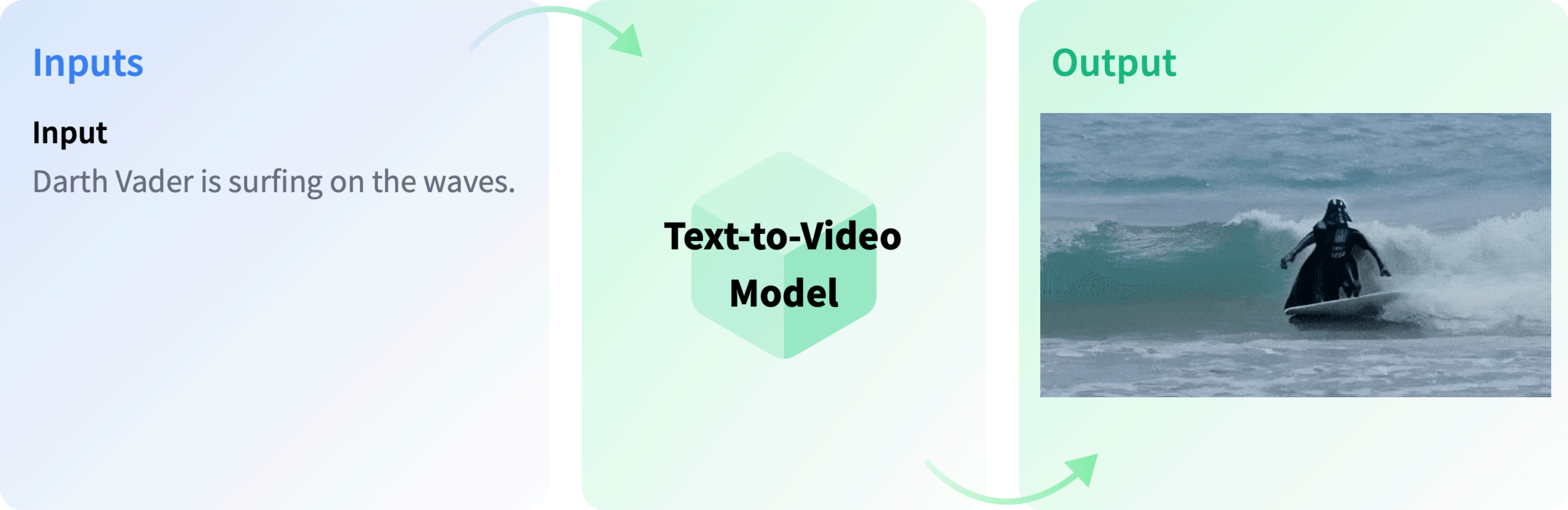
- **Goal.** Build a nice *predictor*—
  - Predict some ***output*** $Y$ given a (jointly distributed) ***input*** $X$.




* image source: HuggingFace

# Setup

- **Goal.** Build a nice ***predictor*—**
  - Predict some ***output*** $Y$ given a (jointly distributed) ***input*** $X$.




* image source: HuggingFace

# Setup

- Find a predictor $f(\cdot)$ such that $f(X) \approx Y$
  - Can rewrite as

$$\text{minimize}\ \mathbb{E}[\ell(f(X), Y)],$$

... over a good set of candidate $f(\cdot)$

for some nice "loss" function $\ell(\cdot, \cdot)$.

- **Problem.** Don't know the joint distribution $P_{XY}$

(if we knew, we can easily choose Bayes-optimal $f$)

# Setup

- **Dataset.** Instead, we can use the ***training dataset***.
  - The dataset consists of many ***input-output*** pairs.
    (i.e., feature-label)

$$D = \left\{ (x_1, y_1), \ldots, (x_n, y_n) \right\}$$

  - We call this scenario *supervised*—
    someone already inspected the data $x_i$ and annotated with $y_i$
    (i.e., supervision for machine)

# Example "Labeled" dataset: ImageNet


n02097047 (196)


n01682714 (40)


n03134739 (522)


n04254777 (806)


n02859443 (449)


n02096177 (192)

imagenet1000_clsidx_to_labels.txt

```
{0: 'tench, Tinca tinca',
 1: 'goldfish, Carassius auratus',
 2: 'great white shark, white shark, man-eater, ma
 3: 'tiger shark, Galeocerdo cuvieri',
 4: 'hammerhead, hammerhead shark',
 5: 'electric ray, crampfish, numbfish, torpedo',
 6: 'stingray',
 7: 'cock',
 8: 'hen',
 9: 'ostrich, Struthio camelus',
 10: 'brambling, Fringilla montifringilla',
 11: 'goldfinch, Carduelis carduelis',
 12: 'house finch, linnet, Carpodacus mexicanus',
 13: 'junco, snowbird',
 14: 'indigo bunting, indigo finch, indigo bird, P
 15: 'robin, American robin, Turdus migratorius',
 16: 'bulbul',
 17: 'jay',
 18: 'magpie',
 19: 'chickadee',
```

# Learning Algorithm

- Summing up, supervised learning is simply doing

$$D = \left\{ (x_1, y_1), \dots, (x_n, y_n) \right\} \longrightarrow \boxed{\textbf{Algo}} \longrightarrow \hat{f}(\,\cdot\,)$$

  with some algorithm.

- **Q. What algorithm?**

# Learning Algorithm

- Typically consist of two elements:
  - A bag of functions (hypothesis space)

$$\mathcal{F} = \left\{ f_1, f_2, \dots \right\}$$

  - An optimizer—the training method
    - (approximately) solves **Empirical Risk Minimization** **(ERM)**

$$\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \ell\Big(y_i, f(x_i)\Big) \quad + \text{ regularizer}$$

# Learning Algorithm

- **Intuition.** Empirical Risk ≈ True Risk (Population Risk)

$$\frac{1}{n}\sum_{i=1}^{n} g(X_i) \longrightarrow \mathbb{E}[g(X)]$$

$$\frac{1}{n}\sum_{i=1}^{n} \ell\Big(y_i, f(x_i)\Big) \longrightarrow \mathbb{E}[\ell(Y, f(X))]$$

(**Note 1.** How fast? consult concentration inequalities)

(**Note 2.** Not 100% required—not all $X_i$ are born equal!)

# Testing

- We hope that $\mathbb{E}[\ell(Y, \hat{f}(X))]$ is small, but how do we know?

- Usually have a ***test dataset*** $D^{\text{test}} = \{(\tilde{x}_1, \tilde{y}_1), \ldots, (\tilde{x}_k, \tilde{y}_k)\}$.
  - We validate the smallness of

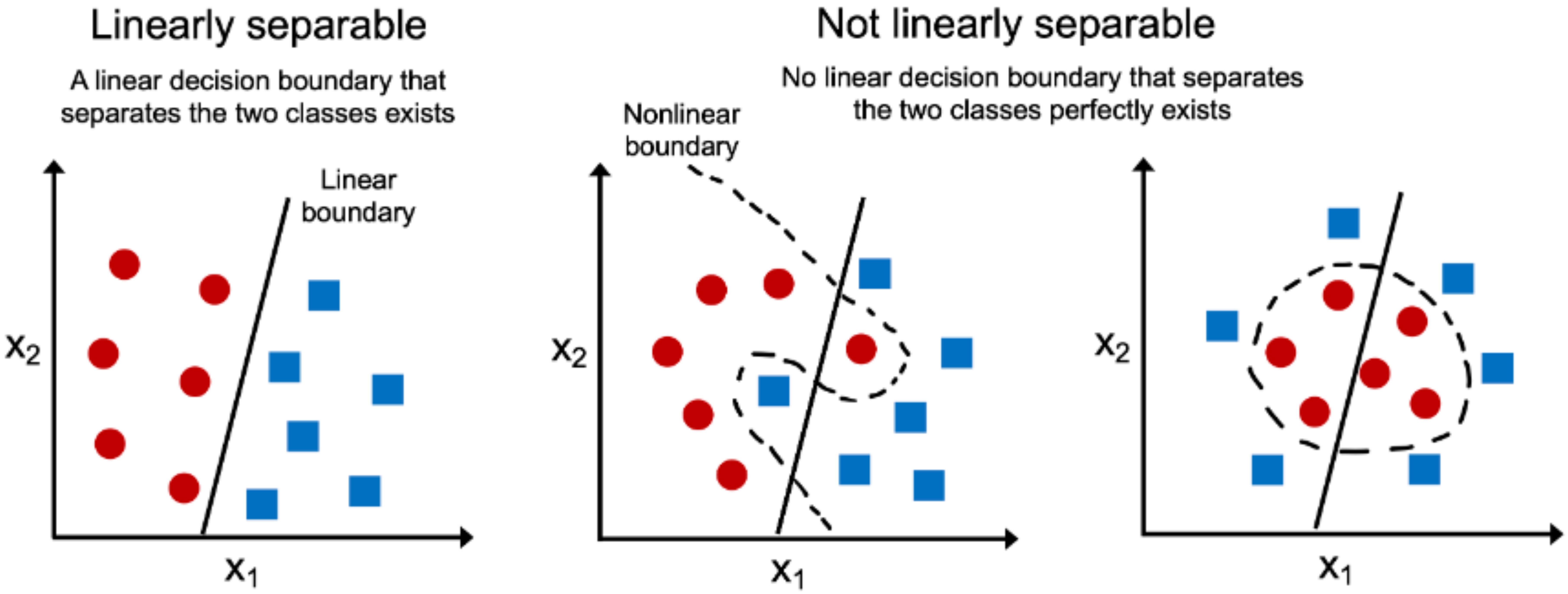
$$\frac{1}{k}\sum_{i=1}^{k} \ell(\hat{f}(\tilde{x}_i), \tilde{y}_i)$$

  - Typically splits train/val*/test into 8:1:1 (or 7:1:2 in the past).
    (cross-validation if the dataset is small)

# Learning algorithm vs Learning algorithm

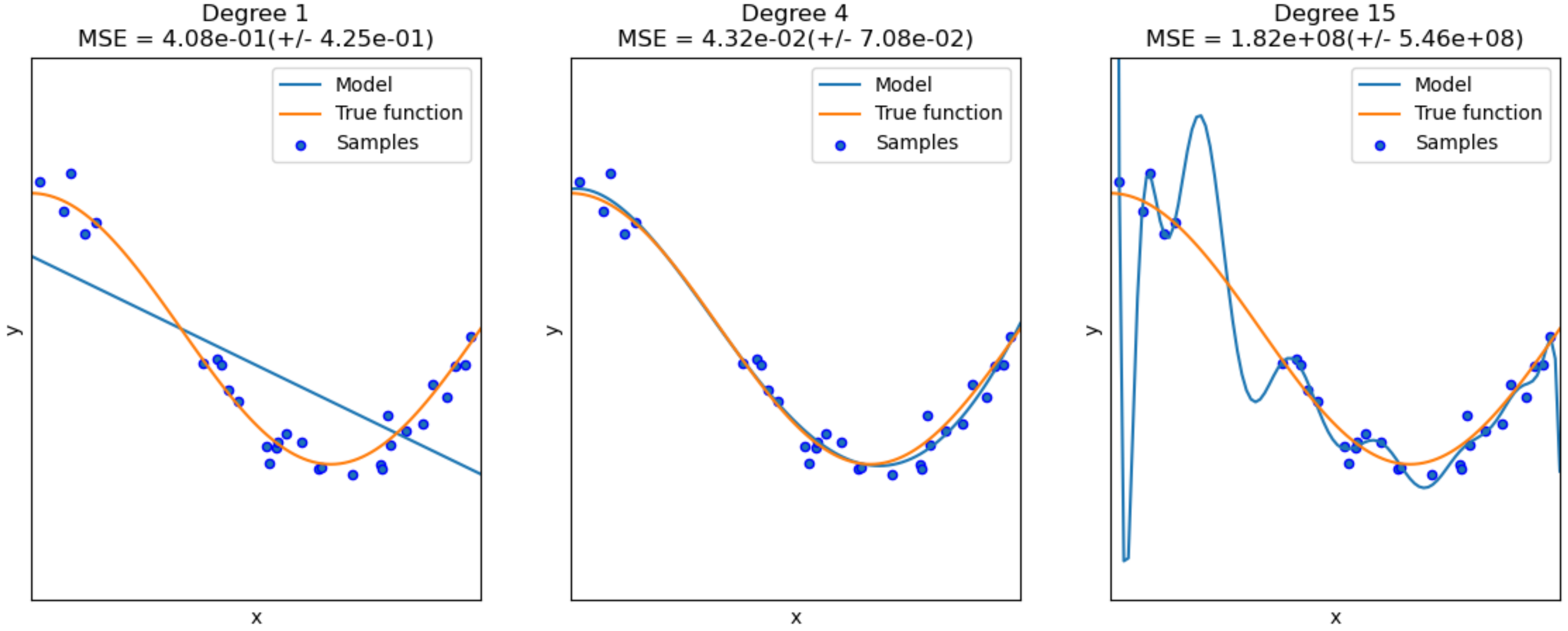
# Which algorithm should we use?

- Some considerations:
  - **Model Size** (= Richness of Hypothesis Space)
    - If too small, even the best $\hat{f}(\cdot)$ cannot fit the reality.

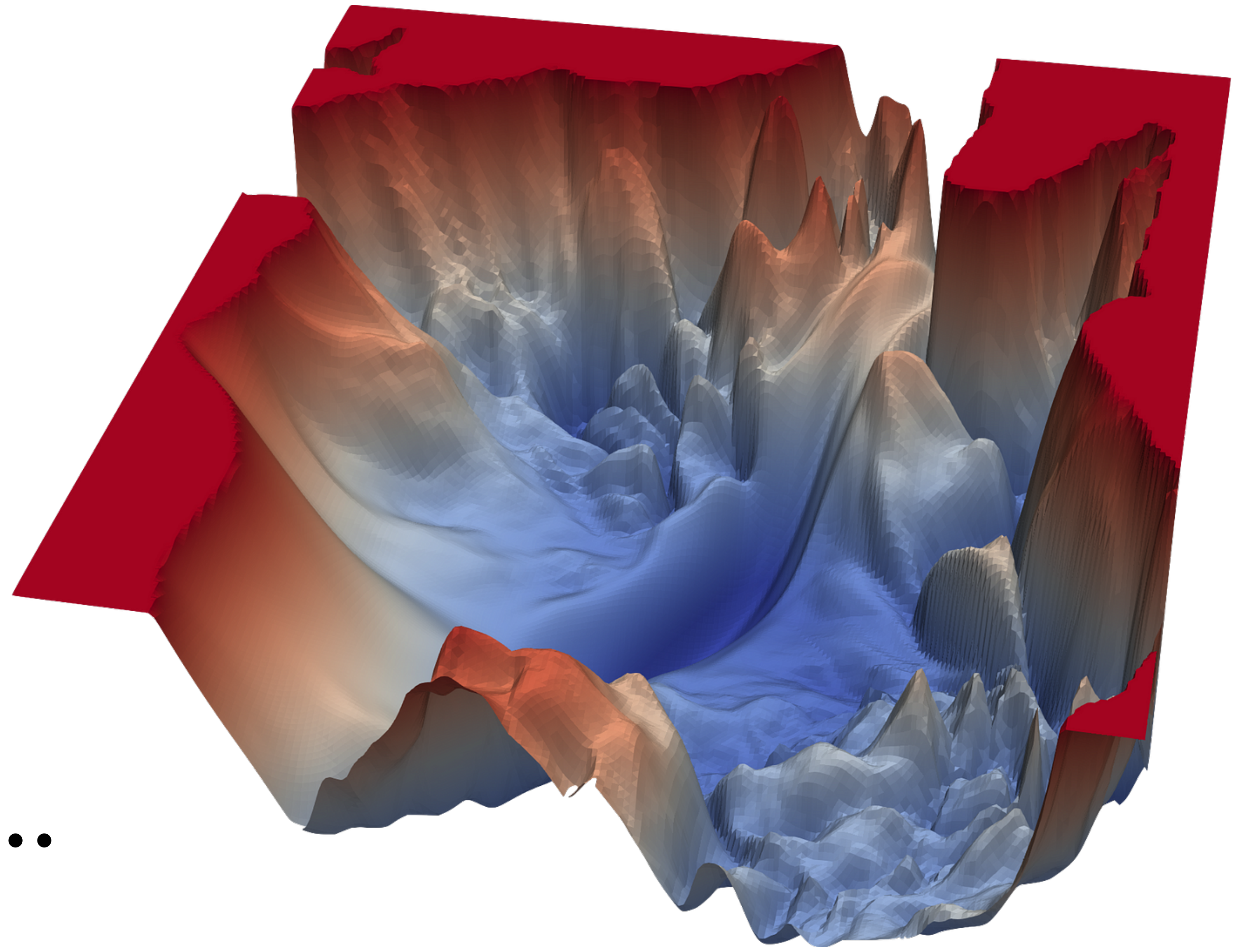


# Which algorithm should we use?

- Some considerations:
  - **Model Size** (= Richness of Hypothesis Space)
    - If too large, can overfit the training data + large inference cost



# Which algorithm should we use?



- Some considerations:
  - **Optimization** (= difficulty of solving ERM)
    - Often highly customized for each "model."
    - For highly complicated, non-linear models…
      - Explicit solution not available.
      - Takes a long time to compute the optimum (high training cost)

# Which algorithm should we use?

- Some considerations:
  - **Loss function / Regularizer**
    - Affects how difficult the optimization is.
    - Affects overfitting.
    - Affects desirable properties (robustness, sparsity)…

# Throughout the course...

- We study popular ML models one-by-one.
  - Which “hypothesis space” it uses.
  - Which “optimizer” it uses.
  - Which “loss/regularizer” it uses.
- **This and Next Class.** Linear models, Naïve Bayes, Nearest Neighbors

*Note.* Many of these choices are heavily dependent on ***task*.**
(regression vs. classification, image vs. text vs. tabular, …)

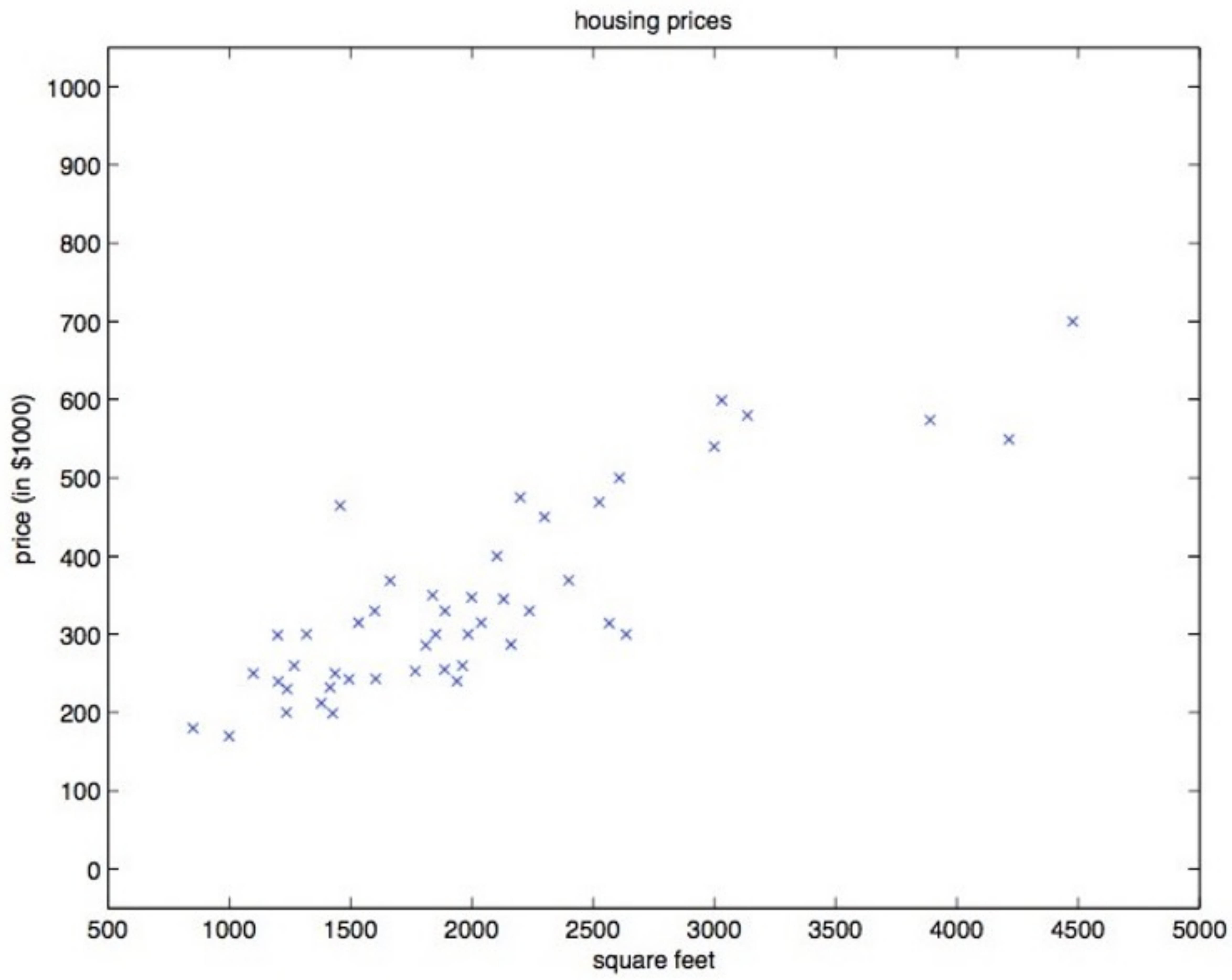
# Linear Regression

# Regression

- **Regression** $\approx$ Predict continuous $y \in \mathbb{R}^m$.
- **Example.** House price prediction.

$$f(\text{area}) = \text{price}$$

| Living area (feet$^2$) | Price (1000\$s) |
|---|---|
| 2104 | 400 |
| 1600 | 330 |
| 2400 | 369 |
| 1416 | 232 |
| 3000 | 540 |
| $\vdots$ | $\vdots$ |



# Linear Regression

- We use linear model $f(\cdot)$.
- If $x \in \mathbb{R}$ and $y \in \mathbb{R}$,

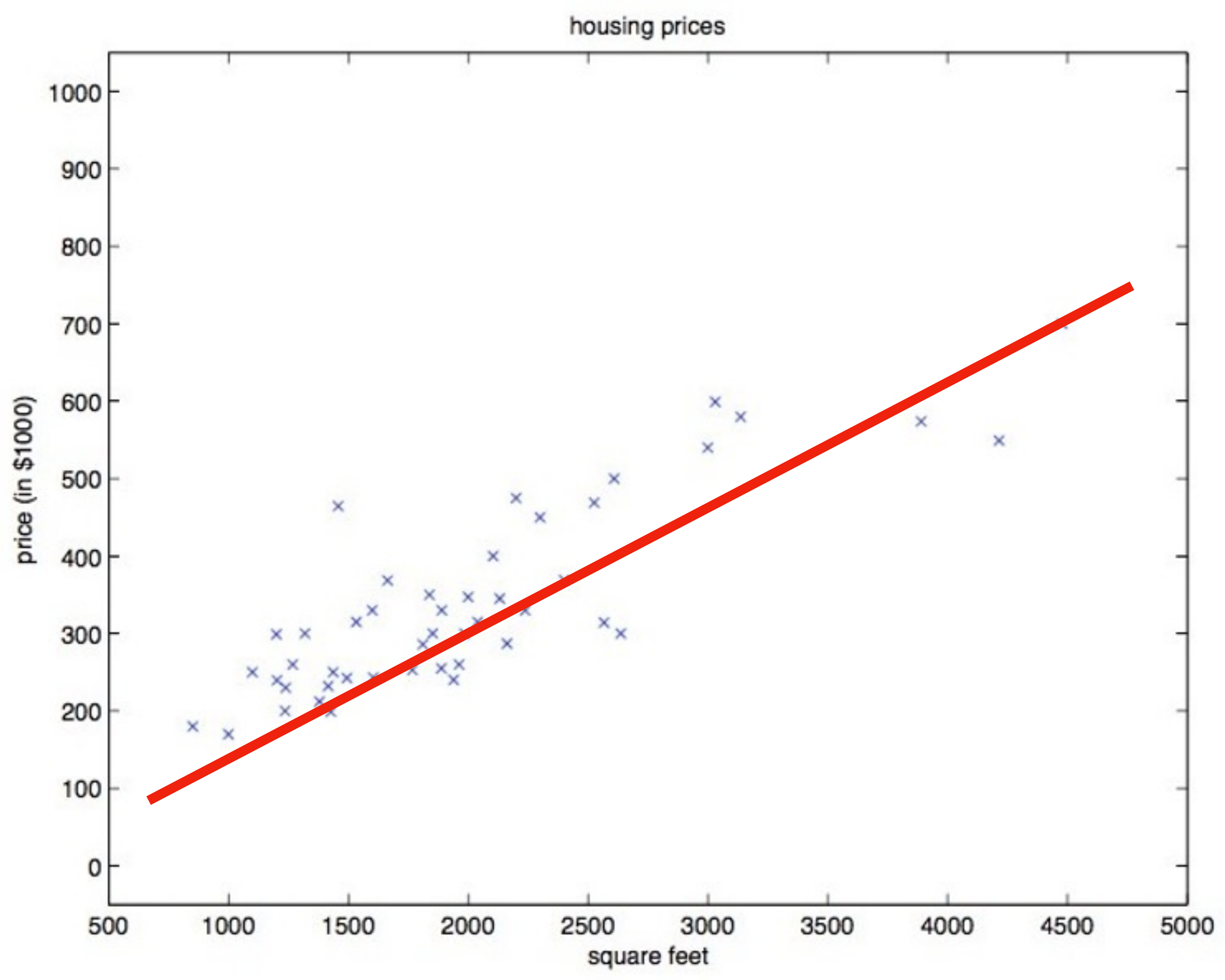
$$f(\mathbf{x}) = w \cdot x + b, \qquad w \in \mathbb{R}, c \in \mathbb{R}$$

- If $\mathbf{x} \in \mathbb{R}^d$ and $y \in \mathbb{R}$,

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b, \qquad \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}$$

- If $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{y} \in \mathbb{R}^m$,

$$f(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}, \qquad \mathbf{W} \in \mathbb{R}^{m \times d}, \mathbf{b} \in \mathbb{R}^m$$



# Linear Regression

- Reflects a belief that the **data-generating distribution** may look like:

$$X \sim P(X)$$

$$Y \sim w_*^\top X + \epsilon$$

where $\epsilon$ is some (zero-mean) noise.

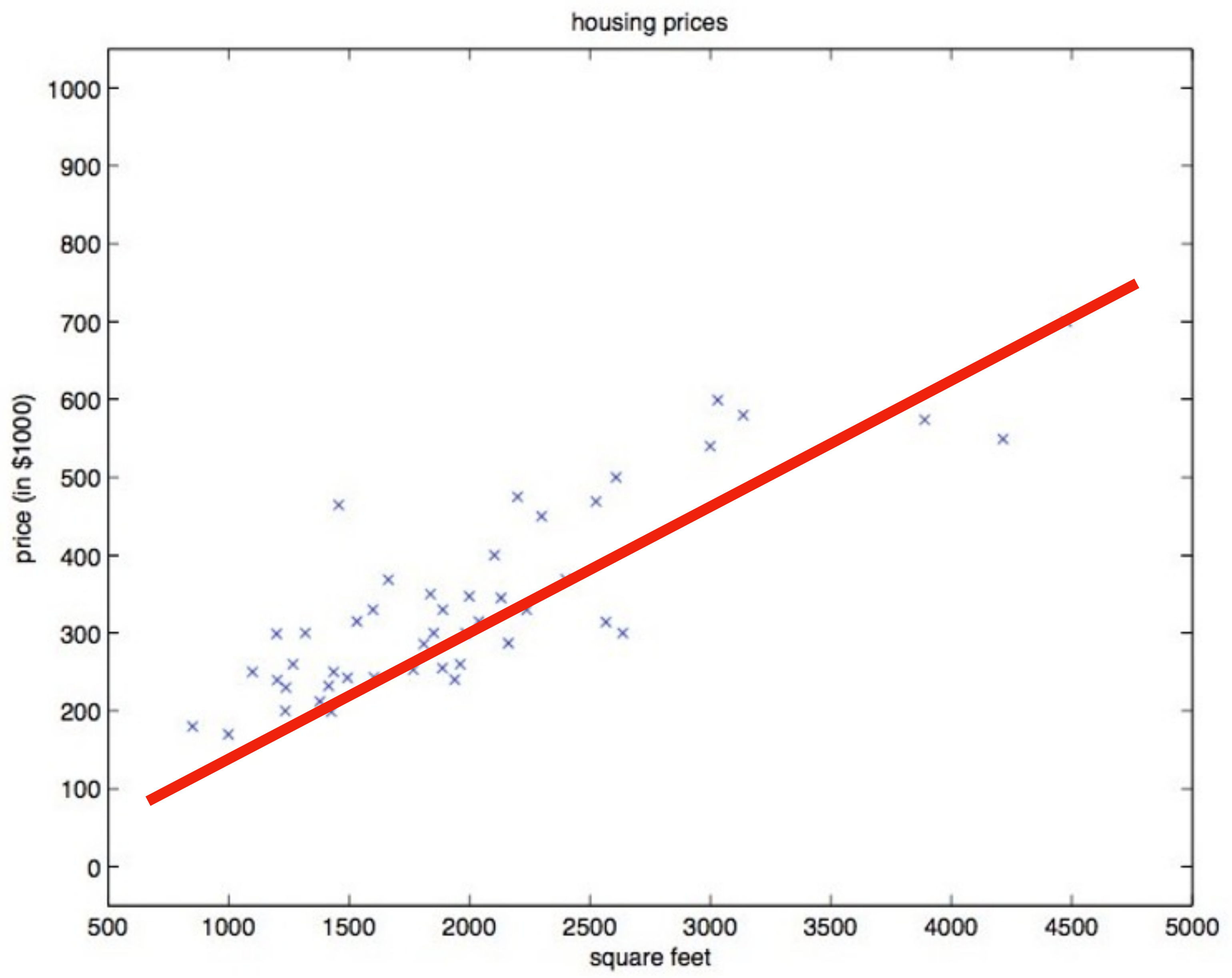
- **Fun fact.** If $X, Y$ are jointly Gaussian, MMSE estimator is always linear!

# Linear Regression: Ordinary Least Squares

- We use squared $\ell_2$ loss $\quad \ell(\mathbf{y}, \hat{\mathbf{y}}) = \|\mathbf{y} - \hat{\mathbf{y}}\|_2^2.$
- For a dataset $D = \{(x_i, y_i)\}_{i=1}^n$, we solve

$$\min_{w,b} \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - (w \cdot x_i + b)\right)^2$$

- **Why least squared?**
  - easy to solve (quadratic)
  - nice interpretation (maximum likelihood solution under linear model + Gaussian noise)



# Solving the Linear Regression

# 1D, bias-free case

$$\min_{w\in\mathbb{R}} \underbrace{\frac{1}{2n}\sum_{i=1}^{n}\Big(y_i - (w\cdot x_i)\Big)^2}_{=:J(w)}$$

- Since this is a quadratic function,
  the minimum is where derivatives are zero (critical point)

$$\frac{\partial J}{\partial w}(w) = 0$$

# 1D, bias-free case

$$\frac{\partial J}{\partial w} = \frac{1}{n}\sum_{i=1}^{n}(w \cdot x_i - y_i)x_i = 0$$

$$\Rightarrow w\left(\sum x_i^2\right) = \sum y_i x_i$$

$$\Rightarrow w = \frac{\sum y_i x_i}{\sum x_i^2}$$

- Explicit solution can be characterized by math (not always possible)
    - No real gradient computation needed (we did math with our brain)
    - Need several multiplications and summations for optimization.

# Solving the minimization: Multivariate

- Consider a slightly more general case of $\mathbf{x} \in \mathbb{R}^d$.

$$\min_{\mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}^1} \frac{1}{2n} \sum_{i=1}^{n} \left( y_i - \mathbf{w}^\top \mathbf{x}_i + b \right)^2$$

- This looks messy, so we want to simplify a bit…

# Solving the minimization: Multivariate

$$\min_{\mathbf{w}\in\mathbb{R}^d, b\in\mathbb{R}^1} \frac{1}{2n}\sum_{i=1}^{n}\left(y_i - \mathbf{w}^\top \mathbf{x}_i + b\right)^2$$

- **Trick #1.**
- Define $\tilde{\mathbf{x}} = \begin{bmatrix}\mathbf{x}\\ 1\end{bmatrix}, \theta = \begin{bmatrix}\mathbf{w}\\ b\end{bmatrix}$.

$$J(\theta) = \frac{1}{2n}\sum_{i=1}^{n}(y - \theta^\top \tilde{\mathbf{x}})^2.$$

# Solving the minimization: Multivariate

$$\min_{\theta \in \mathbb{R}^{d+1}} \frac{1}{2n} \sum_{i=1}^{n} (y - \theta^\top \tilde{\mathbf{x}})^2$$

- **Trick #2.**
- Define $\mathbf{X} = \begin{bmatrix} \tilde{\mathbf{x}}_1^\top \\ \cdots \\ \tilde{\mathbf{x}}_n^\top \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y_1 \\ \cdots \\ y_n \end{bmatrix}.$

$$J(\theta) = \frac{1}{2n} \|\mathbf{y} - \mathbf{X}\theta\|^2.$$

# Solving the minimization: Multivariate

$$J(\theta) = \frac{1}{2n}\|\mathbf{y} - \mathbf{X}\theta\|^2$$

- We examine the critical point—where gradient is zero.

$$\begin{aligned}\nabla J(\theta) &= \frac{1}{2n}\nabla\left((\mathbf{y} - \mathbf{X}\theta)^\top(\mathbf{y} - \mathbf{X}\theta)\right)\\ &= \frac{1}{2n}\nabla\left(\mathbf{y}^\top\mathbf{y} + \theta^\top\mathbf{X}^\top\mathbf{X}\theta - 2\mathbf{y}^\top\mathbf{X}\theta\right)\\ &= \frac{1}{2n}\left(2\theta^\top\mathbf{X}^\top\mathbf{X} - 2\mathbf{y}^\top\mathbf{X}\right) \;= 0\end{aligned}$$

# Solving the minimization: Multivariate

- Thus, critical point is the $\theta$ that satisfies:

$$\mathbf{X}^\top \mathbf{X}\theta = \mathbf{X}^\top \mathbf{y}$$

  - If the matrix $\mathbf{X}^\top \mathbf{X}$ is invertible, we have a unique solution:

$$\hat{\theta} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

    - *Fun exercise*. Count the number of FLOPs?

# Solving the minimization: Multivariate

- Thus, critical point is the $\theta$ that satisfies:

$$\mathbf{X}^\top \mathbf{X}\theta = \mathbf{X}^\top \mathbf{y}$$

  - If not, there are infinite critical points (sadly 😢)
    - Solution. The above takes the form $\mathbf{A}\theta = \mathbf{b}$

      $\Rightarrow$ simply use QR decomposition

      - Gives you Moore-Penrose pseudoinverse $(\mathbf{X}^\top \mathbf{X})^\dagger$,

        which is a minimum norm solution among all possible $\theta$.

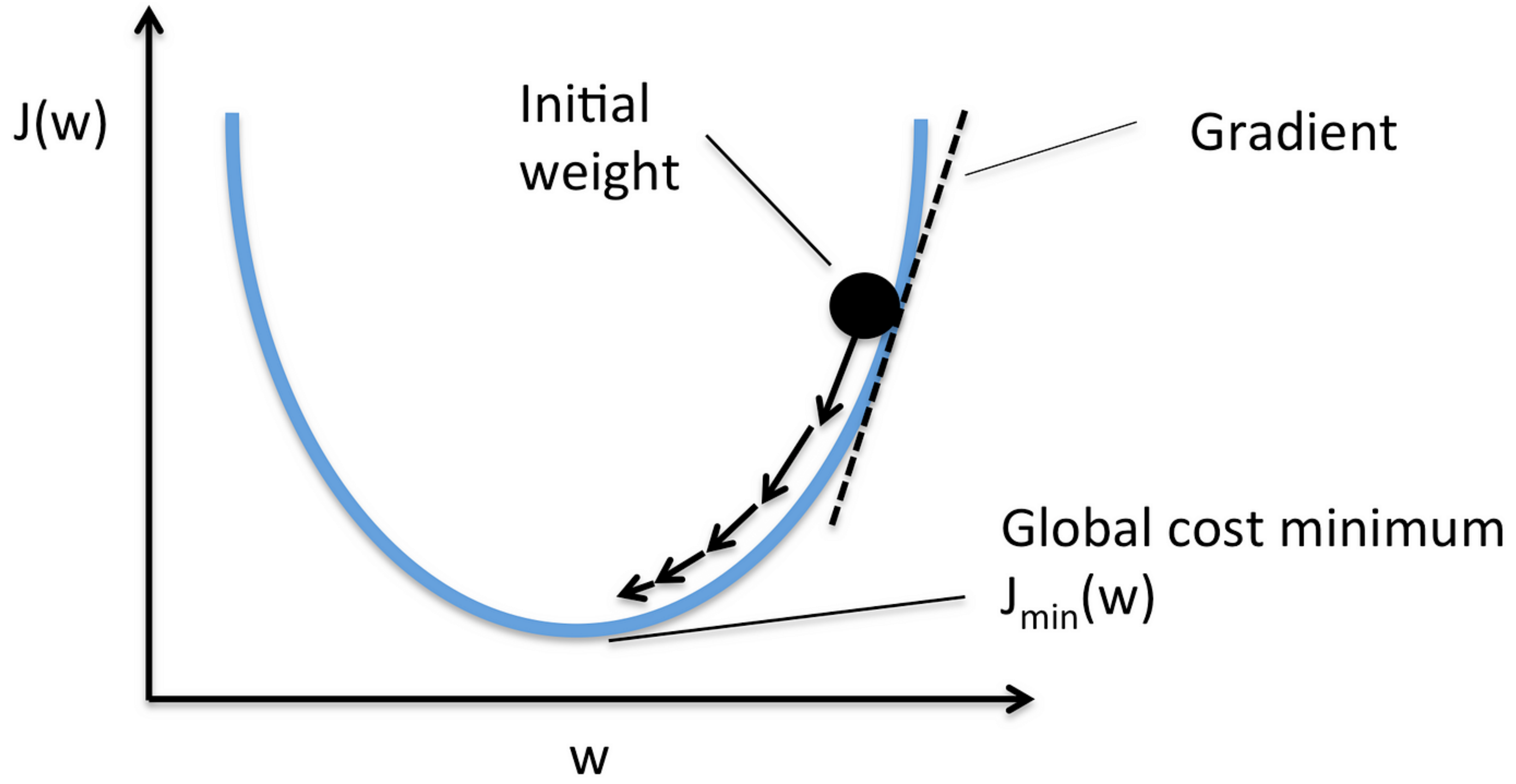
# Solving differently—Gradient Descent

# Gradient Descent

- Repeat taking steps in the downward direction.

# Gradient Descent

- Pick a random $\theta^{(0)}$, and use gradient to update $\theta^{(1)}, \theta^{(2)}, \ldots$



# Gradient Descent

- Pick a random $\theta^{(0)}$, and use gradient to update $\theta^{(1)}, \theta^{(2)}, \ldots$

- **Idea.** Gradient = direction of fastest increase.

  $\Rightarrow$ Negative Gradient = direction of fastest decrease.

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \nabla_\theta J(\theta^{(t)})$$

  - Plug in the previous gradient formula:

$$\theta \leftarrow \theta - \frac{\eta}{n}\left(\mathbf{X}^\top \mathbf{X}\theta - \mathbf{X}^\top \mathbf{y}\right)$$

# Computational Remarks

$$\theta \leftarrow \theta - \frac{\eta}{n}\left(\mathbf{X}^\top \mathbf{X}\theta - \mathbf{X}^\top \mathbf{y}\right)$$

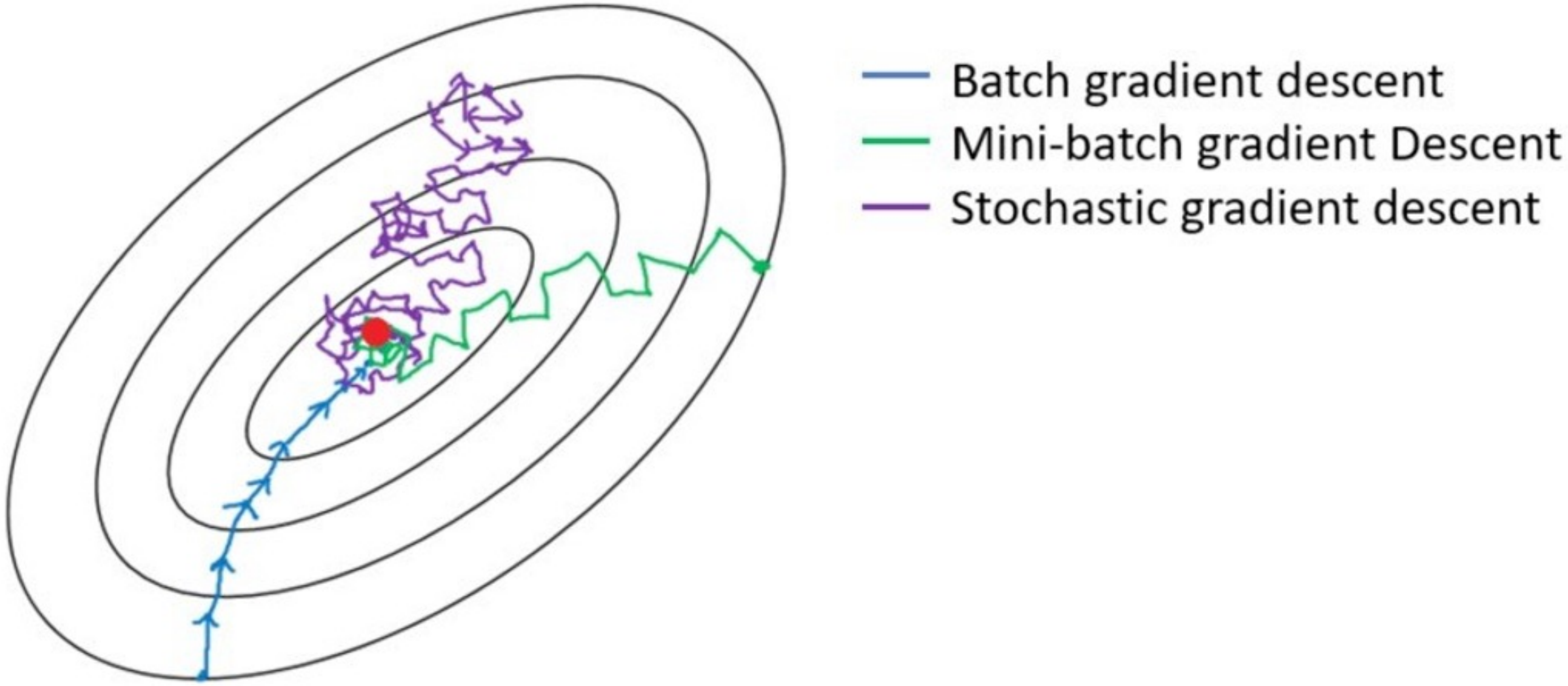
- How computation-heavy?
  - You can pre-compute and re-use $\mathbf{A} := \frac{\eta}{n}\mathbf{X}^\top \mathbf{X}$ and $\mathbf{b} := \frac{\eta}{n}\mathbf{X}^\top \mathbf{y}$ for every GD iteration.

$$\theta \leftarrow (\mathbf{I} - \mathbf{A})\theta - \mathbf{b}$$

    - The pre-computing cost is almost same as solving explicitly (except QR decomposition part).

# Additional Remarks

- You don't need full data for GD—
  using a randomly drawn subset of $k$ samples works ($k \ll n$).
  Called "mini-batch GD." (or "stochastic GD" when $k = 1$).
  - Useful for small RAM!



# Cheers

- <u>*Next up.*</u> Naïve Bayes, Logistic Regression, Nearest Neighbors