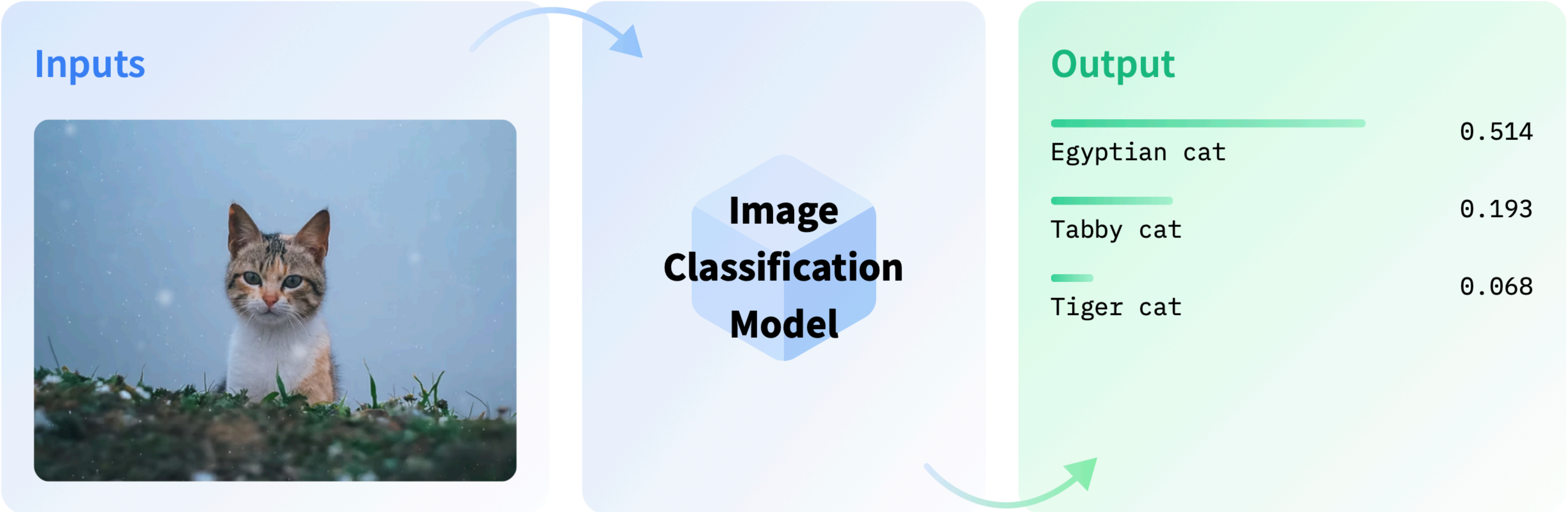
Simple Classifiers

Today

- Various classification algorithms
  - Nearest neighbors
  - Naïve Bayes
  - Linear classifiers
    - Perceptron
    - Logistic regression

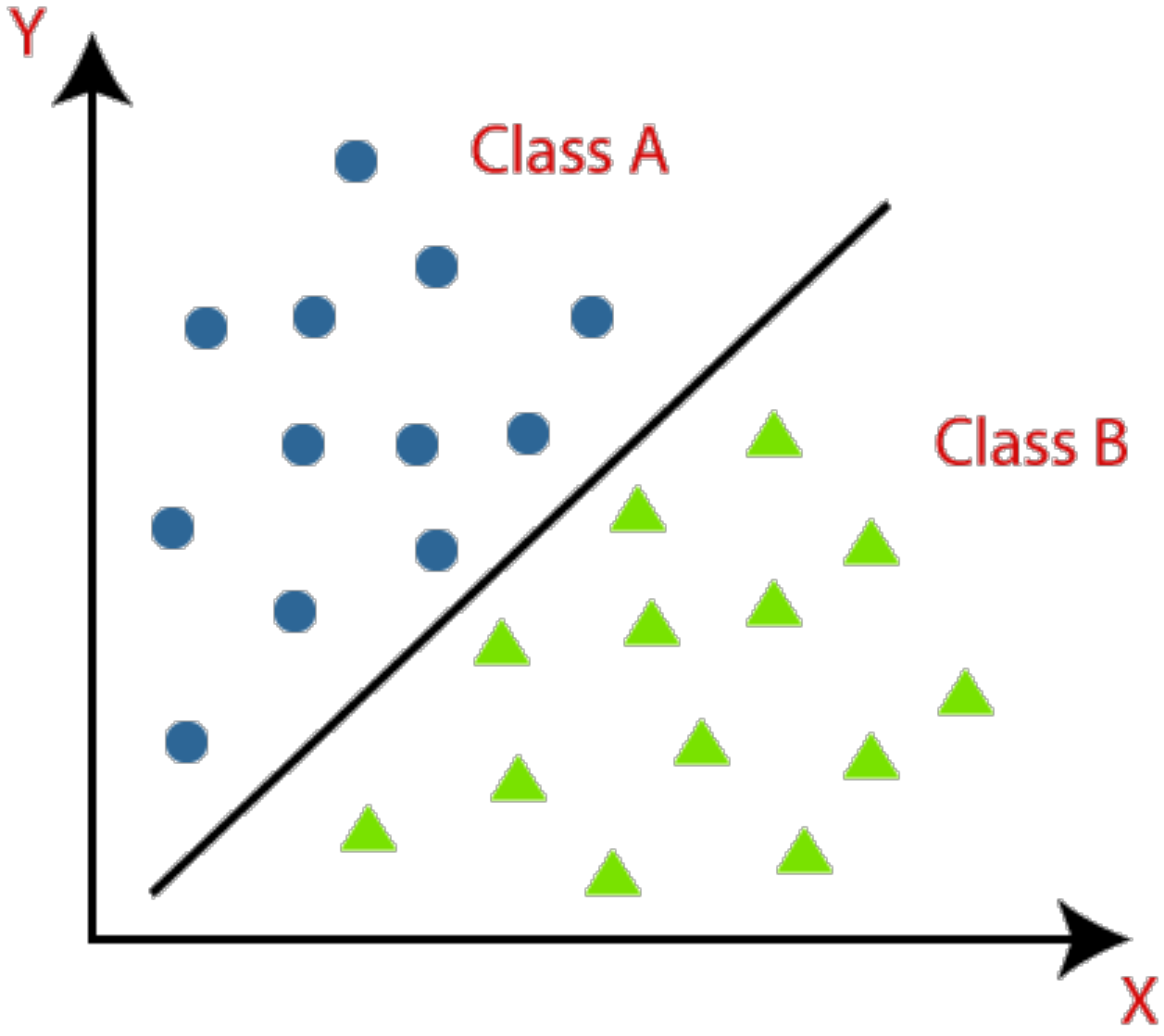
Goal

- Modeling the relationship between
  - continuous input $X \in \mathbb{R}^d$ (or discrete)
  - discrete output $Y \in \{1, \ldots, K\}$
    - called "class"



Binary Classification

- For simplicity, we mostly consider the case of **binary classification**
  - $Y \in \{0,1\}$



Binary Classification

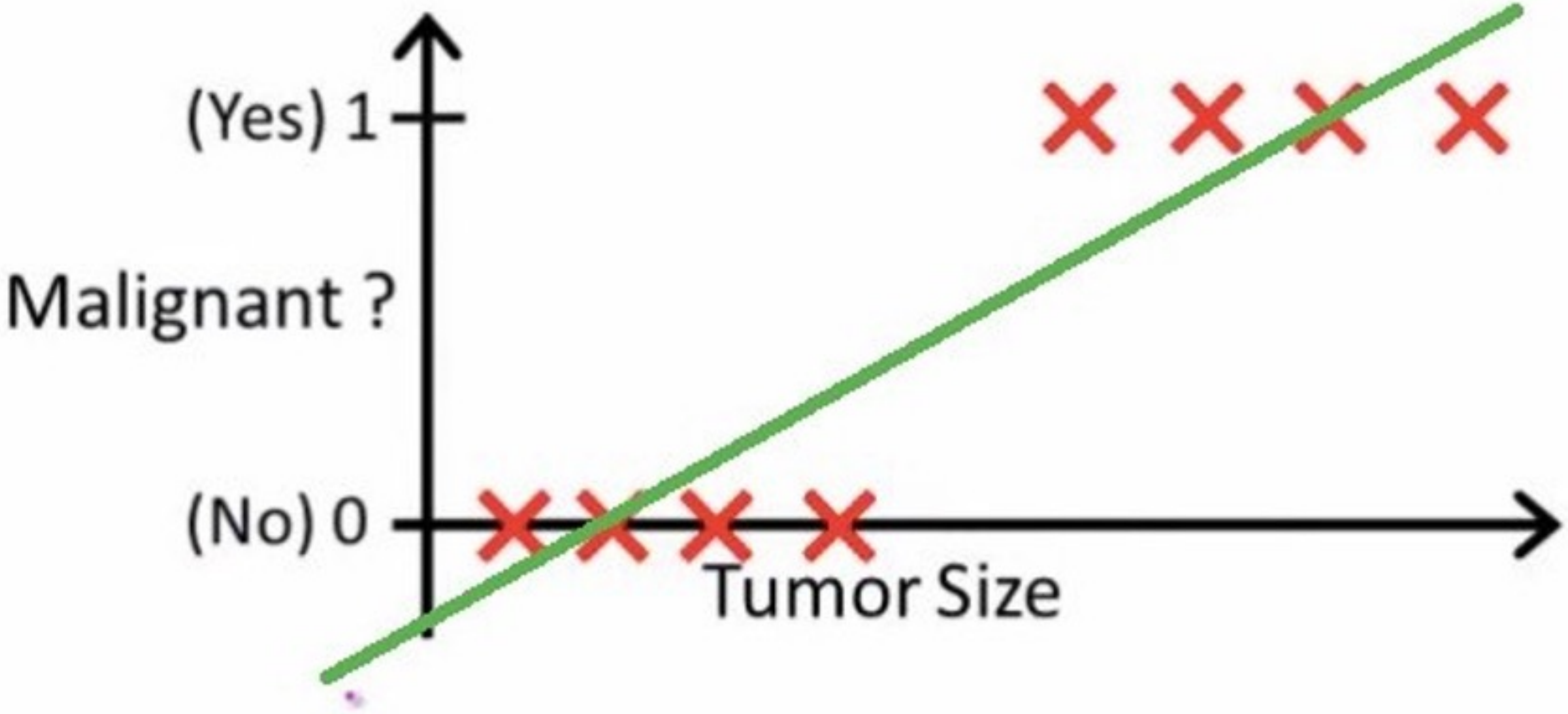
- In binary classification, any classifier can be viewed as selecting a **subset of the input space**

$$f(x) = \begin{cases} 0 & \cdots & x \in \mathscr{R}_0 \\ 1 & \cdots & x \in \mathscr{R}_1 \end{cases}$$

- Decision regions $\mathscr{R}_0, \mathscr{R}_1$ are separated using some **decision boundary**

Classification vs. Regression

- **Fun fact.** Technically, we can use linear regression for classification
  - Simply view $0/1$ class labels as outputs to predict



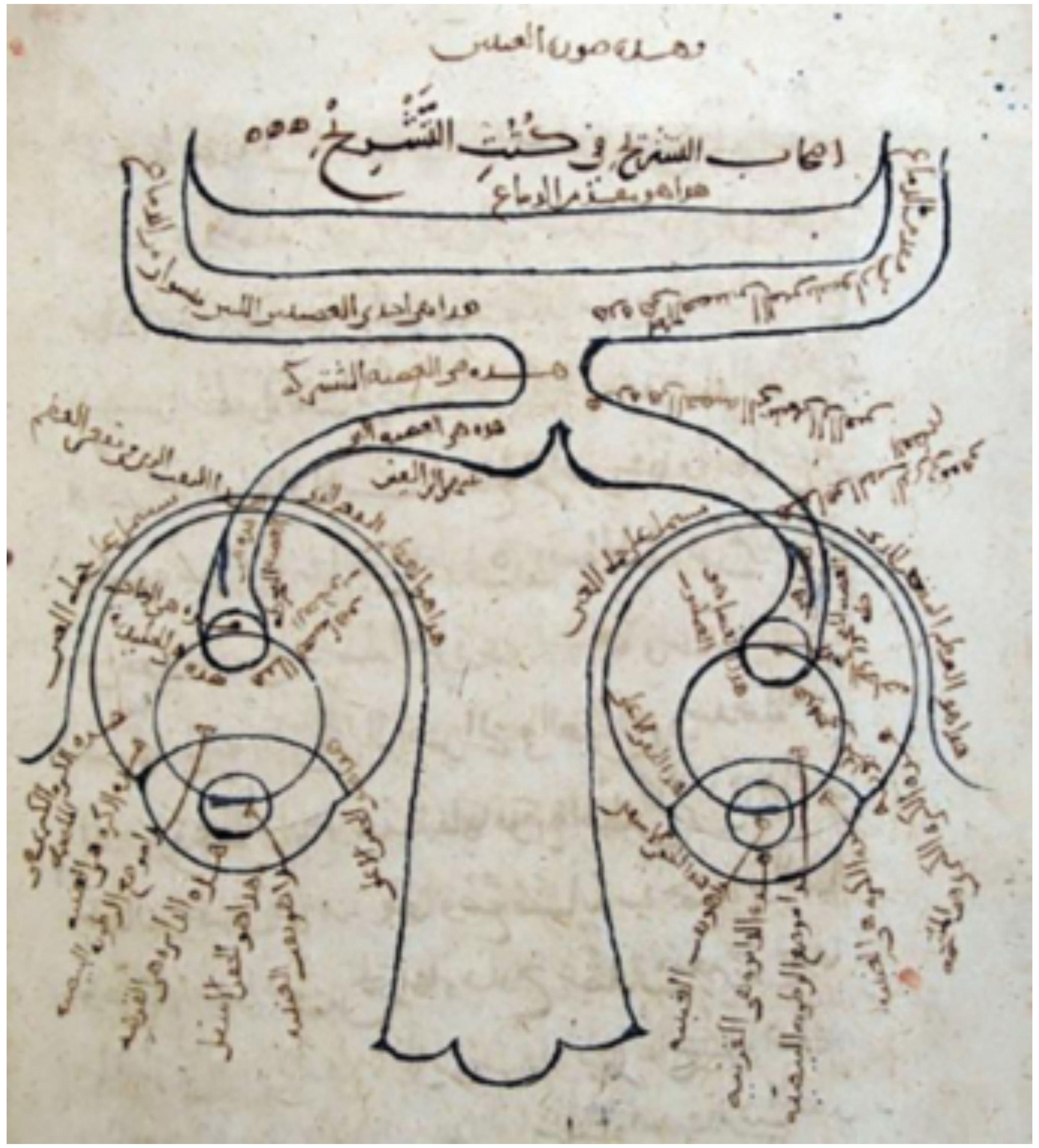
Classification vs. Regression

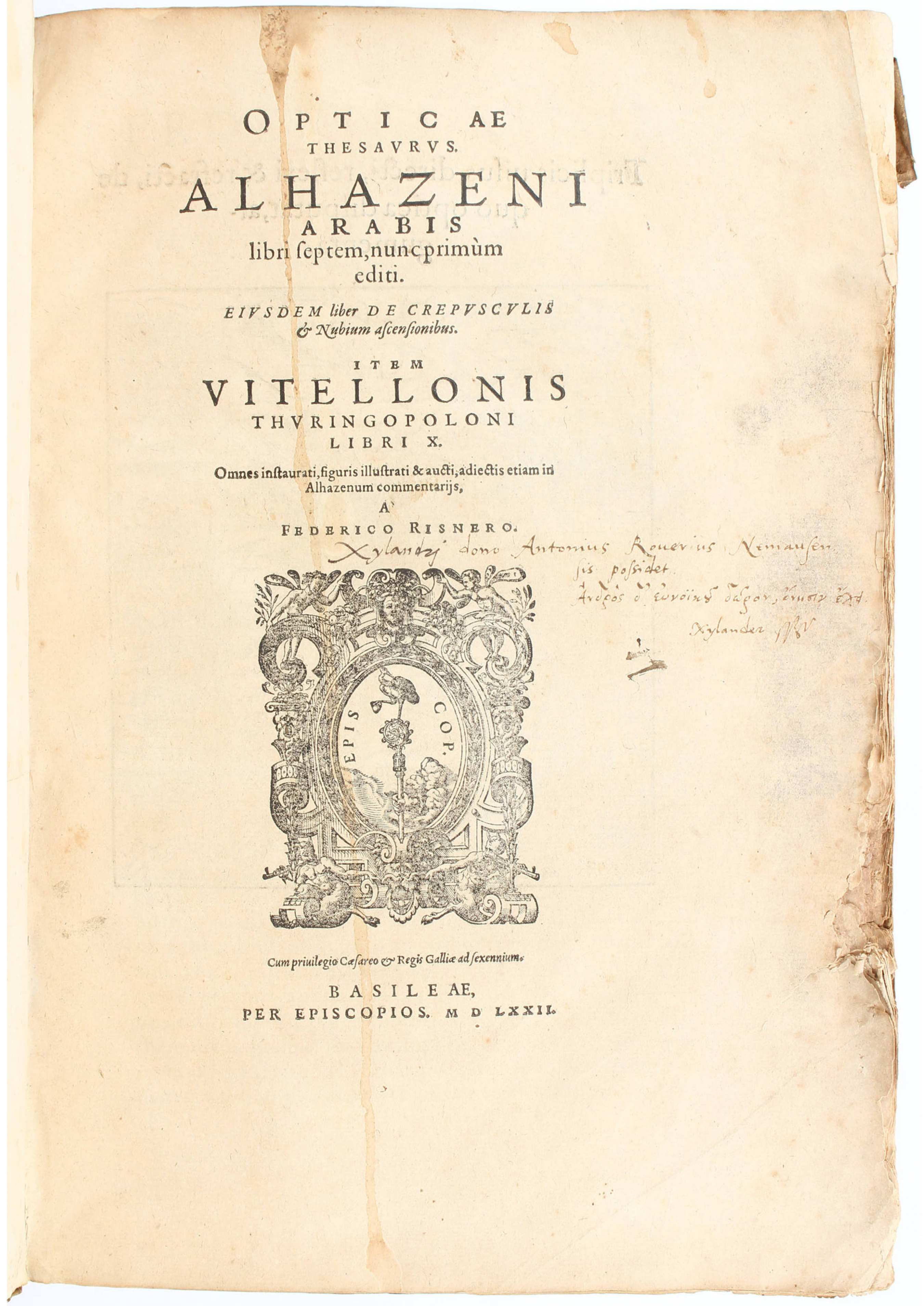
- **However.** This is not a good idea…
  - Very sensitive to “outliers,” e.g., extremely large yet benign tumor
  - Thus we want better tools

Nearest neighbors

Historical bits

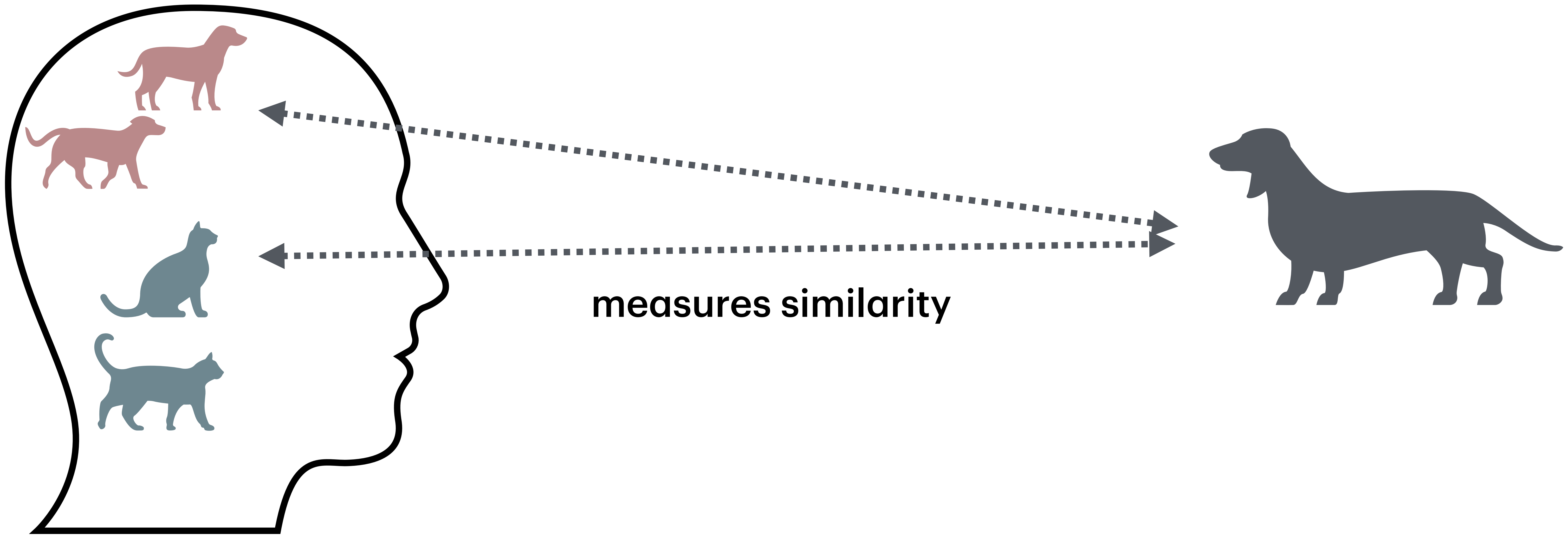
- Can be traced back to a book in 1021
  - كتاب المناظر ("the book of optics") by Ibn al-Haytham





Historical bits

- VIewed human visual recognition as a nearest neighbor
  - “Recognition is the perception of similarity between two forms—i.e., of the form
    (1) sight perceives at the moment of recognition,
    (2) and the form of that visible object, or its like, that it has perceived one or more times before.”



Setup

- We have a labeled dataset

$$D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$$

  - **Features.** $\mathbf{x}_i \in \mathscr{X}$ (continuous, discrete, mixed, …)
  - **Label.** $y_i \in \{1, \ldots, K\}$

Training

- A cool aspect of KNN is that it is **training-free**
  - All we need to do is to **store data** in some database, in a form that we can retrieve them easily

Inference

- Suppose that we are given some test sample $\mathbf{x}^{(\text{new})}$

- Pick k samples with the highest similarity:
  - Equivalently, find the training samples with bottom-k distance:

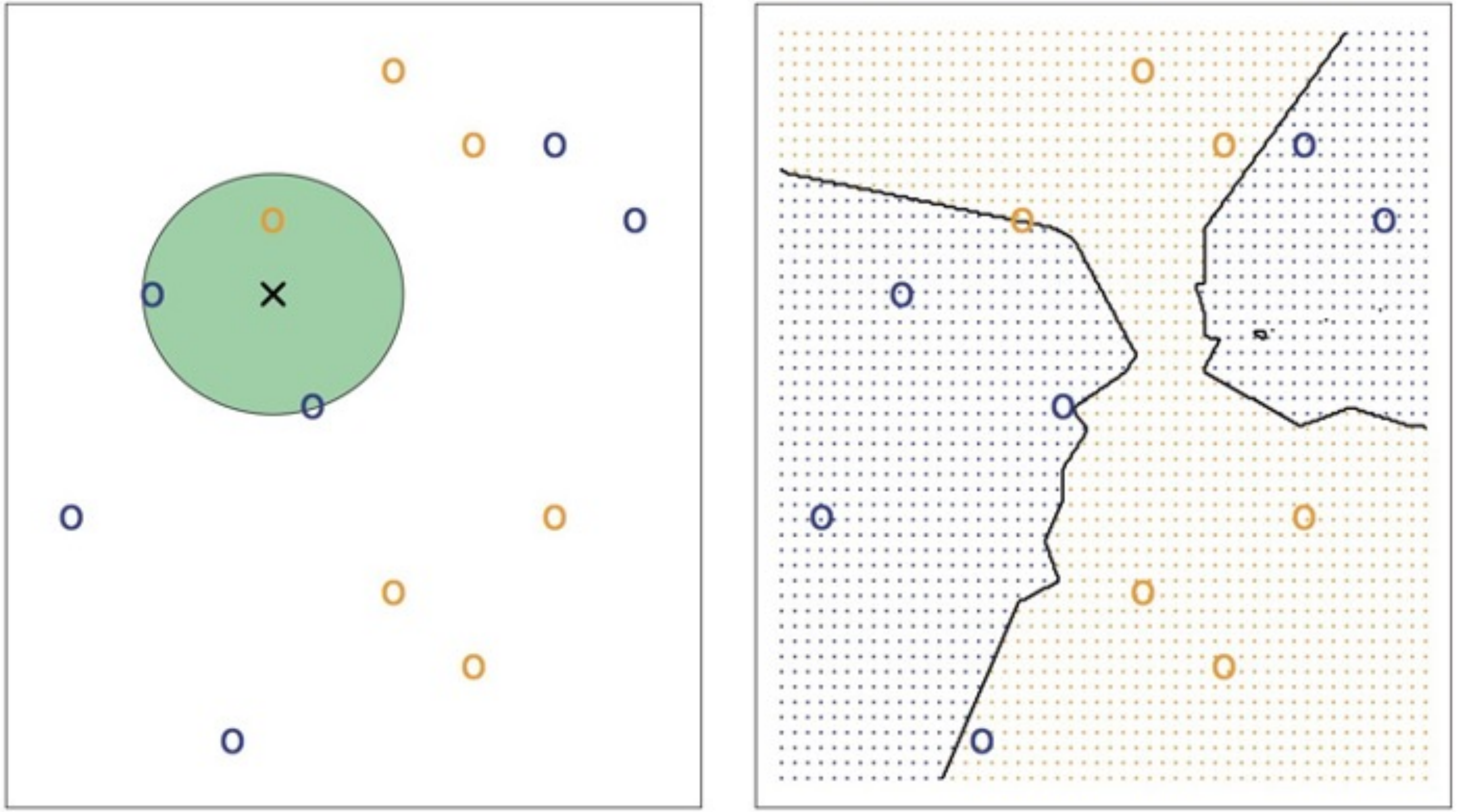
$$\min_i \operatorname{dist}(\mathbf{x}^{(\text{new})}, \mathbf{x}_{(i)})$$

- Then, predict with majority vote

(we can also do regression, via weighted averaging)

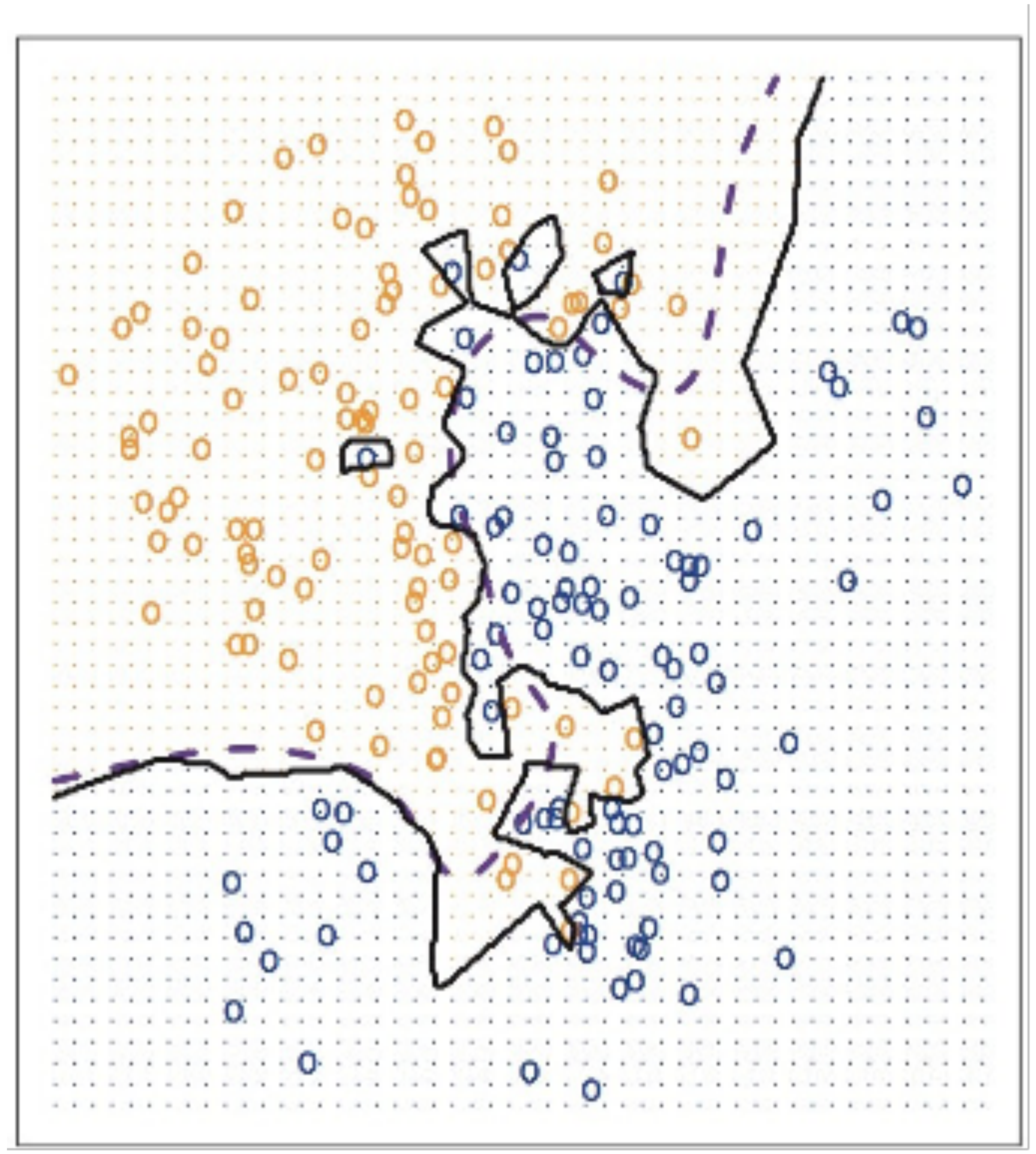
Properties

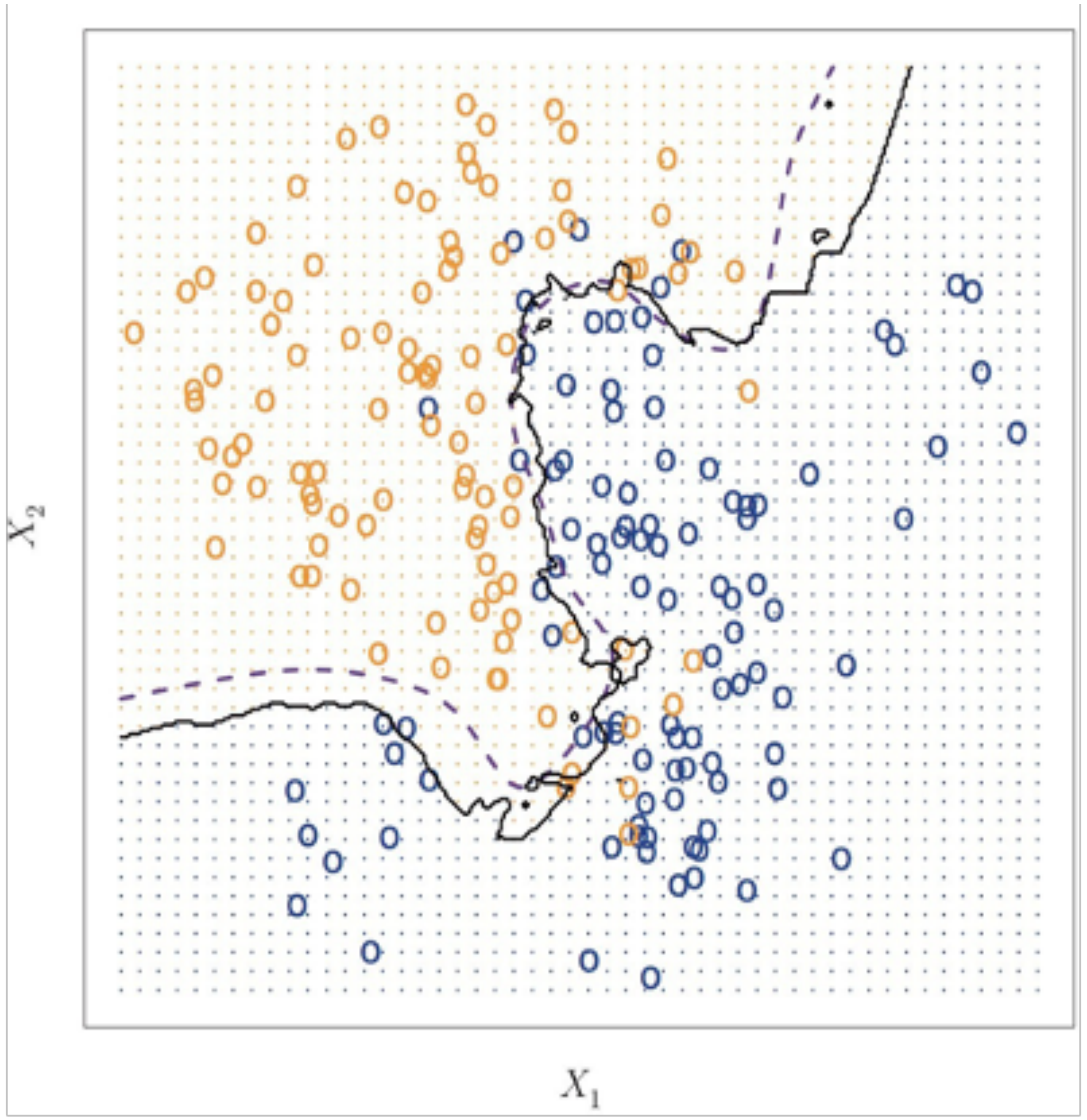
- KNN predictor is **nonlinear**
  - **Example.** $k = 3$

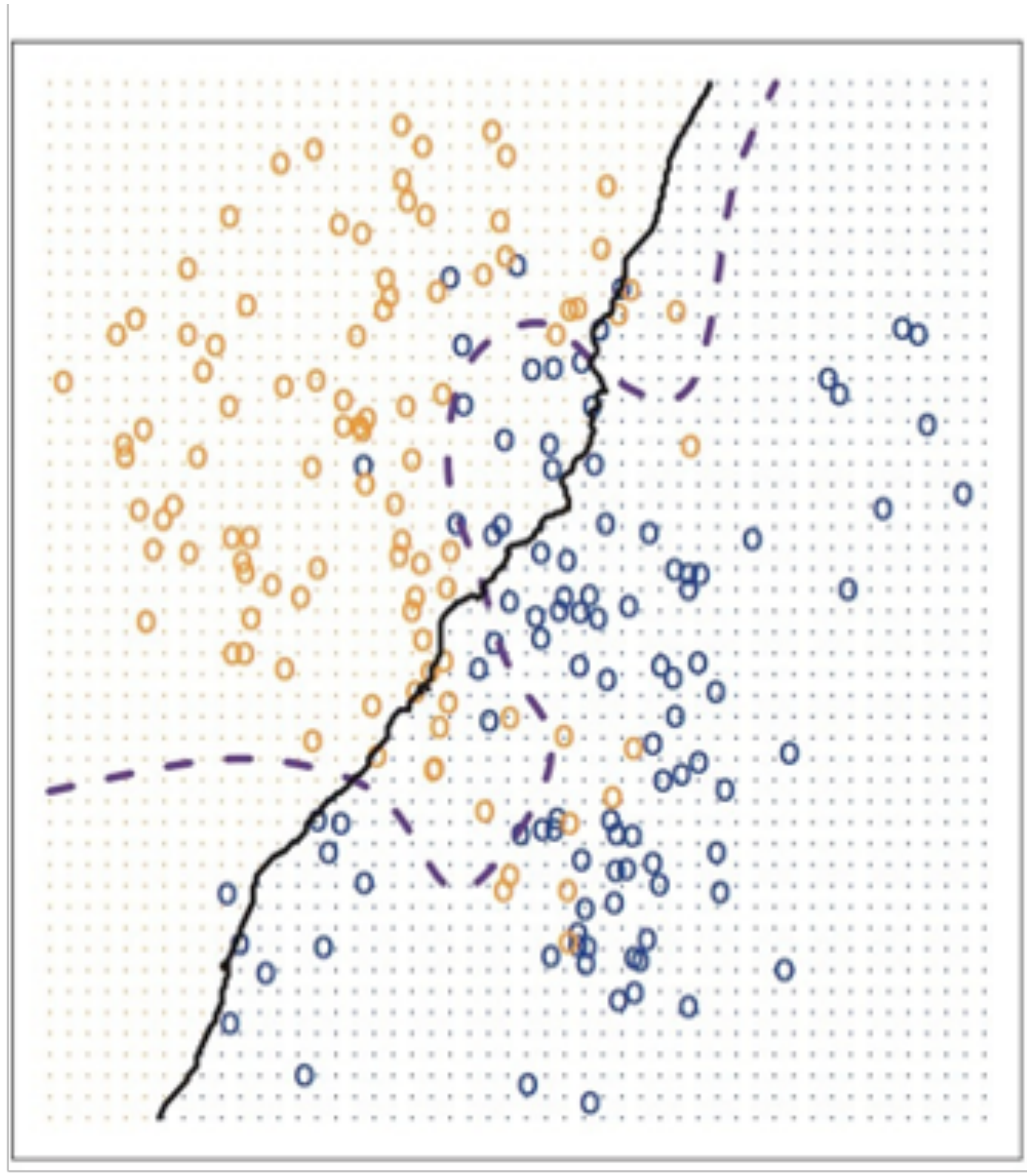


Hyperparameter

- The **neighbor set size** k has a big impact on the predictor
  - Small k: Flexibility   Larger k: Smooth decision boundary







Properties

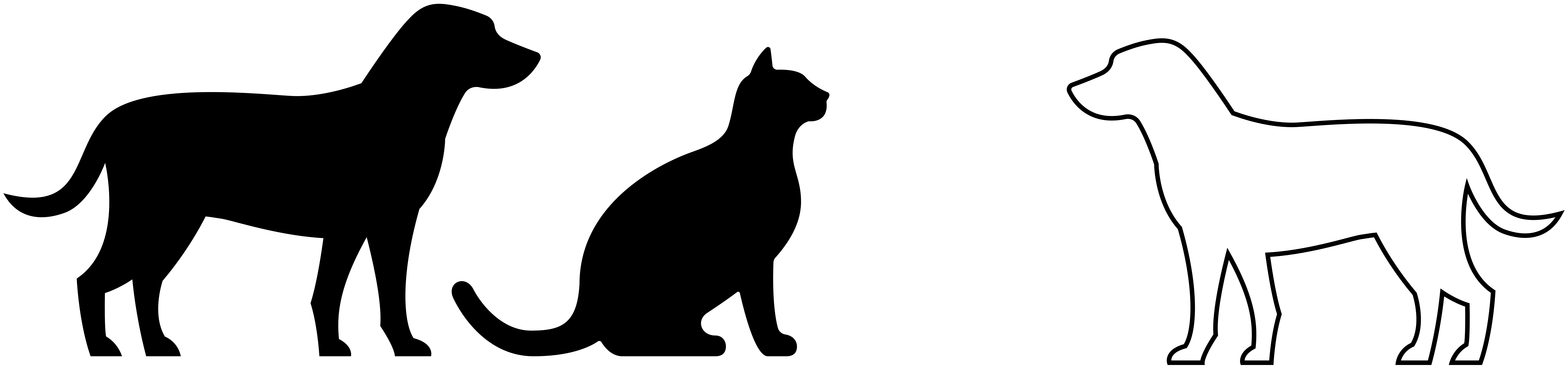
- KNN predictor is **nonparametric**
  - **Nonparametric.** Using flexible number of (or infinite) parameters
    - e.g., k-NN, Decision trees
  - **Parametric.** Parameters are finite-dimensional
    - e.g., linear regression, deep learning

Properties

- **Computation.** K-NN is difficult to scale up to large datasets
  - Pros. No training cost
  - Cons. High inference cost
    - For testing, we need to conduct n comparisons
    - Fortunately, there are many techniques to relieve this
      - Used in modern LLMs with RAG

Limitation

- The success depends critically on the similarity metric
    - The similarity should represent some semantic knowledge
        - From human
        - From data
    - We'll see later how neural nets can do this



Naïve Bayes

Setup

- Suppose that we have a labeled dataset

$$\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$$

  - $\mathbf{x}^{(i)} \in \mathbb{R}^d$
  - $y^{(i)} \in \{0,1\}$

- The data is assumed to have been independently drawn from P_{XY}

Setup

- We assume that entries of each $\mathbf{x}$ are conditionally independent given y

$$p(\mathbf{x} \mid y) = \prod_{i=1}^{d} p(x_i \mid y)$$

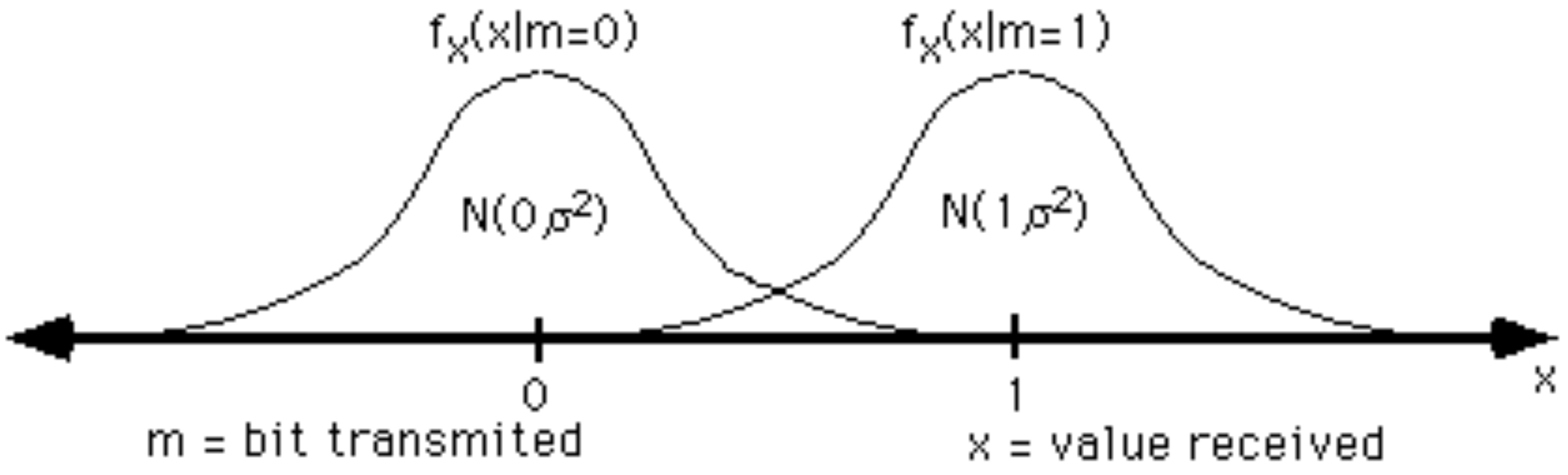
  - Can be true for tabular data, but not for images (thus naïve)

  - From now on, we let $d = 1$, without loss of generality

Bayesian approach

- Based on some human knowledge, we manually design two things:
  - Likelihood model $p(x \mid y)$
  - Prior $p(y)$

- **Example.** We may have a good physical model of the channel output (x) given the channel input (y)



Training

- Estimating parameters of $p(x \mid y), p(y)$ from data
- Example. Gaussian likelihood has four parameters
    - Mean and variance, for each y

$$p(x \mid y) = \frac{1}{\sigma_y \sqrt{2\pi}} \exp\left(-\frac{(x-\mu_y)^2}{2\sigma_y^2}\right)$$

$$\theta_l = (\mu_0, \mu_1, \sigma_0, \sigma_1) \in \mathbb{R}^4$$

- Example. Bernoulli prior has one parameter

$$\theta_p = p(1) \qquad \in [0,1]$$

Training

- To fit the parameters, we maximize the joint probability of the training data given the parameters

$$\max_{\theta} p_{\theta}(\mathbf{x}_1, \ldots, \mathbf{x}_n, y_1, \ldots, y_n)$$

$$= \quad \max_{\theta_\ell, \theta_p} \prod_{i=1}^{n} p_{\theta_\ell}(\mathbf{x}_i \mid y_i) p_{\theta_p}(y_i)$$

- Note. As we have seen last week, this has an ERM interpretation

Training

- We can solve two sub-problems separately

$$\min_{\theta_\ell} \sum_{i=1}^{n} \Big(-\log p_{\theta_\ell}(\mathbf{x}_i \mid y_i) \Big)$$

$$\min_{\theta_p} \sum_{i=1}^{n} \Big(-\log p_{\theta_p}(y_i) \Big)$$

- The solution to the upper optimization problem is what we call the maximum-likelihood estimate (MLE)

Training

- Example. Consider the subproblem for Gaussian likelihood:

$$\min_{\theta_\ell} \sum_{i=1}^{n} \left(- \log p_{\theta_\ell}(\mathbf{x}_i \,|\, y_i) \right)$$

$$\Leftrightarrow \quad \min_{\theta_\ell} \left(\sum_{i=1}^{n} \frac{\|\mathbf{x}_i - \mu_{(y_i)}\|^2}{2\sigma^2_{(y_i)}} + \log(\sigma_{(y_i)}) \right)$$

  - Solving this optimization will give class-wise sample mean and class-wise sample variance (check!)

Training

- Example. Consider the subproblem for Bernoulli prior

$$\min_{\theta_p} \sum_{i=1}^{n} \Big(-\log p_{\theta_p}(y_i) \Big)$$

$$\Leftrightarrow \quad \min_{\theta_p} \left(\sum_{i:y_i=1} -\log(\theta_p) + \sum_{i:y_i=0} -\log(1-\theta_p) \right)$$

  - Solving this optimization will give the sample frequency

$$\theta_p = \frac{\#1\text{s in dataset}}{n}$$

Inference

- We conduct MAP estimation

$$f(\mathbf{x}) = \arg\max_{y} p(y \mid \mathbf{x})$$

$$= \arg\max_{y} p(y) p(\mathbf{x} \mid y)$$

$$= \arg\max_{y} \left(p(y) \prod_{i=1}^{d} p(x_i \mid y) \right)$$

Properties

- **Computation.** Quite simple for popular choice of $p(\mathbf{x} \mid y)$ and $p(y)$
  - Training. Already known, explicit formula
  - Inference. Simply compute $p(y \mid \mathbf{x})$

  - However, these can be very messy for atypical models & priors
    - or if there is any dependency structure

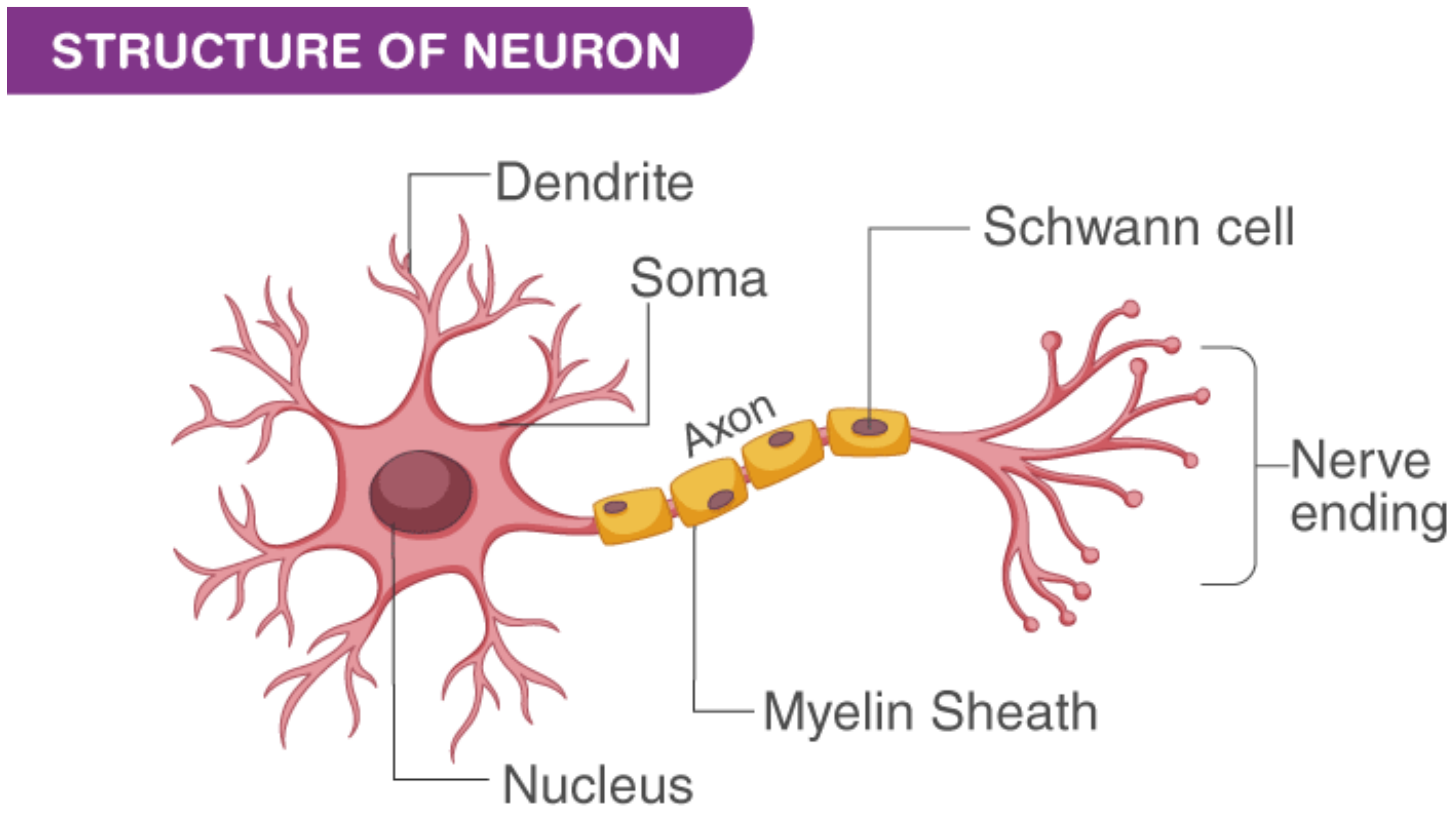
Limitation

- Requires a well-designed prior and likelihood
  - We expect very complicated $p(\mathbf{x} \mid y)$ for, e.g., visual data
  - We want an automated mechanism to design these as well

Perceptrons

Historical bits

- The first “neural network” designed by Rosenblatt (1958)



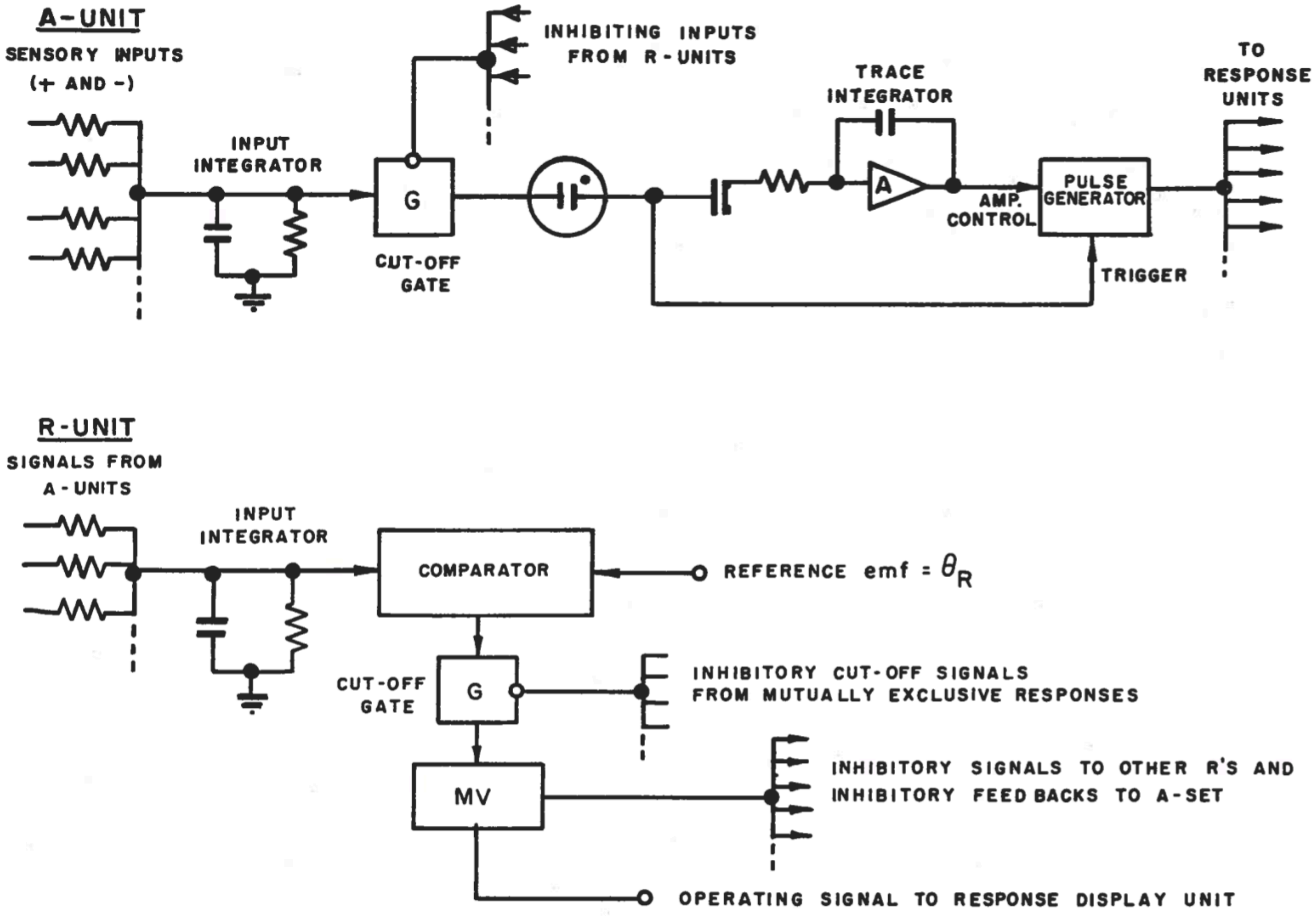
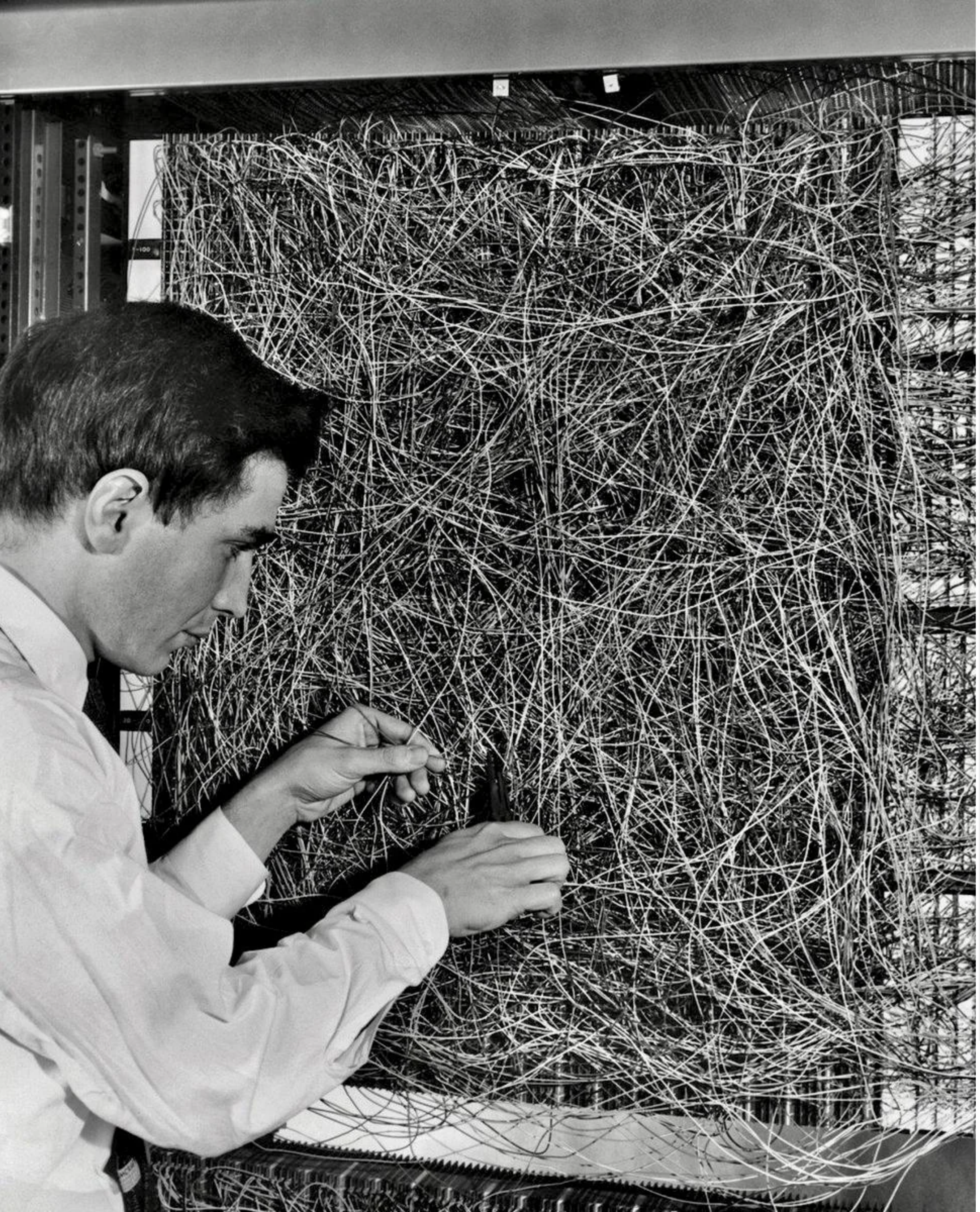
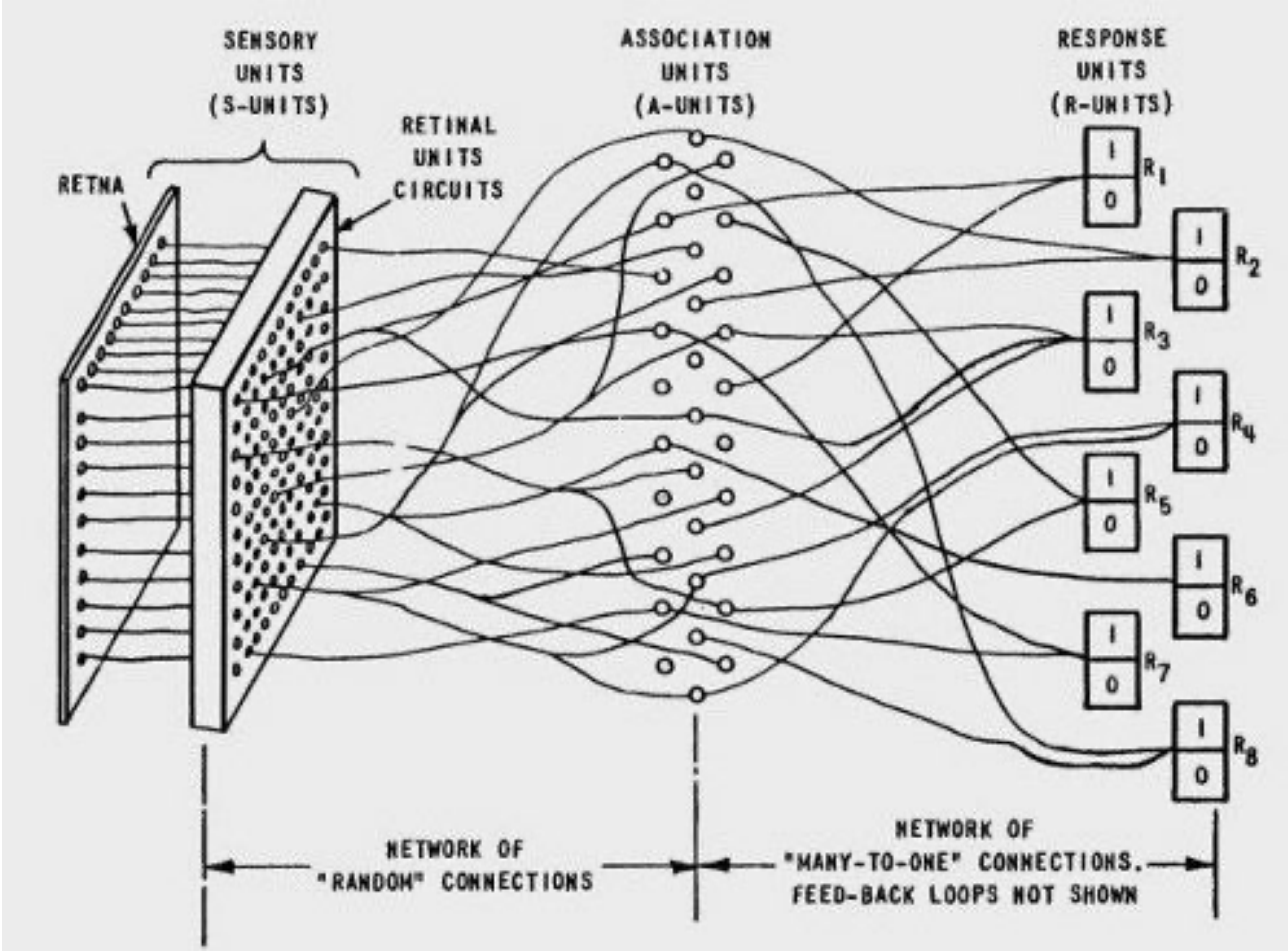


FIGURE 5
DESIGN OF TYPICAL UNITS



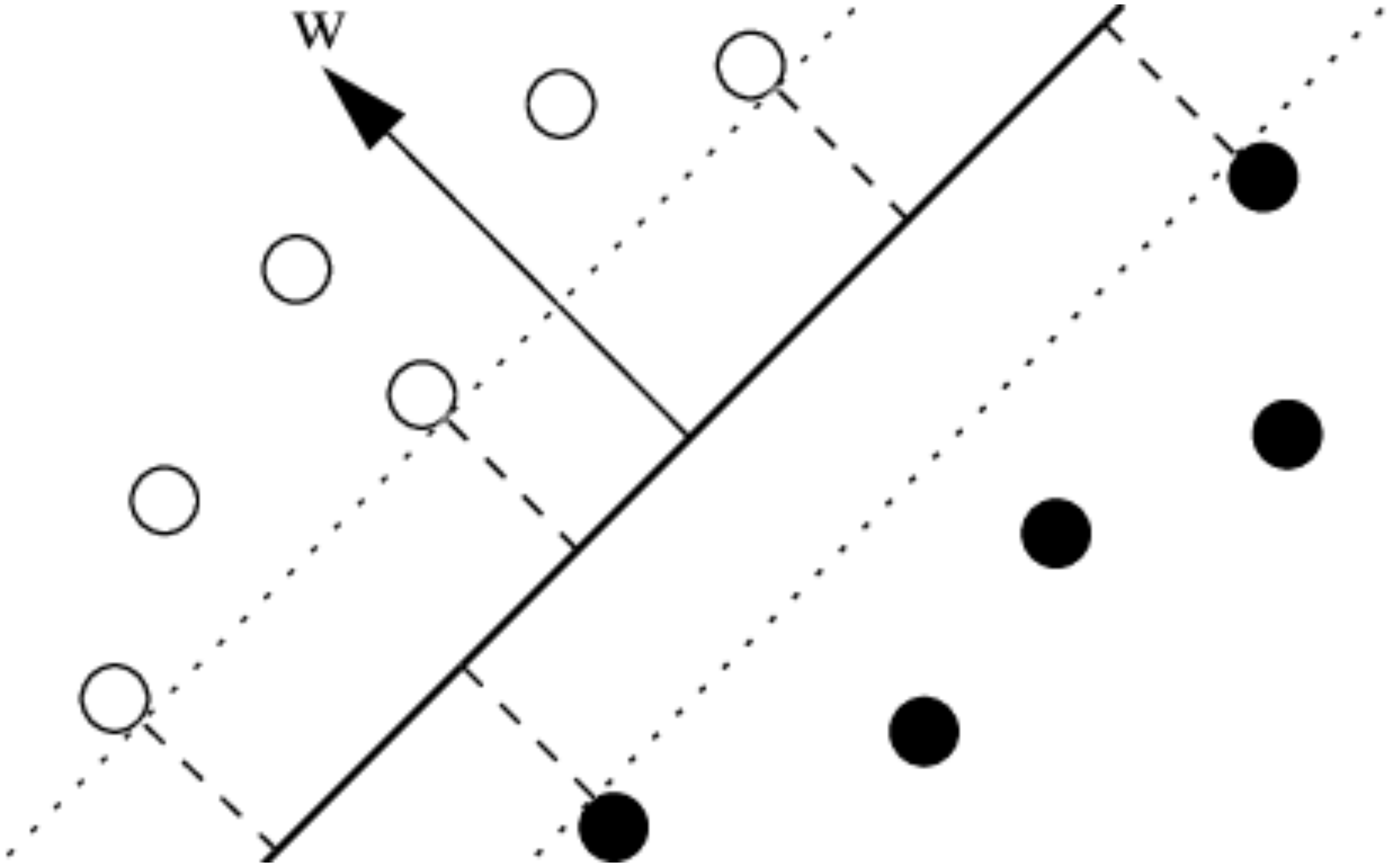

SENSORY UNITS (S-UNITS)
ASSOCIATION UNITS (A-UNITS)
RESPONSE UNITS (R-UNITS)
RETNA
RETINAL UNITS CIRCUITS
R_1
R_2
R_3
R_4
R_5
R_6
R_7
R_8
NETWORK OF "RANDOM" CONNECTIONS
NETWORK OF "MANY-TO-ONE" CONNECTIONS. FEED-BACK LOOPS NOT SHOWN

Linear model

- Perceptron is a method to train a linear classifier
  - Linear classifier is about drawing a linear decision boundary

$$\mathbf{w}^\top \mathbf{x} + b = 0$$

  - This divides two regions:
    - $\{\mathbf{x} \mid \mathbf{w}^\top \mathbf{x} + b > 0\}$
    - $\{\mathbf{x} \mid \mathbf{w}^\top \mathbf{x} + b < 0\}$



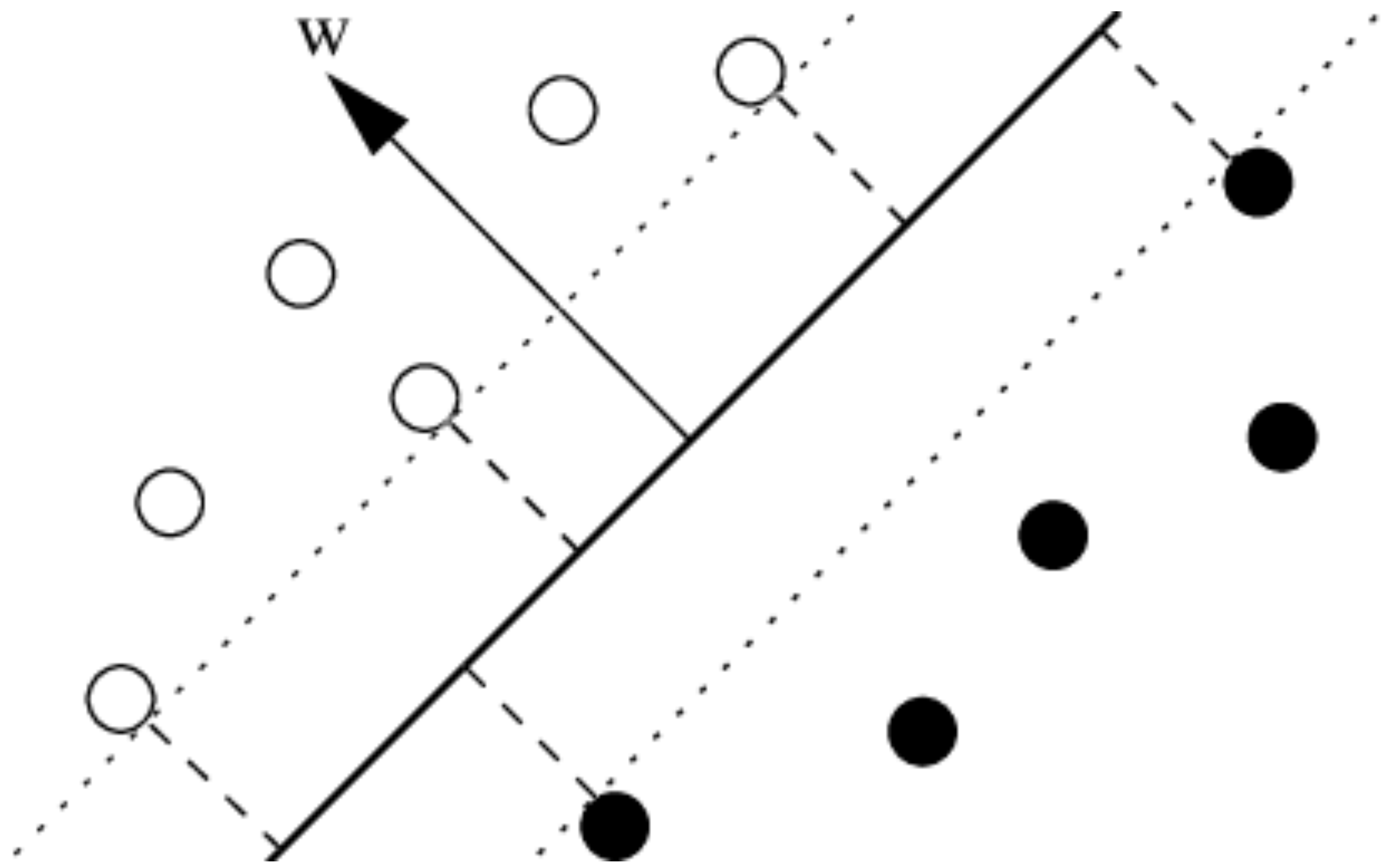
Inference

- For inference, we use the sign of linear models

$$f_\theta(\mathbf{x}) = \mathbf{1}\{\mathbf{w}^\top \mathbf{x} + b > 0\}$$

  - Again, by stacking, we can write more neatly as

$$f_\theta(\mathbf{x}) = \mathbf{1}\{\theta^\top \tilde{\mathbf{x}} > 0\}$$



Training

- The most standard way to find a linear classifier would be to solve:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}\{f_\theta(\mathbf{x}_i) \neq y_i\}$$

- Or more neatly, we can write as:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \left(f_\theta(\mathbf{x}_i)(1 - y_i) + (1 - f_\theta(\mathbf{x}_i))y_i \right)$$

Training

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \left(f_\theta(\mathbf{x}_i)(1 - y_i) + (1 - f_\theta(\mathbf{x}_i))y_i \right)$$

- Problem. Difficult to optimize
  - Explicit solution — not available
  - Gradient descent — difficult to evaluate gradient
    - $f_\theta(\,\cdot\,)$ contains $\mathbf{1}\{\,\cdot\,\}$ — gradient is zero almost everywhere

Training

Rosenblatt's solution.

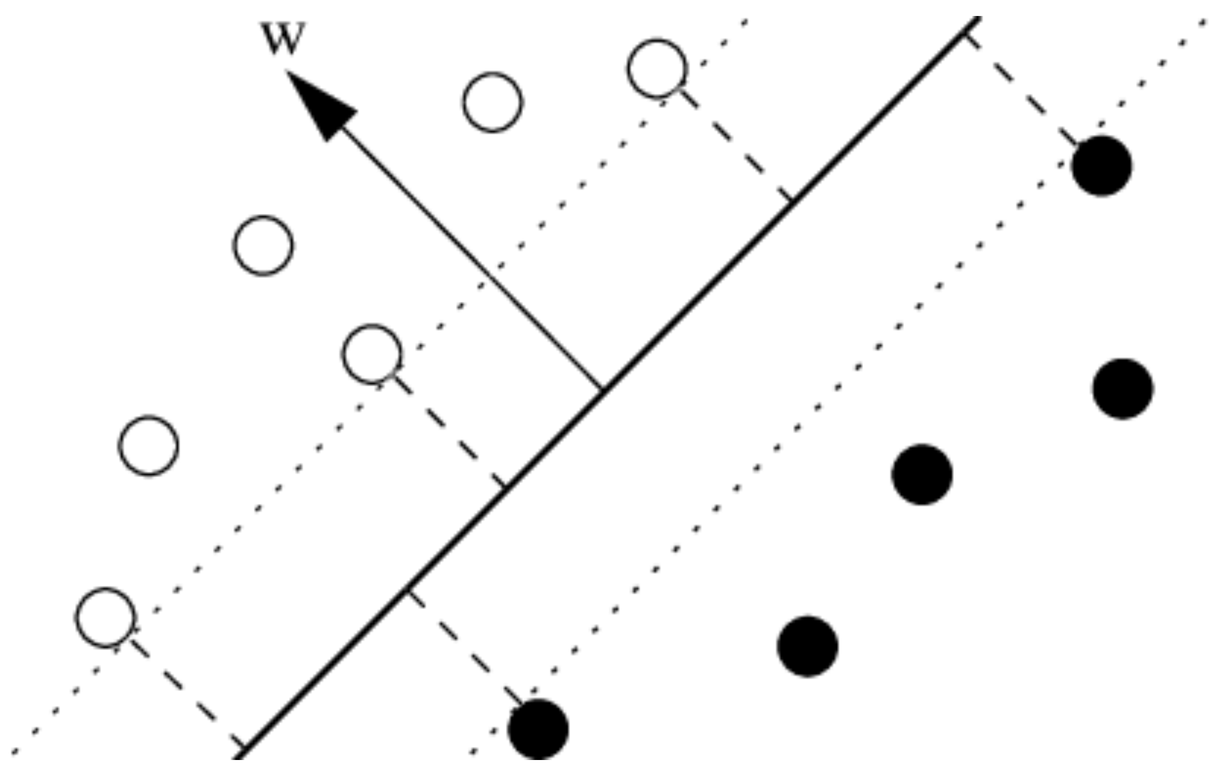
- Instead of the loss

$$\ell(y, f_\theta(\mathbf{x})) = f_\theta(\mathbf{x})(1-y) + (1 - f_\theta(\mathbf{x}))y$$

use this loss instead:

$$\ell(y, f_\theta(\mathbf{x})) = (f_\theta(\mathbf{x}) - y) \cdot \theta^\top \tilde{\mathbf{x}}$$

  - When wrong, the loss is: $|\theta^\top \tilde{\mathbf{x}}|$
  - When correct, the loss is: 0



- **Intuition.** We penalize the "confidence" of misprediction

Training

$$\ell(y, f_\theta(\mathbf{x})) = (f_\theta(\mathbf{x}) - y) \cdot \theta^\top \tilde{\mathbf{x}}$$

- With this new loss, suddenly the gradient is non-zero

$$\nabla_\theta \ell(y, f_\theta(\mathbf{x})) = (f_\theta(\mathbf{x}) - y)\tilde{\mathbf{x}}$$

- The loss like this — not a true loss but helps optimization — is called surrogate loss

Optimization

- The original perceptron paper assumes that:
  - the data comes one-by-one
  - we cannot re-use the past data
- Such scenario is called online learning

Optimization

- Given a sample, the gradient is:

$$\nabla_\theta \ell(y, f_\theta(\mathbf{x})) = (f_\theta(\mathbf{x}) - y)\tilde{\mathbf{x}}$$

  - If wrong for a sample with $y = 1$:

$$\theta^{(i+1)} = \theta^{(i)} + \eta \cdot \tilde{\mathbf{x}}$$

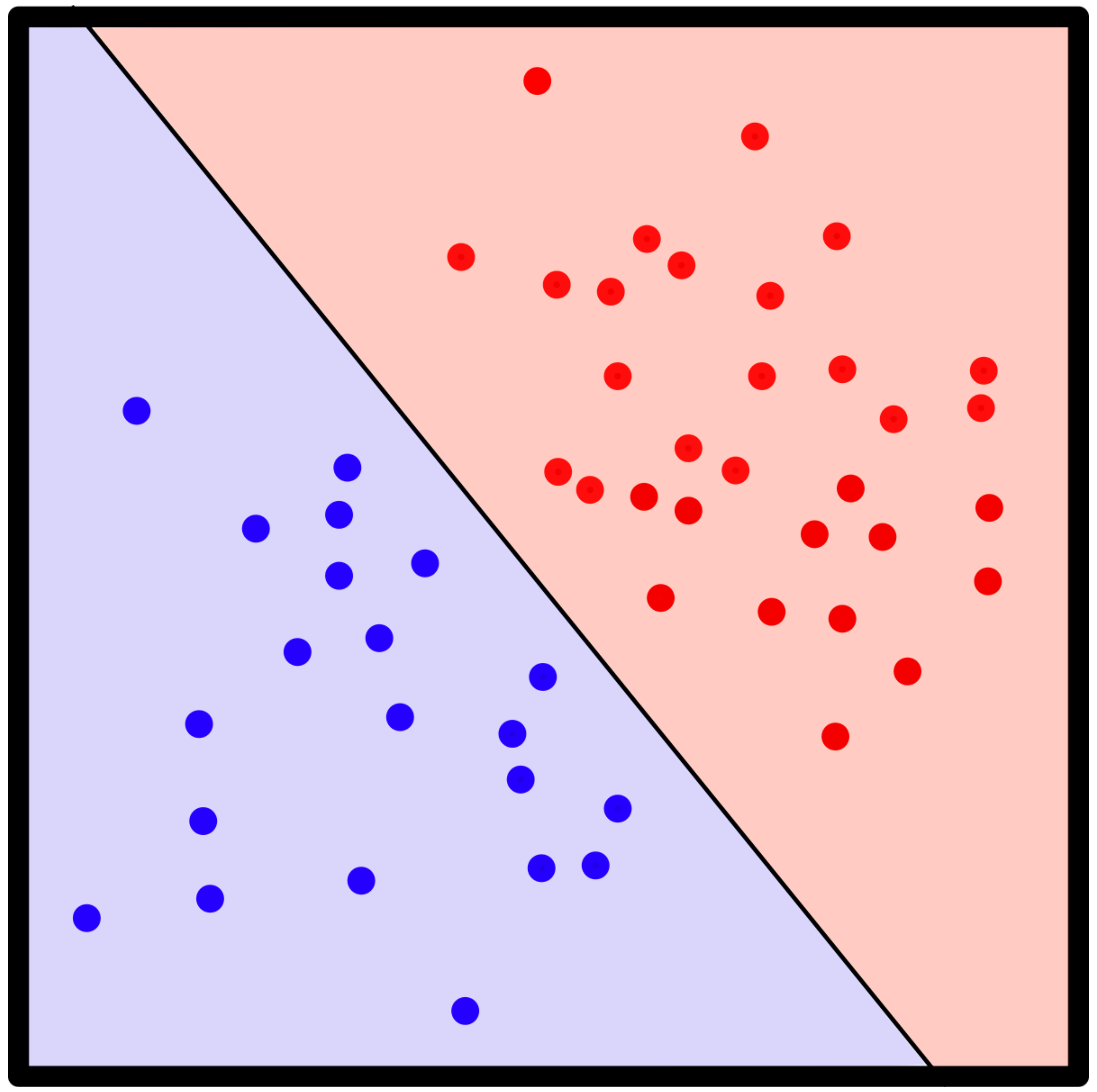
  - If wrong for a sample with $y = 0$:

$$\theta^{(i+1)} = \theta^{(i)} - \eta \cdot \tilde{\mathbf{x}}$$

  - If correct, no change

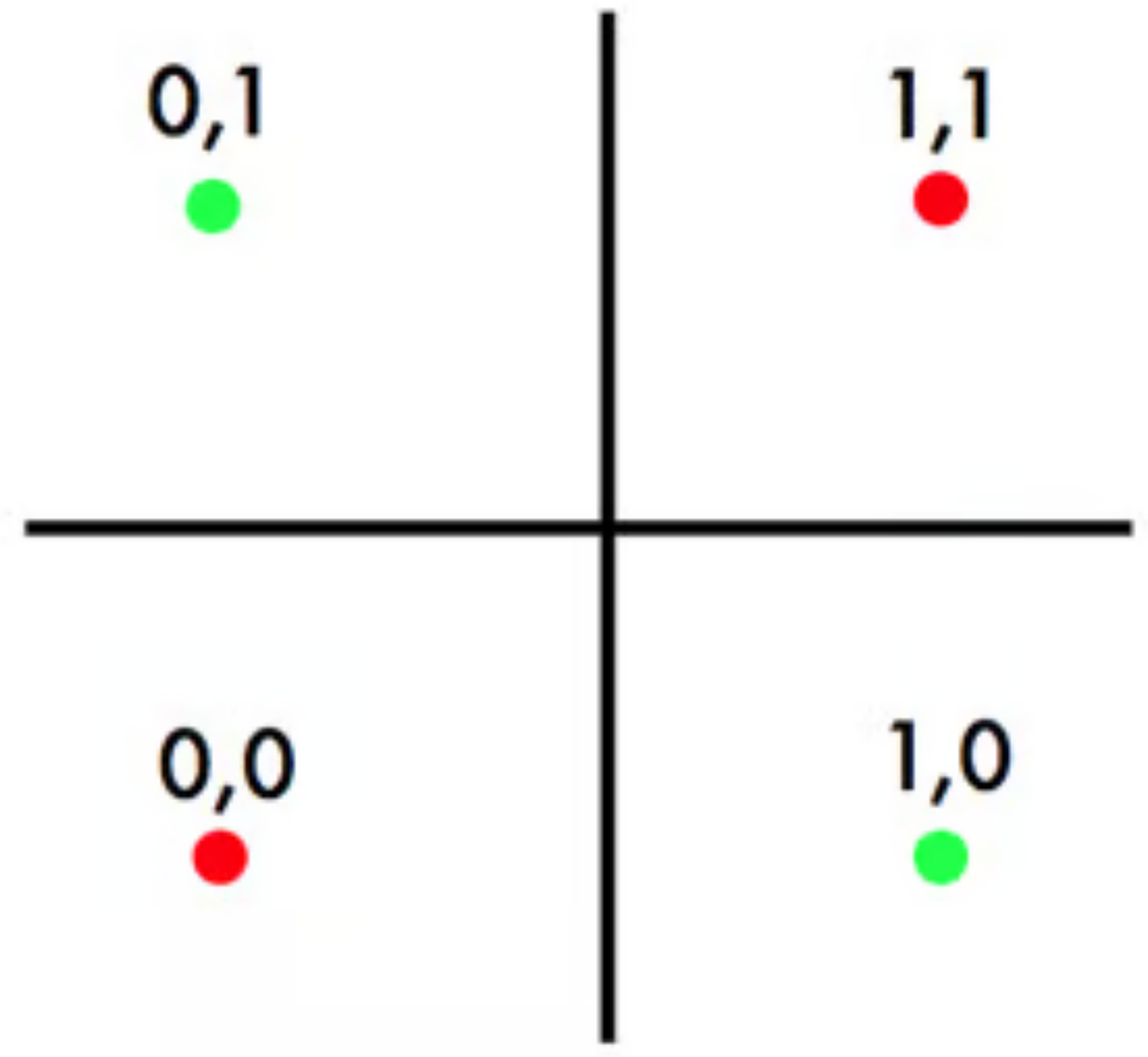
Properties

- **Computation.** Quite easy
  - Training. Simply add or subtract data $\mathbf{x}$
    - Also, provably converges whenever the data is separable
  - Inference. Simply do a dot product



Limitations

- Cannot achieve low training loss on not linearly separable data



Logistic Regression

Logistic Regression

- Another popular version of the linear classifier

$$f_\theta(\mathbf{x}) = \mathbf{1}\{\theta^\top \tilde{\mathbf{x}} \geq 0\}$$

- Unlike Rosenblatt, logistic regression interprets $\theta^\top \tilde{\mathbf{x}}$ as a log-likelihood ratio of the model's internal probability estimate

$$\log\left(\frac{p(y=1\,|\,\mathbf{x})}{p(y=0\,|\,\mathbf{x})}\right) \approx \theta^\top \tilde{\mathbf{x}}$$

- **Brainteaser.** Why not interpret as $p(y=1\,|\,\mathbf{x})$?

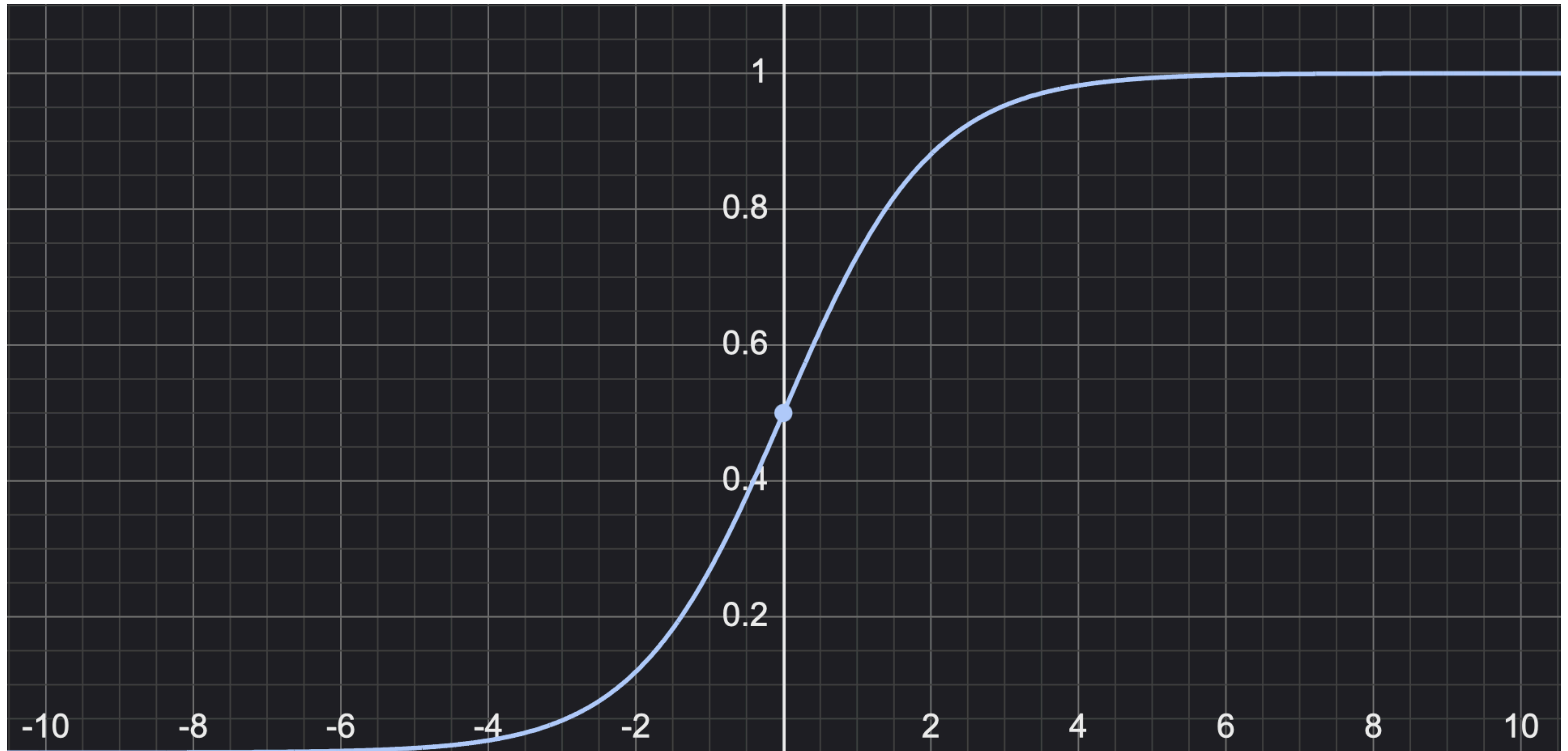
Logistic Regression

$$\log\left(\frac{p(y=1\,|\,\mathbf{x})}{p(y=0\,|\,\mathbf{x})}\right) \approx \theta^\top \tilde{\mathbf{x}}$$

- In other words, we are modeling the posterior distribution as

$$p(y=1\,|\,\mathbf{x}) = \frac{1}{1+\exp(-\theta^\top \tilde{\mathbf{x}})}$$

- The function $\sigma(t) = 1/1 + \exp(-t)$ is the logistic function (a.k.a. sigmoid)



Training

- Given the data, maximize the log-likelihood

$$\max_{\theta} \frac{1}{n} \sum_{i=1}^{n} \log p(y_i \mid \mathbf{x}_i)$$

- Equivalently, minimize the NLL loss:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \log \left(\frac{1}{p(y_i \mid \mathbf{x}_i)} \right)$$

Training

- Equivalently again, we are solving:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f_\theta(\mathbf{x}_i))$$

where

- $f_\theta(\cdot)$ is the sigmoid of the prediction

$$f_\theta(\mathbf{x}) = \sigma(\theta^\top \tilde{\mathbf{x}})$$

- $\ell(\cdot, \cdot)$ is the cross-entropy

$$\ell(y, t) = \mathrm{CE}(\mathbf{1}_y, [t, 1-t]) = \log(t)^{-y} + \log(1-t)^{y-1}$$

Training

- More tediously, this can be written as

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} (-y_i)\log(\sigma(\theta^\top \tilde{\mathbf{x}}_i)) + (y_i - 1)\log(1 - \sigma(\theta^\top \tilde{\mathbf{x}}_i))$$

- No analytic solution, but is convex and can use GD

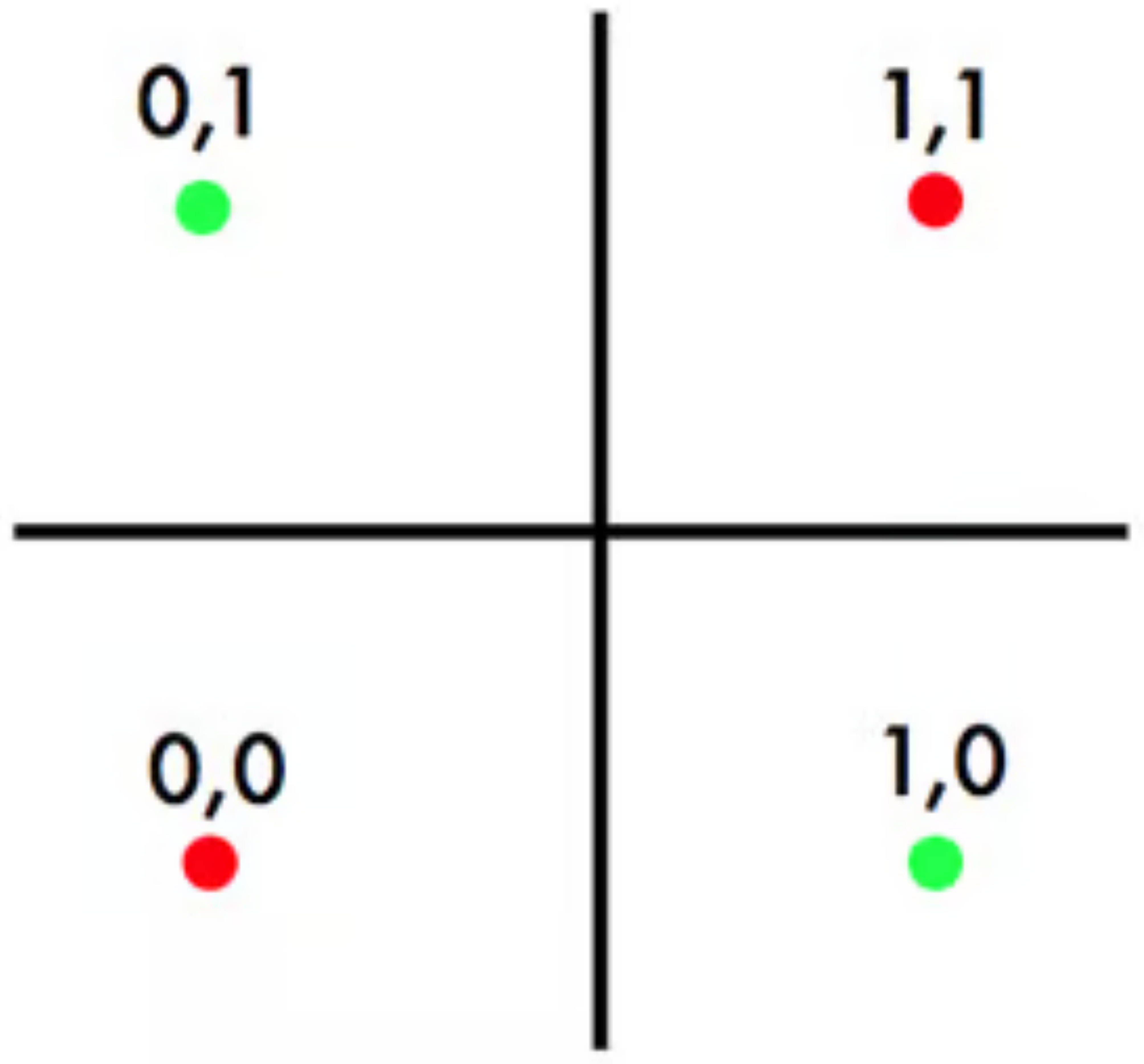
$$\theta^{(\text{new})} = \theta + \eta \cdot \frac{1}{n} \sum_{i=1}^{n} (y_i - \sigma(\theta^\top \tilde{\mathbf{x}}_i))\tilde{\mathbf{x}}_i$$

Properties

- **Computation.** Relatively easy
  - Training. Requires GD, but is convex
  - Inference. Easy — Dot product and apply threshold

Limitation

- Again, cannot fit not-linearly-separable data



Next class

- Sophisticated versions of linear classifiers

</lecture 4>