Simple Classifiers

EECE454 Intro. to Machine Learning Systems

Fall 2024

Notice

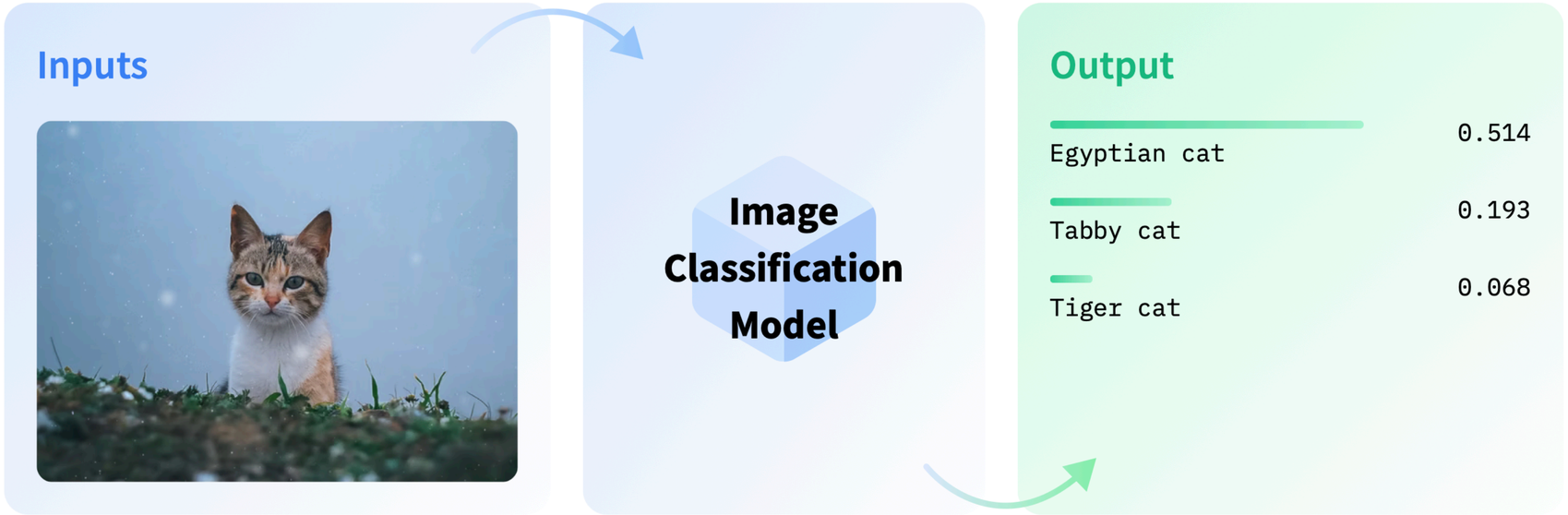
- **Last week.** As you noticed, video lectures are not uploaded yet
    - Sorry!
    - Will cover decision trees, bagging, and boosting
        - quite distinct in style from other ML algorithms
- **Assignment#1.** Also delayed!
    - Uploaded today
    - Due: 10/3

Today

- Basic ML algorithms for **classification**
  - Nearest Neighbors
  - Naïve Bayes
  - Perceptrons
  - Logistic Regression

Classification

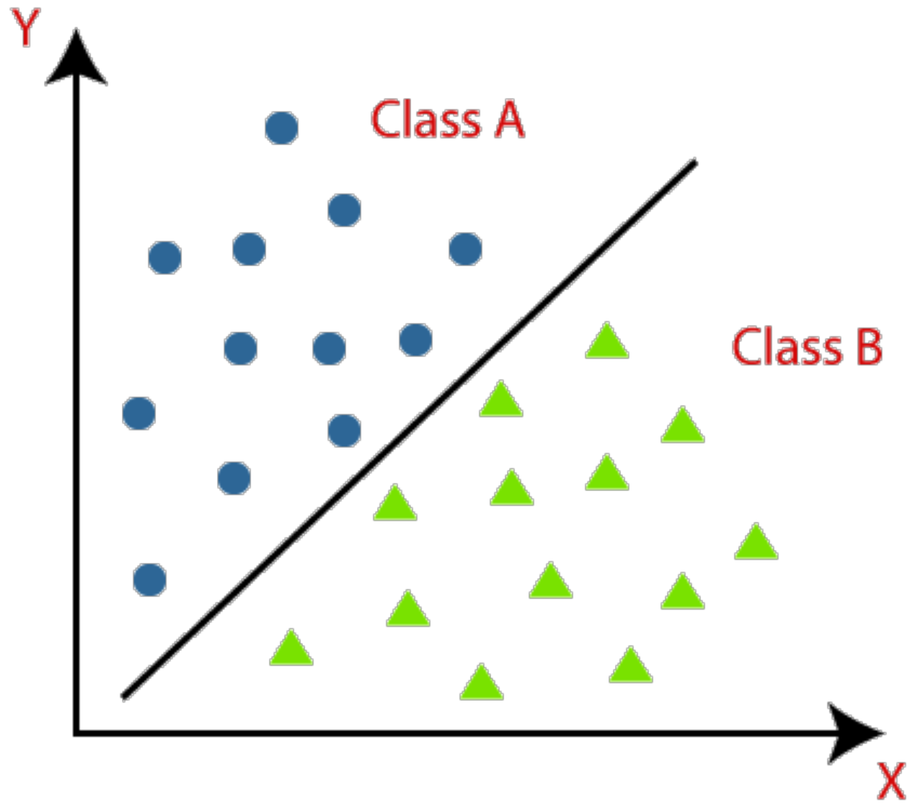
- **Task.** Given some input X, predict an output $Y \in \{1, \ldots, K\}$
  - Y is called "class"
  - c.f., the case of linear regression, where $Y \in \mathbb{R}$



Source: HuggingFace

Binary Classification

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



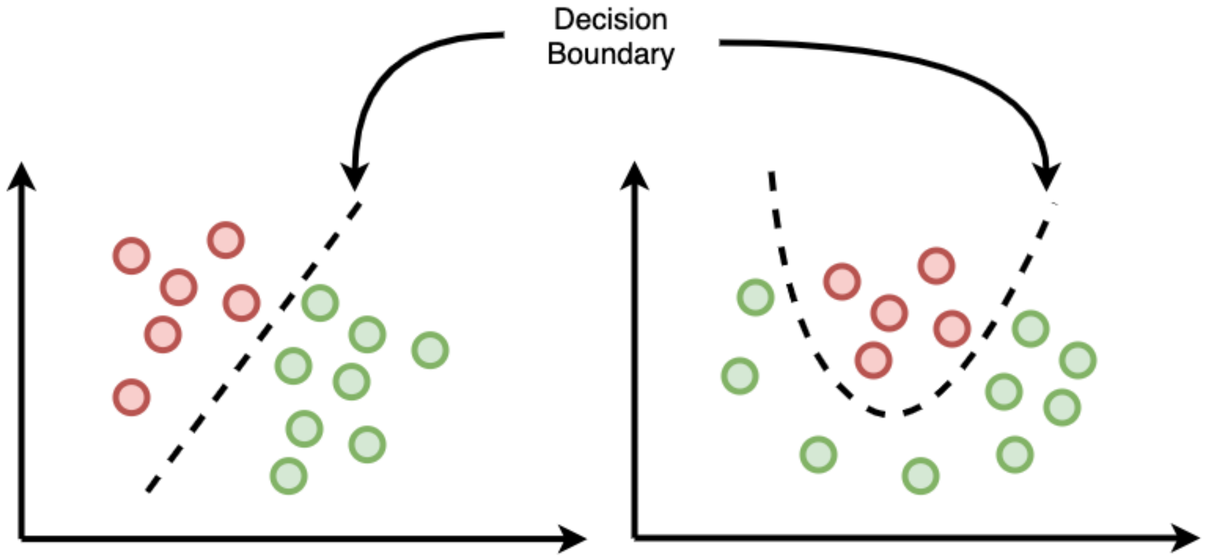
Binary Classification

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

- Any classifier can be viewed as **selecting a subset of the input space**

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

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



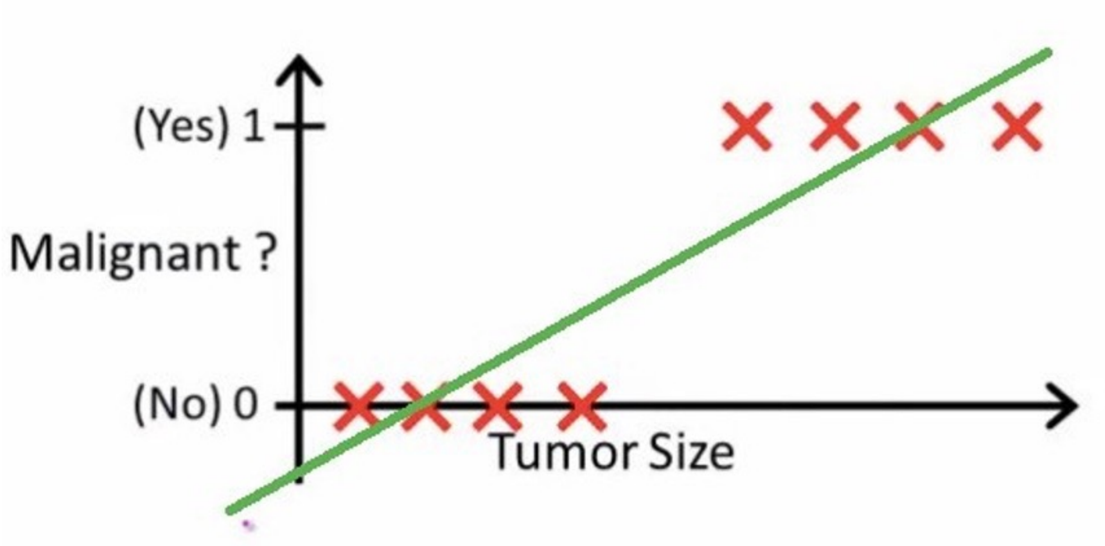
Linear regression, for classification?

- **Question.** Can we use linear regression to solve classification tasks?

Linear regression, for classification?

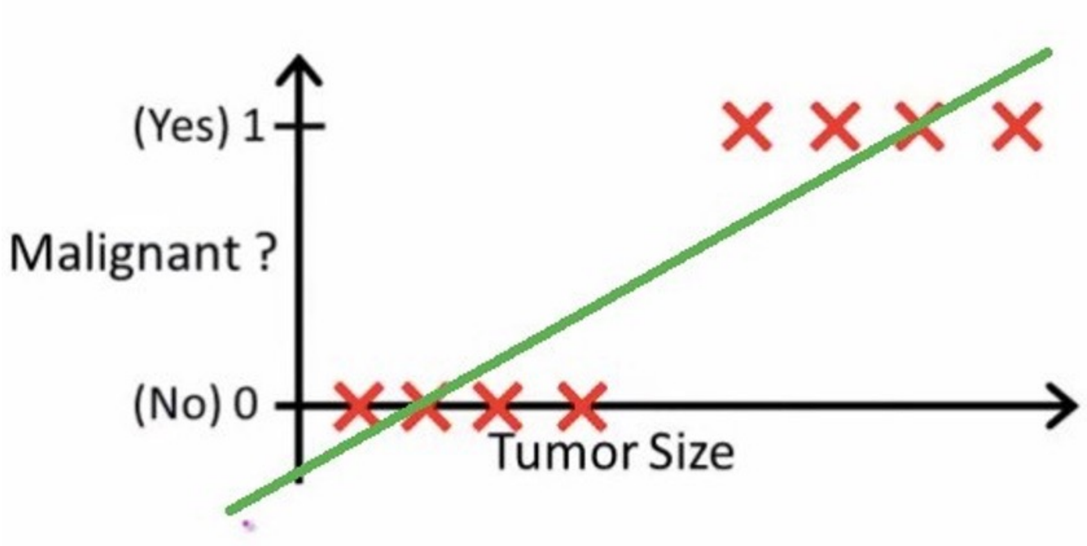
- **Question.** Can we use linear regression to solve classification tasks?

  - Answer. Yes



Linear regression, for classification?

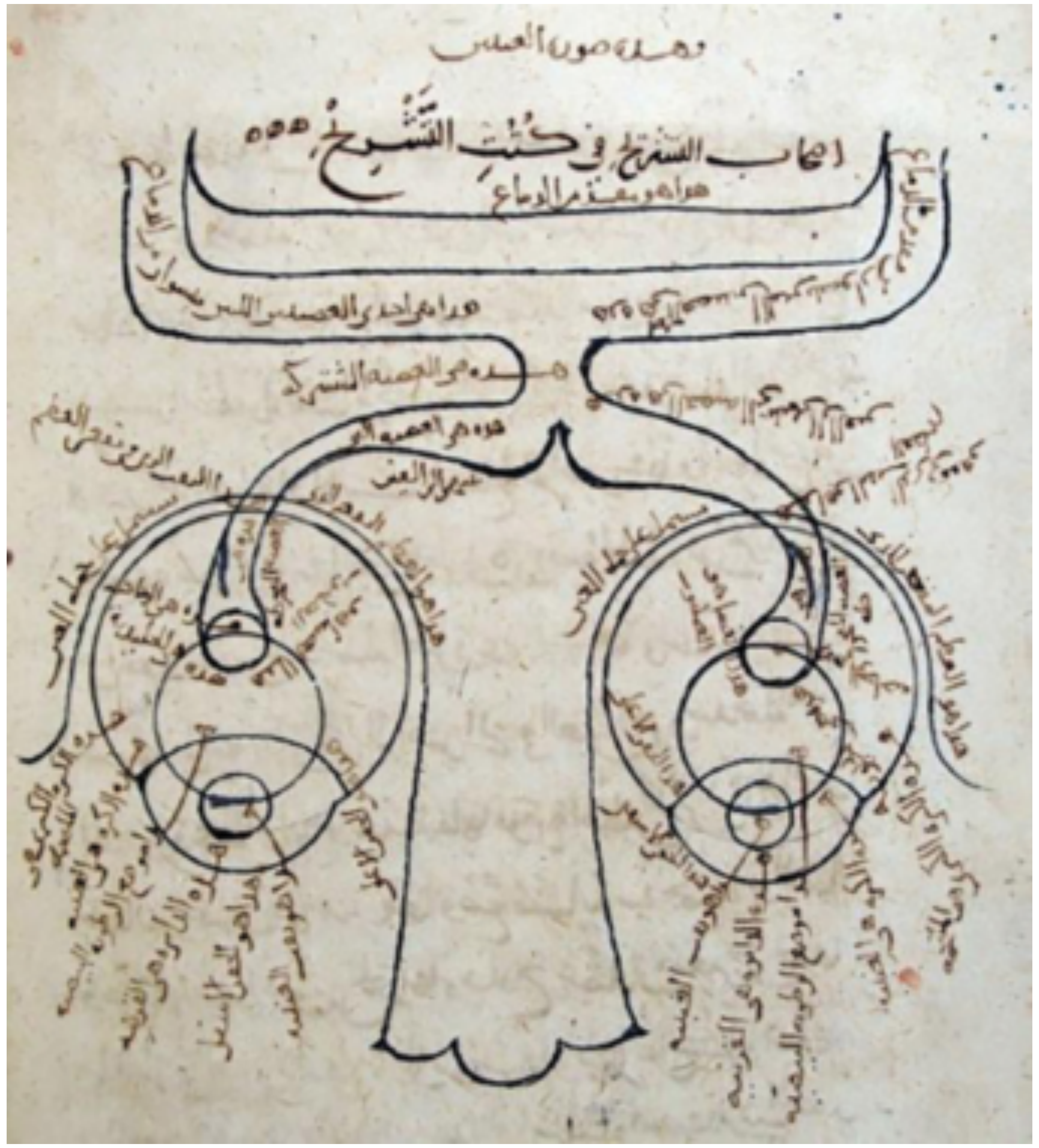
- **Question.** Can we use linear regression to solve classification tasks?
  - Answer. Yes
- **However...** this is a bad choice
  - Very sensitive to "outliers"
    - Consider an extremely large but benign tumor
  - Thus we want better tools

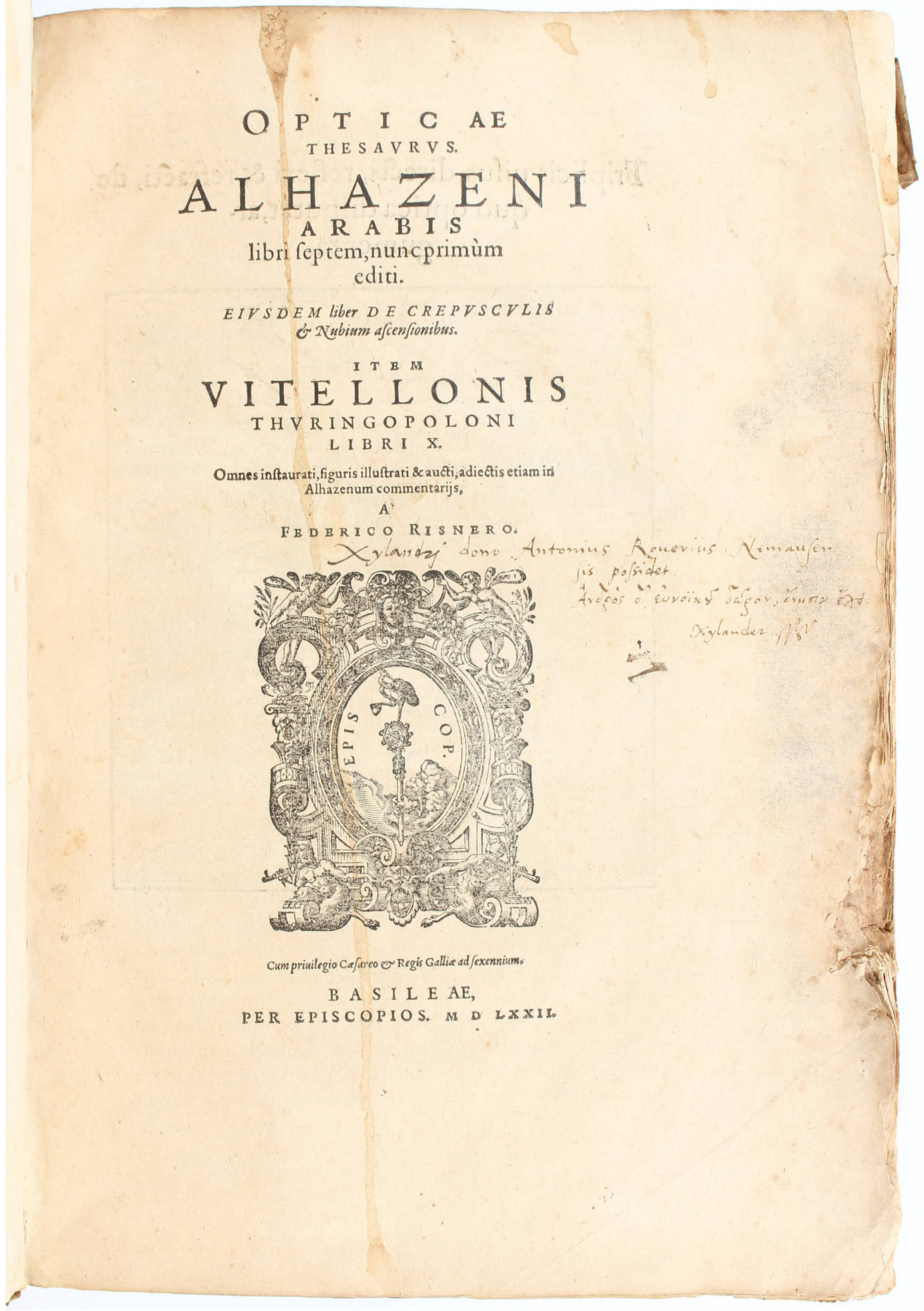


Nearest Neighbors

Historical bits

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





OPTICAE THESAVRVS.

ALHAZENI ARABIS libri ſeptem, nunc primùm editi.

EIVSDEM liber DE CREPVSCVLIS & Nubium aſcenſionibus.

ITEM VITELLONIS THVRINGOPOLONI LIBRI X.

Omnes inſtaurati, figuris illuſtrati & aucti, adiectis etiam in Alhazenum commentarijs,

A FEDERICO RISNERO.

Cum priuilegio Cæſareo & Regis Galliæ ad ſexennium.

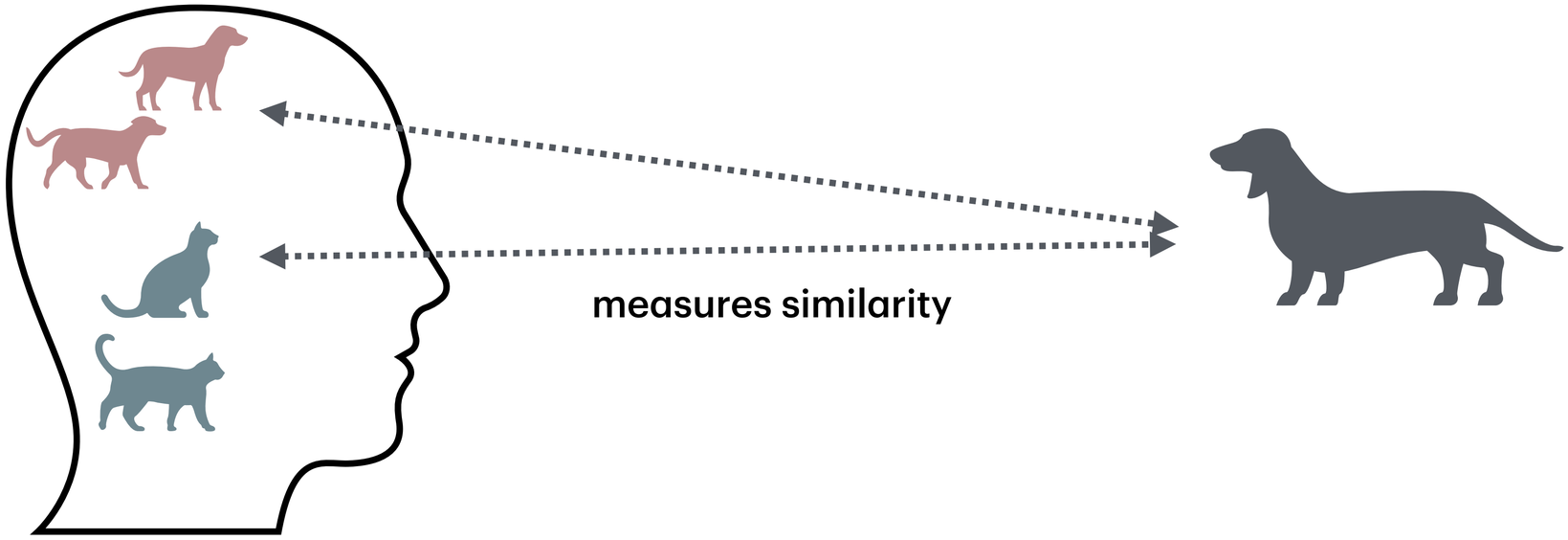
BASILEAE, PER EPISCOPIOS. M D LXXII.

Historical bits

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

- 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.”



K-nearest neighbors

- Suppose that we have a labeled dataset $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$
  - **Training.**
  - **Testing.**

K-nearest neighbors

- Suppose that we have a labeled dataset $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$

  - **Training.** There is no training!

    - Instead, we simply store the training data in an indexable form.

  - **Testing.**

K-nearest neighbors

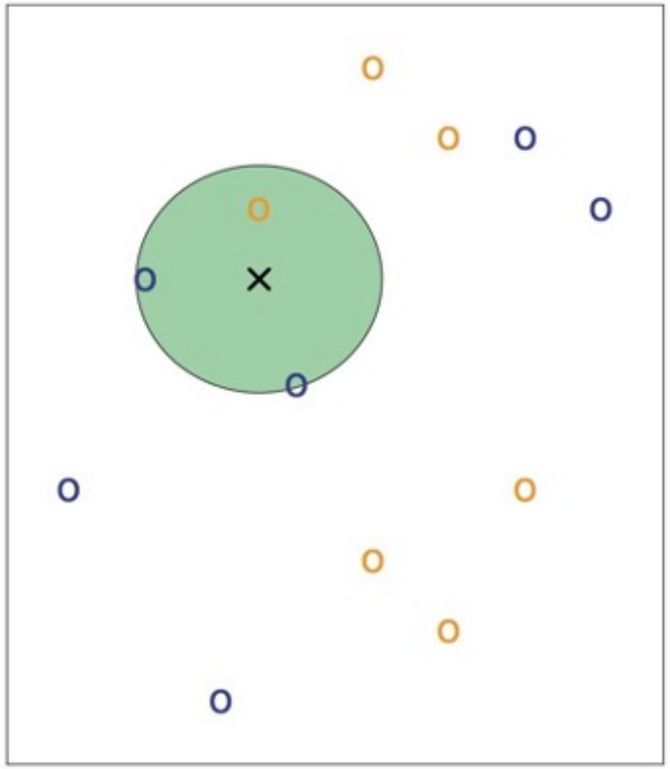
- Suppose that we have a labeled dataset $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$
  - **Training.** There is no training!
    - Instead, we simply store the training data in an indexable form.
  - **Testing.** Whenever a new sample $\mathbf{x}^{(\text{new})}$ comes in:
    - Find k samples $\mathbf{x}_{(1)}, \ldots, \mathbf{x}_{(k)} \in D$ with the **highest similarity**, e.g., have small distance

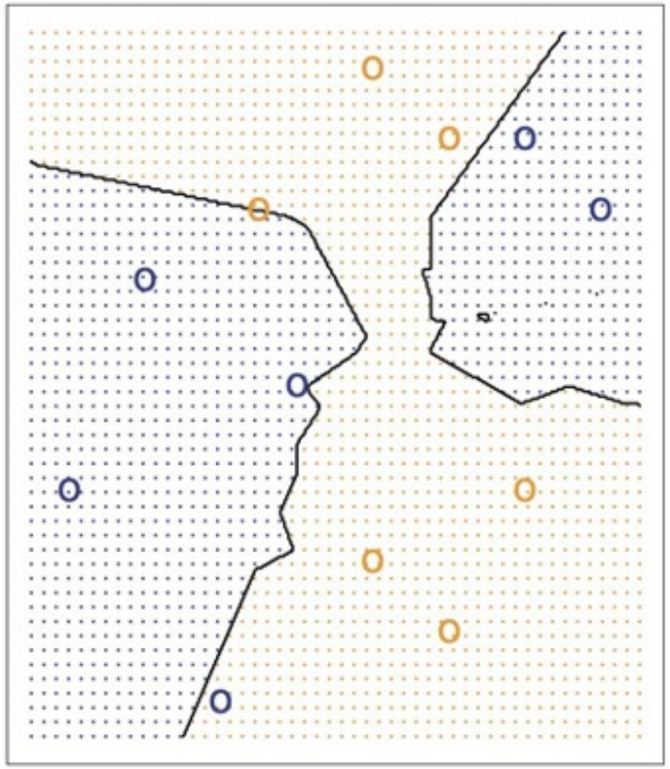
$$\|\mathbf{x}^{(\text{new})} - \mathbf{x}_{(i)}\|$$

    - Predict with the **majority vote**
      - Note. One can also predict real-valued y by (weighted) averaging

K-nearest neighbors

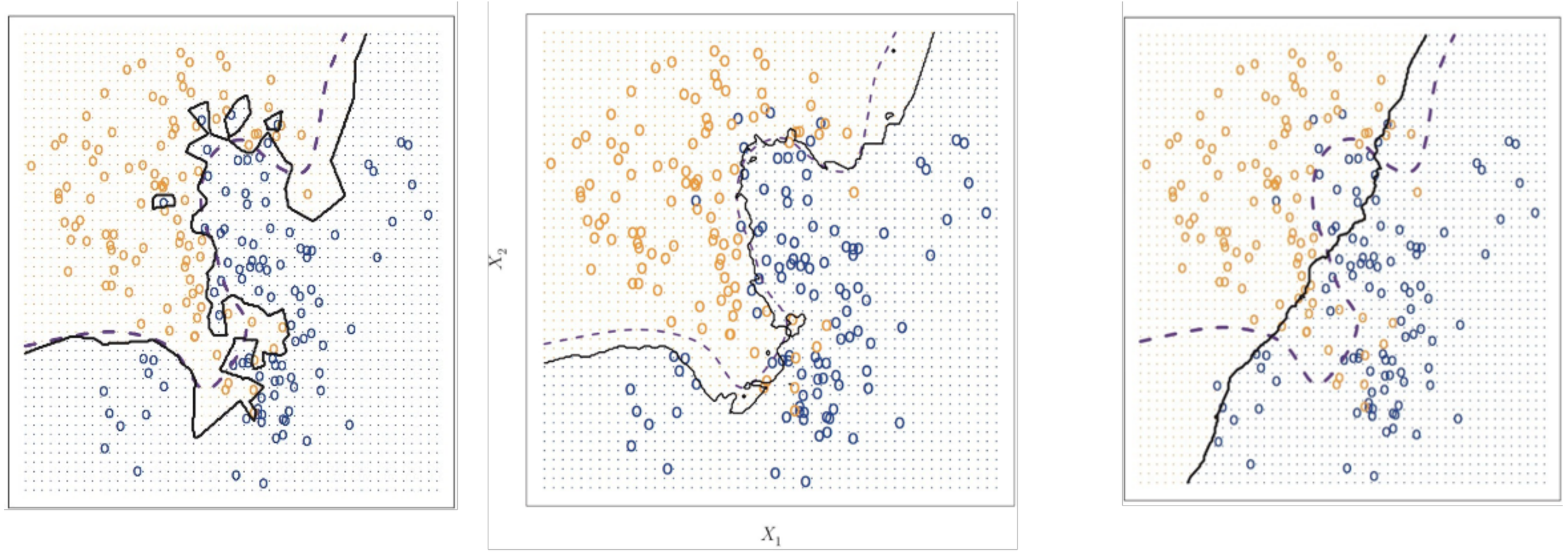
- The resulting predictor is **nonlinear** and **nonparametric** (i.e., not have finite-dimensional params)
  - nonparametric: using flexible number of or infinite-dimensional parameters
    <=> parametric: finite-dimensional parameters
    - Example. k-NN, Trees & Forests
  - Example. K-NN with $k = 3$





Selecting k

- Here, the neighbor set size k has a **big impact** on the model prediction
  - Small k = more flexibility / Larger k = smoother decision boundary



Selecting k

- Here, the neighbor set size k has a **big impact** on the model prediction
  - Small k = more flexibility / Larger k = smoother decision boundary
  - The k is often tuned manually
    - such tunable components are often called **hyperparameters**

Selecting k

- Here, the neighbor set size k has a **big impact** on the model prediction
  - Small k = more flexibility / Larger k = smoother decision boundary
  - The k is often tuned manually
    - such tunable components are often called **hyperparameters**
    - **A basic tuning procedure**
      - Run k-NN on the training data D with different k, to get $f_{k_1}, f_{k_2}, \ldots$
      - Evaluate their performance on validation data, and choose the best $\hat{k}$
      - Measure the test performance with $f_{\hat{k}}$

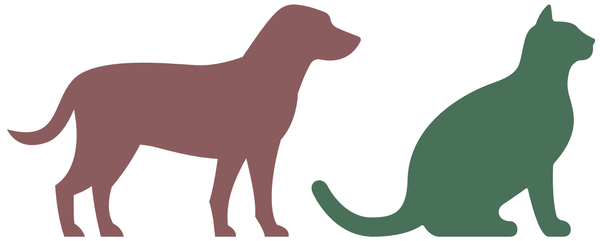
Considerations

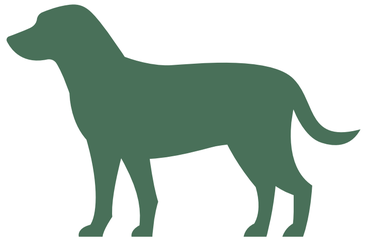
- **Computation.** K-NN is difficult to scale up to large datasets
  - Pros. No training cost
  - Cons. For testing, we need to conduct n comparisons
    - ? How can we make this dependency sublinear?

Considerations

- **Computation.** K-NN is difficult to scale up to large datasets
  - Pros. No training cost
  - Cons. For testing, we need to conduct n comparisons
    - ? How can we make this dependency sublinear?

- **Limitations.** The success depends critically on what similarity metric we use
  - This similarity should represent some knowledge (from human expert, or maybe data)





Later...

- You will find that neural nets provide a way to handle this difficulty
  - Use some training compute to make the comparison simpler (per-class prototypes)
  - Use the similarity metric trained from the dataset

Naïve Bayes

Problem setup

- Suppose that we have a labeled dataset $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$
  - Drawn independently from some P_{XY}
  - $\mathbf{x}^{(i)} \in \mathbb{R}^d$, $y^{(i)} \in \{0,1\}$

Problem setup

- Suppose that we have a labeled dataset $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$
  - Drawn independently from some P_{XY}
  - $\mathbf{x}^{(i)} \in \mathbb{R}^d$, $y^{(i)} \in \{0,1\}$
- **Assumption.** Entries of each $\mathbf{x}$ are conditionally independent given y

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

  - Note. Wrong for images (thus naïve), but can be true for tabular data
  - From now on, we let $d = 1$, WLOG

Bayesian approach

- Based on some human expert knowledge, we manually design two things.
  - **Likelihood models.** $p(x \mid y)$
  - **Priors.** $p(y)$

Bayesian approach

- Based on some human expert knowledge, we manually design two things
  - **Likelihood models.** $p(x \mid y)$
  - **Priors.** $p(y)$
- Each of these have some parameters to tune using the data
  - Example. Gaussian likelihood has two parameters $\mu, \sigma \in \mathbb{R}$, for each y

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

Bayesian approach

- Based on some human expert knowledge, we manually design two things
  - **Likelihood models.** $p(x \mid y)$
  - **Priors.** $p(y)$
- Each of these have some parameters to tune using the data
  - Example. Gaussian likelihood has two parameters $\mu, \sigma \in \mathbb{R}$, for each y

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

  - Example. Bernoulli prior has one parameter $p(y = 1)$

Bayesian predictor

- **Predictor.** After fitting the $p(y)$ and $p(\mathbf{x} \mid y)$ with data, we construct the MAP estimator
  (Maximum a Posteriori)

  - Choose the y with the maximum posterior probability $p(y \mid \mathbf{x})$

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

Bayesian predictor

- **Predictor.** After fitting the $p(y)$ and $p(\mathbf{x} \mid y)$ with data, we construct the MAP estimator (Maximum a Posteriori)
  - Choose the y with the maximum posterior probability $p(y \mid \mathbf{x})$

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

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

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

- ? Is MAP the only choice?

Bayesian training

- **Hypothesis space.** Constructed by selecting parameters for $p_{\theta_l}(\mathbf{x} \mid y)$ and $p_{\theta_p}(y)$

  - Example. Gaussian likelihood —> $\theta_l = (\mu_0, \mu_1, \sigma_0, \sigma_1) \in \mathbb{R}^4$

    Bernoulli prior —> $\theta_p \in [0,1]$

Bayesian training

- **Hypothesis space.** Constructed by selecting parameters for $p_{\theta_l}(\mathbf{x} \mid y)$ and $p_{\theta_p}(y)$

  - Example. Gaussian likelihood —> $\theta_l = (\mu_0, \mu_1, \sigma_0, \sigma_1) \in \mathbb{R}^4$

    Bernoulli prior —> $\theta_p \in [0,1]$

- **Training.** To fit the parameters, we maximize the joint probability of the training data

$$\max_{\theta} p_\theta(\mathbf{x}_1, \ldots, \mathbf{x}_n, y_1, \ldots, y_n) \quad = \quad \max_{\theta_\ell, \theta_p} \prod_{i=1}^{n} p_{\theta_\ell}(\mathbf{x}_i \mid y_i) p_{\theta_p}(y_i)$$

Bayesian training

$$\max_{\theta} p_\theta(\mathbf{x}_1, \ldots, \mathbf{x}_n, y_1, \ldots, y_n) \quad = \quad \max_{\theta_\ell, \theta_p} \prod_{i=1}^{n} p_{\theta_\ell}(\mathbf{x}_i \,|\, y_i) p_{\theta_p}(y_i)$$

- This is equivalent to performing ERM

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

So-called ***negative log-likelihood (NLL) loss***

Bayesian training

$$\max_{\theta} p_{\theta}(\mathbf{x}_1, \ldots, \mathbf{x}_n, y_1, \ldots, y_n) \quad = \quad \max_{\theta_\ell, \theta_p} \prod_{i=1}^{n} p_{\theta_\ell}(\mathbf{x}_i \,|\, y_i) p_{\theta_p}(y_i)$$

- This is equivalent to performing ERM

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

- Solving this is equivalent to conducting **two optimizations separately**:

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

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

← such θ_ℓ is the ***maximum likelihood estimate (MLE)***

Bayesian training

- For popular choices of likelihoods & priors, these ERM solutions are quite simple:
  - Example. Gaussian Likelihood
    - Use class-wise sample mean and classwise sample variance for $\mu_0, \mu_1, \sigma_0^2, \sigma_1^2$
  - Example. Bernoulli Prior
    - Simply use the frequency $$p = \frac{\#1\text{s in dataset}}{n}$$

Considerations

- **Computation.** Quite simple for popular choices of $p(\mathbf{x} \mid y)$ and $p(y)$
  - Training. Simple, by explicit formula
  - Test. Simply compute $p(y \mid \mathbf{x})$
  - These can be very messy for atypical choices, or any dependency structures!

Considerations

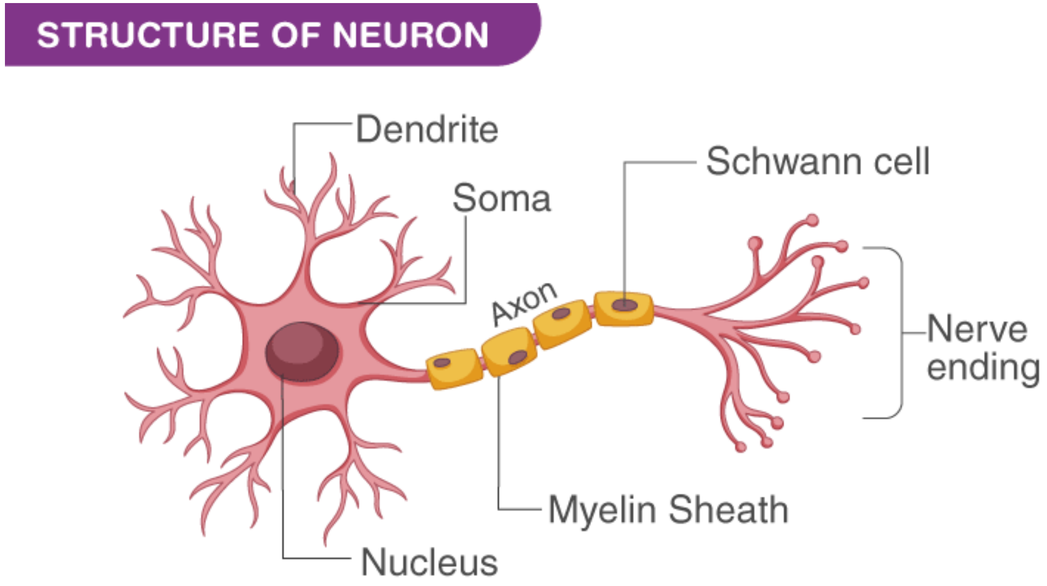
- **Computation.** Quite simple for popular choices of $p(\mathbf{x} \mid y)$ and $p(y)$
  - Training. Simple, by explicit formula
  - Test. Simply compute $p(y \mid \mathbf{x})$
  - These can be very messy for atypical choices, or any dependency structures!

- **Limitation.** Requires a good prior and likelihood to be designed
  - We expect very complicated $p(\mathbf{x} \mid y)$
  - Wish to replace human knowledge with some data-driven mechanisms...

Perceptrons

Perceptrons

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



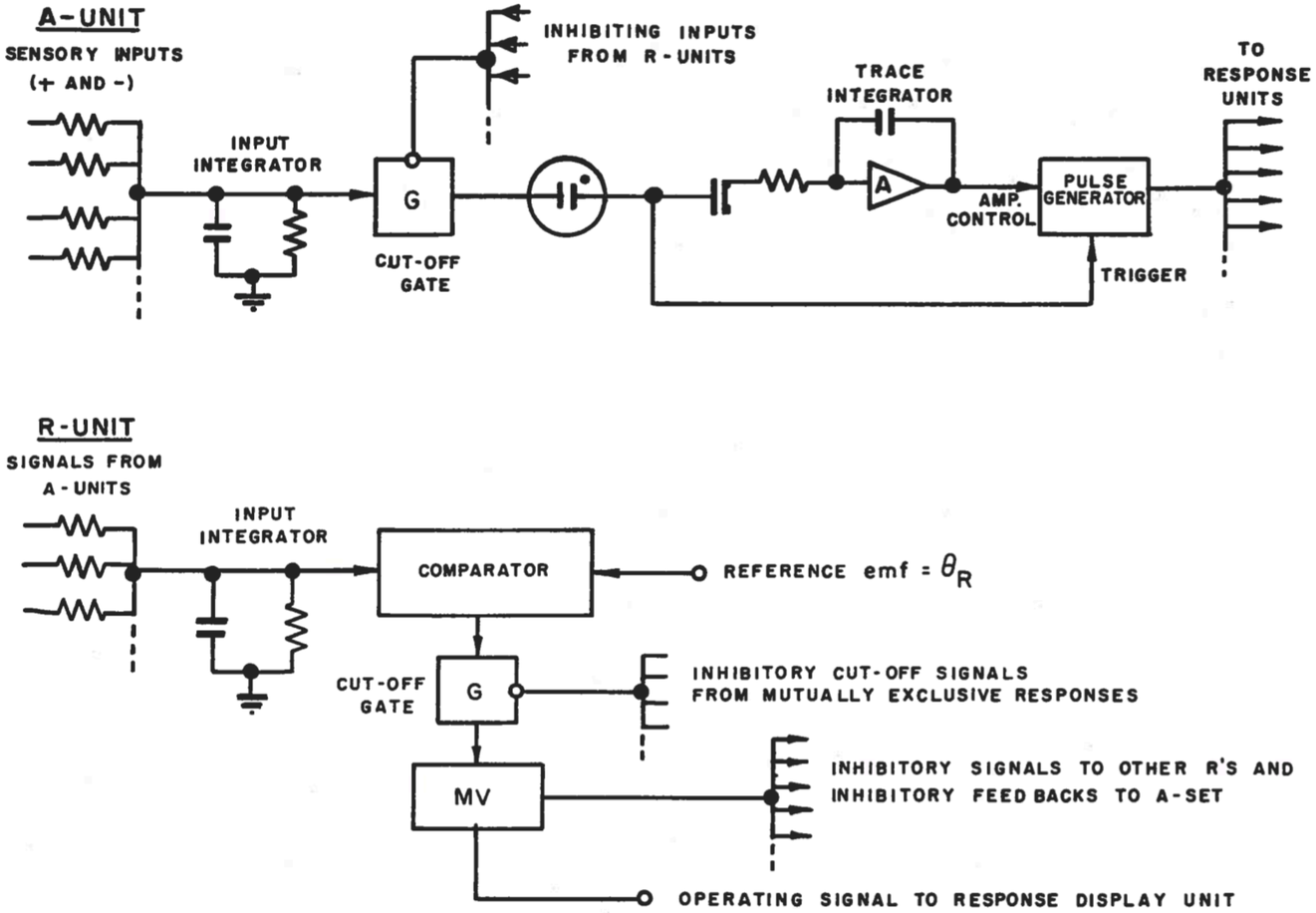
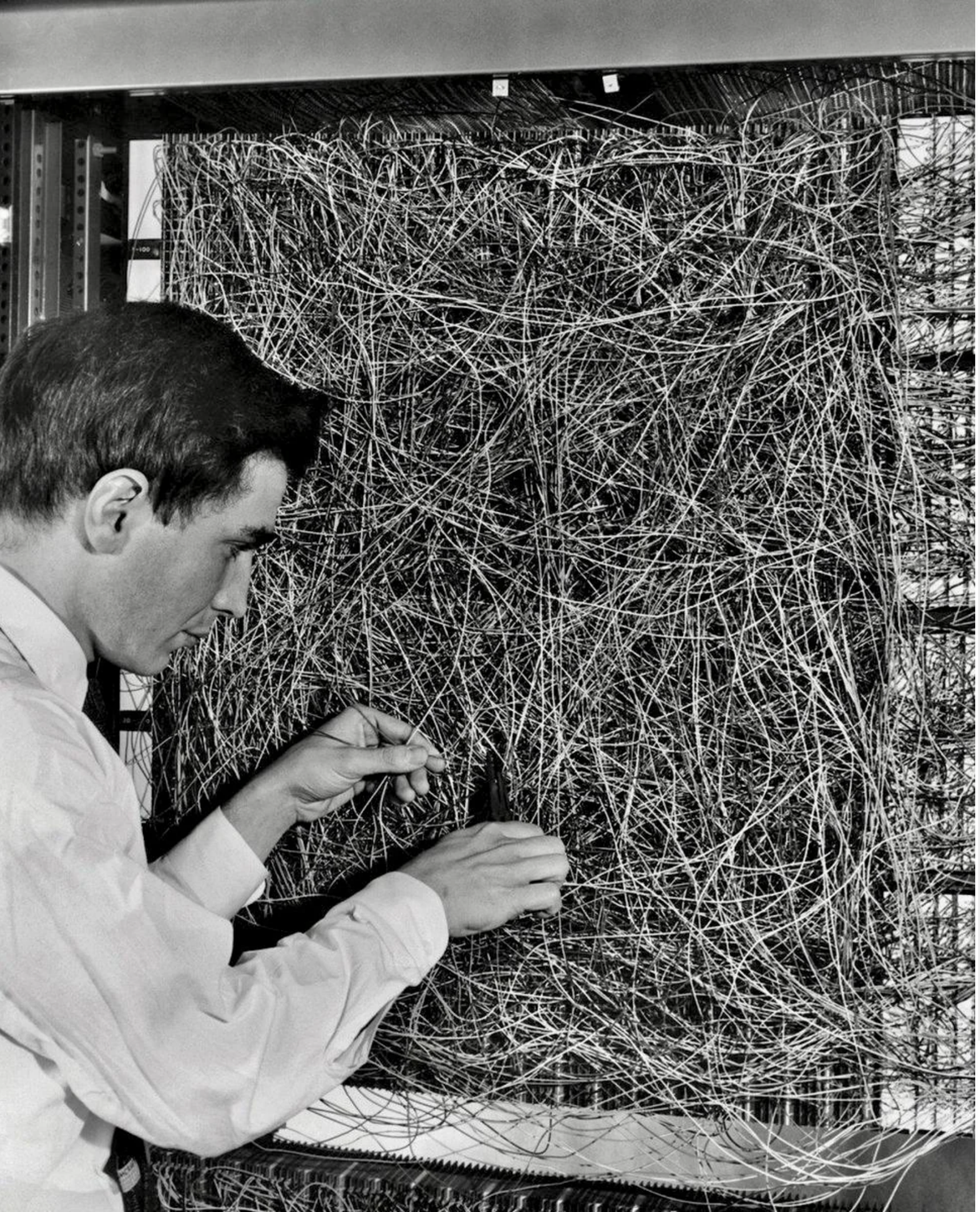
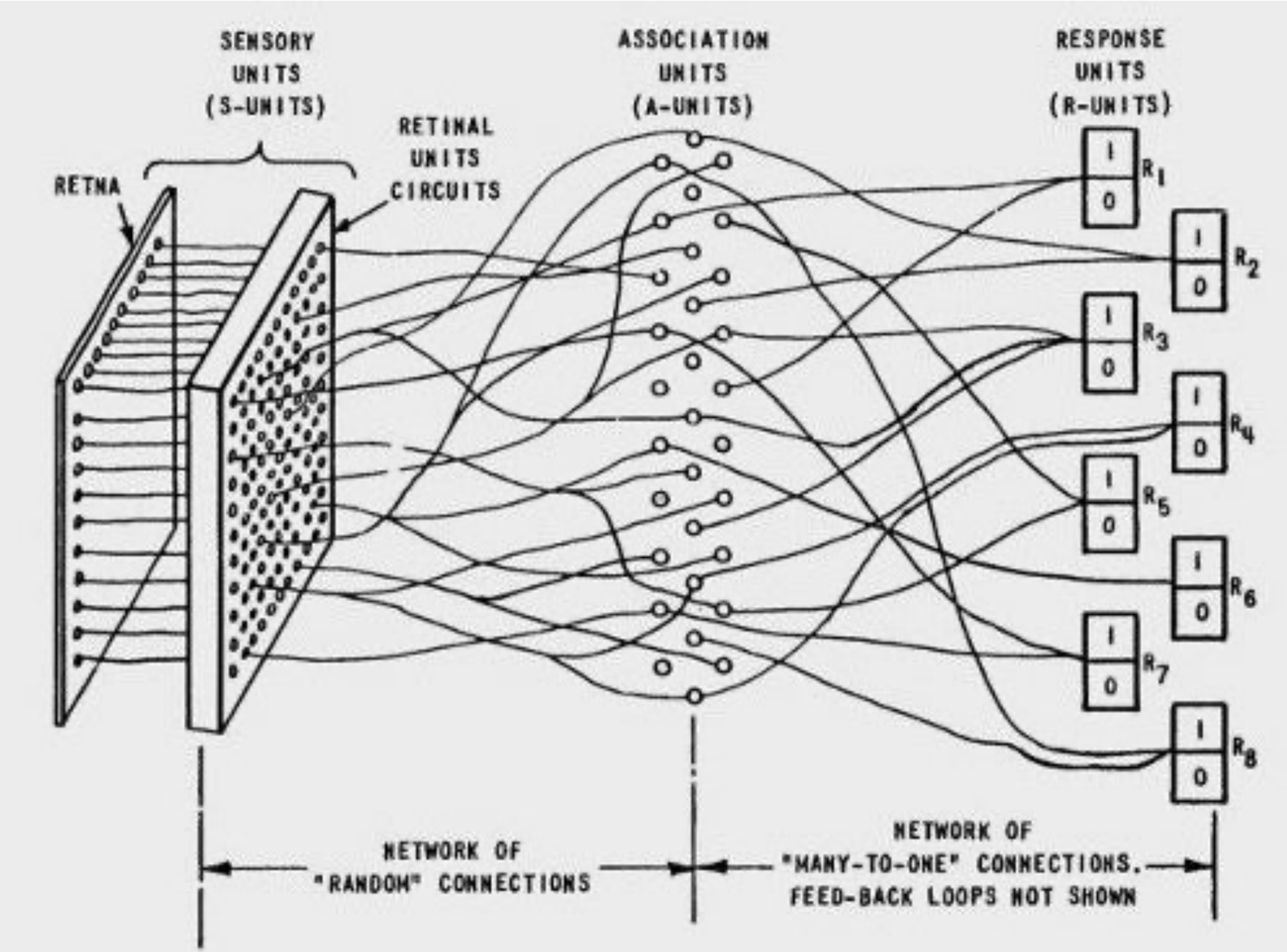


FIGURE 5
DESIGN OF TYPICAL UNITS

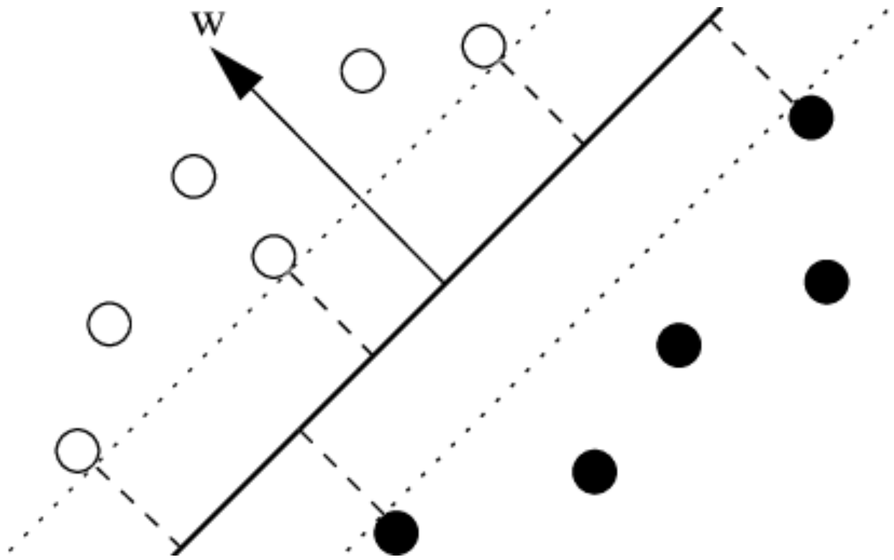



SENSORY
UNITS
(S-UNITS)
RETINAL
UNITS
CIRCUITS
ASSOCIATION
UNITS
(A-UNITS)
RESPONSE
UNITS
(R-UNITS)
RETNA
I
O
R1
R2
R3
R4
R5
R6
R7
R8
NETWORK OF
"RANDOM" CONNECTIONS
NETWORK OF
"MANY-TO-ONE" CONNECTIONS.
FEED-BACK LOOPS NOT SHOWN

Perceptrons

- Mathematically, quite simple
  - Again, we are given some dataset $D = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$
    - $\mathbf{x} \in \mathbb{R}^d$ and $y \in \{0,1\}$

Perceptrons

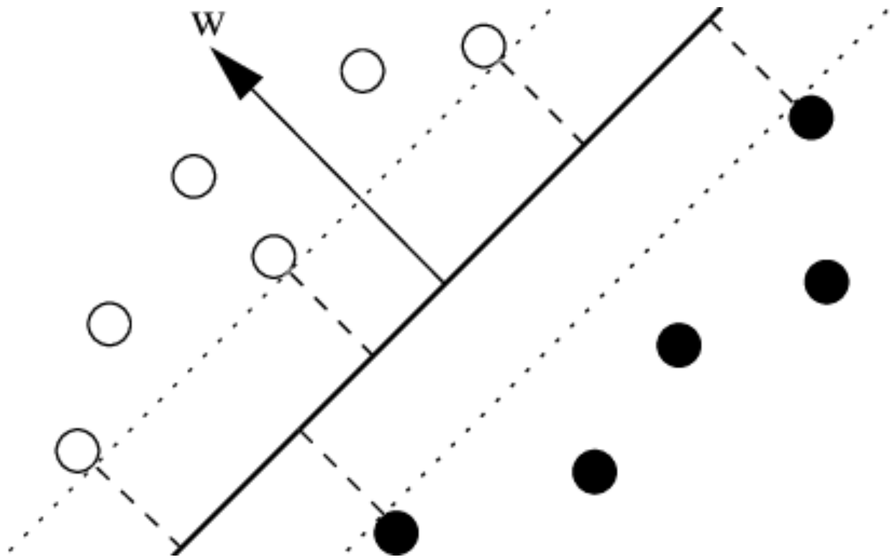


- Mathematically, quite simple
  - Again, we are given some dataset $D = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$
    - $\mathbf{x} \in \mathbb{R}^d$ and $y \in \{0,1\}$
  - **Predictors.** Use the sign of linear models

$$\left\{ f_\theta(\cdot) \,\middle|\, f_\theta(\mathbf{x}) = \mathbf{1}\left[\theta_1{}^\top \mathbf{x} + \theta_0 > 0\right] \right\} \quad = \left\{ f_\theta(\cdot) \,\middle|\, f_\theta(\mathbf{x}) = \mathbf{1}\left[\theta^\top \tilde{\mathbf{x}} > 0\right] \right\}$$

(indicator function; 1 if the bracket is true, 0 if false)

Perceptrons



- Mathematically, quite simple
  - Again, we are given some dataset $D = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$
    - $\mathbf{x} \in \mathbb{R}^d$ and $y \in \{0,1\}$
  - **Predictors.** Use the sign of linear models

$$\left\{ f_\theta(\cdot) \;\middle|\; f_\theta(\mathbf{x}) = \mathbf{1}\left[\theta_1{}^\top \mathbf{x} + \theta_0 > 0\right] \right\} \quad = \left\{ f_\theta(\cdot) \;\middle|\; f_\theta(\mathbf{x}) = \mathbf{1}\left[\theta^\top \tilde{\mathbf{x}} > 0\right] \right\}$$

  - **Training.** Difficult to find an explicit solution.
    - Want to do something like gradient descent... but taking derivative w.r.t. $\mathbf{1}[\,\cdot\,]$ is nasty.

Training Perceptrons

- **Loss.** To optimize, we use the loss

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

  - That is, we have loss $|\theta^\top \mathbf{x}|$ when wrong (penalizing confidence)
    0 when correct

Training Perceptrons

- **Loss.** To optimize, we use the loss

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

  - That is, we have loss $|\theta^\top \mathbf{x}|$ when wrong (penalizing confidence)
    0 when correct

  - Note. Using such surrogate loss is quite common in ML
    (i.e., loss functions different from the performance criterion)

Training Perceptrons

- **Loss.** To optimize, we use the loss

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

  - That is, we have loss $|\theta^\top \mathbf{x}|$ when wrong (penalizing confidence)
    0 when correct

  - Note. Using such surrogate loss is quite common in ML
    (i.e., loss functions different from the performance criterion)

? If $\theta = 0$, the loss is zero but our classifier is bad!
Can we still train a good model?

Training Perceptrons

- **Optimization.** The original perceptron paper assumes that the data comes one-by-one.
    - Called online learning

Training Perceptrons

- **Optimization.** The original perceptron paper assumes that the data comes one-by-one.
  - Called online learning
  - The gradient is

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

  - If wrong for a sample with $y = 1$ $\quad\quad$ $\theta^{(i+1)} = \theta^{(i)} + \eta \cdot \mathbf{x}$
  - If wrong for a sample with $y = 0$ $\quad\quad$ $\theta^{(i+1)} = \theta^{(i)} - \eta \cdot \mathbf{x}$
  - If correct, no change

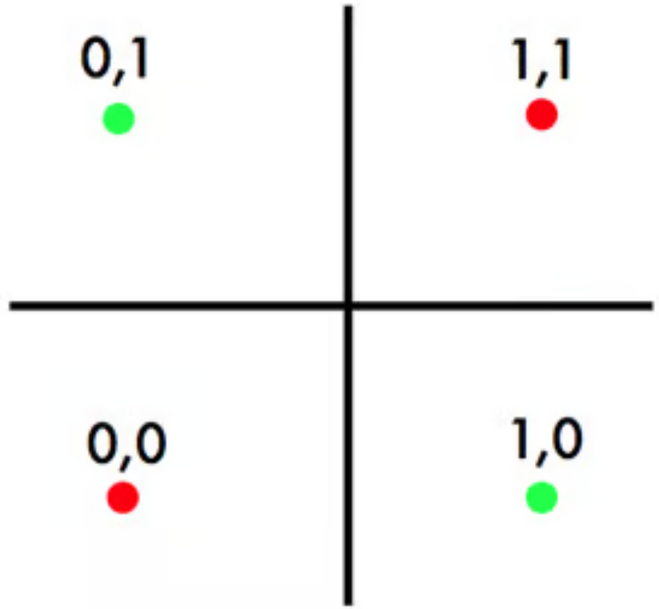
Remarks

- **Computation.** Quite easy
    - Training. Provably converges whenever the data is separable, luckily
    - Test. Simply do the dot product

Remarks

- **Computation.** Quite easy
  - Training. Provably converges whenever the data is separable, luckily
  - Test. Simply do the dot product

- **Limitations.** Cannot achieve low training loss on not linearly separable data



Logistic Regression

Logistic regression

- Solve the classification, just like linear regression
  - **Idea.** Do not predict the label directly, but predict the log likelihood ratio (note the direction)

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

  - **Question.** Why don't we simply predict $p(y=1\,|\,\mathbf{x})$?

Logistic regression

- Solve the classification, just like linear regression
  - **Idea.** Do not predict the label directly, but predict the log likelihood ratio (note the direction)

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

    - **Question.** Why don't we simply predict $p(y=1\,|\,\mathbf{x})$?
      - Answer. To utilize the full range; $p(y=1\,|\,\mathbf{x}) \in [0,1]$, but $\theta^{\mathsf{T}}\tilde{\mathbf{x}} \in (-\infty, +\infty)$

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

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

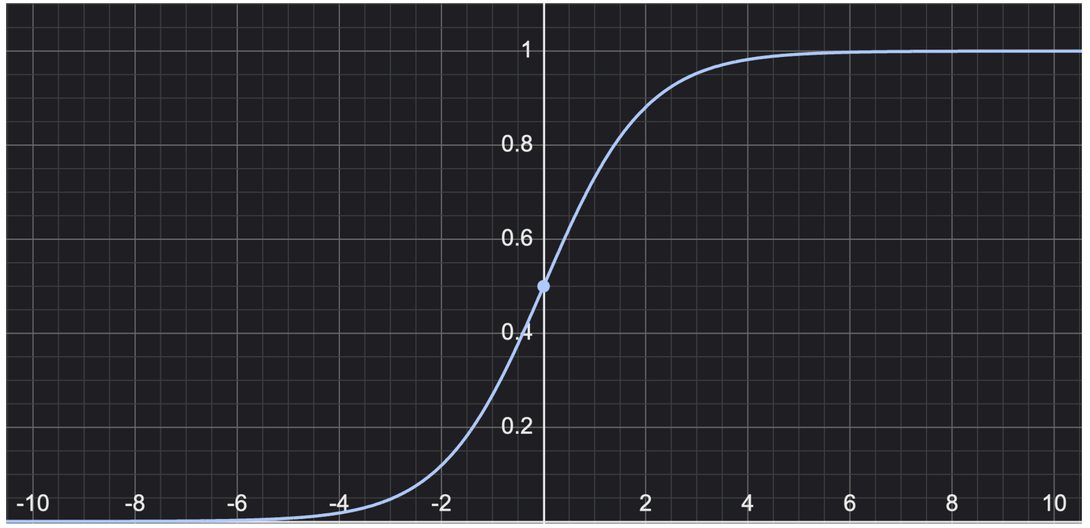
Logistic regression

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

- 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 called the logistic function



Training

- **Training.** Given the data $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, we 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, solve an ERM with:
  - Hypothesis space $\{f_\theta(\mathbf{x}) = \sigma(\theta^\top \tilde{\mathbf{x}})\}$
  - Loss is the cross-entropy $\ell(y, t) = \mathrm{CE}(\mathbf{1}_y, [t, 1-t]) = \log(t)^{-y} + \log(1-t)^{y-1}$

Training

- Equivalently again, solve an ERM with:
  - Hypothesis space $\{f_\theta(\mathbf{x}) = \sigma(\theta^\top \tilde{\mathbf{x}})\}$
  - Loss is the cross-entropy $\ell(y, t) = \mathrm{CE}(\mathbf{1}_y, [t, 1-t]) = \log(t)^{-y} + \log(1-t)^{y-1}$
- More tediously, minimize

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

  - Convex, but no general closed-form solution —> use gradient descent

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

Remarks

- **Computation.** Relatively easy
    - Training. Requires solving GD, but is convex
    - Testing. Dot product, and apply some threshold

Remarks

- **Computation.** Relatively easy
  - Training. Requires solving GD, but is convex
  - Testing. Dot product, and apply some threshold

- **Limitations.** Again, limited expressive power

Wrapping up

- Looked at very simple classification algorithms
    - Easy to train and use
    - Cannot capture big, complicated data (except k-NN)

- **Next class.** A bit more sophisticated version of linear classification models

Cheers