CSE 417T
Introduction to Machine Learning

Instructor: Chien-Ju (CJ) Ho

Logistics: Homework

- HW 0:
  - Due by **11:30am next Tuesday**
  - Submit via Gradescope
  - Only waitlisted students need to submit
  - No late days can be used
  - The rules on academic integrity apply

- HW 1: Will be announced next week
  - The questions in HW0 will appear in HW1 as well

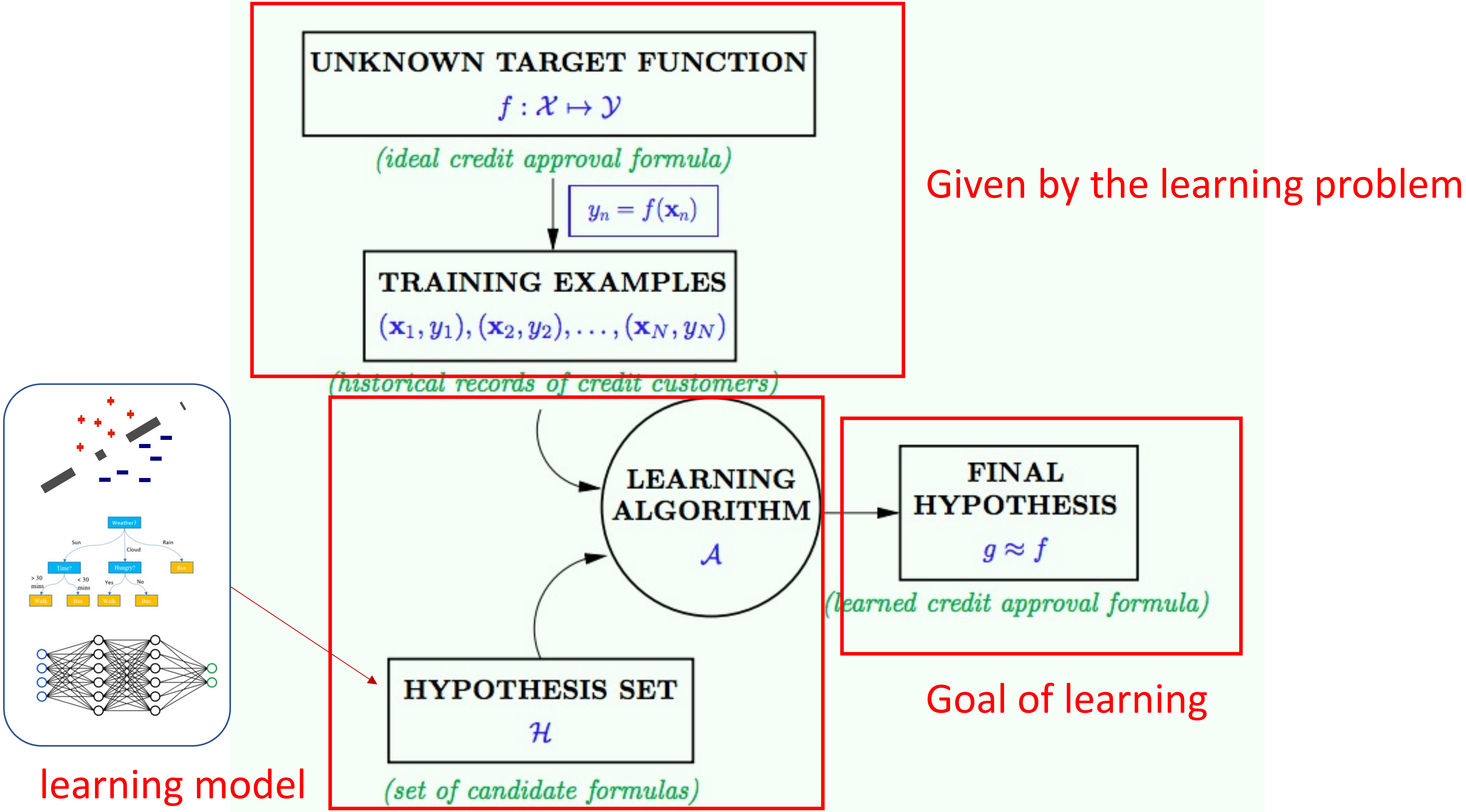
Logistics: Academic Integrity

- Discussion (conceptually) about course content and homework assignments is encouraged.

- How to make sure to not violate academic integrity?
- Rule of thumb:
  - You **must** write down the answers/codes entirely on your own.
  - Can’t look at the write-up / codes by others.

- Ask if you are not sure.

Recap

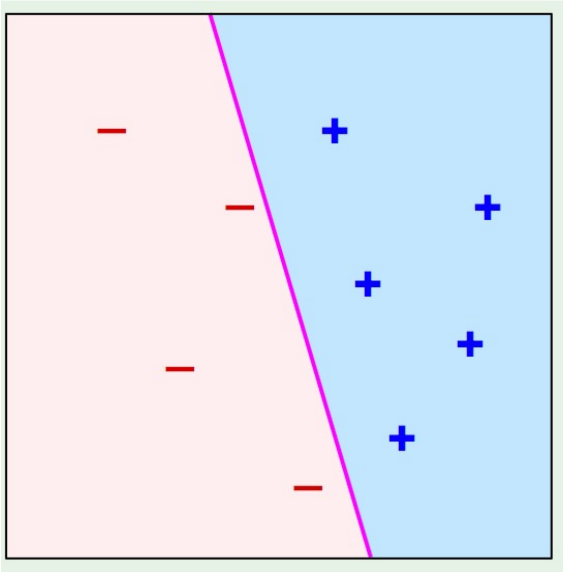

UNKNOWN TARGET FUNCTION
$f : \mathcal{X} \mapsto \mathcal{Y}$
(ideal credit approval formula)
$y_n = f(\mathbf{x}_n)$
TRAINING EXAMPLES
$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$
(historical records of credit customers)
Given by the learning problem
LEARNING ALGORITHM
$\mathcal{A}$
FINAL HYPOTHESIS
$g \approx f$
(learned credit approval formula)
Goal of learning
HYPOTHESIS SET
$\mathcal{H}$
(set of candidate formulas)
Weather?
Sun
Cloud
Rain
Time?
Hungry?
Bus
> 30 mins
< 30 mins
Yes
No
Walk
Bus
Walk
Bus
learning model

Linear Hypothesis Space (Perceptron)

- Input $\vec{x} = (x_1, x_2, \ldots, x_d)$
- Output $y \in \{-1, +1\}$

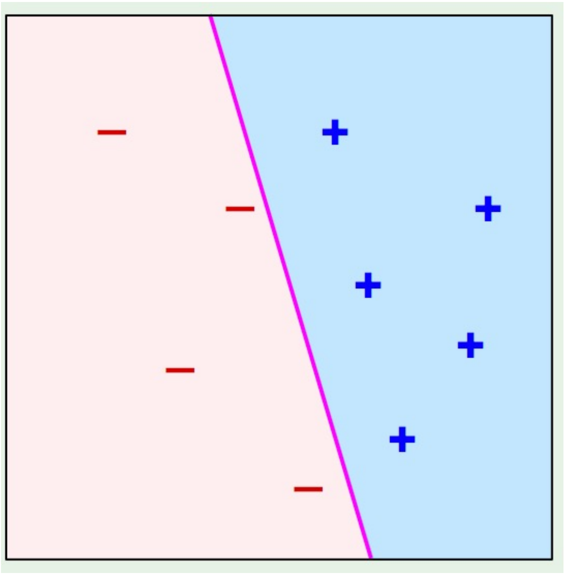
- A hypothesis h is a linear separator $\vec{w}^T\vec{x} = b$, characterized by
  - weight vector $\vec{w} = (w_1, \ldots w_d)$
  - threshold b

- $h(\vec{x}) = sign\left(\sum_{i=1}^{d} w_i x_i - b\right) = sign(\vec{w}^T\vec{x} - b)$
  - Predict $+1$ if $\vec{w}^T\vec{x} > b$
  - Predict -1 if $\vec{w}^T\vec{x} < b$



Linear Hypothesis Space (Perceptron)

- To simplify $h(\vec{x}) = sign(\vec{w}^T\vec{x} - b)$, define
  - $x_0 = 1$
  - $w_0 = -b$



- And we implicitly let
  - $\vec{x} = (x_0, x_1, \dots, x_d)$
  - $\vec{w} = (w_0, w_1, \dots, w_d)$

- A hypothesis can then be written as
  - $h(\vec{x}) = sign(\vec{w}^T\vec{x})$
    - We will interchangeably use h and $\vec{w}$ to express a hypothesis in Perceptron

Perceptron Learning Algorithm (PLA)

- Given a dataset $D = \{(\vec{x}_1, y_1), \dots, (\vec{x}_N, y_N)\}$
- Assume the dataset is **linearly separable**
- Want to find a hypothesis that separates data in D

- Perceptron Learning Algorithm
  - Initialize $\vec{w}(0) = \vec{0}$
  - For $t = 0, \dots$
    - Find a misclassified data point $(\vec{x}(t), y(t))$ in D
      - That is, $\mathrm{sign}(\vec{w}(t)^T \vec{x}(t)) \neq y(t)$
    - If no such data point exists
      - Return $\vec{w}(t)$
    - Else
      - $\vec{w}(t+1) \leftarrow \vec{w}(t) + y(t)\vec{x}(t)$

Notation:
We use $\vec{w}(t)$ to denote the value of $\vec{w}$ at step t of the algorithm.

Similarly, we use $(\vec{x}(t), y(t))$ to denote the data point found at step t.

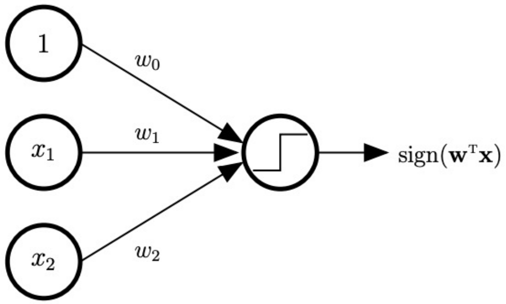
Perceptron Learning Algorithm (PLA)

- Theorem (informal):
  - If a dataset D is linearly separable, PLA find a linear separator that separates the data in D within a finite number of steps.

- You will prove the above theorem in HW0

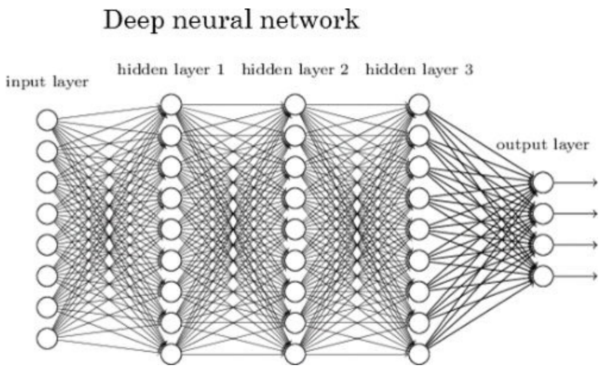
Perceptron

- Graphical Representation



Inspired by neurons:
The output signal is triggered when the weighted combination of the inputs is larger than some threshold

- Deep learning (neural network with many layers)



Common Notations in This Course

- Data point with augmented x_0: $\vec{x} = (x_0, \dots, x_d)$
  - We often use d to specify the dimensions of data points
  - We augment $x_0 = 1$ for each data point (Check Lecture 1 for the reasoning)

- Dataset: $D = \{(\vec{x}_1, y_1), \dots, (\vec{x}_N, y_N)\}$
  - We often use N to specify the number of data points in the dataset

- Hypothesis set H
  - We use $h \in H$ to specify an arbitrary hypothesis
  - We use $g \in H$ to specify the hypothesis output by the learning algorithm

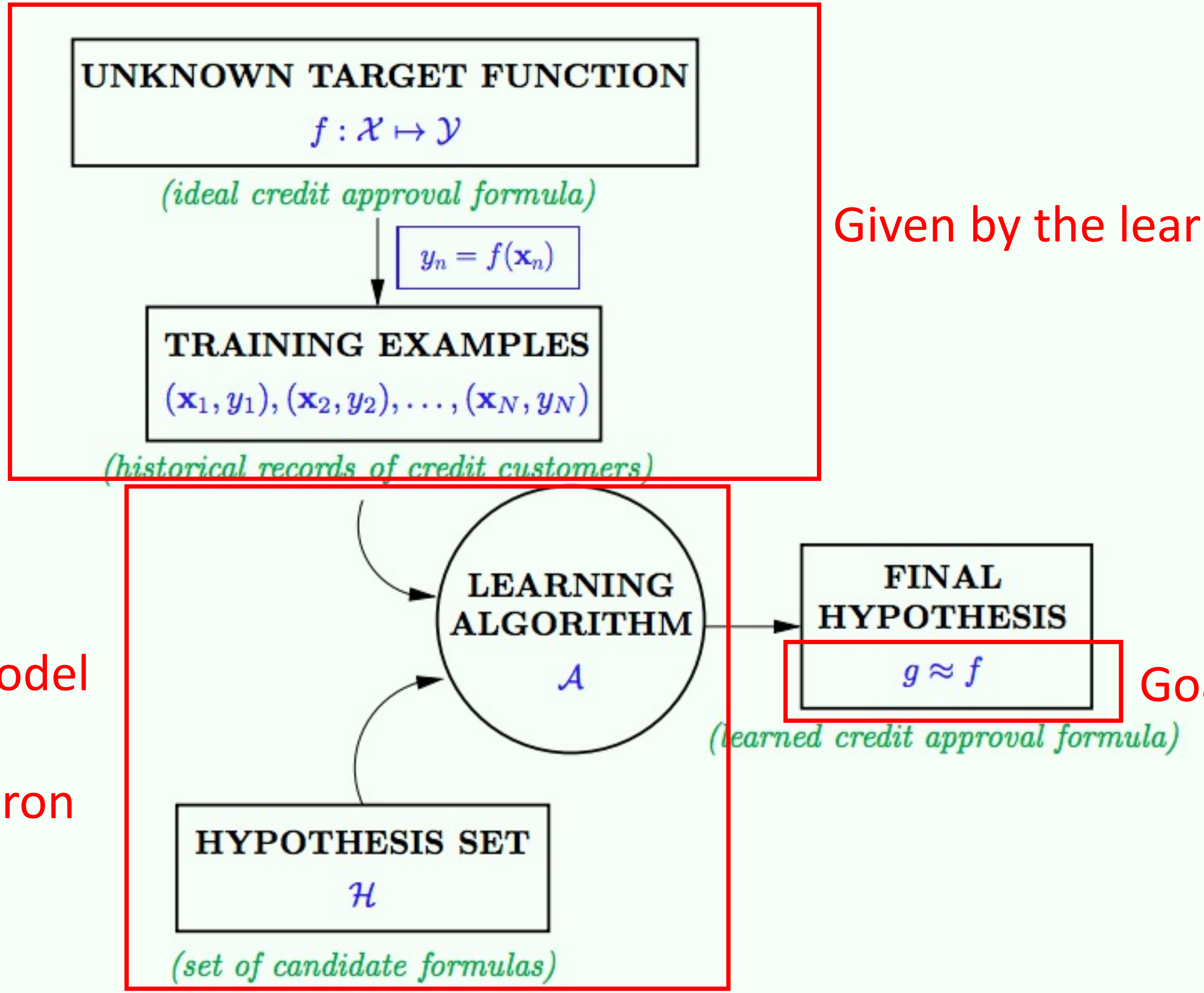
- Indicator variable:
  - $\mathbb{I}[\text{event}] = \begin{cases} 1 & \text{if event is true} \\ 0 & \text{if event is false} \end{cases}$

Example: $\mathbb{I}[h(\vec{x}) \neq f(\vec{x})] = \begin{cases} 1 & \text{if } h(\vec{x}) \neq f(\vec{x}) \\ 0 & \text{if } h(\vec{x}) = f(\vec{x}) \end{cases}$

Lecture Today

The notes are not intended to be comprehensive.

Let me know if you spot errors.



Given by the learning problem

Goal of learning?

learning model
(example:
H: Perceptron
A: PLA)

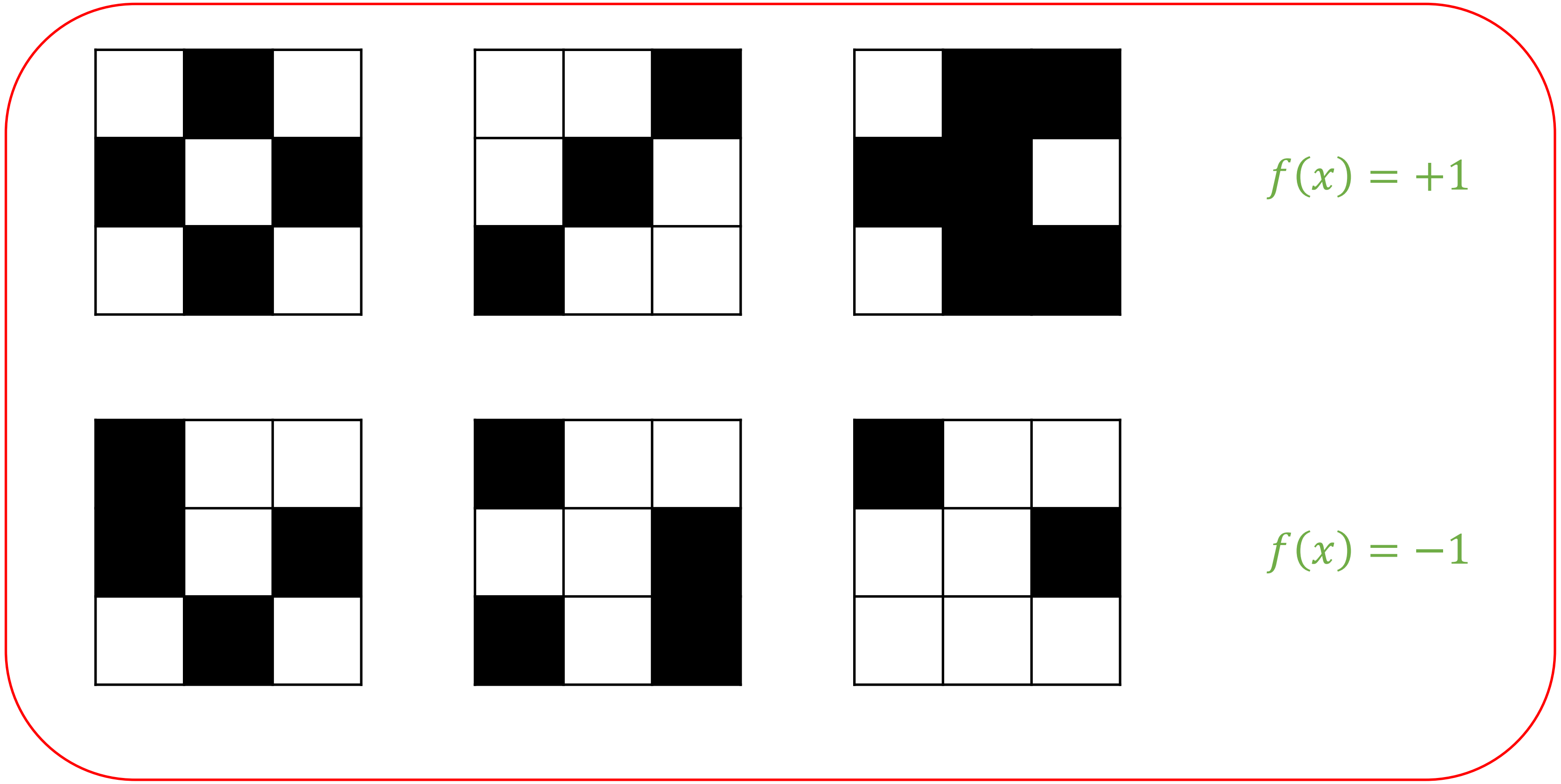
How Do We Formally Characterize the Goal?

- Goal of learning: find $g \approx f$
  - f: unknown target function
  - g: output of the learning algorithm
  - What do we mean by $g \approx f$?

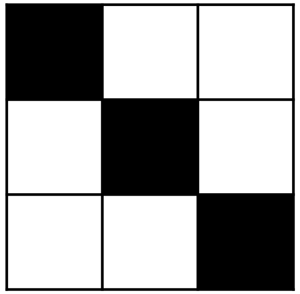
- Main idea: **Generalization**
  - Want g to make predictions similar to f for **unseen data points**

Focus of today's lecture:
- Feasibility of learning
- Can we achieve generalization?

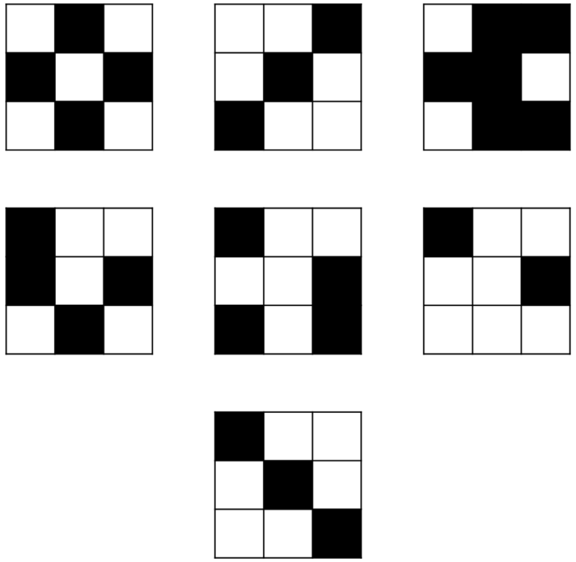
Training Dataset



Predict for unseen points
(Generalization)



$f(x) = ???$



$f(x) = +1$

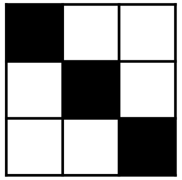
$f(x) = -1$

$f(x) = ???$

Hypothesis 1

$$h(x) = \begin{cases} +1 \text{ if symmetric} \\ -1 \text{ otherwise} \end{cases}$$

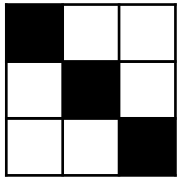
$$\downarrow$$

$h\Bigg($  $\Bigg) = +1$

Hypothesis 2

$$h(x) = \begin{cases} +1 \text{ if top left is white} \\ -1 \text{ otherwise} \end{cases}$$

$$\downarrow$$

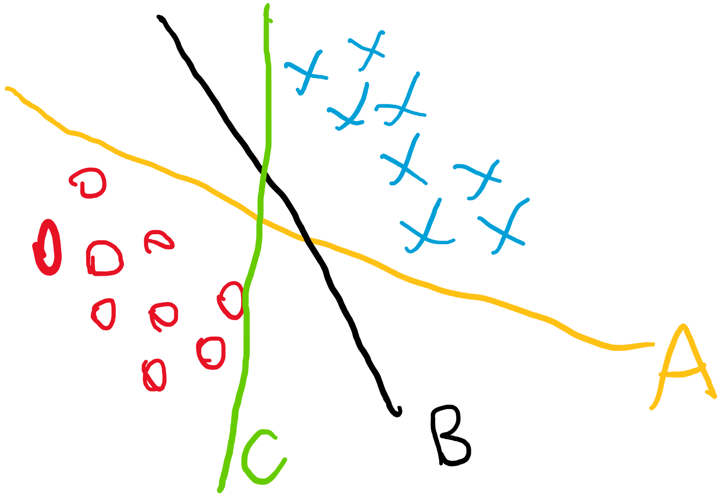
$h\Bigg($  $\Bigg) = -1$

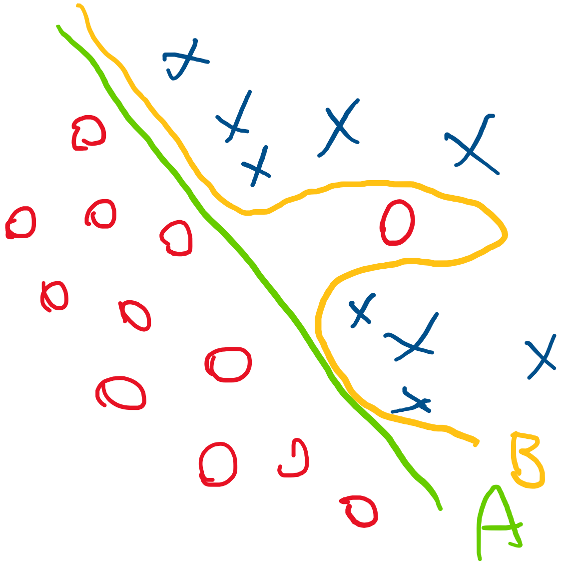
You can come up with many more hypothesis

Feasibility of Learning

- Is learning feasible (can we generalize the learning)?
  - Cannot know anything **for sure** about f outside the data without assumptions
  - We might need to give up the **"for sure"** and make additional assumptions

- Thought experiments: Which hypothesis would you choose? Why?

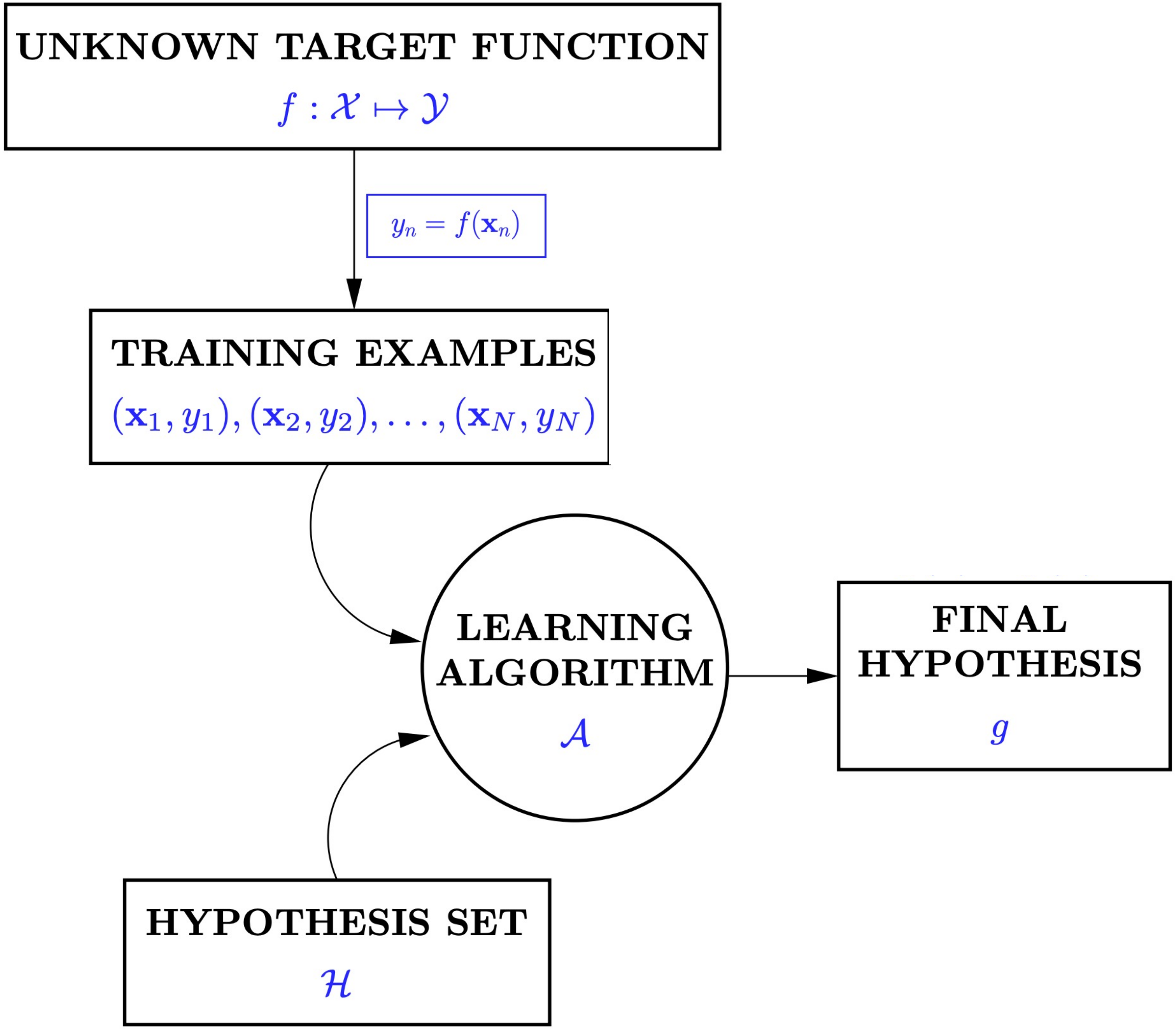


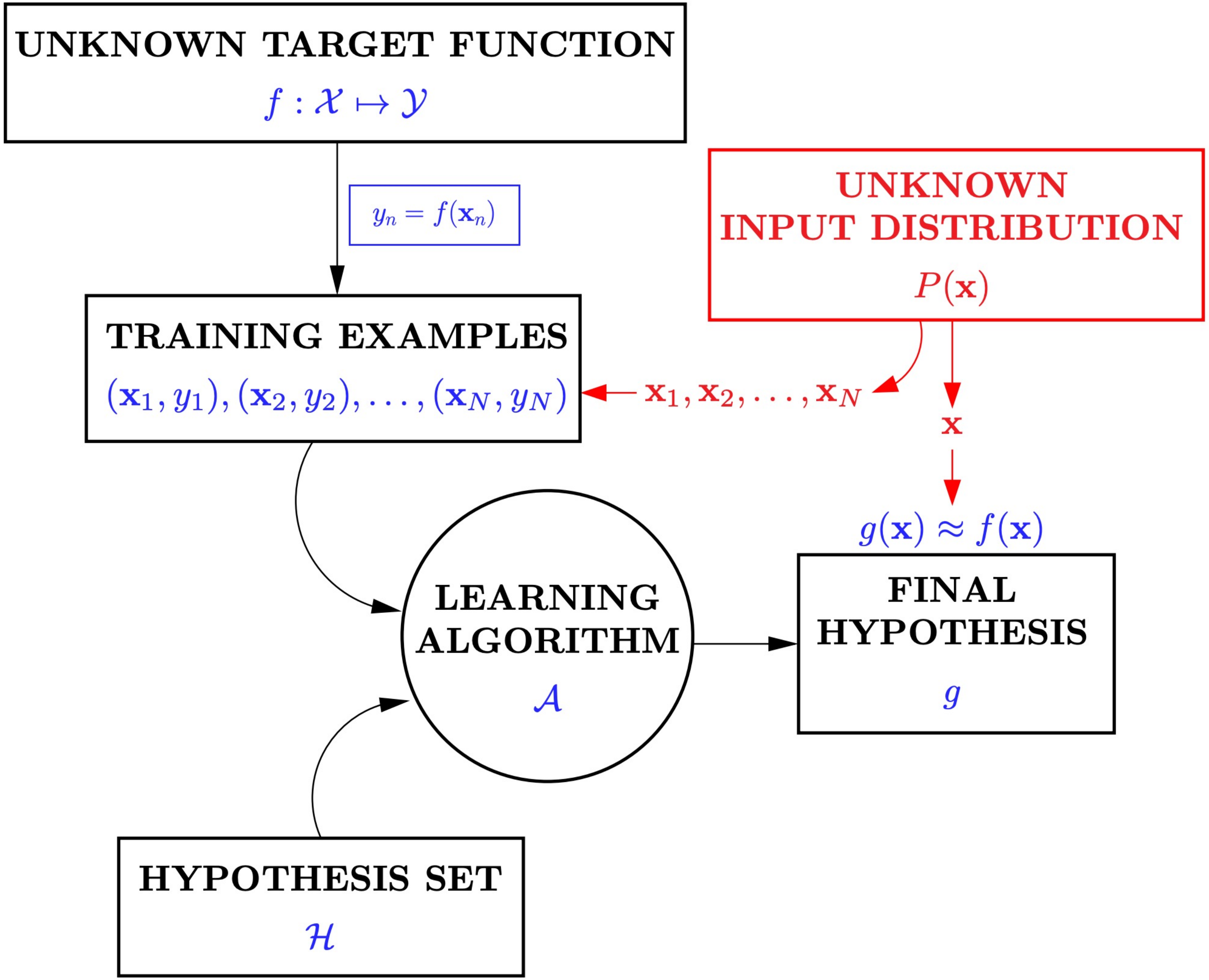


Key assumption of ML

Training data points and testing data points are i.i.d. drawn from the same (unknown) distribution

- Remarks
  - Modern ML is built on probabilistic inference with this assumption
  - The assumption is a reasonable approximation in many useful scenarios
  - The assumption might not hold in other cases
    - There are various research efforts on this, but it’s outside of the scope of this course

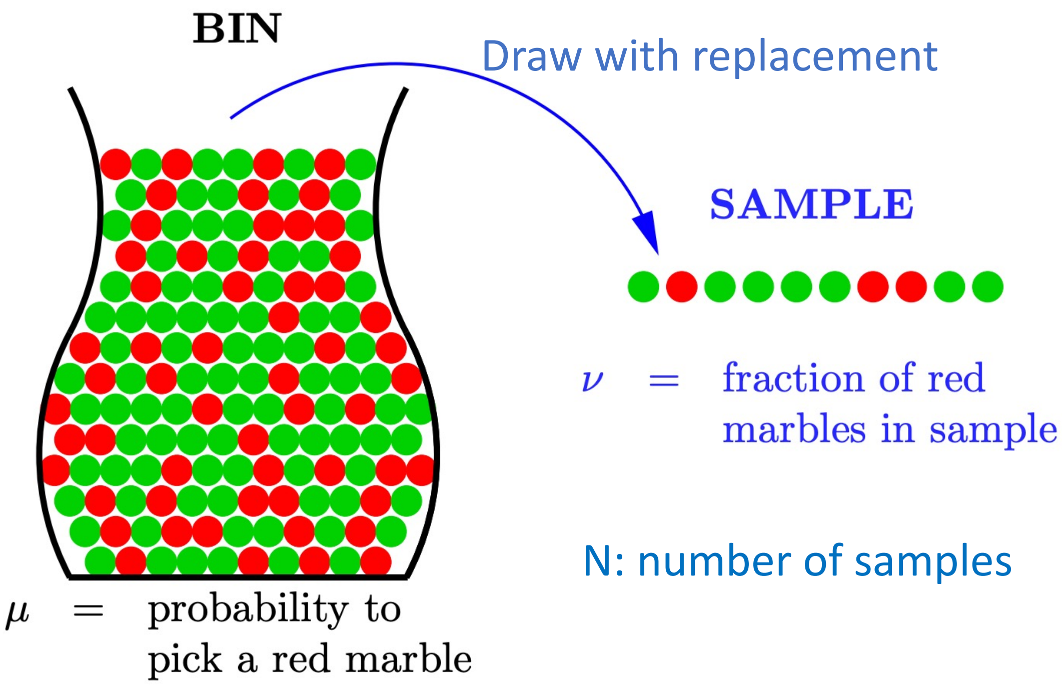

UNKNOWN TARGET FUNCTION
$f : \mathcal{X} \mapsto \mathcal{Y}$
$y_n = f(\mathbf{x}_n)$
TRAINING EXAMPLES
$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$
LEARNING ALGORITHM
$\mathcal{A}$
FINAL HYPOTHESIS
g
HYPOTHESIS SET
$\mathcal{H}$


UNKNOWN TARGET FUNCTION
$f : \mathcal{X} \mapsto \mathcal{Y}$
$y_n = f(\mathbf{x}_n)$
UNKNOWN
INPUT DISTRIBUTION
$P(\mathbf{x})$
TRAINING EXAMPLES
$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$
$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$
$\mathbf{x}$
$g(\mathbf{x}) \approx f(\mathbf{x})$
LEARNING
ALGORITHM
$\mathcal{A}$
FINAL
HYPOTHESIS
g
HYPOTHESIS SET
$\mathcal{H}$

Let's discuss probability first

We'll then talk about how it connects back to machine learning

A Thought Experiment about Probability



N: number of samples

What can we say about μ from ν?

Law of large numbers

- When $N \to \infty$, $\nu \to \mu$

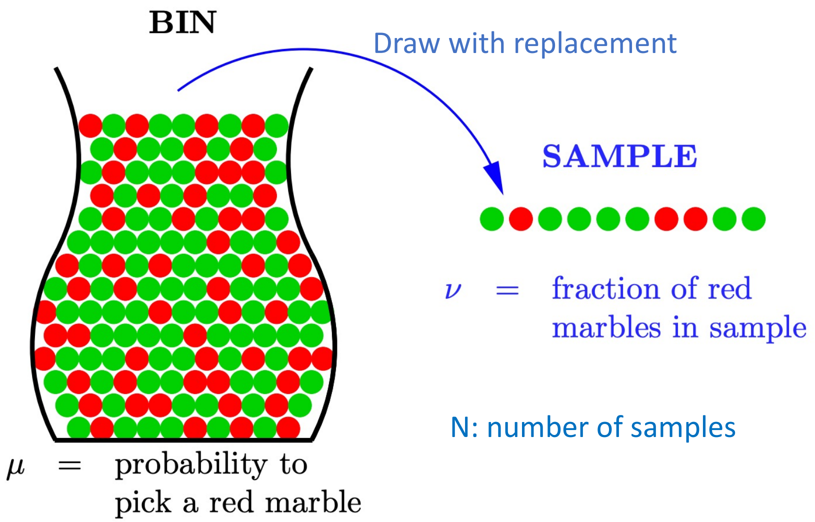
Hoeffding's Inequality

- $\Pr[|\mu - \nu| > \epsilon] \leq 2e^{-2\epsilon^2 N}$ for any $\epsilon > 0$

Interpretations

$$\Pr[|\mu - \nu| > \epsilon] \leq 2e^{-2\epsilon^2 N}$$

- Define $\delta = \Pr[\,|\mu - \nu| > \epsilon\,]$
  - Probability of the bad event
  - Probability of the bad event is bounded by $2e^{-2\epsilon^2 N}$
- A tradeoff between δ, ϵ, N
  - Fix $\epsilon, \delta = O(e^{-N})$
  - Fix $N, \delta = O\left(e^{-\epsilon^2}\right)$
  - Fix $\delta, \epsilon = O\left(\sqrt{1/N}\right)$

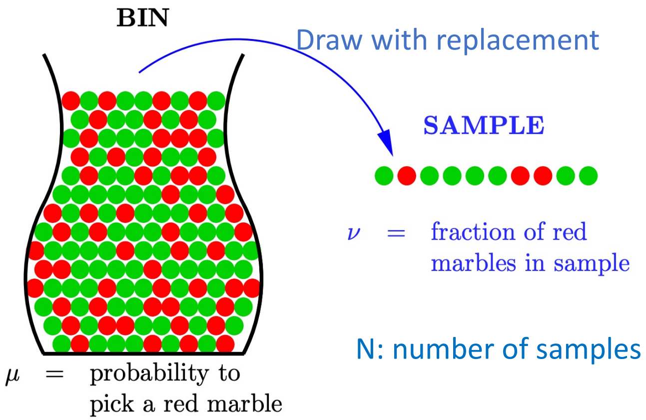


- For example, N=1000
  - $\mu - 0.05 \leq \nu \leq \mu + 0.05$ with 99% chance
  - $\mu - 0.10 \leq \nu \leq \mu - 0.10$ with 99.9999996% chance

Interpretations

$$\Pr[|\mu - \nu| > \epsilon] \le 2e^{-2\epsilon^2 N}$$

- Define $\delta = \Pr[\, |\mu - \nu| > \epsilon \,]$
  - Probability of the bad event



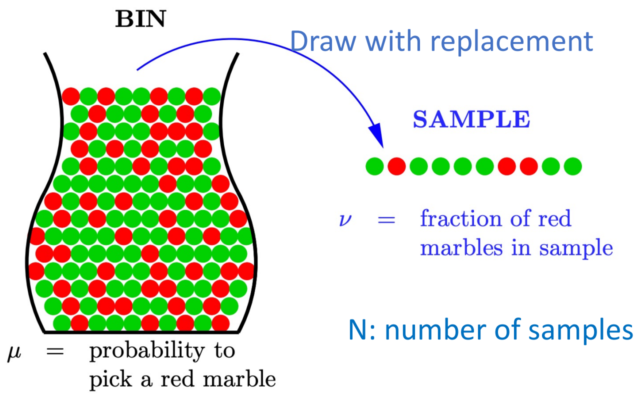
- For example, N=1000
  - $\mu - 0.05 \le \nu \le \mu + 0.05$ with 99% chance
  - $\mu - 0.10 \le \nu \le \mu - 0.10$ with 99.9999996% chance

- ν is approximately close to μ with high probability
- ν as an estimate of μ is **p**robably **a**pproximately **c**orrect (P.A.C.)

PAC learning is proposed by Leslie Valiant, who wins the Turing award in 2010.

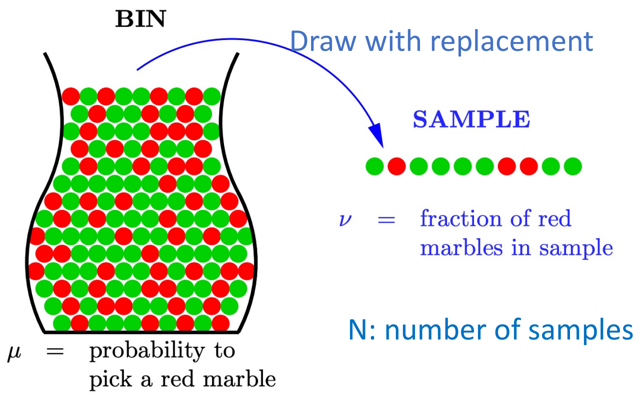
Connection to Learning

- Let each marble represent a point $\vec{x}$, drawn from unknown $P(\vec{x})$
  - Dataset $D = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}$
  - Recall that $y_n = f(\vec{x}_n)$ (will discuss noisy target function f later in the semester)
- "Fix" a hypothesis h
  - For each marble $\vec{x}$, color it as below
    - If $h(\vec{x}) = f(\vec{x})$, color it as green marble [h is correct on $\vec{x}$]
    - If $h(\vec{x}) \neq f(\vec{x})$, color it as red marble [h is wrong on $\vec{x}$]



Connection to Learning

- Let each marble represent a point $\vec{x}$, drawn from unknown $P(\vec{x})$
  - Dataset $D = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}$
  - Recall that $y_n = f(\vec{x}_n)$ (will discuss noisy target function f later in the semester)
- "Fix" a hypothesis h
  - For each marble $\vec{x}$, color it as below
    - If $h(\vec{x}) = f(\vec{x})$, color it as green marble [h is correct on $\vec{x}$]
    - If $h(\vec{x}) \neq f(\vec{x})$, color it as red marble [h is wrong on $\vec{x}$]



- With the above coloring

$$\nu = \frac{1}{N}\sum_{n=1}^{N} \mathbb{1}[h(\vec{x}_n) \neq f(\vec{x}_n)]$$

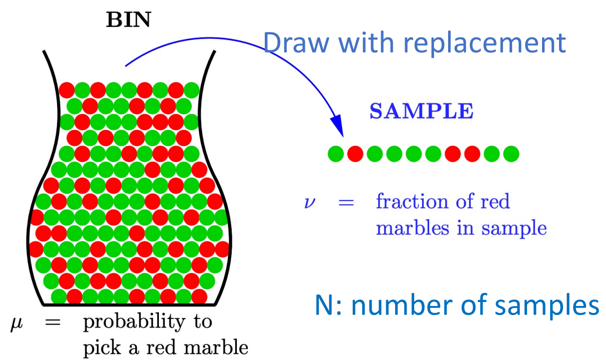
$\stackrel{\text{def}}{=} E_{in}(h)$ **[in-sample error of h]**

$$\mu = \Pr_{\vec{x}\sim P(\vec{x})}[h(\vec{x}) \neq f(\vec{x})]$$

$\stackrel{\text{def}}{=} E_{out}(h)$ **[Out-of-sample error of h]**

Connection to Learning

$$\Pr[|\mu - \nu| > \epsilon] \leq 2e^{-2\epsilon^2 N}$$



- Look at the error again
  - $E_{out}(h)$: What we really care about but unknown to us
  - $E_{in}(h)$: What we can calculate from dataset D

- Fixed a h, What can we say about $E_{out}(h)$ from $E_{in}(h)$?

<u>Hoeffding's Inequality</u>

$\Pr[|E_{out}(h) - E_{in}(h)| > \epsilon] \leq 2e^{-2\epsilon^2 N}$ for any $\epsilon > 0$

- Are we done?
  - Not really, this is verification, not learning

Verification vs. Learning

- Verification
  - I have a hypothesis h
  - I know $E_{in}(h)$, i.e., how well h performs in my dataset
  - I can infer what $E_{out}(h)$ (how well h will perform for unseen data) might be

- Learning
  - Given a dataset D and hypothesis set H
  - Apply some learning algorithm, that outputs a $g \in H$
  - Know $E_{in}(g)$
  - Want to infer $E_{out}(g)$

Connection to "Real" Learning

- Given a finite hypothesis set $H = \{h_1, \ldots, h_M\}$
- Apply some learning algorithm on D, output a $g \in H$
    - For example, choosing the hypothesis that minimizes in-sample error
        - $g = argmin_{h \in H} E_{in}(h)$

- Can we apply Hoeffding's inequality and claim

$$\Pr[|E_{out}(g) - E_{in}(g)| > \epsilon] \leq 2e^{-2\epsilon^2 N} \quad \text{for any } \epsilon > 0$$

- **No!**

Consider this example

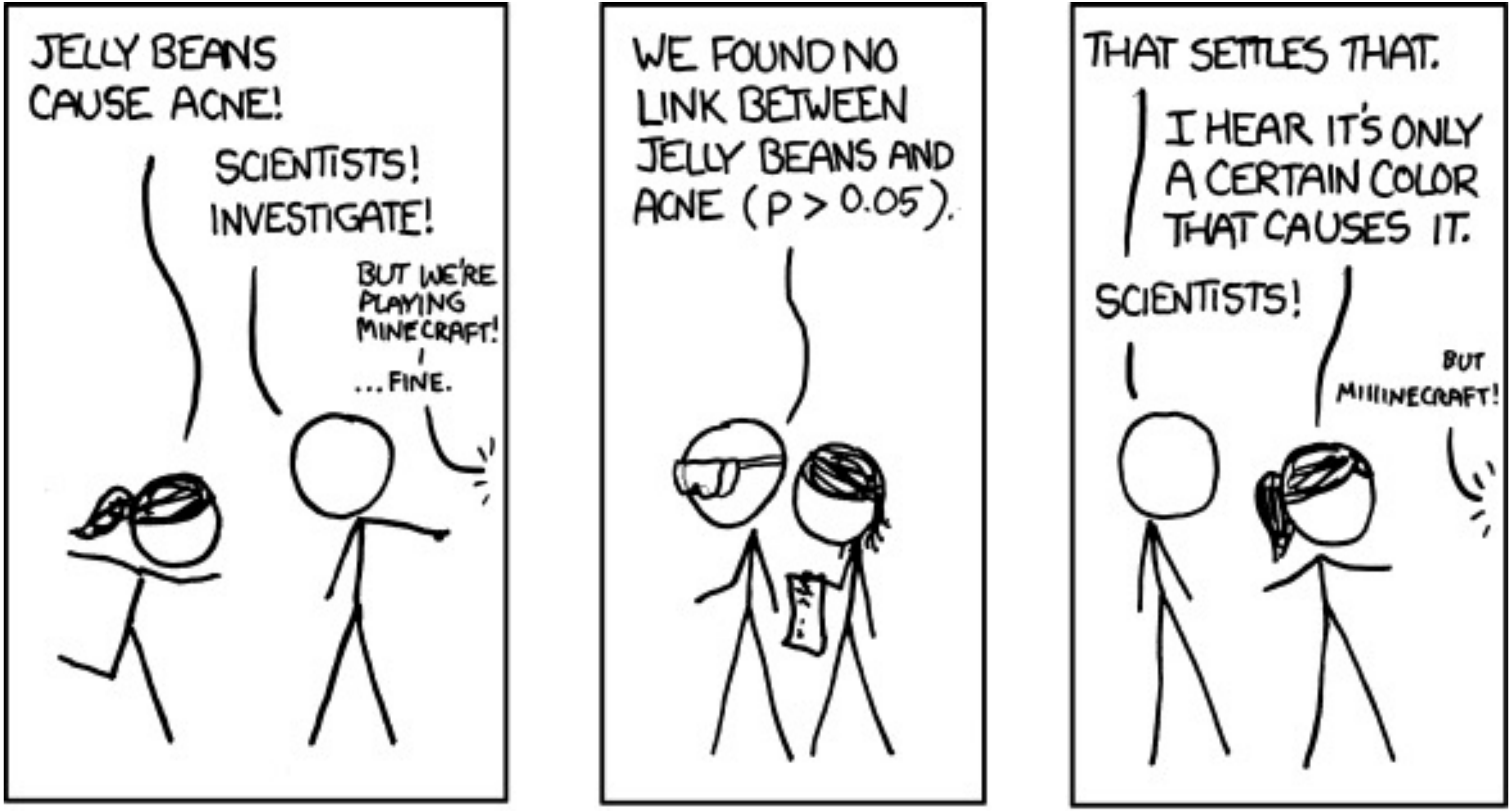
- If you toss a fair coin 10 times, the prob that you get heads 10 times is

$$2^{-10} = \frac{1}{1024}$$

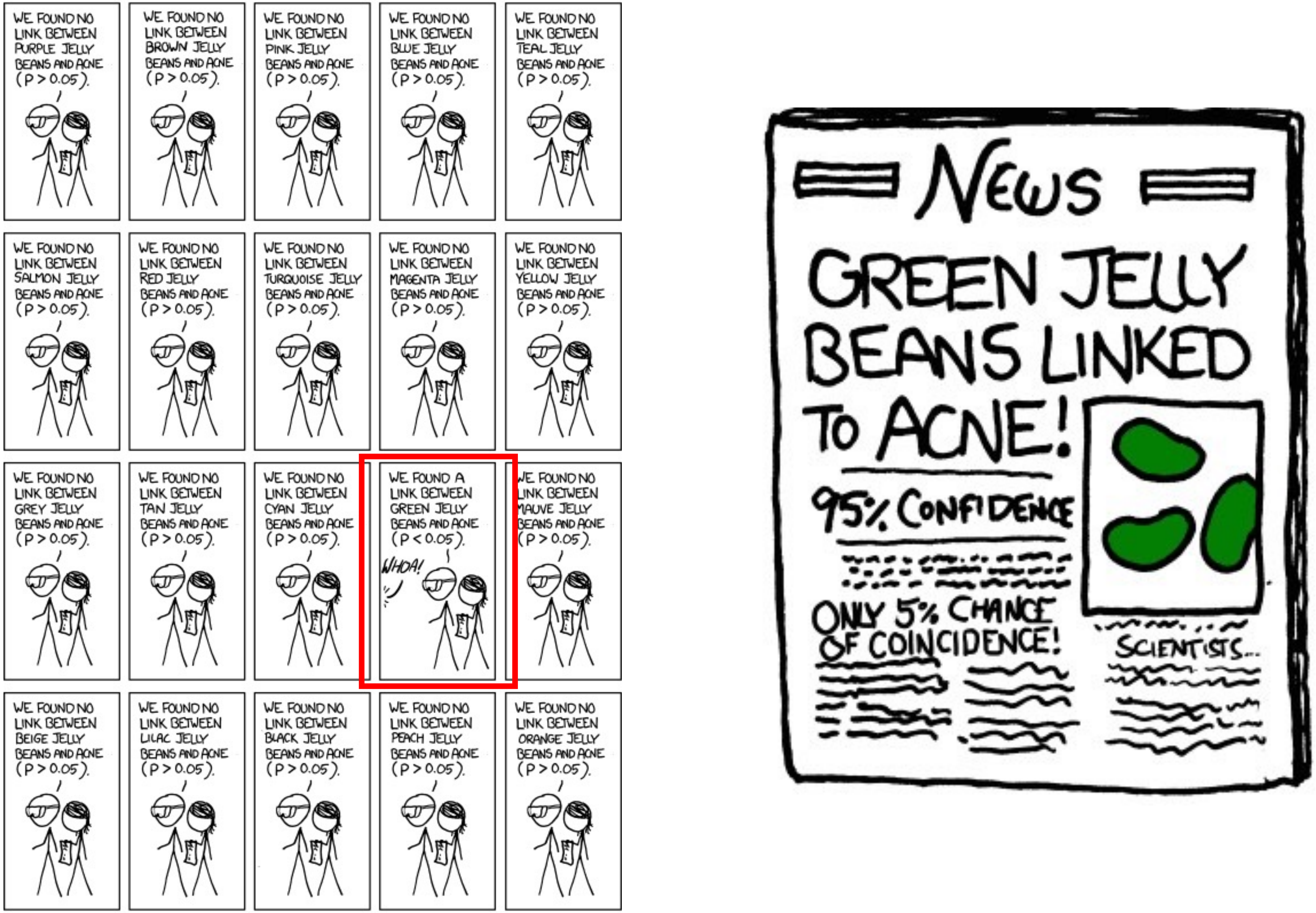
- If you toss 1000 fair coins 10 times each, the probability that at least one coin comes up heads 10 times is

$$1 - \left(\frac{1023}{1024}\right)^{1000} \approx 62.36\%$$

- If each hypothesis is doing random guessing (i.e., tossing a fair coin), if we have 1000 hypothesis with 10 data points, more than 60% chance there will be at least one hypothesis with **zero in-sample error**
  - But that hypothesis is still random guessing and has 50% out-of-sample error



From xkcd, by Randall Munroe: http://xkcd.com/882



From xkcd, by Randall Munroe: http://xkcd.com/882

Connection to “Real” Learning

- Given a finite hypothesis set $H = \{h_1, \dots, h_M\}$
- Apply some learning algorithm on D, output a $g \in H$

- Question: What can we say about $E_{out}(g)$ from $E_{in}(g)$?

Derivations

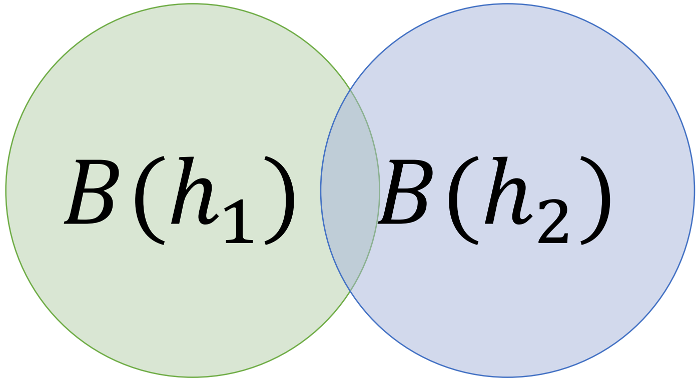
- Define "bad event of h" $B(h)$ as $|E_{out}(h) - E_{in}(h)| > \epsilon$
  - Informally, you can interpret "bad event of h" as the event that we draw a "unrepresentative dataset D" that makes the in-sample errors of h to be far away from out-of-sample error of h

For each fixed $h \in H$, we have $\Pr[B(h)] \leq 2e^{-2\epsilon^2 N}$

- Recall g is selected from H (it could be any $h \in H$)
- What can we say about $\Pr[B(g)]$?

Bounding $\Pr[B(g)]$?

- If g is selected from $\{h_1, h_2\}$



$$B(g) \subseteq B(h_1) \cup B(h_2)$$

$$\begin{aligned} \Pr[B(g)] &\leq \Pr[B(h_1) \text{ or } B(h_2)] \\ &\leq \Pr[B(h_1)] + \Pr[B(h_2)] \quad \text{(Union Bound)} \end{aligned}$$

Derivations

- Define "bad event of h" $B(h)$ as $|E_{out}(h) - E_{in}(h)| > \epsilon$
  - Informally, you can interpret "bad event of h" as the event that we draw a "unrepresentative dataset D" that makes the in-sample errors of h to be far away from out-of-sample error of h

For each fixed $h \in H$, we have $\Pr[B(h)] \leq 2e^{-2\epsilon^2 N}$

- Recall g is selected from H (it could be any $h \in H$)
- What can we say about $\Pr[B(g)]$?

$$
\begin{aligned}
\Pr[B(g)] &\leq \Pr[B(h_1)\ or\ B(h_2)\ or\ \dots or\ B(h_M)] \\
&\leq \Pr[B(h_1)] + \Pr[B(h_2)] + \cdots + \Pr[B(h_M)] \\
&\leq M\ 2e^{-2\epsilon^2 N}
\end{aligned}
$$

Connection to "Real" Learning

- Given a finite hypothesis set $H = \{h_1, \dots, h_M\}$
- Apply some learning algorithm on D, output a $g \in H$

- Question: What can we say about $E_{out}(g)$ from $E_{in}(g)$?

$$\Pr[|E_{out}(g) - E_{in}(g)| > \epsilon] \le 2\boldsymbol{M}e^{-2\epsilon^2 N} \quad \text{for any } \epsilon > 0$$

- M can be considered as a proxy of the "complexity" of the hypothesis set
  - Will talk about what happens when $M \to \infty$ in the next few lectures

Interpreting $\Pr[|E_{out}(g) - E_{in}(g)| > \epsilon] \leq 2Me^{-2\epsilon^2 N}$

- Playing around with the math
  - Define $\delta = \Pr[|E_{out}(g) - E_{in}(g)| > \epsilon]$
  - We have $\delta \leq 2Me^{-2\epsilon^2 N}$ => $\epsilon \leq \sqrt{\frac{1}{2N}\ln\frac{2M}{\delta}}$

- This means, with probability at least $1 - \delta$
  - $E_{out}(g) \leq E_{in}(g) + \epsilon \leq E_{in}(g) + \sqrt{\frac{1}{2N}\ln\frac{2M}{\delta}}$

More Discussion

- With probability at least $1 - \delta$

$$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{1}{2N} \ln \frac{2M}{\delta}}$$

  Consider M as a proxy measure on the "complexity" of H

- Our ultimate goal is to have a small $E_{out}(g)$
  - There is a tradeoff of choosing M (what "learning model" to use)
    - Increase M -> Smaller $E_{in}(g)$ (more hypothesis to "fit" the training data)
    - Increase M -> Larger ϵ
  - It also depends on N, the number of data points you have
    - A small number of data points => use simple models (e.g., linear models)
    - Complex models (e.g., deep learning) work when you have a lot of data