CSE 417T
Introduction to Machine Learning

Lecture 3

Instructor: Chien-Ju (CJ) Ho

Logistics

- Course website and Piazza
  - Website: http://chienjuho.com/courses/cse417t/
  - Piazza: http://piazza.com/wustl/spring2020/cse417t
  - Make sure you follow both regularly.

- I have synchronized the roster on Piazza with that on Canvas.
  - It should be associated with your @wustl.edu email.
  - Let me know if you can't get access

Logistics

- HW 1:
  - http://chienjuho.com/courses/cse417t/hw1.pdf
  - Due: **Feb 7 (Friday), 2020**.
  - You are strongly encouraged to work on it before the drop deadline (this Thu)
  - There will be two submission links: Report and Code.
    - Report: Answer all questions, including the implementation question
      - **Grades are based solely on the report**
    - Code: Complete and submit **two matlab files** in Problem 2
      - Only need to submit these two files
      - The code will only be used for correctness checking (when in doubts) and plagiarism checking.
  - Reserve time if you never used Gradescope.
    - Make sure to **specify the pages for each problem**. You **won't get points** otherwise.

Logistics: Academic Integrity

- Don’t violate the rules of academic integrity

Resources related to Matlab

Matlab: Access

- Check if you can get access to MATLAB

- Notes:
  - If you are an undergraduate student, follow the instructions here. (Portal)
  - If you are an engineering graduate student, you should also have access using the instructions above. If not, contact softwarelicensing@wustl.edu.
  - If you are a non-engineering graduate student, check the portal first. If it's not working,
    - Try computer labs or use remote desktop.
    - You can also purchase a student copy here at $53.

Matlab: Basics

- You should be able to pick up basic ideas relatively quickly if you are familiar with at least one of the popular languages.

- Some useful resources
  - Basic introduction by Prof. Marion Neumann
    https://sites.wustl.edu/neumann/resources/intro-to-matlab/
  - Cheatsheet
    http://web.mit.edu/18.06/www/Spring09/matlab-cheatsheet.pdf

Tutorial by Marion Neumann

- https://sites.wustl.edu/neumann/resources/intro-to-matlab/

TOPICS AND NOTES

- Unit 1: Getting started (MATLAB environment, help, variables, built-in functions, scripts (example_script.m), save & load)
- Unit 2: Arrays (vectors, matrices, and strings)
- Unit 3: Functions (exampleFunction.m, *Advanced topics on functions)
- Unit 4: Logicals and Random Numbers
- Unit 5: If-else and Loops (example_sinTolerance.m)
- Unit6: Plotting
- Materials (zip-file containing all examples and data for excercises)

A Matlab Cheat-sheet (MIT 18.06, Fall 2007)

Basics:

`save 'file.mat'`	save variables to *file.mat*
`load 'file.mat'`	load variables from *file.mat*
`diary on`	record input/output to file *diary*
`diary off`	stop recording
`whos`	list all variables currenly defined
`clear`	delete/undefine all variables
`help command`	quick help on a given *command*
`doc command`	extensive help on a given *command*

Defining/changing variables:

`x = 3`	define variable x to be 3
`x = [1 2 3]`	set x to the 1×3 row-vector (1,2,3)
`x = [1 2 3];`	same, but don't echo x to output
`x = [1;2;3]`	set x to the 3×1 column-vector (1,2,3)
`A = [1 2 3 4;5 6 7 8;9 10 11 12];`	set A to the 3×4 matrix with rows 1,2,3,4 etc.
`x(2) = 7`	change x from (1,2,3) to (1,7,3)
`A(2,1) = 0`	change $A_{2,1}$ from 5 to 0

Arithmetic and functions of numbers:

`3*4, 7+4, 2-6 8/3`	multiply, add, subtract, and divide numbers
`3^7, 3^(8+2i)`	compute 3 to the 7th power, or 3 to the 8+2i power
`sqrt(-5)`	compute the square root of –5
`exp(12)`	compute e^{12}
`log(3), log10(100)`	compute the natural log (ln) and base-10 log ($\log_{10}$)
`abs(-5)`	compute the absolute value |–5|
`sin(5*pi/3)`	compute the sine of 5π/3
`besselj(2,6)`	compute the Bessel function $J_2(6)$

Arithmetic and functions of vectors and matrices:

`x * 3`	multiply every element of x by 3
`x + 2`	add 2 to every element of x
`x + y`	element-wise addition of two vectors x and y
`A * y`	product of a matrix A and a vector y
`A * B`	product of two matrices A and B
`x * y`	not allowed if x and y are two column vectors!
`x .* y`	element-wise product of vectors x and y
`A^3`	the square matrix A to the 3rd power
`x^3`	not allowed if x is not a square matrix!
`x.^3`	every element of x is taken to the 3rd power
`cos(x)`	the cosine of every element of x
`abs(A)`	the absolute value of every element of A
`exp(A)`	e to the power of every element of A
`sqrt(A)`	the square root of every element of A
`expm(A)`	the matrix exponential e^A
`sqrtm(A)`	the matrix whose square is A

Constructing a few simple matrices:

`rand(12,4)`	a 12×4 matrix with uniform random numbers in [0,1)
`randn(12,4)`	a 12×4 matrix with Gaussian random (center 0, variance 1)
`zeros(12,4)`	a 12×4 matrix of zeros
`ones(12,4)`	a 12×4 matrix of ones
`eye(5)`	a 5×5 identity matrix I (“eye”)
`eye(12,4)`	a 12×4 matrix whose first 4 rows are the 4×4 identity
`linspace(1.2,4.7,100)`	row vector of 100 equally-spaced numbers from 1.2 to 4.7
`7:15`	row vector of 7,8,9,…,14,15
`diag(x)`	matrix whose diagonal is the entries of x (and other elements = 0)

Portions of matrices and vectors:

`x(2:12)`	the 2nd to the 12th elements of x
`x(2:end)`	the 2nd to the last elements of x
`x(1:3:end)`	every third element of x, from 1st to the last
`x(:)`	all the elements of x
`A(5,:)`	the row vector of every element in the 5th row of A
`A(5,1:3)`	the row vector of the first 3 elements in the 5th row of A
`A(:,2)`	the column vector of every element in the 2nd column of A
`diag(A)`	column vector of the diagonal elements of A

Solving linear equations:

`A \ b`	for A a matrix and b a column vector, the solution x to $Ax=b$
`inv(A)`	the inverse matrix A^{-1}
`[L,U,P] = lu(A)`	the LU factorization $PA=LU$
`eig(A)`	the eigenvalues of A
`[V,D] = eig(A)`	the columns of V are the eigenvectors of A, and the diagonals `diag(D)` are the eigenvalues of A

Plotting:

`plot(y)`	plot y as the y axis, with 1,2,3,… as the x axis
`plot(x,y)`	plot y versus x (must have same length)
`plot(x,A)`	plot columns of A versus x (must have same # rows)
`loglog(x,y)`	plot y versus x on a log-log scale
`semilogx(x,y)`	plot y versus x with x on a log scale
`semilogy(x,y)`	plot y versus x with y on a log scale
`fplot(@(x) …expression…,[a,b])`	plot some expression in x from $x=a$ to $x=b$
`axis equal`	force the x and y axes of the current plot to be scaled equally
`title('A Title')`	add a title *A Title* at the top of the plot
`xlabel('blah')`	label the x axis as *blah*
`ylabel('blah')`	label the y axis as *blah*
`legend('foo','bar')`	label 2 curves in the plot *foo* and *bar*
`grid`	include a grid in the plot
`figure`	open up a new figure window

Pay attention to matrix operations

Transposes and dot products:

`x.', A.'`	the transposes of x and A
`x', A'`	the complex-conjugate of the transposes of x and A
`x' * y`	the dot (inner) product of two *column* vectors x and y
`dot(x,y), sum(x.*y)`	…two other ways to write the dot product
`x * y'`	the *outer* product of two *column* vectors x and y

Specialty of Matlab

- Matlab (Matrix Laboratory) is optimized for performing **matrix operations**
  - If you don't care about efficiency, treating it as another language works
  - But, try to take advantage of what Matlab is good at
    - There will be big differences in both runtime efficiency and code simplicity

- Example: As part of Problem 3 in HW 1
  - You need to simulate the process of flipping 1,000 fair coins, 10 times each, and calculate the ratio of heads for each coin.
  - You can do this with a single line of codes in Matlab.
    - randi to generate random integers of a 1000 x 10 matrix
    - mean to calculate the mean of rows
    - help [command] to see how to use [command]

Matlab: Vectorization

- Vectorization: Take advantage of Matlab's ability for matrix operations
  - https://www.mathworks.com/help/matlab/matlab_prog/vectorization.html

- Ex: Computes the value of sines for 1,001 values from 0 to 10

```
i = 0;
for t = 0:.01:10
    i = i + 1;
    y(i) = sin(t);
end
```

$\xrightarrow{\text{Vectorize}}$

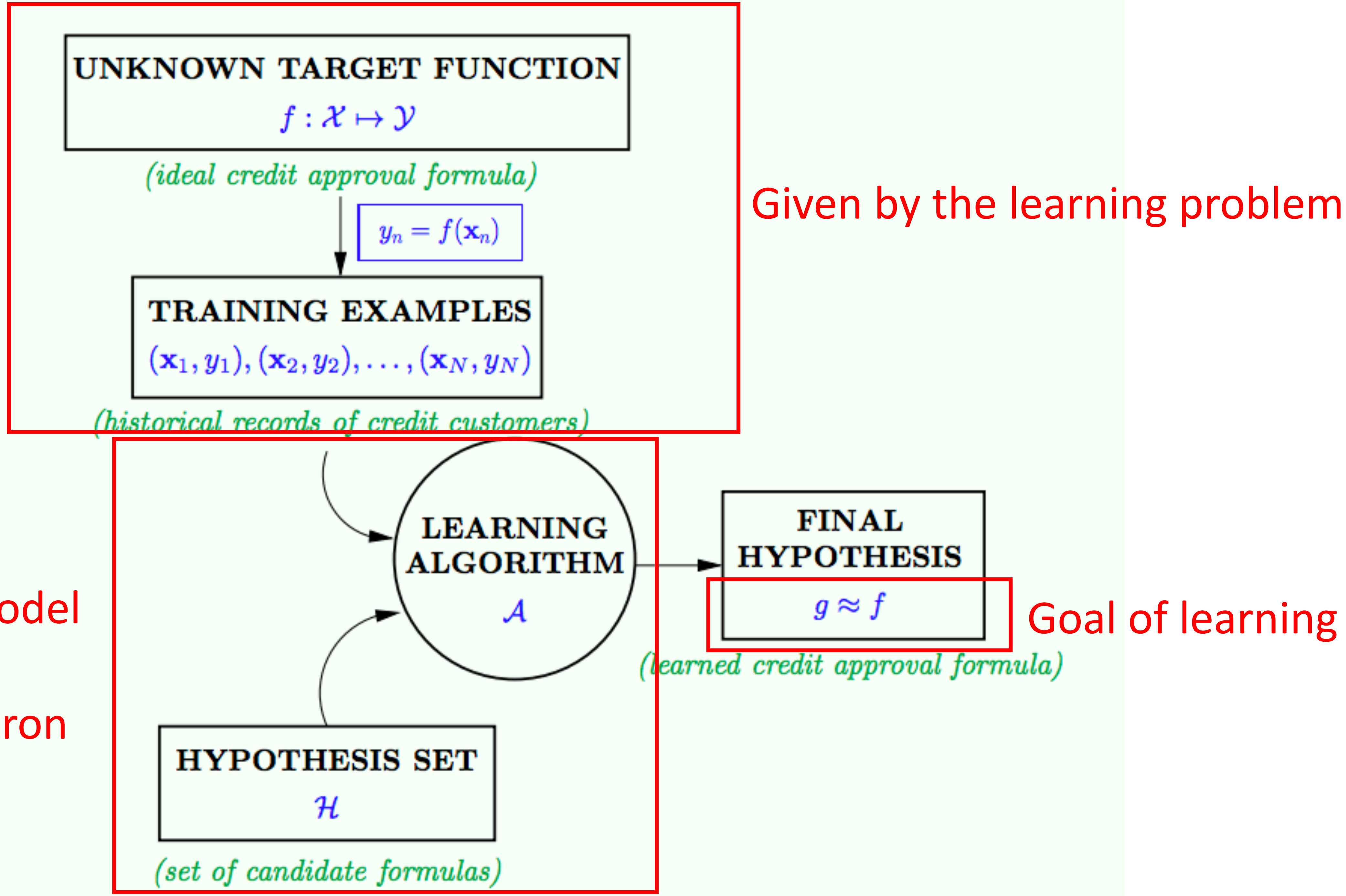
```
t = 0:.01:10;
y = sin(t);
```

- Matlab can often directly operates on matrix and vector, treating them as variables for basic commands.

Matlab: Homework Requirement

- Required to use Matlab for Problem 2 of HW 1
  - Generate random datasets.
  - Implement PLA and observe its performance.
- A good practice to get you familiar with Matlab
  - A lot of computations can be implemented as matrix operations
- Code submission
  - Only need to fill in and submit the two stub files

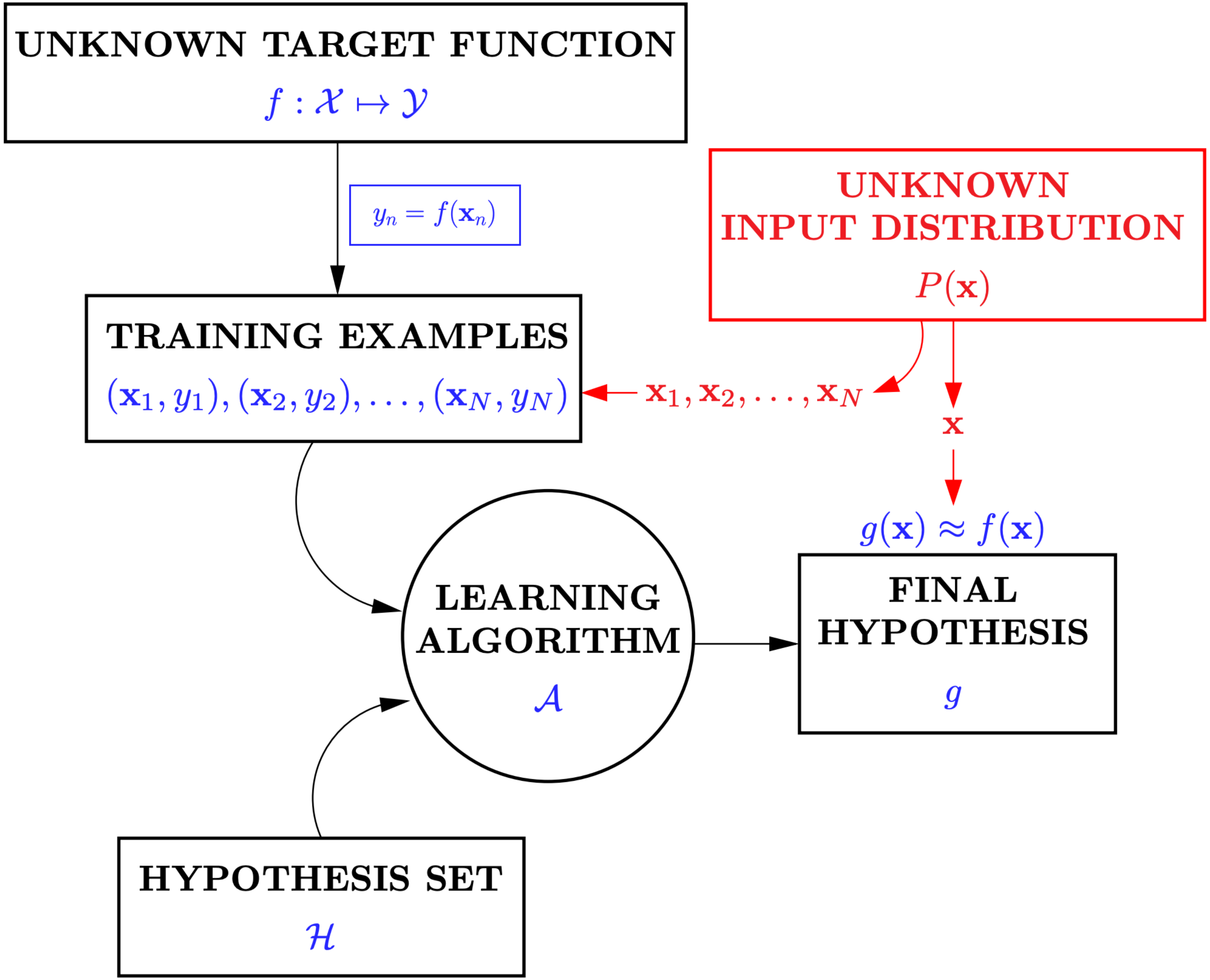
Recap



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

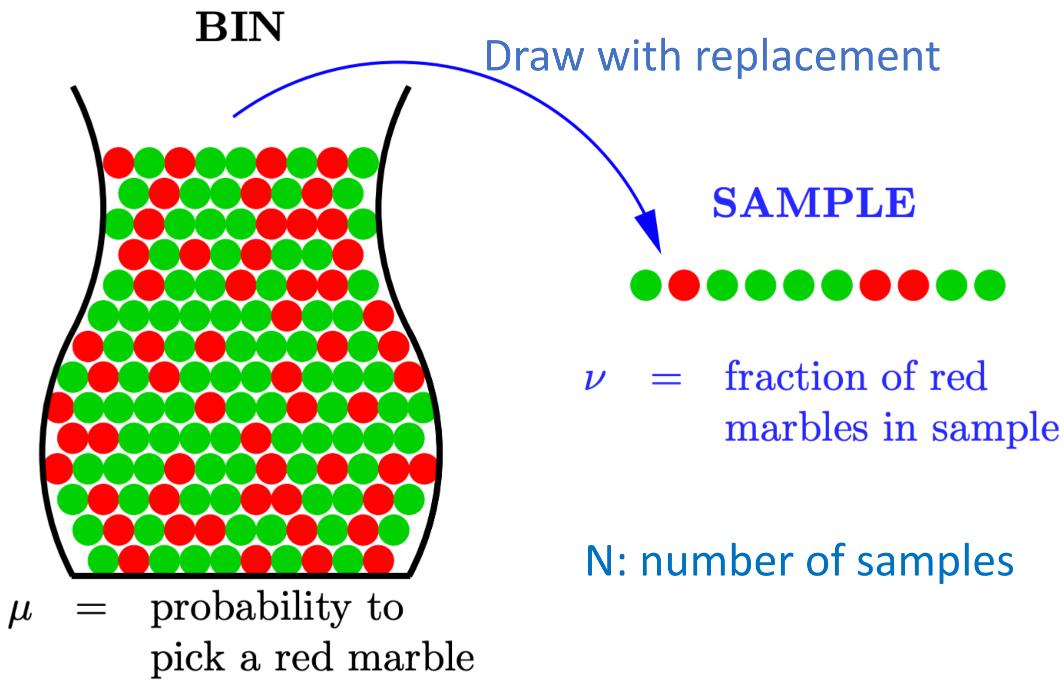
Goal of Learning: Generalization

- Given **training data**, find a hypothesis that is close to the unknown target function on the **unseen testing data**.
- This goal is generally impossible without assumptions.
- Key assumption in machine learning:
  - Training and testing data are i.i.d. drawn from the same (unknown) distribution.



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
$\mathcal{H}$

A Thought Experiment about Probability



What can we say about μ from ν?

Law of large numbers

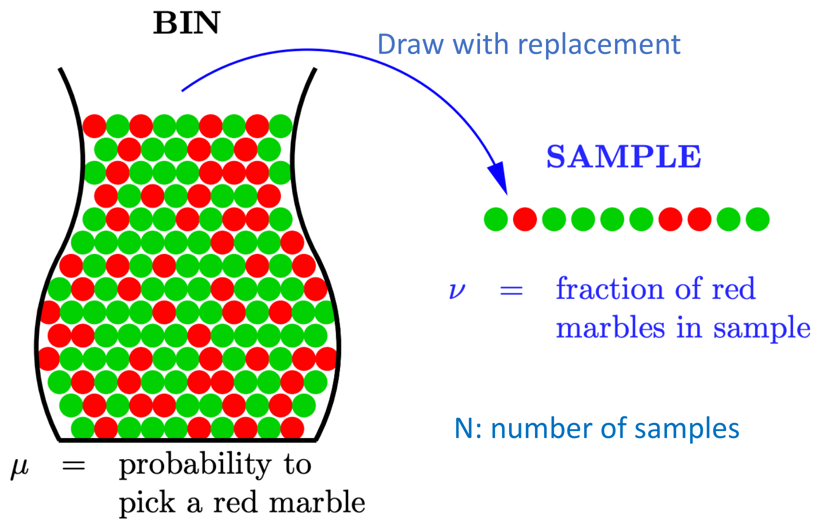
- When $N \to \infty, \nu \to \mu$

Hoeffding's Inequality

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

Connection to Learning

- Let each marble represent a point $\vec{x}$, drawn from unknown $P(\vec{x})$
  - We assume f is perfect for now (i.e., $f(\vec{x}) = y$)



- "Fix" a hypothesis h
  - For each marble $\vec{x}$,
    - If $h(\vec{x}) = f(\vec{x})$, color it as green marble (h is correct)
    - If $h(\vec{x}) \neq f(\vec{x})$, color it as red marble (h makes error)

- With the above coloring
  - $\mu = \Pr_{\vec{x} \sim P(\vec{x})}[h(\vec{x}) \neq f(\vec{x})] \stackrel{\text{def}}{=} \boldsymbol{E_{out}(h)}$ [Out-of-sample error of h]
  - $\nu = \frac{1}{N} \sum_{n=1}^{N} \mathbb{1}[h(\vec{x}_n) \neq f(\vec{x}_n)] \stackrel{\text{def}}{=} \boldsymbol{E_{in}(h)}$ [in-sample error of h]

Connection to Learning

- $\boldsymbol{E_{out}(h)}$: What we really want to know but unknown to us
- $\boldsymbol{E_{in}(h)}$: What we can calculate from dataset

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

  Hoeffding's Inequality

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

- 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.

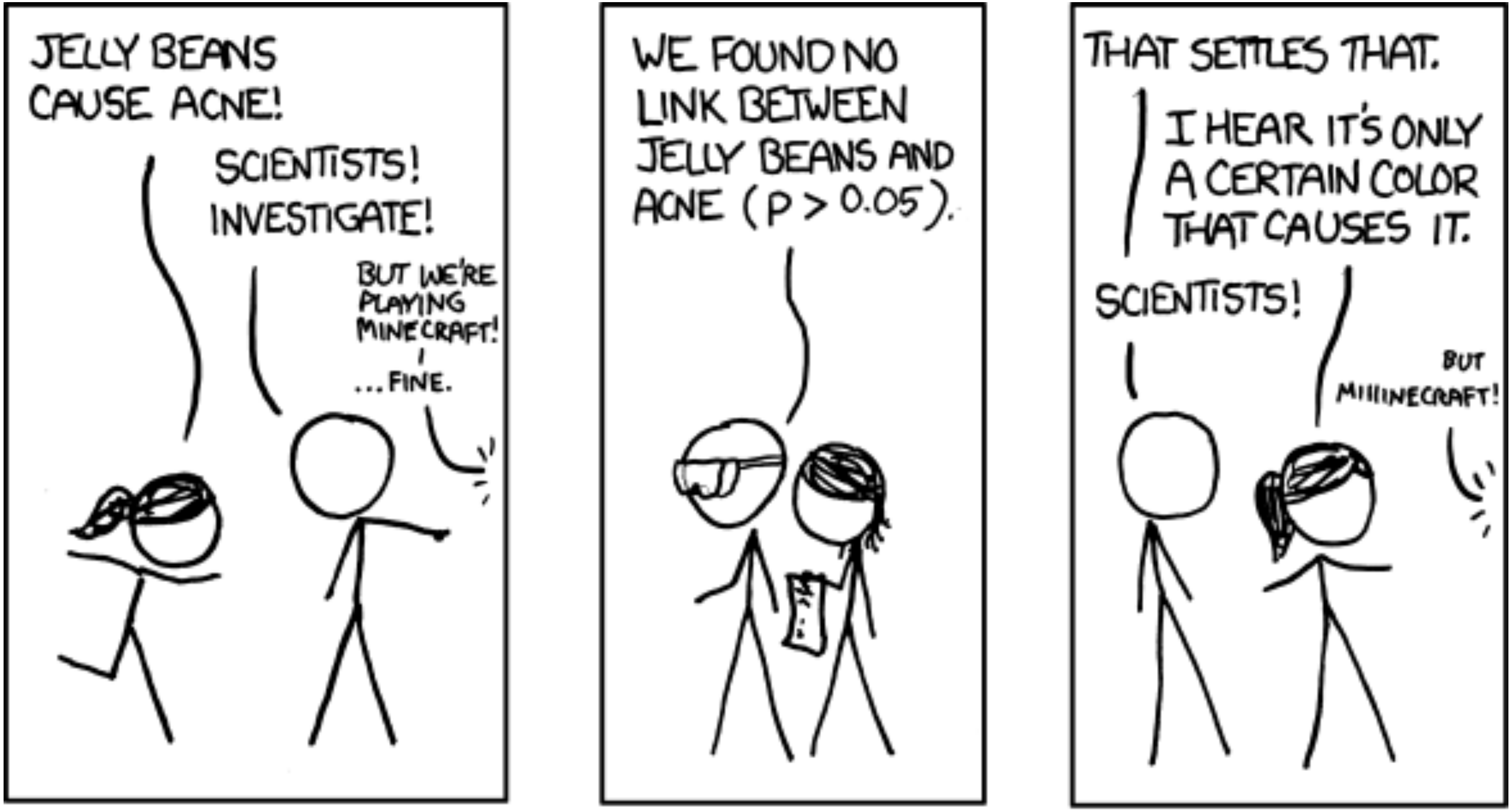
- Learning
  - Given a dataset D and hypothesis set H.
  - Apply some learning algorithm, that outputs a $g \in \mathrm{H}$.
  - Know $E_{in}(g)$.
  - Want to infer $E_{out}(g)$

Connection to "Real" Learning

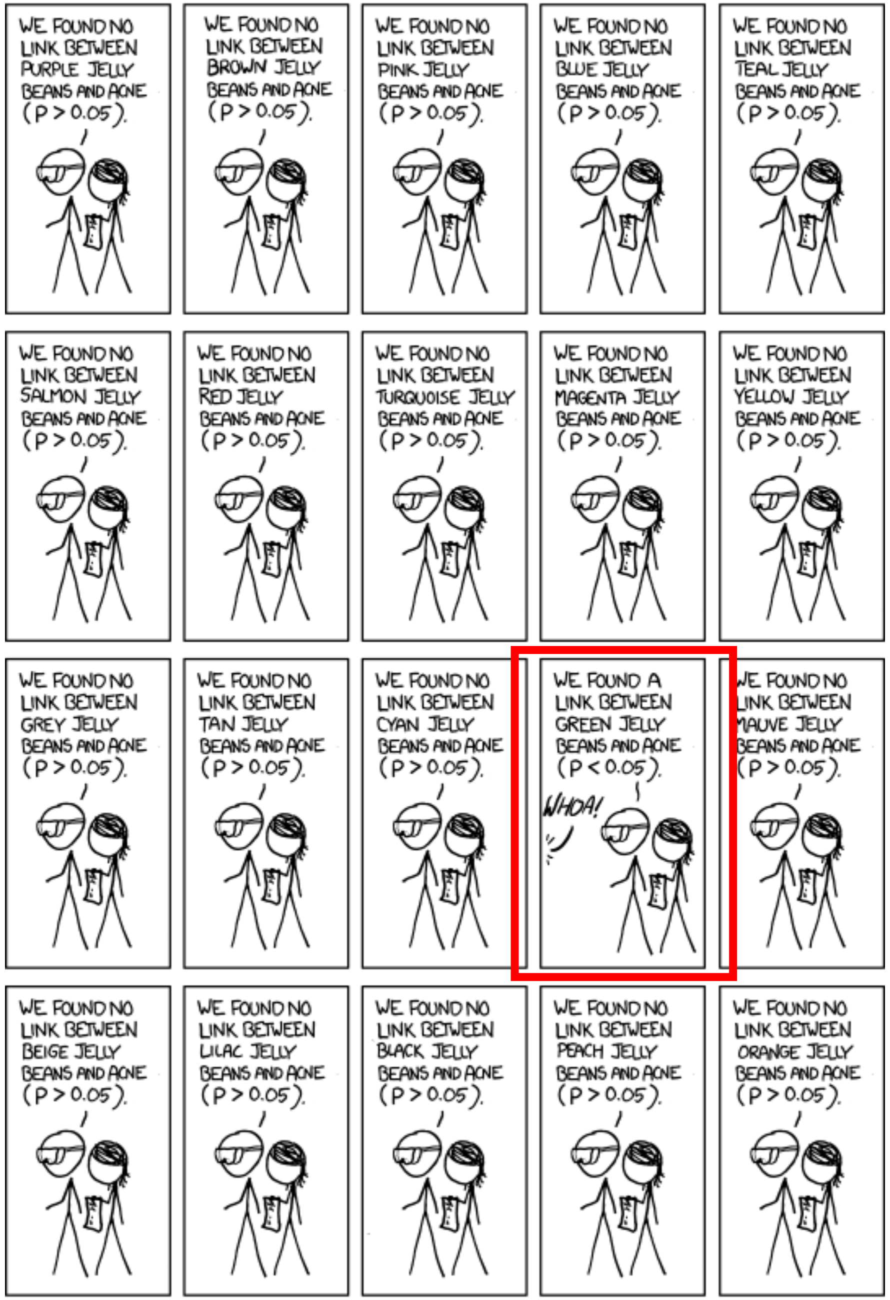
- Given a finite hypothesis set $H = \{\mathrm{h}_1, \ldots, \mathrm{h}_\mathrm{M}\}$
  - Will discuss the infinite case in the next few lectures.
- Apply some learning algorithm on D, output a $g \in H$
  - For example, choosing the hypothesis that minimizes in-sample error
    - $g = argmin_{h \in H} E_{in}(h)$
- Can we apply Hoeffding's inequality and claim

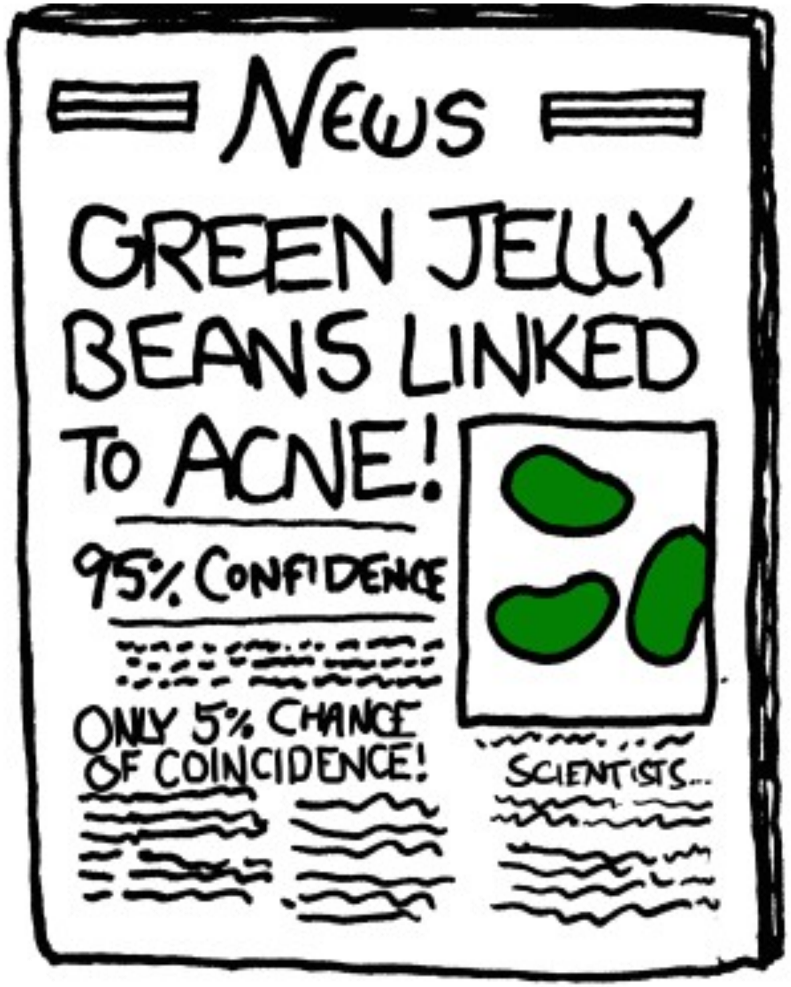
$$\Pr[|E_{out}(g) - E_{in}(g)| > \epsilon] \leq 2e^{-2\epsilon^2 N} \quad \text{for any } \epsilon > 0$$

- **No!**



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





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

Another Analogy

- If you toss a fair coin 10 times, the probability that it comes up heads 10 times is $2^{-10} \approx$ 1e-3

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

$$1 - \left(\frac{1023}{1025}\right)^{1000} \approx 62.36\%$$

- 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

One More Analogy

- Three fair coins, numbered by 1, 2, 3. Each flipped 10 times
- Question: (choosing from >5, =5, or <5)
  - What's the expected number of heads of the 10 flips for coin 1?

    Ans: = 5
  - Randomly choose a coin, what's the expected number of heads of the 10 flips for this one?

    Ans: = 5
  - After observing the flips, choosing the coin with the smallest number of heads, what is the expected number of heads of the 10 flips for the coin?

    Ans: < 5
  - Without observing the flips, choose the coin anyway you like, what is the expected number of heads of the 10 flips for this oin?

    Ans: = 5
- You will implement a simulation for this process (with 1,000 coins) in Problem 3 of HW1.

One More Analogy

- Analogy to learning
  - Coin -> Hypothesis
  - Coin flips -> Performance of hypothesis in training data D

- Choosing the hypothesis “before” or “after” looking at the data (knowing the realization of the data drawing) makes a very big difference!

Brief Lecture Notes

The notes are not intended to be comprehensive.

Let me know if you spot errors.

Connection to "Real" Learning

- Given a finite hypothesis set $H = \{\mathrm{h}_1, \ldots, \mathrm{h}_\mathrm{M}\}$
- Apply some learning algorithm on D, output a $g \in H$

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

Connection to “Real” Learning

- Since g is selected from H (it could be any $h \in H$)
- 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

- What can we say about $\Pr[B(g)]$?

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

Connection to "Real" Learning

- Given a finite hypothesis set $H = \{\mathrm{h}_1, \dots, \mathrm{h}_\mathrm{M}\}$
- Apply some learning algorithm on D, output a $g \in H$

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

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

- 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

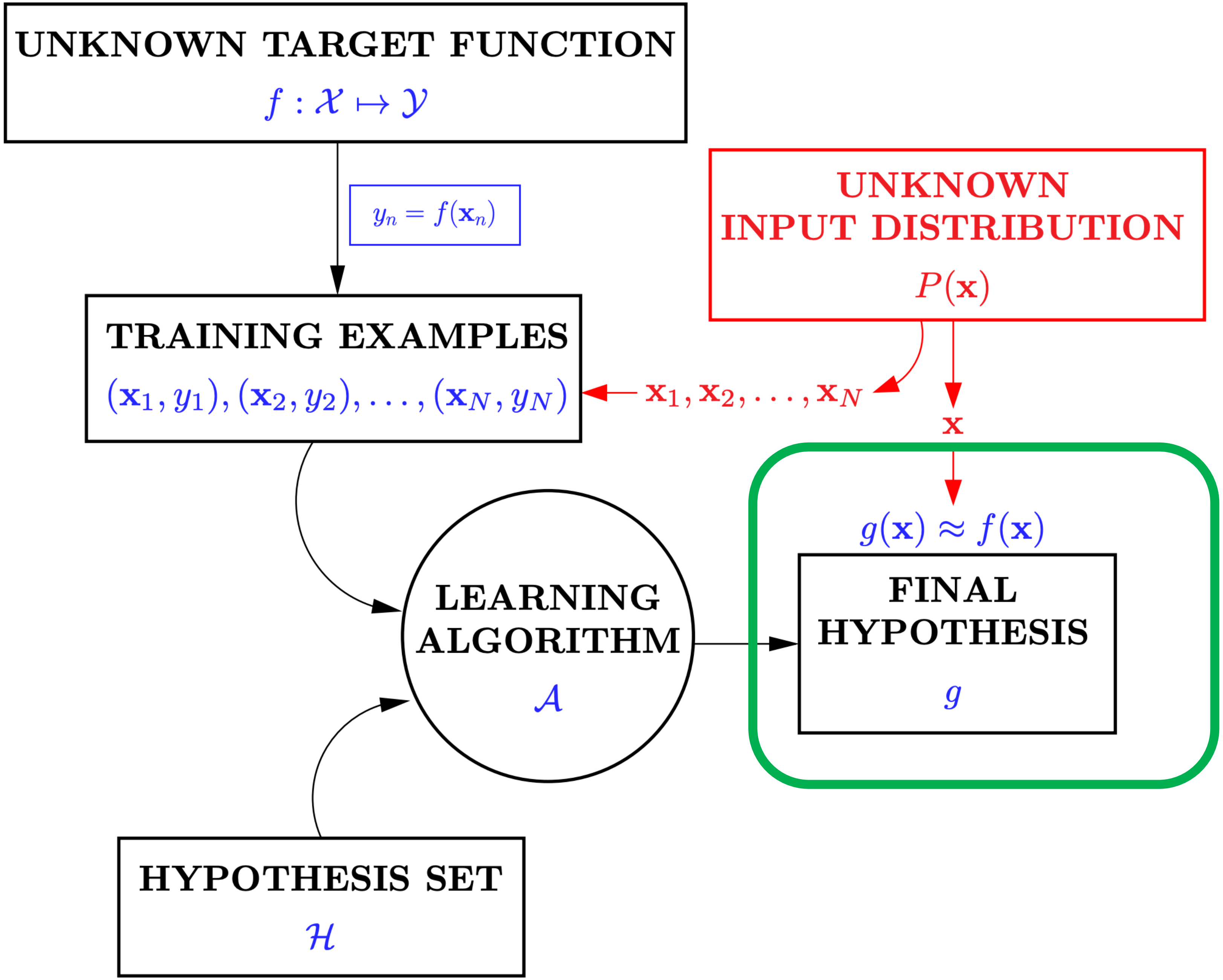
More Interpretations

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

- Note that our goal is to have a small $E_{out}(g)$
  - A tradeoff of choosing M: the complexity of hypothesis set
    - Increase M -> Smaller $E_{in}(g)$ (more hypothesis to “fit” the training data)
    - Increase M -> Larger ϵ

Revisit the learning problem


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
$\mathcal{H}$

Goal: $g \approx f$

- A general approach:
  - Define a error function $E(h, f)$ that quantify how far away g is to f
  - Choose the one with the smallest error
  - For example: $g = \underset{h \in \mathcal{H}}{\operatorname{argmin}} E(h, f)$

- E is usually defined in terms of a pointwise error function $e(h, f, \vec{x})$
  - Binary error (classification): $e(h, f, \vec{x}) = \mathbb{I}[h(\vec{x}_n) \neq f(\vec{x}_n)]$ (What we have discussed so far)
  - Squared error (regression): $e(h, f, \vec{x}) = \left(f(\vec{x}) - h(\vec{x})\right)^2$

How to choose the error function?

- Domain specific: Specified by domain experts
- Fingerprint recognition example:
  - Input: fingerprints
  - Outputs: whether the person is authorized

		$f(\vec{x})$	
		+1	-1
$h(\vec{x})$	+1	No error	False positive
	-1	False negative	No error

- Errors assigned to false negative/positive differ depending on applications
  - Supermarket coupons vs FBI
  - False positive is a big issue for FBI but probably fine for supermarket coupons

How to choose the error function?

- Domain specific: Specified by domain experts
- Computation considerations
  - ML Algorithm is essentially doing optimization (finding g with smallest error)

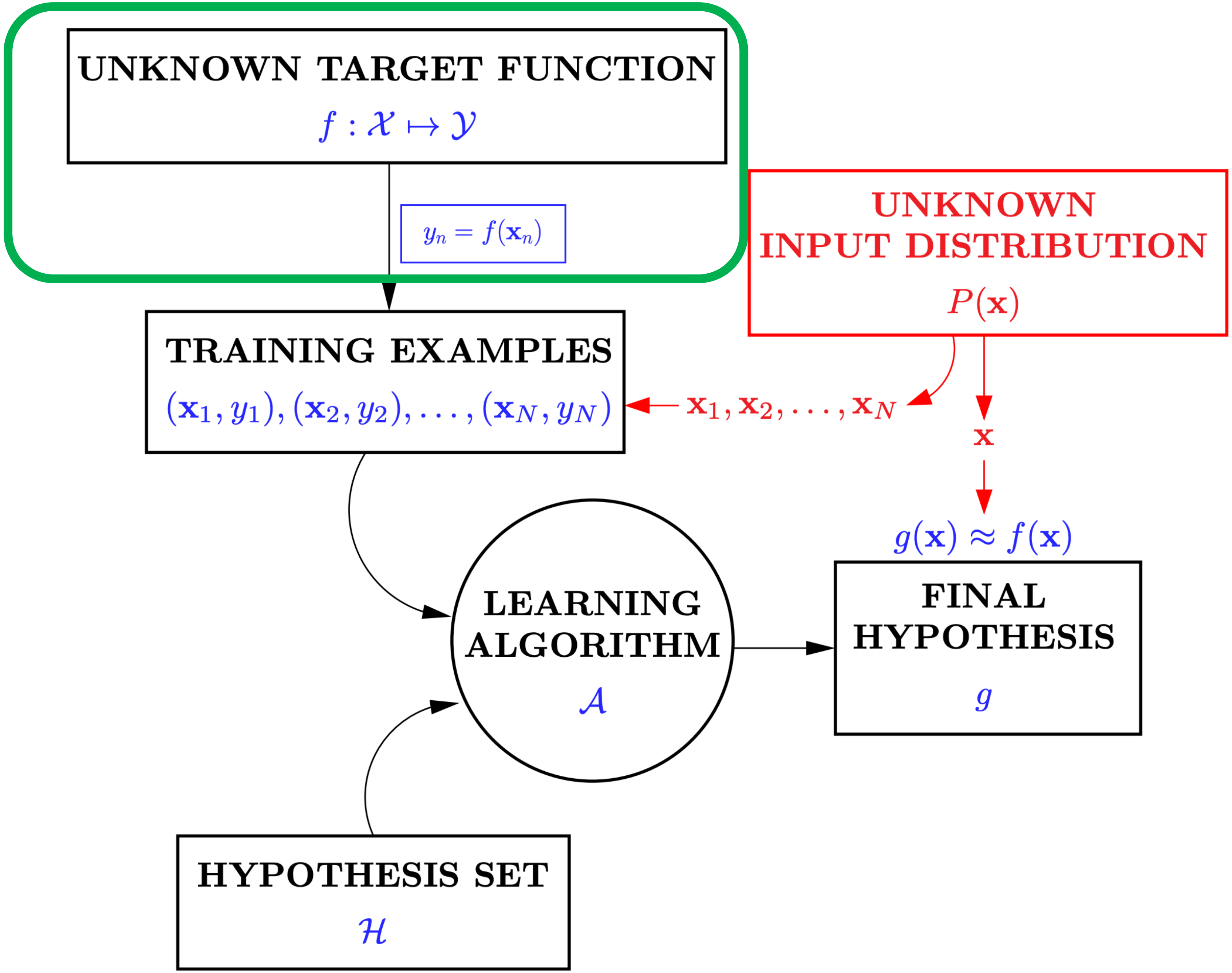
$$g = \operatorname*{argmin}_{h \in \mathcal{H}} E(h, f)$$

  - Choosing the error that is “easier” to optimize

How to choose the error function?

- Domain specific: Specified by domain experts
- Computation considerations

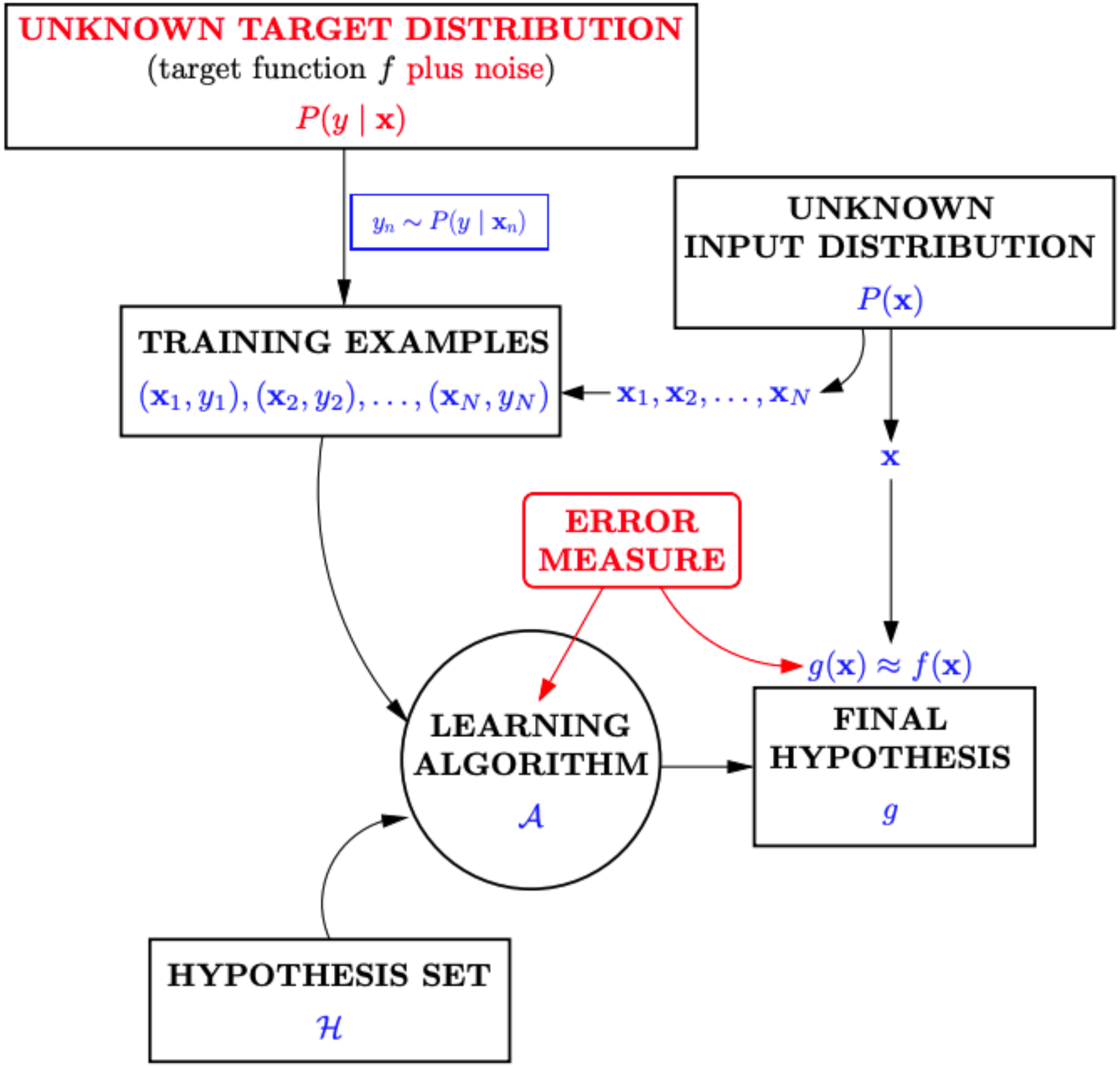
- Specifying the error function is part of setting up the learning problem
  - It impact what you eventually learn.


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
$\mathcal{H}$

Noisy Target

- What if there doesn't exist f such that $y = f(\vec{x})$?
  - f is stochastic instead of deterministic

- Common approach
  - Instead of a target function, define a target **distribution**
  - Instead of $y = f(\vec{x})$, y is drawn from a conditional distribution $P(y|\vec{x})$
  - $y = f(\vec{x}) + \epsilon$ where $\epsilon \sim N(0, \sigma^2)$



UNKNOWN TARGET DISTRIBUTION
(target function f plus noise)
$P(y \mid \mathbf{x})$
$y_n \sim P(y \mid \mathbf{x}_n)$
UNKNOWN INPUT DISTRIBUTION
$P(\mathbf{x})$
TRAINING EXAMPLES
$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$
$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$
$\mathbf{x}$
ERROR MEASURE
$g(\mathbf{x}) \approx f(\mathbf{x})$
LEARNING ALGORITHM
$\mathcal{A}$
FINAL HYPOTHESIS
g
HYPOTHESIS SET
$\mathcal{H}$

Theory of Generalization

This is going to be the most “abstract” part of this course.