CSE 417T
Introduction to Machine Learning

Lecture 7

Instructor: Chien-Ju (CJ) Ho

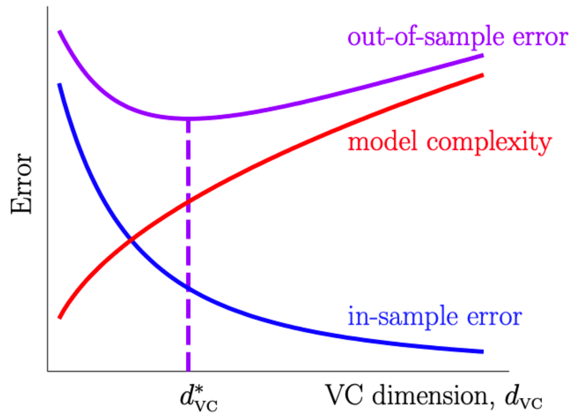
Logistics

- HW1: Due Sep 23
  - Reserve time if you have never used Gradescope
  - Check that submission is readable (if you scan your handwriting)
  - **Correctly assign pages** to each problem (you won't get points otherwise)
- HW2: Will be announce later this week
- Exam dates
  - Exam 1: announce later this week (most likely in the week before spring break)
  - Exam 2: last lecture of the semester

Recap

VC Generalization Bound

- VC Bound: $E_{out}(g) \leq E_{in}(g) + O\left(\sqrt{d_{vc}\frac{\ln N}{N}}\right)$
- Theoretically characterize the feasibility of learning
- The performance of your learning, i.e., $E_{out}(g)$, depends on
  - How well you fit your data ($E_{in}(g)$)
  - How well your $E_{in}(g)$ generalizes to $E_{out}(g)$

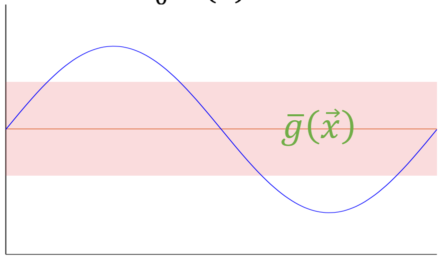


Bias-Variance Decomposition

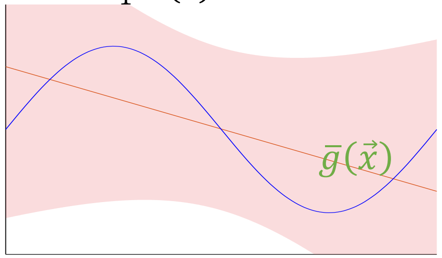
- $\mathbb{E}_D\left[E_{out}\left(g^{(D)}\right)\right] = \mathbb{E}_{\vec{x}}\left[\overbrace{\left(\bar{g}(\vec{x}) - f(\vec{x})\right)^2}^{\text{Bias}(\vec{x})}\right] + \mathbb{E}_{\vec{x}}\left[\mathbb{E}_D\left[\overbrace{\left(g^{(D)}(\vec{x}) - \bar{g}(\vec{x})\right)^2}^{\text{Var}(\vec{x})}\right]\right]$

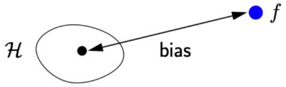
- The performance of your learning, i.e.,$\mathbb{E}_D\left[E_{out}\left(g^{(D)}\right)\right]$, depends on
  - How well you can fit your data using your hypothesis set (bias)
  - How stable your learning is for a randomly drawn dataset (variance)

H_0: $h(x) = b$

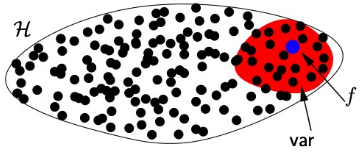


H_1: $h(x) = ax + b$





Very small model



Very large model

Today’s Lecture

The notes are not intended to be comprehensive. They should be accompanied by lectures and/or textbook.

Let me know if you spot errors.

Two Theories of Generalization

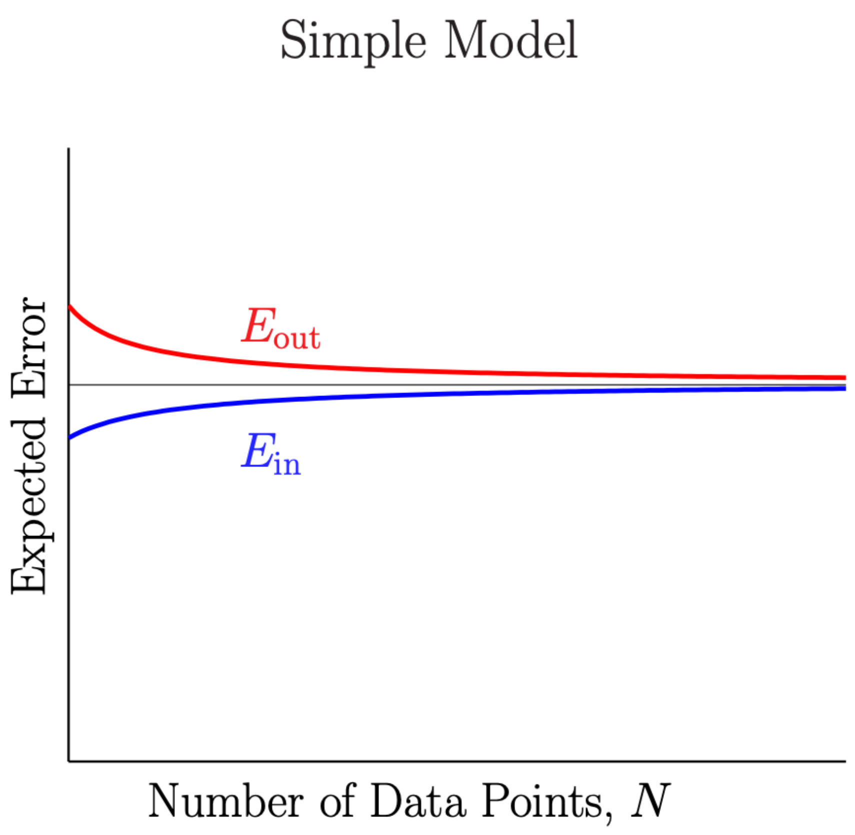
- VC Generalization Bound

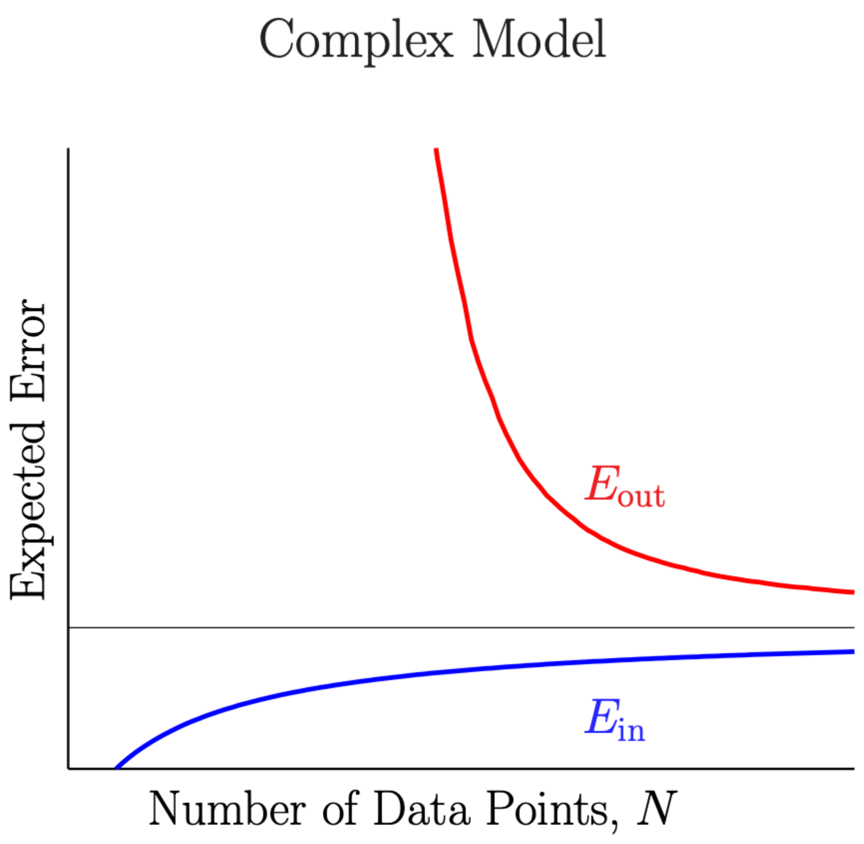
$$E_{out}(g) \leq E_{in}(g) + O\left(\sqrt{d_{vc}\frac{\ln N}{N}}\right)$$

- Bias-Variance Tradeoff

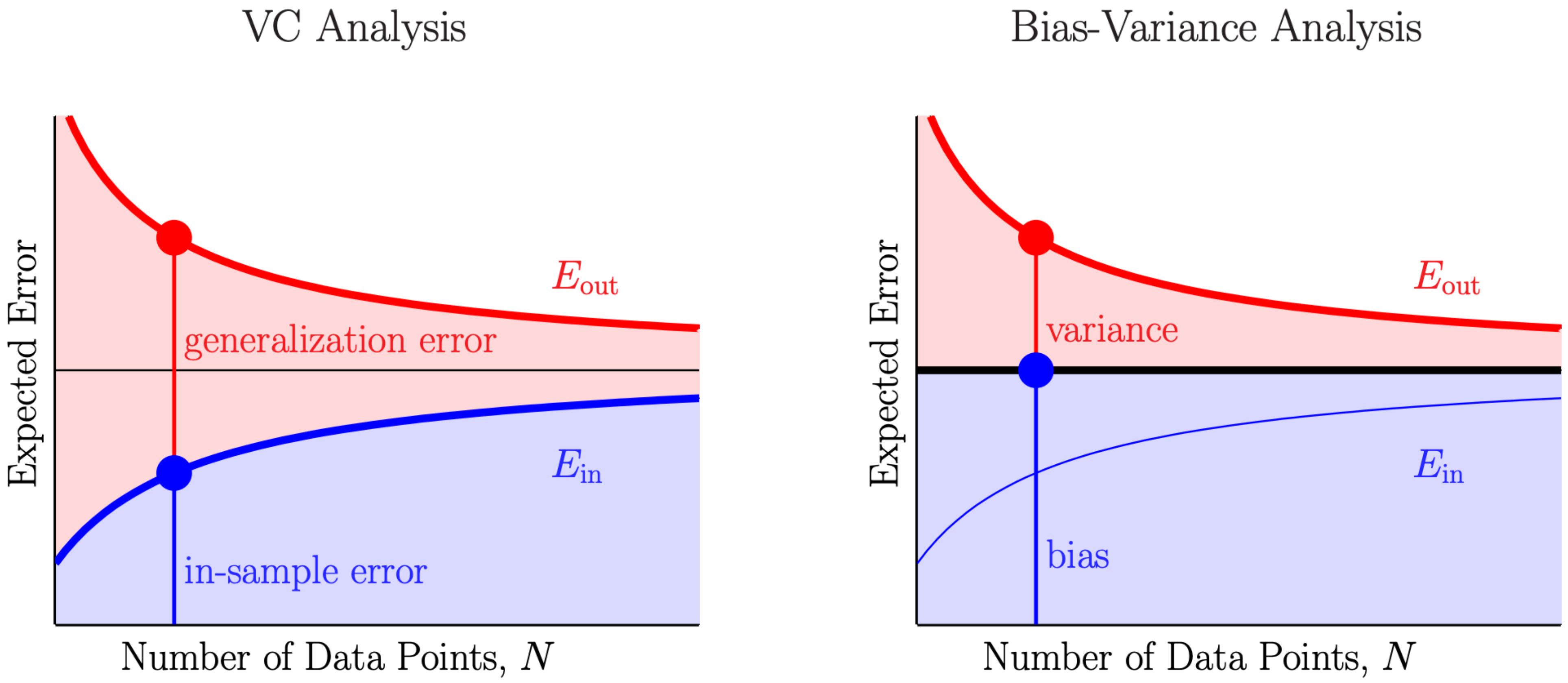
$$\mathbb{E}_D\left[E_{out}\left(g^{(D)}\right)\right] = \mathbb{E}_{\vec{x}}\left[\left(\bar{g}(\vec{x}) - f(\vec{x})\right)^2\right] + \mathbb{E}_{\vec{x}}\left[\mathbb{E}_D\left[\left(g^{(D)}(\vec{x}) - \bar{g}(\vec{x})\right)^2\right]\right]$$

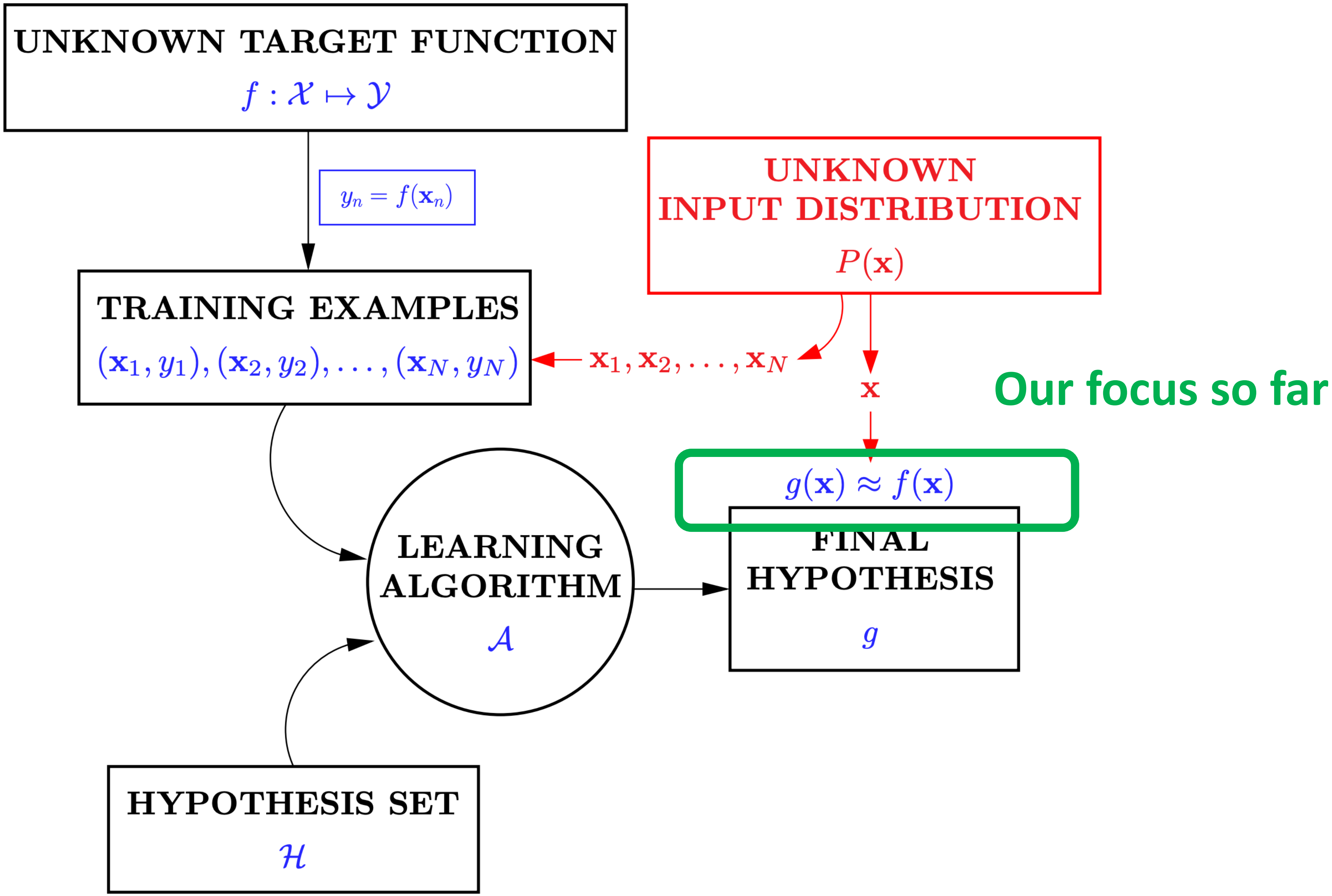
Learning Curves

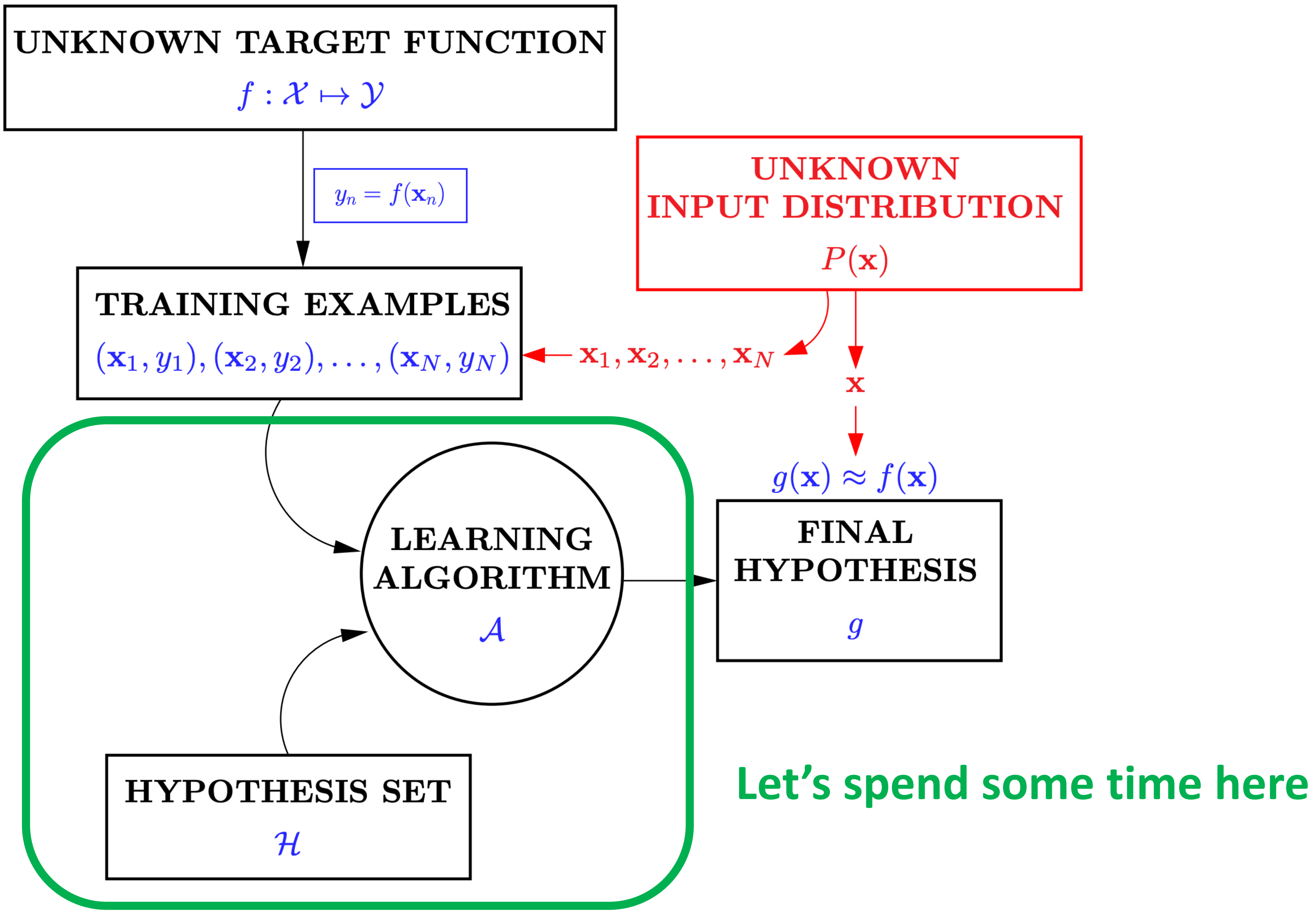




Learning Curves




UNKNOWN TARGET FUNCTION
$f: \mathcal{X} \mapsto \mathcal{Y}$
$y_n = f(\mathbf{x}_n)$
UNKNOWN
INPUT DISTRIBUTION
$P(\mathbf{x})$
TRAINING EXAMPLES
$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$
$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$
$\mathbf{x}$
Our focus so far
$g(\mathbf{x}) \approx f(\mathbf{x})$
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 spend some time here

Linear Models

Linear Models

This is why it's called linear models

- H contains hypothesis $h(\vec{x})$ as **some function of** $\boldsymbol{\vec{w}^T\vec{x}}$

	Domain	Model
Linear Classification	$y \in \{-1, +1\}$	$H = \{h(\vec{x}) = sign(\vec{w}^T\vec{x})\}$
Linear Regression	$y \in \mathbb{R}$	$H = \{h(\vec{x}) = \vec{w}^T\vec{x}\}$
Logistic Regression	$y \in [0,1]$	$H = \{h(\vec{x}) = \theta(\vec{w}^T\vec{x})\}$

<u>Credit Card Example</u>

Approve or not

Credit line

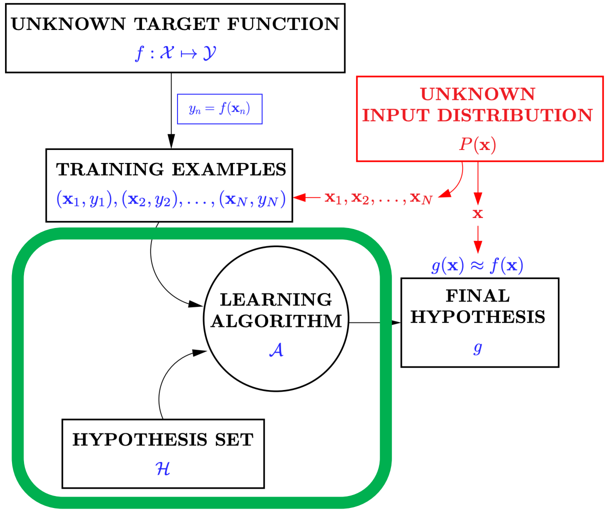
Prob. of default

$$\theta(s) = \frac{e^s}{1 + e^s}$$

- Linear models:
  - Simple models => Good generalization error

- Reminder:
  - We will **interchangeably use** $\boldsymbol{h}$ **and** $\boldsymbol{\vec{w}}$ to represent a hypothesis in linear models

Learning Algorithm?



- Goal of the algorithm: Find $g \in H$ that minimizes $E_{out}(g)$
  (We don't know E_{out})

- Common algorithms:
  - $g = argmin_{h \in H} E_{in}(h)$
    - Works well when the model is simple (generalization error is small)
    - Will focus on this in the discussion of linear models
  - $g = argmin_{h \in H} \{E_{in}(h) + \Omega(h)\}$
    - $\Omega(h)$: penalty for complex h
    - Will discuss this when we get to LFD Section 4

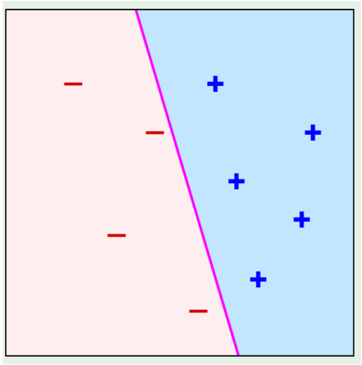
VC Bound: $E_{out}(g) \leq E_{in}(g) + O\left(\sqrt{d_{vc} \frac{\ln N}{N}}\right)$

- Optimization is a key component in machine learning

Linear Classification

	Domain	Model
Linear Classification	$y \in \{-1, +1\}$	$H = \{h(\vec{x}) = sign(\vec{w}^T \vec{x})\}$
Linear Regression	$y \in \mathbb{R}$	$H = \{h(\vec{x}) = \vec{w}^T \vec{x}\}$
Logistic Regression	$y \in [0,1]$	$H = \{h(\vec{x}) = \theta(\vec{w}^T \vec{x})\}$

Linear Classification (Perceptron)

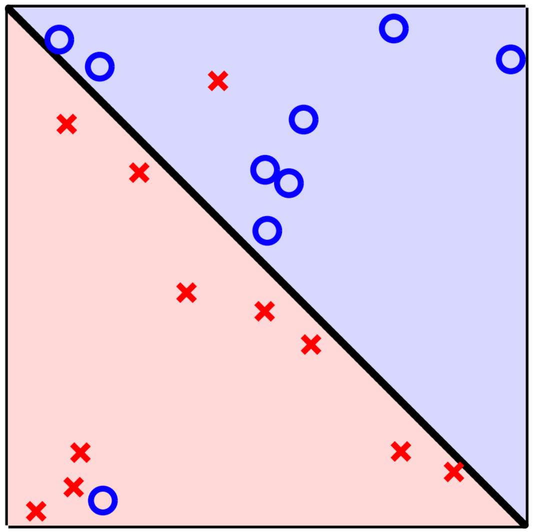


- Formulation
  - Hypothesis set $H = \{h(\vec{x}) = sign(\vec{w}^T \vec{x})\}$
  - Error measure: binary error $e(h(\vec{x}), y) = \mathbb{I}[h(\vec{x}) \neq y]$

- Property
  - Simple model (Fact: the VC dimension of d-dim perceptron is d+1)
  - Good generalization error

- When data is linearly separable
  - Run PLA
    => find g with $E_{in}(g) = 0$
    => $E_{out}(g)$ is close to $E_{in}(g) = 0$

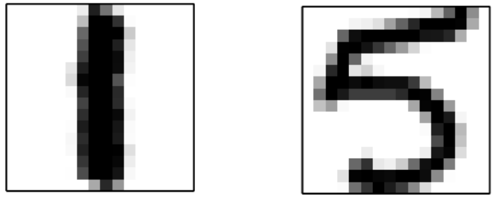
Non-Separable Data



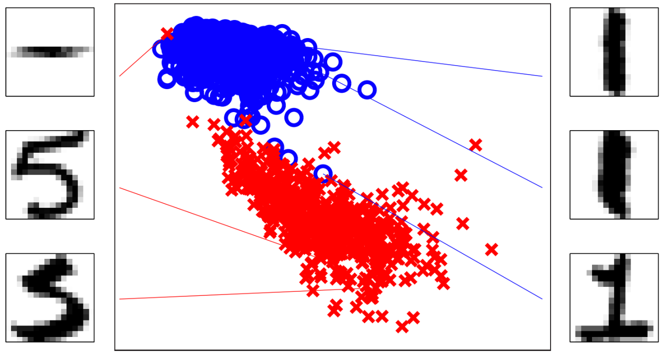
- Generally a hard problem
  - Minimizing E_{in} is NP-hard
  - Reason: binary error is discrete and hard to optimize
- Alternative approaches
  - Pocket algorithm
    - Run PLA for a finite pre-determined T rounds
    - Keep track of the best weights $\vec{w}^*$ ($\vec{w}(t)$ that minimizes E_{in})
  - Engineering the features to make data closer to be separable
    - Feature engineering (requiring domain knowledge, e.g., see LFD Example 3.1)
  - Non-linear transformation (will discuss this in later lectures)
  - Changing the problem formulation
    - Treat it as a logistic regression problem (what's the probability for the label to be +1)
    - Another example: Support vector machines in 2nd half of the semester

Example on Feature Engineering

- Task: Classify handwritten digits of 1 and 5



- Linearly separable?
  - What are the features $\vec{x}$?
    - Each pixel as a feature (deep neural network takes this approach. requires a lot of data)
    - $\vec{x} = (\text{intensity}, \text{symmtry})$



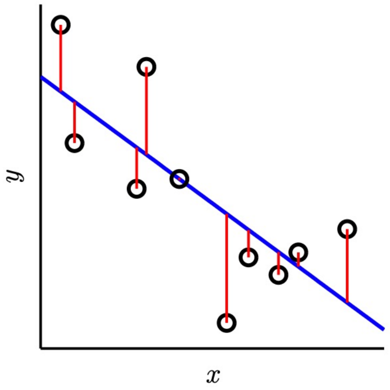
Feature engineering is a practical issue in applied ML but not the focus of this course (requires domain knowledge).

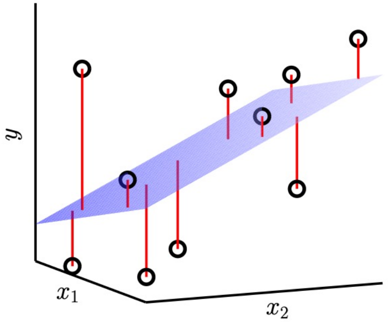
Linear Regression

	Domain	Model
Linear Classification	$y \in \{-1, +1\}$	$H = \{h(\vec{x}) = sign(\vec{w}^T\vec{x})\}$
Linear Regression	$y \in \mathbb{R}$	$H = \{h(\vec{x}) = \vec{w}^T\vec{x}\}$
Logistic Regression	$y \in [0,1]$	$H = \{h(\vec{x}) = \theta(\vec{w}^T\vec{x})\}$

Linear Regression

- Formulation
  - Hypothesis set $H = \{h(\vec{x}) = \vec{w}^T\vec{x}\}$
  - Squared error $e(h(\vec{x}), y) = (h(\vec{x}) - y)^2$





- Given dataset $D = \{(\vec{x}_1, y_1), \dots, (\vec{x}_N, y_N)\}$
  - $E_{in}(\vec{w}) = \frac{1}{N}\sum_{n=1}^{N}(\vec{w}^T\vec{x}_n - y_n)^2$

- Goal: find $\vec{w}_{lin} = argmin_{\vec{w}}\, E_{in}(\vec{w})$

Matrix Representation

- $D = \{(\vec{x}_1, y_1), \dots, (\vec{x}_N, y_N)\}$
- $X = \begin{bmatrix} \vec{x}_1^T \\ \vdots \\ \vec{x}_N^T \end{bmatrix} = \begin{bmatrix} x_{1,0} & x_{1,1} & \cdots & x_{1,d} \\ x_{2,0} & x_{2,1} & \cdots & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,0} & x_{N,1} & \cdots & x_{N,d} \end{bmatrix}$

$x_{n,i}$: the i-th element of vector $\vec{x}_n$

- $\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$

Predictions made by hypothesis $\vec{w}$

$$X\vec{w} = \begin{bmatrix} \vec{x}_1^T \vec{w} \\ \vdots \\ \vec{x}_N^T \vec{w} \end{bmatrix}$$

$$X\vec{w} - \vec{y} = \begin{bmatrix} \vec{x}_1^T \vec{w} - y_1 \\ \vdots \\ \vec{x}_N^T \vec{w} - y_N \end{bmatrix}$$

Rewriting the In-Sample Error In Matrix Form

$$E_{in}(\vec{w}) = \frac{1}{N}\sum_{n=1}^{N}(\vec{w}^T\vec{x}_n - y_n)^2$$

$$= \frac{1}{N}\sum_{n=1}^{N}(\vec{x}_n^T\vec{w} - y_n)^2$$

$$= \frac{1}{N}\|X\vec{w} - \vec{y}\|^2$$

$$= \frac{1}{N}(X\vec{w} - \vec{y})^{\mathrm{T}}(X\vec{w} - \vec{y})$$

$$X = \begin{bmatrix} \vec{x}_1^T \\ \vdots \\ \vec{x}_N^T \end{bmatrix}; \quad \vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

$$X\vec{w} = \begin{bmatrix} \vec{x}_1^T\vec{w} \\ \vdots \\ \vec{x}_N^T\vec{w} \end{bmatrix}$$

$$X\vec{w} - \vec{y} = \begin{bmatrix} \vec{x}_1^T\vec{w} - y_1 \\ \vdots \\ \vec{x}_N^T\vec{w} - y_N \end{bmatrix}$$

$$\|\vec{z}\| = \sqrt{\vec{z}^T\vec{z}} = \sqrt{\Sigma_{i=1}^{d} z_i^2}$$

$$\|\vec{z}\|^2 = \vec{z}^T\vec{z} = \Sigma_{i=1}^{d} z_i^2$$

$$\longrightarrow \quad E_{in}(\vec{w}) = \frac{1}{N}\left((X\vec{w})^T - \vec{y}^{\mathrm{T}}\right)(X\vec{w} - \vec{y})$$

$$= \frac{1}{N}(\vec{w}^T X^T X\vec{w} - 2\vec{w}^T X^T\vec{y} + \vec{y}^T\vec{y})$$

How to find $\vec{w}_{lin} = argmin_{\vec{w}}\ E_{in}(\vec{w})$?

- Given $E_{in}(\vec{w}) = \frac{1}{N}(\vec{w}^T X^T X\vec{w} - 2\vec{w}^T X^T \vec{y} + \vec{y}^T \vec{y})$
- Solve for $\nabla_{\vec{w}} E_{in}(\vec{w}) = 0$
  - Think about what you'll do for one-dimensional case

$$\nabla f(\vec{w}) = \nabla_{\vec{w}} f(\vec{w}) = \begin{bmatrix} \frac{\partial}{\partial w_0} f(\vec{w}) \\ \frac{\partial}{\partial w_1} f(\vec{w}) \\ \vdots \\ \frac{\partial}{\partial w_d} f(\vec{w}) \end{bmatrix}$$

- Derivations
  - $E_{in}(\vec{w}) = \frac{1}{N}(\vec{w}^T X^T X\vec{w} - 2\vec{w}^T X^T \vec{y} + \vec{y}^T \vec{y})$
  - $\nabla_{\vec{w}} E_{in}(\vec{w}) = \frac{1}{N}(2X^T X\vec{w} - 2X^T \vec{y})$
  - $\nabla_{\vec{w}} E_{in}(\vec{w}_{lin}) = 0$ ==> $X^T X\vec{w}_{lin} = X^T \vec{y}$

- $X^T X \vec{w}_{lin} = X^T \vec{y}$

- Two cases:
  - If $X^T X$ is invertible (When $N \gg d$, most of the time, it is invertible)
    - $\vec{w}_{lin} = (X^T X)^{-1} X^T \vec{y}$
  - If $X^T X$ is not invertible
    - Requires special handling (See LFD Problem 3.15 for an example)

- In practice
  - Define $X^\dagger$ as the pseudo-inverse of X
    - When $X^T X$ is invertible, $X^\dagger = (X^T X)^{-1} X^T$
    - When $X^T X$ is not invertible, "handle" it appropriately (usually done in the library for you)

- Linear regression algorithm (a single step algorithm):
  - $\vec{w}_{lin} = X^\dagger \vec{y}$

Linear Regression “Algorithm”

- Input: $D = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \dots, (\vec{x}_N, y_N)\}$

1. Construct X and $\vec{y}$
2. Compute the pseudo-inverse of X: $X^\dagger$
   ($X^\dagger = (X^T X)^{-1} X^T$ when $(X^T X)$ is invertible)
3. Compute $\vec{w}_{lin} = \mathrm{X}^\dagger \vec{y}$

- Output: $\vec{w}_{lin}$

Short Discussion

Linear Regression "Algorithm"

- Input: $D = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \dots, (\vec{x}_N, y_N)\}$

1. Construct X and $\vec{y}$
2. Compute the pseudo-inverse of X: $X^\dagger$
   ($X^\dagger = (X^T X)^{-1} X^T$ when $(X^T X)$ is invertible)
3. Compute $\vec{w}_{lin} = \mathrm{X}^\dagger \vec{y}$

- Output: $\vec{w}_{lin}$

- What happens in 0-dimensional model
  - $\vec{x} = (x_0)$
  - Given $D = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \dots, (\vec{x}_N, y_N)\}$
  - What's $\vec{w}_{lin}$

Short Discussion

Linear Regression "Algorithm"

- Input: $D = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \dots, (\vec{x}_N, y_N)\}$

1. Construct X and $\vec{y}$
2. Compute the pseudo-inverse of X: $X^\dagger$
   ($X^\dagger = (X^TX)^{-1}X^T$ when (X^TX) is invertible)
3. Compute $\vec{w}_{lin} = \mathrm{X}^\dagger \vec{y}$

- Output: $\vec{w}_{lin}$

- Special case of zero–dimensional space

$$X = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \Rightarrow X^TX = N \Rightarrow (X^TX)^{-1} = 1/\mathrm{N}$$

$$\vec{w}_{lin} = (X^TX)^{-1}X^T\vec{y}$$

$$= \begin{bmatrix} \frac{1}{N} \cdots \frac{1}{N} \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = \frac{1}{N}\sum_{n=1}^{N} y_n$$

Squared error => mean

Discussion

- Linear regression generalizes very well
  - Under mild conditions (See LFD Exercise 3.4 for an example)

$$E_{out}(g) = E_{in}(g) + O\left(\frac{d}{N}\right)$$

- Use regression for classification
  - Note that $\{-1, +1\} \subset \mathbb{R}$
  - Use linear regression to find $\vec{w}_{lin} = (X^T X)^{-1} X^T \vec{y}$ for data with $y \in \{-1, +1\}$
  - Use $\vec{w}_{lin}$ for classification: $g(\vec{x}) = \text{sign}\left(\vec{w}_{lin}^T \vec{x}\right)$

  - Alternatively, use $\vec{w}_{lin}$ as the initialization for Pocket Algorithm

Logistic Regression

	Domain	Model
Linear Classification	$y \in \{-1, +1\}$	$H = \{h(\vec{x}) = sign(\vec{w}^T\vec{x})\}$
Linear Regression	$y \in \mathbb{R}$	$H = \{h(\vec{x}) = \vec{w}^T\vec{x}\}$
Logistic Regression	$y \in [0,1]$	$H = \{h(\vec{x}) = \theta(\vec{w}^T\vec{x})\}$

Logistic Regression: Predicting a Probability

- Will this patient have a heart attack within the next year?

age	62 years
gender	male
blood sugar	120 mg/dL40,000
HDL	50
LDL	120
Mass	190 lbs
Height	$5' \ 10''$
...	...

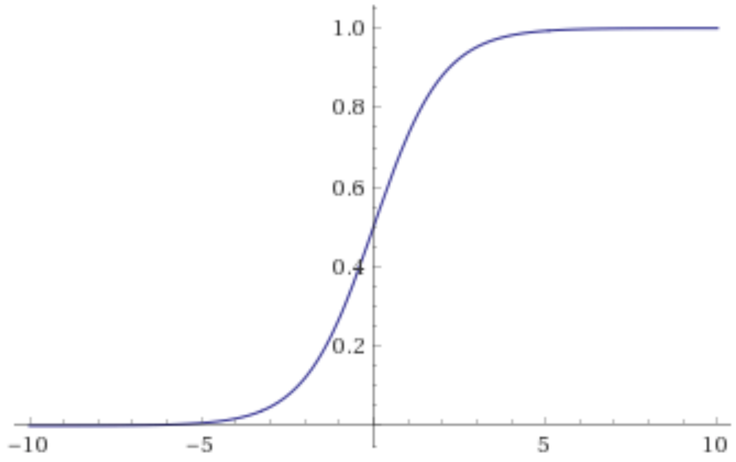
Classification: Yes/No

Logistic regression: Probability of Yes

- A hypothesis $h(\vec{x})$ outputs a value in $[0,1]$
  - Interpreting it as the probability of yes

Logistic Regression: Predicting a Probability

- Hypothesis set $H = \{h(\vec{x}) = \theta(\vec{w}^T \vec{x})\}$
  - Want θ to map from $(-\infty, \infty)$ to $[0,1]$
- $\theta(s) = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}}$
  - A sigmoid function (”S”-shaped function)
  - $\theta(s) = \begin{cases} 1 & \text{when } s \to \infty \\ 0.5 & \text{when } s = 0 \\ 0 & \text{when } s \to -\infty \end{cases}$
- Useful property
  - $1 - \theta(s) = \frac{1+e^s}{1+e^s} - \frac{e^s}{1+e^s} = \frac{1}{1+e^s} = \theta(-s)$



What Kind of Dataset do We Get?

- Dataset $D = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}$

$\vec{x} =$

age	62 years
gender	male
blood sugar	120 mg/dL40,000
HDL	50
LDL	120
Mass	190 lbs
Height	$5'\ 10''$
...	...

- What are the values of y_n?
  - Ideally, we want to have y_n to be the probability value
  - In practice, we cannot measure a probability
  - We can only see the occurrence of an event and infer the probability
  - (We often only observe whether the person had heart attack, we don't observe the "probability")

- Need to address the case when $y_n \in \{-1, +1\}$ in the given dataset D

Error Measure: Quantifying $g \approx f$

Side note:
You probably can guess why the property $1 - \theta(s) = \theta(-s)$ might be helpful

- Target function $f(\vec{x}) = \Pr(y = +1|\vec{x})$
  - Another way to write it: $\Pr(y|\vec{x}) = \begin{cases} f(\vec{x}) & \text{for } y = +1 \\ 1 - f(\vec{x}) & \text{for } y = -1 \end{cases}$

- How do we define the error measure to quantify $g \approx f$
  - Ideally, we want it to be meaningful
    - Binary error for classification: tell us the number of mistakes we make
    - Squared error for regression: the error minimizer is the ”mean (average)”
  - We also want it to be easy to optimize

Cross Entropy Error

$$E_{in}(\vec{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln(1 + e^{-y_n \vec{w}^T \vec{x}_n})$$

- It looks complicated, but
  - It has nice interpretations (min error => max likelihood)
  - It is easy to optimize (continuous, differentiable, convex)

Minimizing Cross Entropy Error

Maximizing Likelihood

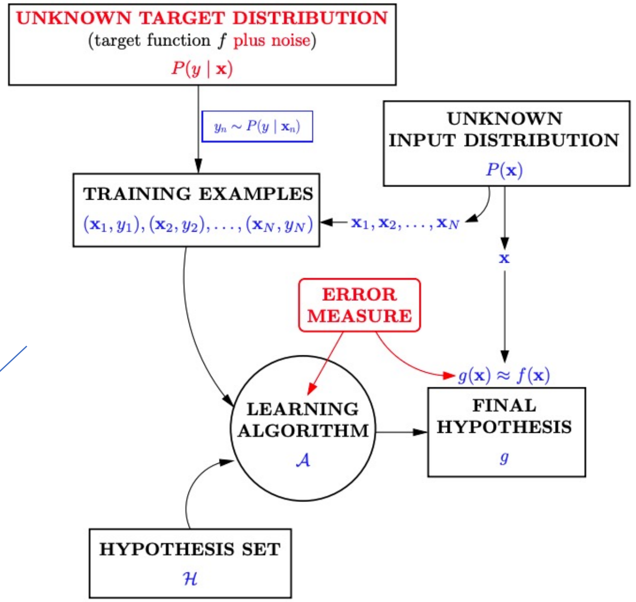
Maximum Likelihood Estimation

- Likelihood $\Pr(D|h)$
  - The probability of seeing dataset D if D is generated according to h
  - $\Pr(D|h) = \Pr((\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)|h)$
  - Maximum likelihood estimation (MLE)
    - $g = argmax_{h \in H} \Pr(D|h)$

- Sidenote: Two different concepts in ML
  - Likelihood: $\Pr(D|h)$ [Focus of this course]
  - Posterior: $\Pr(h|D)$ [Focus of Bayesian machine learning: More in 515T]

  - Connection: $\Pr(h|D) = \frac{\Pr(h)\Pr(D|h)}{\Pr(D)}$
    - Prior $\Pr(h)$: the additional assumption Bayesian ML makes

Write Down the Likelihood



- How are $D = \{(\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)\}$ generated?
  - $(\vec{x}_1, \ldots, \vec{x}_N)$ are i.i.d. drawn from a distribution
  - $(y_1, \ldots, y_N)$ are labeled according to target function $f(\vec{x})$

- Likelihood $\Pr(D|h)$
  - The probability of seeing dataset D if D is generated according to h
  - $\Pr(D|h) = \Pr((\vec{x}_1, y_1), \ldots, (\vec{x}_N, y_N)|h)$

$$= \Pr(\vec{x}_1, \ldots, \vec{x}_N) \Pr\big((y_1, \ldots, y_N)|(\vec{x}_1, \ldots, \vec{x}_N), h\big)$$

$$= \prod_{n=1}^{N} \Pr(\vec{x}_n) \prod_{n=1}^{N} \Pr(y_n|\vec{x}_n, h)$$

(Assumption of independent data)

Maximum Likelihood

- Choosing the hypothesis that maximizes the likelihood
  - $g = argmax_{h \in H} \Pr(D|h)$
    $= argmax_{h \in H} \prod_{n=1}^{N} \Pr(\vec{x}_n) \ \prod_{n=1}^{N} \Pr(y_n|\vec{x}_n, h)$
    $= argmax_{h \in H} \prod_{n=1}^{N} \Pr(y_n|\vec{x}_n, h)$

$\prod_{n=1}^{N} \Pr(\vec{x}_n)$ doesn't depend on h

- We interpret $h(\vec{x})$ as the probability of $y = +1$
  - $\Pr(y|\vec{x}, h) = \begin{cases} h(\vec{x}) = \theta(\vec{w}^T \vec{x}) & \text{for } y = +1 \\ 1 - h(\vec{x}) = 1 - \theta(\vec{w}^T \vec{x}) & \text{for } y = -1 \end{cases}$

- Since $1 - \theta(s) = \theta(-s)$
  - $\Pr(y|\vec{x}, h) = \theta(y\, \vec{w}^T \vec{x})$

Maximum Likelihood

- Choosing the hypothesis that maximizes the likelihood
  - $g = argmax_{h \in H} \Pr(D|h)$
    $= argmax_{h \in H} \prod_{n=1}^{N} \Pr(y_n|\vec{x}_n, h)$

  - $\vec{w}^* = argmax_{\vec{w}} \prod_{n=1}^{N} \theta(y_n \vec{w}^T \vec{x}_n)$
    $= argmax_{\vec{w}} \ln(\prod_{n=1}^{N} \theta(y_n \vec{w}^T \vec{x}_n)\ )$
    $= argmax_{\vec{w}} \sum_{n=1}^{N} \ln(\theta(y_n \vec{w}^T \vec{x}_n))$
    $= argmin_{\vec{w}} - \sum_{n=1}^{N} \ln(\theta(y_n \vec{w}^T \vec{x}_n))$
    $= argmin_{\vec{w}} \sum_{n=1}^{N} \ln \frac{1}{\theta(y_n \vec{w}^T \vec{x}_n)}$
    $= argmin_{\vec{w}} \sum_{n=1}^{N} \ln(1 + e^{-y_n \vec{w}^T \vec{x}_n})$
    $= argmin_{\vec{w}} \boxed{\frac{1}{N} \sum_{n=1}^{N} \ln(1 + e^{-y_n \vec{w}^T \vec{x}_n})}$

$$\boxed{\theta(s) = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}}}$$

$$E_{in}(\vec{w}) = \frac{1}{N} \sum_{n=1}^{N} \ln(1 + e^{-y_n \vec{w}^T \vec{x}_n})$$

Cross Entropy Error

$$E_{in}(\vec{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln(1 + e^{-y_n\vec{w}^T\vec{x}_n})$$

- Minimizing $E_{in}(\vec{w})$ is the same as maximizing likelihood
- Next question: How to solve $\vec{w}^* = argmin_{\vec{w}} E_{in}(\vec{w})$
    - Answer: Solve for $\nabla_{\vec{w}} E_{in}(\vec{w}) = 0$
    - No single-step solution like we have in linear regression