

CSE 417T

# Introduction to Machine Learning

Lecture 8

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: October 27
    - We expect to finish the content for exam1 several lectures before the exam.
  - Exam 2: December 8

Recap

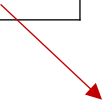
# Linear Models

This is why it's called linear models



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

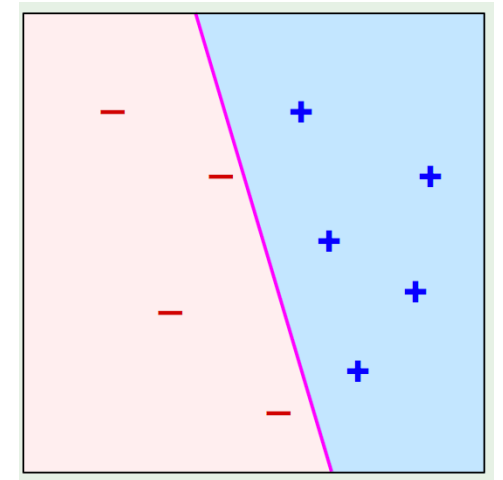
	Domain	Model	Credit Card Example
Linear Classification	$y \in \{-1, +1\}$	$H = \{h(\vec{x}) = \text{sign}(\vec{w}^T \vec{x})\}$	Approve or not
Linear Regression	$y \in \mathbb{R}$	$H = \{h(\vec{x}) = \vec{w}^T \vec{x}\}$	Credit line
Logistic Regression	$y \in [0,1]$	$H = \{h(\vec{x}) = \theta(\vec{w}^T \vec{x})\}$	Prob. of default


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

- Algorithm:
  - Focus on  $g = \operatorname{argmin}_{h \in H} E_{in}(h)$

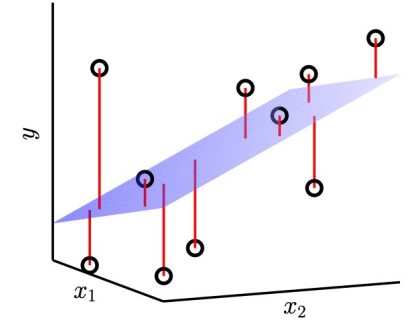
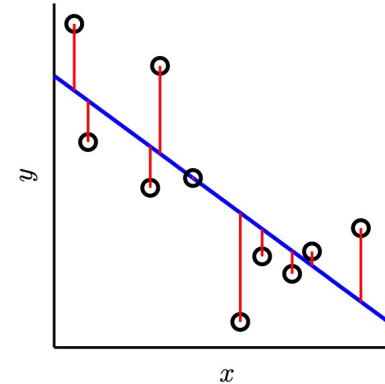
# Linear Classification (Perceptron)

- Formulation
  - Hypothesis set  $H = \{h(\vec{x}) = \text{sign}(\vec{w}^T \vec{x})\}$
  - Error measure: binary error  $e(h(\vec{x}), y) = \mathbb{I}[h(\vec{x}) \neq y]$
- Data is linearly separable
  - Run PLA  $\Rightarrow E_{in} = 0 \Rightarrow$  Low  $E_{out}$
- Data is not linearly separable
  - Minimizing  $E_{in}$  is NP hard
  - Pocket algorithm
  - Engineering the features (e.g., handwritten digits)
  - More discussion later in the semester



# 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  $\vec{w}_{lin} = \operatorname{argmin}_{\vec{w}} E_{in}(\vec{w})$



# Linear Regression “Algorithm”

- There is a closed-form solution for minimizing  $E_{in}$ 
  - Closed-form solution for  $\nabla_{\vec{w}} E_{in}(\vec{w}) = 0$  ( $E_{in}$  is convex)
- One-step algorithm
  - Given  $D = \{(\vec{x}_1, y_1), \dots, (\vec{x}_N, y_N)\}$

- Construct  $X = \begin{bmatrix} \vec{x}_1^T \\ \vdots \\ \vec{x}_N^T \end{bmatrix} = \begin{bmatrix} x_{1,0} & x_{1,1} & \cdots & x_{1,d} \\ x_{2,0} & x_{2,1} & \cdots & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,0} & x_{N,1} & \cdots & x_{N,d} \end{bmatrix}$  and  $\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$

- Output  $\vec{w}_{lin} = (X^T X)^{-1} X^T \vec{y}$  (Assume  $X^T X$  is invertible)

# 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.



# Logistic Regression

	Domain	Model
Linear Classification	$y \in \{-1, +1\}$	$H = \{h(\vec{x}) = \text{sign}(\vec{w}^T \vec{x})\}$
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  $\theta$  to map from  $(-\infty, \infty)$  to  $[0,1]$

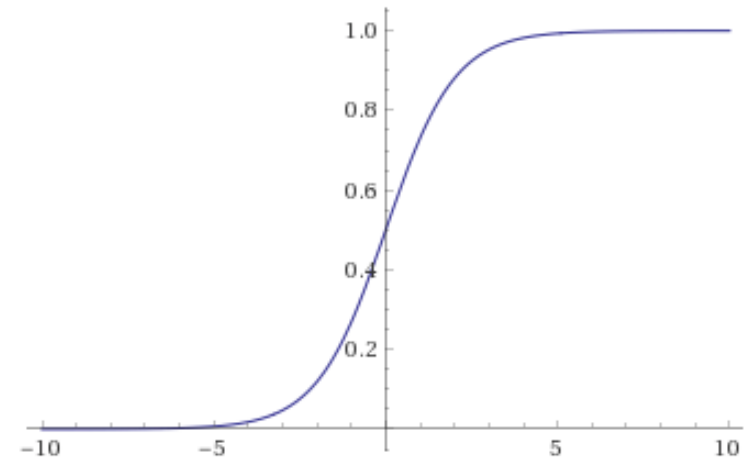
- $\theta(s) = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}}$

- A sigmoid function ("S"-shaped function)

- $\theta(s) = \begin{cases} 1 & \text{when } s \rightarrow \infty \\ 0.5 & \text{when } s = 0 \\ 0 & \text{when } s \rightarrow -\infty \end{cases}$

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

$\vec{x} =$

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  $h$
  - $\Pr(D|h) = \Pr((\vec{x}_1, y_1), \dots, (\vec{x}_N, y_N)|h)$
  - Maximum likelihood estimation (MLE)
    - $g = \operatorname{argmax}_{h \in H} \Pr(D|h)$
- 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  $D = \{(\vec{x}_1, y_1), \dots, (\vec{x}_N, y_N)\}$  generated?
  - $(\vec{x}_1, \dots, \vec{x}_N)$  are i.i.d. drawn from a distribution
  - $(y_1, \dots, y_N)$  are labeled according to target function  $f(\vec{x})$

- Likelihood  $\Pr(D|h)$

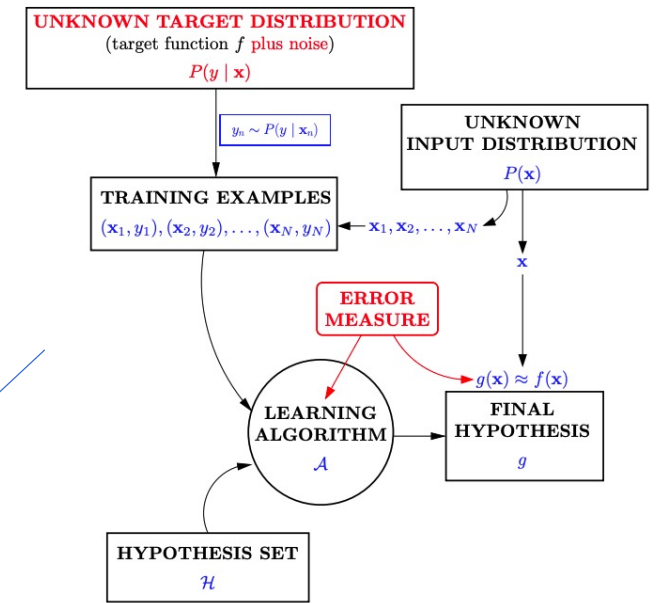
- The probability of seeing dataset  $D$  if  $D$  is generated according to  $h$

- $\Pr(D|h) = \Pr((\vec{x}_1, y_1), \dots, (\vec{x}_N, y_N)|h)$

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

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

(Assumption of independent data)



# Maximum Likelihood

- Choosing the hypothesis that maximizes the likelihood

- $g = \operatorname{argmax}_{h \in H} \Pr(D|h)$   
 $= \operatorname{argmax}_{h \in H} \prod_{n=1}^N \Pr(\vec{x}_n) \prod_{n=1}^N \Pr(y_n|\vec{x}_n, h)$   
 $= \operatorname{argmax}_{h \in H} \prod_{n=1}^N \Pr(y_n|\vec{x}_n, h)$

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

- $g = \operatorname{argmax}_{h \in H} \Pr(D|h)$   
 $= \operatorname{argmax}_{h \in H} \prod_{n=1}^N \Pr(y_n|\vec{x}_n, h)$

- $\vec{w}^* = \operatorname{argmax}_{\vec{w}} \prod_{n=1}^N \theta(y_n \vec{w}^T \vec{x}_n)$   
 $= \operatorname{argmax}_{\vec{w}} \ln(\prod_{n=1}^N \theta(y_n \vec{w}^T \vec{x}_n))$   
 $= \operatorname{argmax}_{\vec{w}} \sum_{n=1}^N \ln(\theta(y_n \vec{w}^T \vec{x}_n))$   
 $= \operatorname{argmin}_{\vec{w}} - \sum_{n=1}^N \ln(\theta(y_n \vec{w}^T \vec{x}_n))$   
 $= \operatorname{argmin}_{\vec{w}} \sum_{n=1}^N \ln \frac{1}{\theta(y_n \vec{w}^T \vec{x}_n)}$   
 $= \operatorname{argmin}_{\vec{w}} \sum_{n=1}^N \ln(1 + e^{-y_n \vec{w}^T \vec{x}_n})$   
 $= \operatorname{argmin}_{\vec{w}} \frac{1}{N} \sum_{n=1}^N \ln(1 + e^{-y_n \vec{w}^T \vec{x}_n})$

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

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

# 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  $\vec{w}^* = \operatorname{argmin}_{\vec{w}} E_{in}(\vec{w})$ 
  - Answer: Solve for  $\nabla_{\vec{w}} E_{in}(\vec{w}) = 0$
  - No single-step solution like we have in linear regression

# Using Logistic Regression for Classification

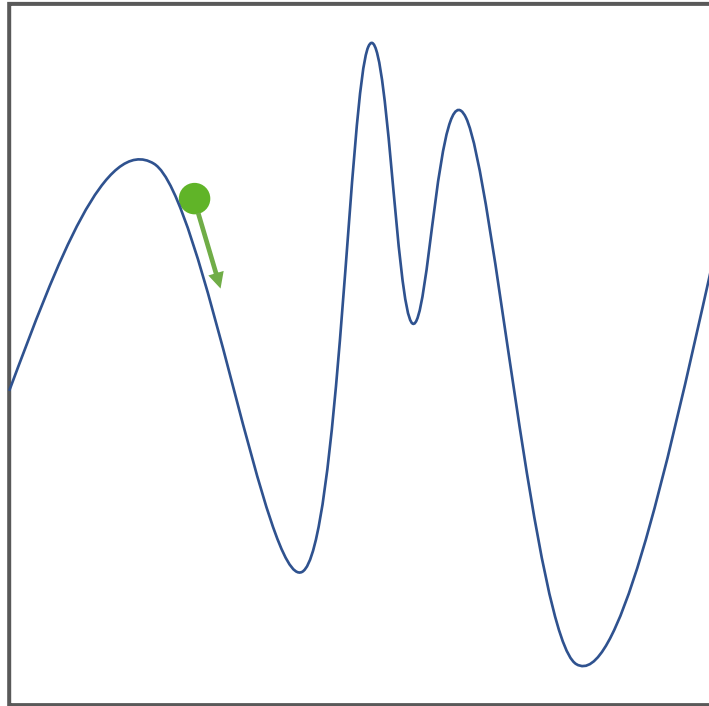
- Let  $\vec{w}^*$  or  $g$  be the learned logistic regression model, how can we make classification predictions using  $\vec{w}^*$ ?
- Set a cutoff probability  $C\%$  (e.g., 50%).
  - Classify +1 if  $g(\vec{x}) = \theta(\vec{w}^{*T} \vec{x}) > C\%$
  - Classify -1 if  $g(\vec{x}) = \theta(\vec{w}^{*T} \vec{x}) < C\%$
- When  $C$  is 50 (a common choice)
  - $\theta(\vec{w}^{*T} \vec{x}) > 50\% \Rightarrow \vec{w}^{*T} \vec{x} > 0$
  - Equivalent to using  $\vec{w}^*$  as the linear classification hypothesis, i.e.,  $g(\vec{x}) = \text{sign}(\vec{w}^{*T} \vec{x})$

# Gradient Descent

A general optimization technique

# Gradient Descent

- A technique for optimizing functions that **gradients exist everywhere**.



- An iterative method that converges to local optimum.
- Converge to global optimum if the function is convex (since there is only one local optimum).

# Gradient Descent: Minimizing $E_{in}(\vec{w})$

- An iterative method of the form:

$$\vec{w}(t + 1) \leftarrow \vec{w}(t) + \eta_t \vec{v}_t$$

- $\vec{v}_t$ : a unit vector, determining the direction of the update
- $\eta_t$ : a scalar, determining how much to update
- How to choose  $\vec{v}_t$  and  $\eta_t$ ?



# Choosing $\vec{v}_t$ in $\vec{w}(t+1) \leftarrow \vec{w}(t) + \eta_t \vec{v}_t$

- Intuition: Choose  $\vec{v}_t$  that moves towards the “steepest” direction
  - Approaching the minimum faster

- Taylor’s approximation:

- $E_{in}(\vec{w}(t) + \eta_t \vec{v}_t) = E_{in}(\vec{w}(t)) + \eta_t \nabla_{\vec{w}} E_{in}(\vec{w}(t))^T \vec{v}_t + O(\eta_t^2)$

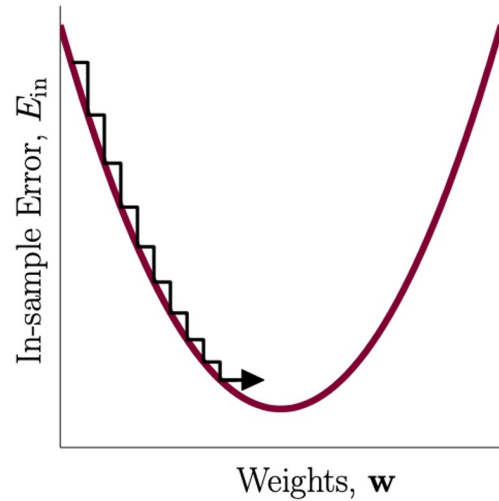
- $E_{in}(\vec{w}(t+1)) - E_{in}(\vec{w}(t)) \approx \eta_t \nabla_{\vec{w}} E_{in}(\vec{w}(t))^T \vec{v}_t$

$\eta_t$  is usually small, so ignore this term

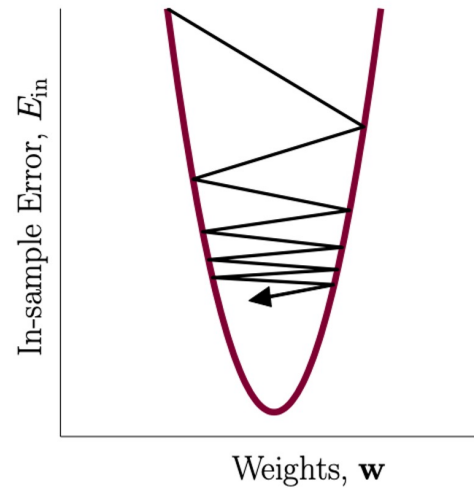
- Choose unit vector  $\vec{v}_t$  that minimizes  $\nabla_{\vec{w}} E_{in}(\vec{w}(t))^T \vec{v}_t$ 
  - $\vec{v}_t$  should be in the opposite direction of  $\nabla_{\vec{w}} E_{in}(\vec{w}(t))$
  - $\vec{v}_t = \frac{-\nabla_{\vec{w}} E_{in}(\vec{w}(t))}{\|\nabla_{\vec{w}} E_{in}(\vec{w}(t))\|}$

Choosing  $\eta_t$  in  $\vec{w}(t+1) \leftarrow \vec{w}(t) + \eta_t \vec{v}_t$

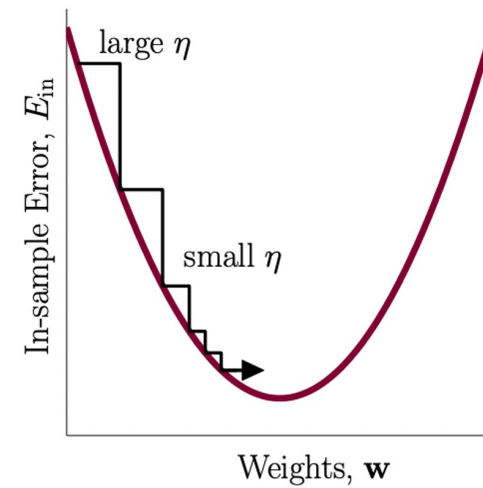
$\eta$  too small



$\eta$  too large

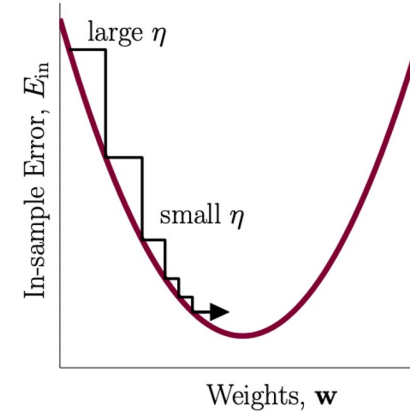


variable  $\eta_t$  – just right



Choosing  $\eta_t$  in  $\vec{w}(t + 1) \leftarrow \vec{w}(t) + \eta_t \vec{v}_t$

- Intuition (for convex  $E_{in}$ )
  - When  $E_{in}$  is closer to the minimum,
    - $\nabla_{\vec{w}} E_{in}(\vec{w}(t))$  is smaller
    - We should set  $\eta_t$  smaller
- Therefore, set  $\eta_t = \eta \|\nabla_{\vec{w}} E_{in}(\vec{w}(t))\|$



# Putting Them Together

- Iterative update rule:  $\vec{w}(t + 1) \leftarrow \vec{w}(t) + \eta_t \vec{v}_t$

- $\vec{w}(t + 1) \leftarrow \vec{w}(t) - \eta \nabla_{\vec{w}} E_{in}(\vec{w}(t))$

$$\vec{v}_t = \frac{-\nabla_{\vec{w}} E_{in}(\vec{w}(t))}{\|\nabla_{\vec{w}} E_{in}(\vec{w}(t))\|}$$

$$\eta_t = \eta \|\nabla_{\vec{w}} E_{in}(\vec{w}(t))\|$$

- Gradient calculations

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

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

# Gradient Descent for Logistic Regression

- Initialize  $\vec{w}(0)$
- For  $t = 0, \dots$ 
  - Compute  $\nabla_{\vec{w}} E_{in}(\vec{w}(t)) = -\frac{1}{N} \sum_{n=1}^N \frac{y_n \vec{x}_n}{1 + e^{y_n \vec{w}(t)^T \vec{x}_n}}$
  - $\vec{w}(t + 1) \leftarrow \vec{w}(t) - \eta \nabla_{\vec{w}} E_{in}(\vec{w}(t))$
  - Terminate if the **stop conditions** are met
- Return the final weights

$\eta$ : learning rate.  
A parameter the learner can choose.

# Gradient Descent for Logistic Regression

- Initialization
  - Random initialization is a good idea and a common approach
  - (we specify the initialization in HW2 mostly for grading purposes)
- Stop conditions (a mix of the following criteria)
  - When the **number of iteration** exceeds the pre-set threshold
  - When the **improvement on  $E_{in}$**  (e.g., check  $\nabla_{\vec{w}} E_{in}$ ) is too small
  - When  **$E_{in}$  is small** enough

# Computation of Gradient Descent

- Gradient Descent for Logistic Regression

- Initialize  $\vec{w}(0)$

- For  $t = 0, \dots$

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

- $\vec{w}(t+1) \leftarrow \vec{w}(t) - \eta \nabla_{\vec{w}} E_{in}(\vec{w}(t))$

- Terminate if the **stop conditions** are met

- Return the final weights

- Which step requires the most computation?

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

- The time complexity is  $O(N)$

- $N$  is large for big datasets

# Stochastic Gradient Descent



# Deal with Heavy Computation of $\nabla_{\vec{w}} E_{in}(\vec{w})$

- Speed up the implementation of  $\nabla_{\vec{w}} E_{in}(\vec{w}) = -\frac{1}{N} \sum_{n=1}^N \frac{y_n \vec{x}_n}{1 + e^{y_n \vec{w}^T \vec{x}_n}}$ 
  - **Vectorization** can make your HW2 running time in **several order of magnitudes** faster
- Example:
  - Given  $[x_1, \dots, x_N]$ , want to calculate  $[e^{x_1}, \dots, e^{x_N}]$
  - Using for loop:
    - Loop from  $n=1$  to  $N$ , calculate  $e^{x_n}$
  - Vectorized method:
    - Using numpy library: `np.exp([x1, ..., xN])`
- Why? Matrix operations are optimized in a low level using numpy operations (or other scientific computing libraries).
  - Try to replace loops with numpy matrix operations in your HW2

# Deal with Heavy Computation of $\nabla_{\vec{w}} E_{in}(\vec{w})$

- Speed up the implementation of  $\nabla_{\vec{w}} E_{in}(\vec{w}) = -\frac{1}{N} \sum_{n=1}^N \frac{y_n \vec{x}_n}{1 + e^{y_n \vec{w}^T \vec{x}_n}}$ 
  - **Vectorization** can make your HW2 running time in **several order of magnitudes** faster
- Solve  $\nabla_{\vec{w}} E_{in}(\vec{w})$  "in expectation"
  - Define  $e_n(\vec{w}) = \ln(1 + e^{-y_n \vec{w}^T \vec{x}_n})$ , the point-wise error caused by  $(\vec{x}_n, y_n)$
  - Observe that
    - $E_{in}(\vec{w}) = \frac{1}{N} \sum_{n=1}^N e_n(\vec{w})$
    - $\nabla_{\vec{w}} E_{in}(\vec{w}) = \frac{1}{N} \sum_{n=1}^N \nabla_{\vec{w}} e_n(\vec{w})$
  - Draw a point  $\vec{x}_n$  from  $\{\vec{x}_1, \dots, \vec{x}_N\}$  uniformly at random
    - $E_{\vec{x}_n}[\nabla_{\vec{w}} e_n(\vec{w})] = \nabla_{\vec{w}} E_{in}(\vec{w})$

# Stochastic Gradient Descent (SGD)

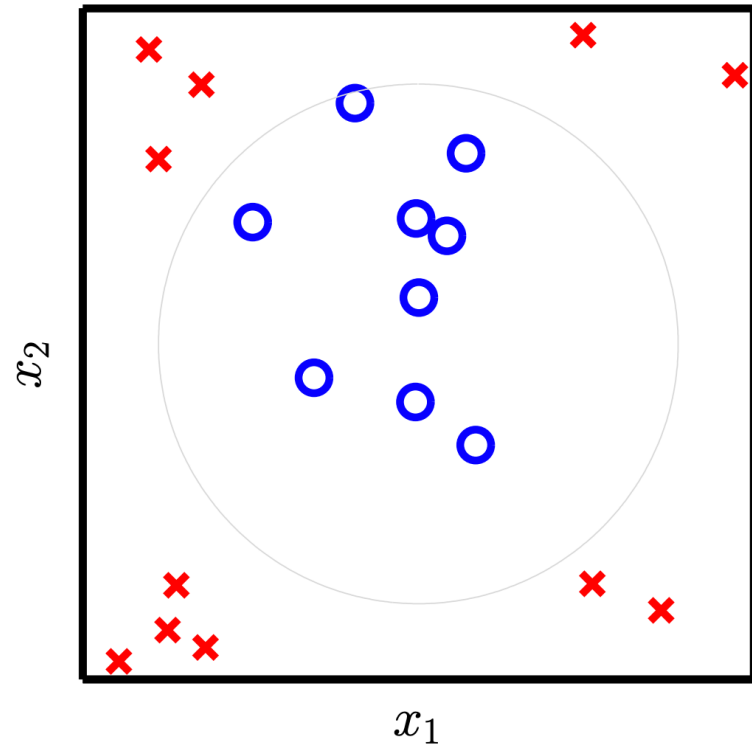
- Algorithm
  - Initialize  $\vec{w}(0)$
  - For  $t = 0, \dots$ 
    - Randomly choose  $n$  from  $\{1, \dots, N\}$
    - $\vec{w}(t + 1) \leftarrow \vec{w}(t) - \eta \nabla_{\vec{w}} e_n(\vec{w}(t))$
    - Terminate if the stop conditions are met
  - Return the final weights
- $\mathbb{E}[\nabla_{\vec{w}} e_n(\vec{w})] = \nabla_{\vec{w}} E_{in}(\vec{w})$ 
  - SGD is doing the same thing as GD **in expectation**
    - More efficient (scale to large dataset), suitable for online data, helps escaping local min, etc.
    - Noisier, harder to define stop criteria

# Mini-Batch Gradient Descent

- GD: Computationally heavy, stable updates
- SGD: Computationally light, noisy updates
- Middle ground: Mini-Batch Gradient Descent
  - In each iteration, randomly choose  $k$  points  $\{n(1), \dots, n(k)\}$
  - Update rule
    - $\vec{w}(t + 1) \leftarrow \vec{w}(t) - \eta \frac{1}{k} \sum_{i=1}^k \nabla_{\vec{w}} e_{n(i)}(\vec{w}(t))$
- Side note about HW2
  - Please report your results on GD (non-stochastic version)
    - You should feel free to play around with SGD or mini-batch on your own

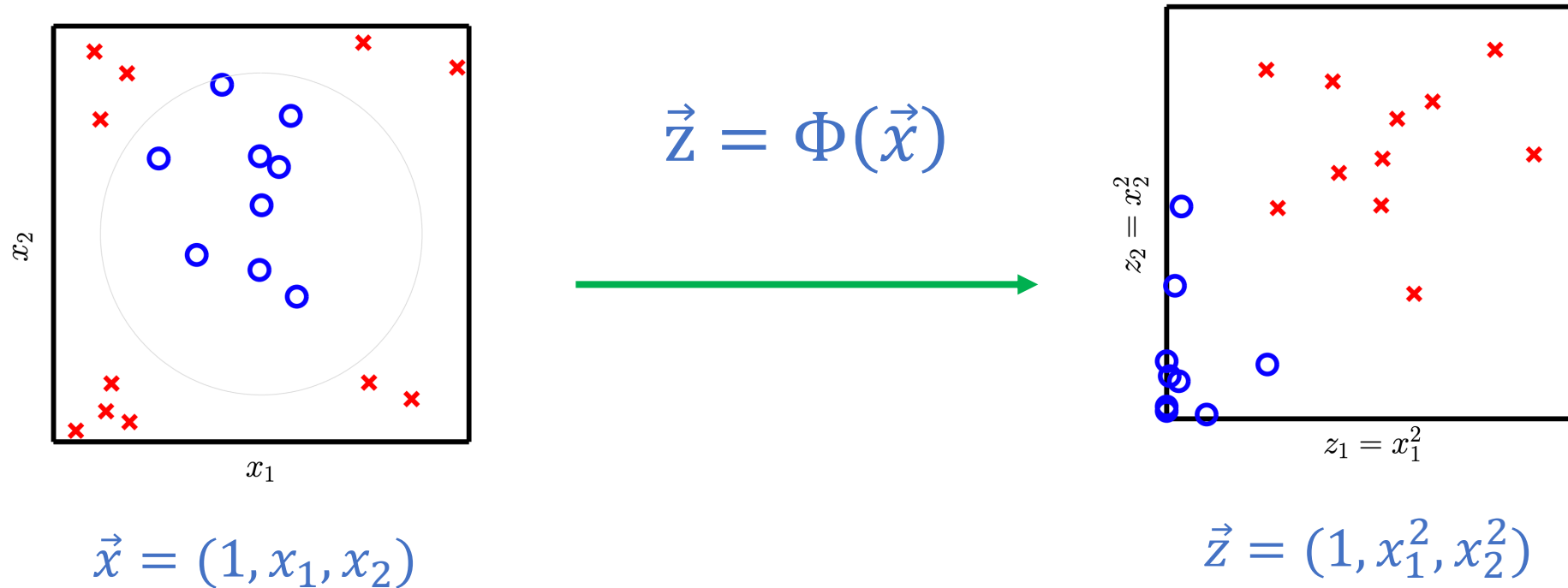
# Non-Linear Transformation

# Limitations of Linear Models



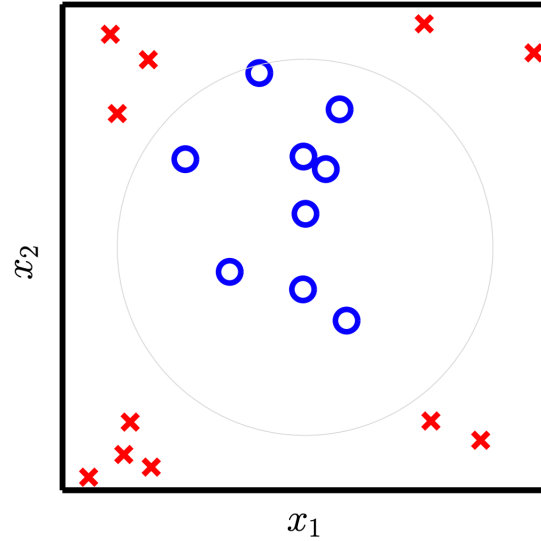
# Using Non-Linear Transformations

- Find a feature transform  $\Phi$  that maps data from  $\vec{x}$  space to  $\vec{z}$  space



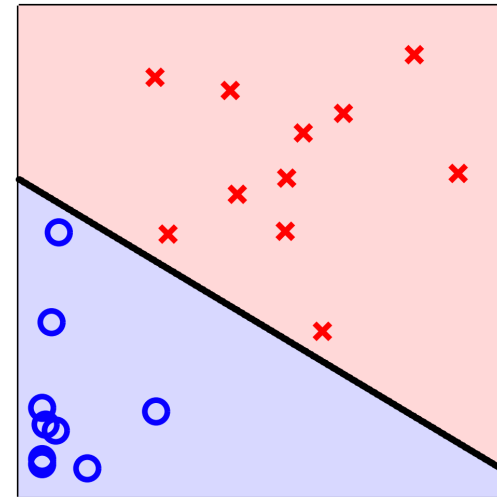
# Using Non-Linear Transformations

- Learn a linear classifier in  $\vec{z}$  space:  $g^{(z)}(\vec{z}) = \text{sign}(\vec{w}^{(z)T} \vec{z})$



$$\vec{x} = (1, x_1, x_2)$$

$$\vec{z} = \Phi(\vec{x})$$



$$\vec{z} = (1, x_1^2, x_2^2)$$

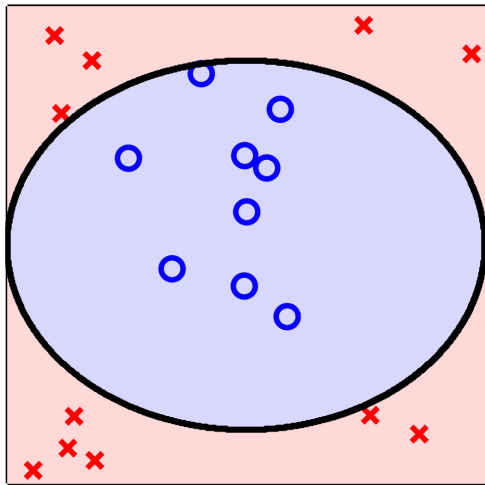
$$g^{(z)}(\vec{z}) = \text{sign}(-0.6 + z_1 + z_2)$$



# Using Non-Linear Transformations

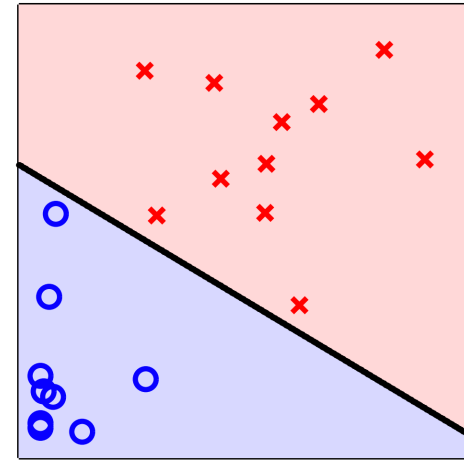
- Transform the learned hypothesis back to  $\vec{x}$  space

- $g(\vec{x}) = g^{(z)}(\Phi(\vec{x})) = \text{sign}\left(\vec{w}^{(z)T} \Phi(\vec{x})\right)$



$$\vec{x} = (1, x_1, x_2)$$

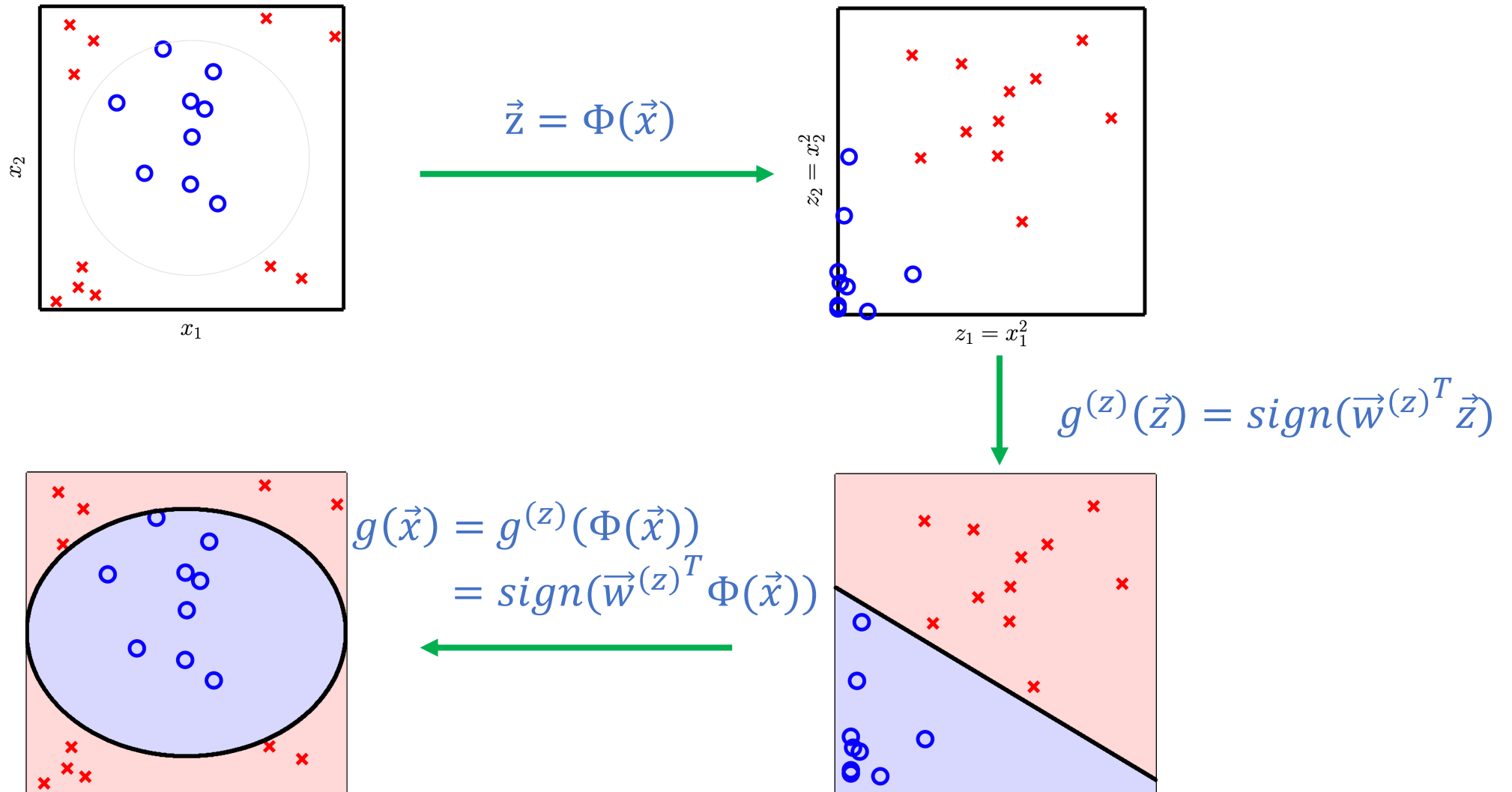
$$g(\vec{x}) = \text{sign}(-0.6 + x_1^2 + x_2^2)$$



$$\vec{z} = (1, x_1^2, x_2^2)$$

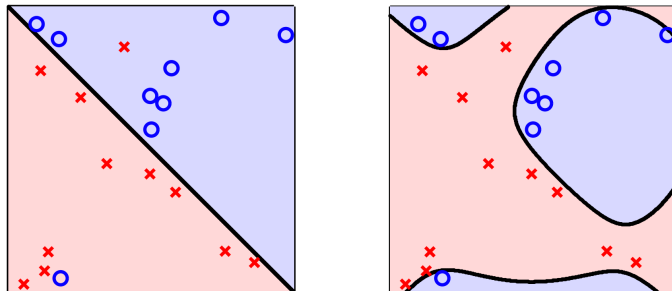
$$g^{(z)}(\vec{z}) = \text{sign}(-0.6 + z_1 + z_2)$$

# Nonlinear Transformation

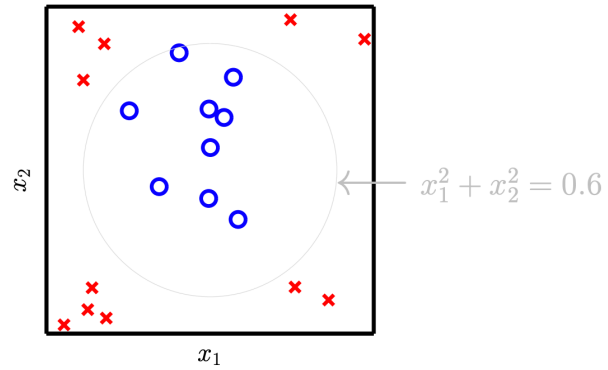


# Generalization of Nonlinear Transformation

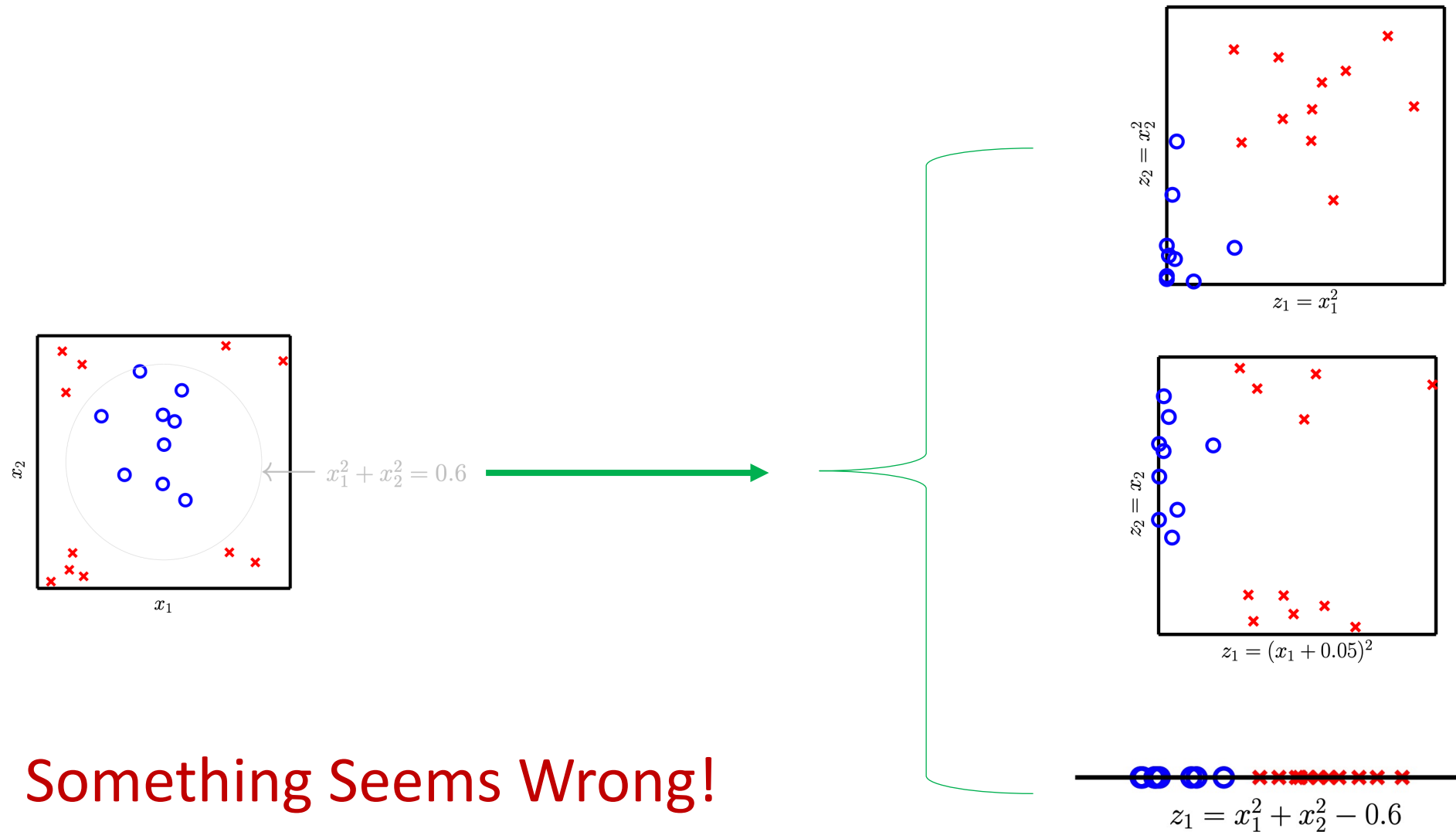
- Fact (We'll prove this later)
  - The VC Dimension of d-dim perceptron is  $d + 1$
- VC dimension of perceptron on input space  $\vec{x} = (x_0, \dots, x_d)$ 
  - $d+1$
- VC dimension of perceptron on input space  $\vec{z} = (z_0, \dots, z_{d'})$ 
  - $\leq d' + 1$  (usually treated as  $\approx d' + 1$ )
- Careful: Non-linear transform might lead to "nonsense" behavior



# How to Choose Feature Transform $\Phi$



# How to Choose Feature Transform $\Phi$



Must choose  $\Phi$   
**BEFORE** looking at the data

Otherwise, you are doing “data snooping”

The hypothesis set  $H$  is as large as anything your brain can think of

# Choose $\Phi$ Before Seeing Data

- Rely on domain knowledge (feature engineering)
  - Handwriting digit recognition example
- Use common sets of feature transformation
  - Polynomial transformation
  - 2nd order Polynomial transformation
    - $\vec{x} = (1, x_1, x_2)$
    - $\Phi_2(\vec{x}) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$
    - Pros: more powerful (contains circle, ellipse, hyperbola, etc)
    - Cons: 2-d  $\Rightarrow$  5-d
      - More computation/storage
      - Worse generalization error

The VC dimension of d-dim perceptron is  $d+1$