CSE 417T
Introduction to Machine Learning

Lecture 9
Instructor: Chien-Ju (CJ) Ho

Logistics: Homework 2

- Homework 2 is due on **Feb 24, Monday**
  - One implementation question (in Matlab)
  - Several math questions
- Return of Homework 1
  - We plan to return each homework 2 weeks after the deadline
- Regrade requests
  - You will have up to 7 days to submit regrade requests after homework return.
  - We might check the entire homework for each request, so the grades might go down as well if we find new mistakes.

Recap

Linear Models

This is why it's called linear models

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

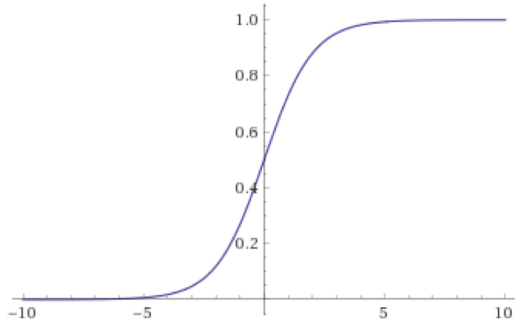
	Domain	Model
Linear Classification	$y \in \{-1, +1\}$	$H = \{h(\vec{x}) = sign(\vec{w}^T\vec{x})\}$
Linear Regression	$y \in \mathbb{R}$	$H = \{h(\vec{x}) = \vec{w}^T\vec{x}\}$
Logistic Regression	$y \in [0,1]$	$H = \{h(\vec{x}) = \theta(\vec{w}^T\vec{x})\}$

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

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

Logistic Regression

- Hypothesis set $H = \{h(\vec{x}) = \theta(\vec{w}^T \vec{x})\}$
  - $\theta(s) = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}}$



- Predict a probability
  - Interpreting $h(\vec{x})$ as the prob for $y = +1$ given $\vec{x}$ when h is the target function
- Algorithm
  - Find $g = argmin_{h \in H} E_{in}(h)$
- Two key questions
  - How to define $E_{in}(h)$?
  - How to perform the optimization (minimizing E_{in})?

Define $E_{in}(\vec{w})$: 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 cross entropy error is the same as maximizing likelihood

- Likelihood: $\Pr(D|\vec{w})$
  - $\vec{w}^* = argmax_{\vec{w}} \Pr(D|\vec{w})$ (maximizing likelihood)
    $= argmin_{\vec{w}} E_{in}(\vec{w})$ (minimizing cross-entropy error)

Min Cross-Entropy Error <=> Max Likelihood

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

1. $1 - \theta(s) = \theta(-s)$
2. data independence assumption

$argmax\ A(x)B(x)$
$= argmax \ln A(x) + \ln B(x)$

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

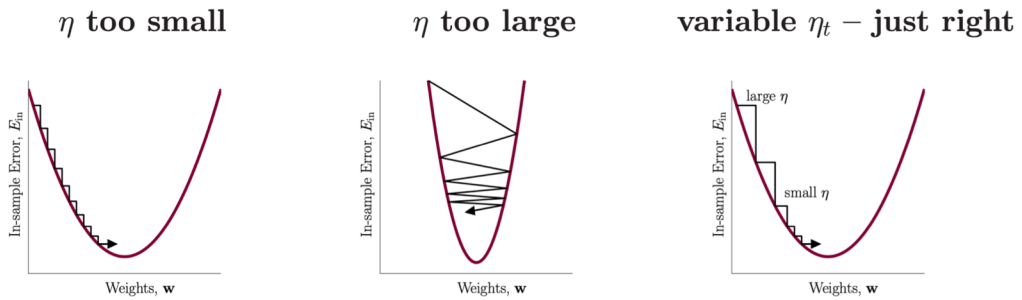
Optimizing $E_{in}(\vec{w})$: Gradient Descent

An iterative method: $\vec{w}(t+1) \leftarrow \vec{w}(t) + \eta_t \vec{v}_t$

- $\vec{v}_t$: an unit vector, determining the direction of the update
- η_t: an scalar, determining how much to update

- How to choose $\vec{v}_t$
  - Move towards the "steepest" direction
  - Approaching the minimum faster
  - Taylor approximation:
    - $E_{in}(\vec{w}(t+1)) - E_{in}(\vec{w}(t)) \approx \eta_t \nabla_{\vec{w}} E_{in}(\vec{w}(t))^T \vec{v}_t$
  - Choose $\vec{v}_t$ to be the opposite direction of $\nabla_{\vec{w}} E_{in}$
    - $\vec{v}_t = \frac{-\nabla_{\vec{w}} E_{in}(\vec{w}(t))}{\|\nabla_{\vec{w}} E_{in}(\vec{w}(t))\|}$

- How to choose η_t



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

Gradient Descent for Logistic Regression

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

η: learning rate.
A parameter the learner can choose.

Gradient Descent for Logistic Regression

- Initialization
  - In HW2, you are asked to initialize $\vec{w}(0)$ to $\vec{0}$.
  - In practice, random initialization is a good idea and a common approach.

- 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
  - (We use the first two in HW2)

Using Logistic Regression for Classification

- Let $\vec{w}^*$ or g be the learned logistic regression model, can we make classification predictions using $\vec{w}^*$?

- Yes
  - 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)
    - $\theta({\vec{w}^*}^T \vec{x}) > 50\%$ => ${\vec{w}^*}^T \vec{x} > 0$
    - Equivalent to use $\vec{w}^*$ as the linear classification hypothesis, i.e., $g(\vec{x}) = sign({\vec{w}^*}^T \vec{x})$

Brief Lecture Notes Today

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

Let me know if you spot errors.

More on Cross Entropy [This Page is Safe to Skip]

- Cross entropy of q related to p: $H(p,q) = \sum_{i=1}^{n} p(x_i) \log \frac{1}{q(x_i)}$
  - Fix $p, H(p,q)$ is minimized when $q = p$ [Solve for $\nabla_q H(p,q) = 0$]
- Cross-entropy error
  - $E_{in}(\vec{w}) = \frac{1}{N} \sum_{n=1}^{N} \ln(1 + e^{-y_n \vec{w}^T \vec{x}_n})$
  - $E_{in}(\vec{w}) = \frac{1}{N} \sum_{n=1}^{N} \left[\mathbb{I}[y_n = 1] \ln \frac{1}{\theta(\vec{w}^T \vec{x})} + \mathbb{I}[y_n \neq 1] \ln \frac{1}{1-\theta(\vec{w}^T \vec{x})} \right]$
  - Interpretations
    - p: empirical distribution of y_n in training data
    - q: predicted probability distribution of y_n of hypothesis h
  - Minimizing E_{in} => Make $q \approx p$ => Make prediction align with data
    - E_{in} is minimized if
      - When $y_n = 1$, $h(\vec{x}) = \theta(\vec{w}^T \vec{x}) \approx 1$
      - When $y_n = -1$, $h(\vec{x}) = \theta(\vec{w}^T \vec{x}) \approx 0$

Computational Considerations for Gradient Descent

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

- Which step is the most computationally heavy?
  - 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

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

- Speed up the implementation of $\nabla_{\vec{w}} E_{in}(\vec{w}) = -\frac{1}{N}\sum_{n=1}^{N} \frac{y_n \vec{x}_n}{1+e^{y_n \vec{w}^T \vec{x}_n}}$
  - E.g., doing vectorization in Matlab (Problem 1 in HW2)
  - Not required to do this in HW2, but it greatly improves the time efficiency

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

Stochastic Gradient Descent (SGD)

- Algorithm
  - Initialize $\vec{w}(0)$
  - For $t = 0, \ldots$
    - Randomly choose n from $\{1, \ldots, N\}$
    - $\vec{w}(t+1) \leftarrow \vec{w}(t) - \eta \nabla_{\vec{w}} e_n(\vec{w}(t))$
    - 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

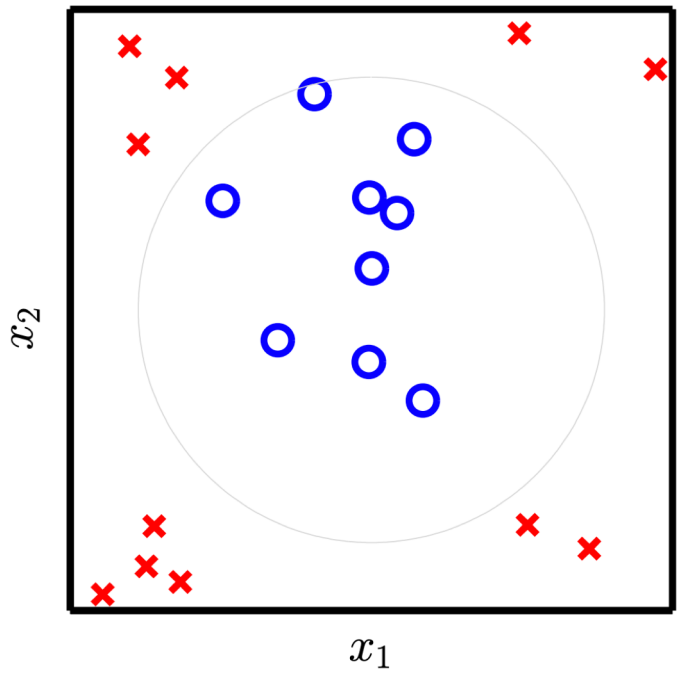
- GD: Computationally heavy, stable updates
- SGD: Computationally light, noisy updates

- 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.

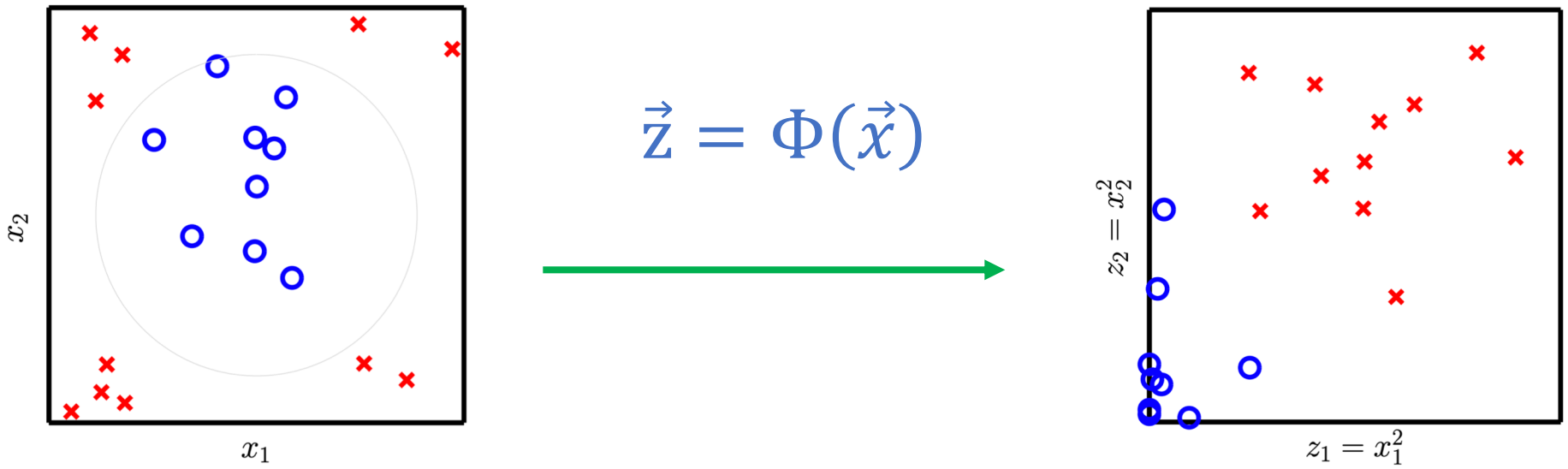
Non-Linear Transformation

Limitations of Linear Models



Using Non-Linear Transformations

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

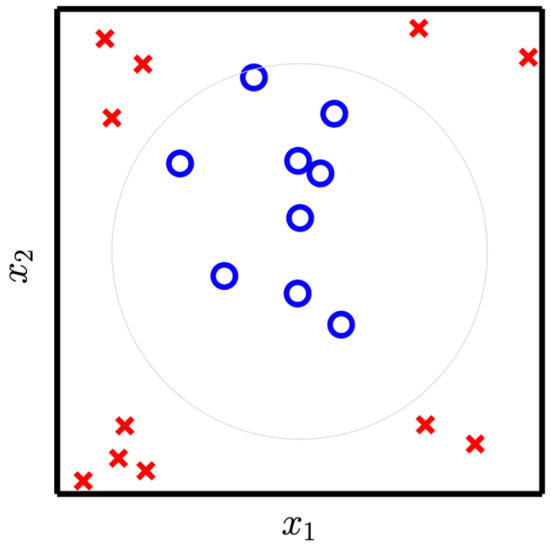


$\vec{x} = (1, x_1, x_2)$

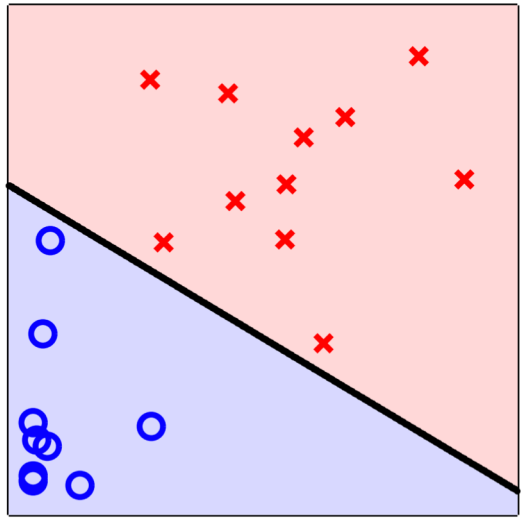
$\vec{z} = (1, x_1^2, x_2^2)$

Using Non-Linear Transformations

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



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

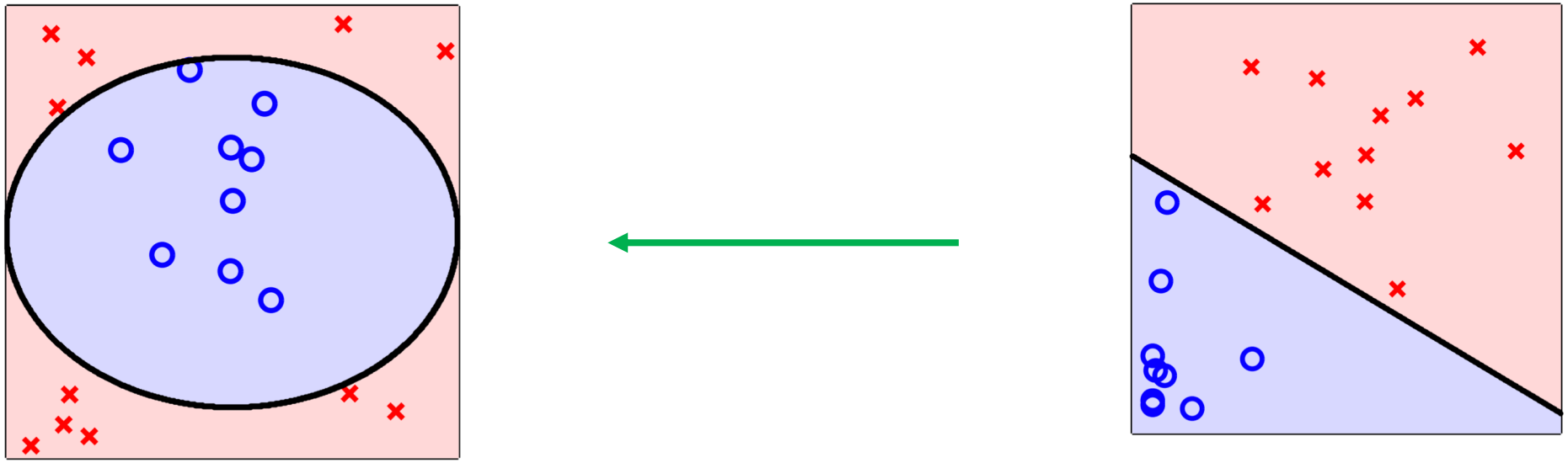


$$\vec{x} = (1, x_1, x_2)$$

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

Using Non-Linear Transformations

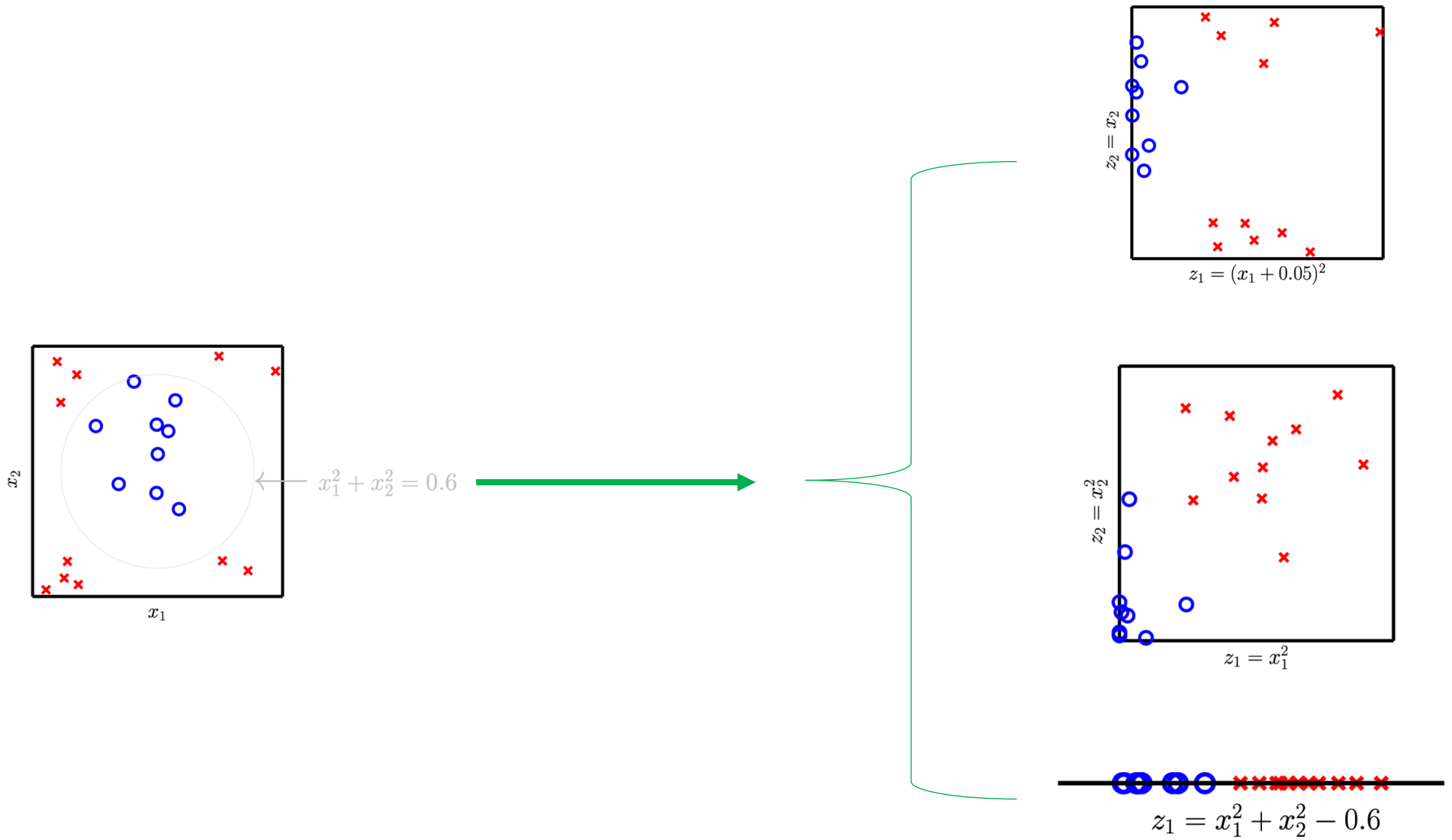
- Transform the learned hypothesis back to $\vec{x}$ space
  - $g(\vec{x}) = g^{(z)}(\Phi(\vec{x})) = sign(\vec{w}^{(z)^T}\Phi(\vec{x}))$



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

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

How to Choose Feature Transform Φ



MUST choose Φ
BEFORE looking at the data

Choose Φ Before Seeing Data

- Rely on domain knowledge (feature engineering)
  - Handwriting digit recognition example

- Use common sets of feature transformation
  - Polynomial transformation
  - E.g., 2nd order Polynomial transformation
    - $\vec{x} = (1, x_1, x_2)$
    - $\Phi_2(\vec{x}) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$
    - Plus: more powerful (contains circle, ellipse, hyperbola, etc)
    - Minus: 2-d => 5-d
      - More computation/storage
      - Worse generalization error

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

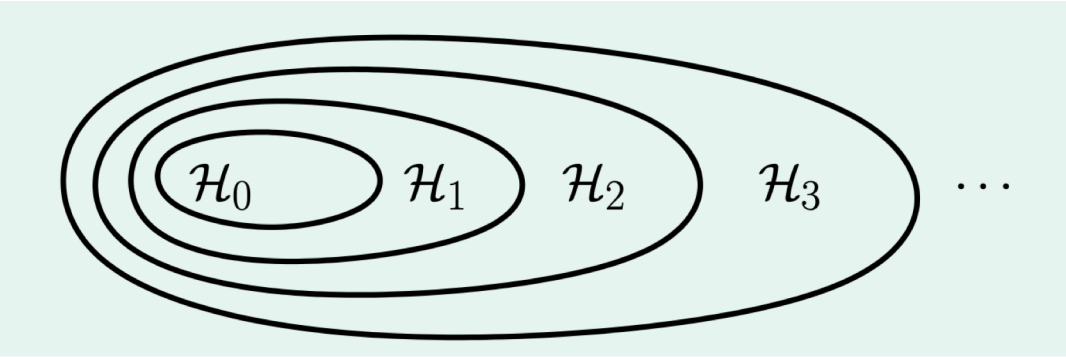
Q-th Order Polynomial Transform

- $\vec{x} = (1, x_1, \dots, x_d)$
- From 1-st order to Q-th order polynomial transform:
  - $\Phi_1(\vec{x}) = \vec{x}$
  - $\Phi_2(\vec{x}) = (\Phi_1(\vec{x}), x_1^2, x_1x_2, x_1x_3, \dots, x_d^2)$
  - …
  - $\Phi_Q(\vec{x}) = (\Phi_{Q-1}(\vec{x}), x_1^Q, x_1^{Q-1}x_2, \dots, x_d^Q)$

- Number of elements in $\Phi_Q(\vec{x})$: $\binom{Q+d}{Q}$

Structural Hypothesis Sets

- Let H_Q be the linear model for the $\Phi_Q(\vec{x})$ space



- Let $g_Q = argmin_{h \in H_Q} E_{in}(h)$
  - $H_0 \subset H_1 \subset H_2 \subset \cdots$
  - $d_{vc}(H_0) \leq d_{vc}(H_1) \leq \cdots$
  - $E_{in}(g_1) \geq E_{in}(g_2) \geq \cdots$