CSE 417T
Introduction to Machine Learning

Lecture 9

Instructor: Chien-Ju (CJ) Ho

Logistics

- Homework 2 is due on **Mar 8, Monday**
  - Implement gradient descent for logistic regression
  - Several math questions
- Return of Homework
  - We plan to return each homework around 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
- Exam 1: **Mar 23 (Tuesday)**

Recap

Linear Models

This is why it's called linear models

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

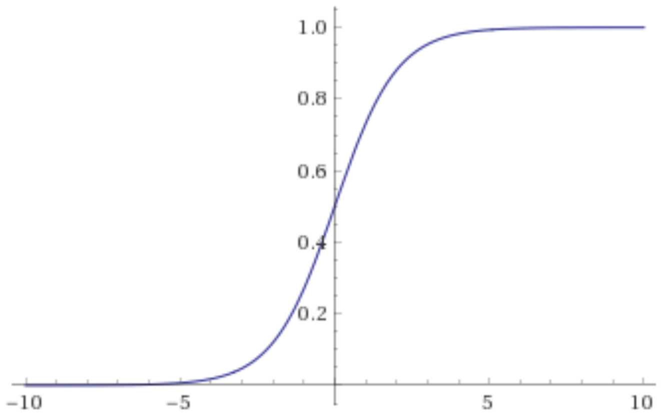
	Domain	Model	Credit Card Example
Linear Classification	$y \in \{-1, +1\}$	$H = \{h(\vec{x}) = sign(\vec{w}^T\vec{x})\}$	Approve or not
Linear Regression	$y \in \mathbb{R}$	$H = \{h(\vec{x}) = \vec{w}^T\vec{x}\}$	Credit line
Logistic Regression	$y \in [0,1]$	$H = \{h(\vec{x}) = \theta(\vec{w}^T\vec{x})\}$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

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

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

Logistic Regression

- Predict a probability
  - Interpreting $h(\vec{x}) \in [0,1]$ as the prob for $y = +1$ given $\vec{x}$
- Hypothesis set $H = \{h(\vec{x}) = \theta(\vec{w}^T\vec{x})\}$
  - $\theta(s) = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}}$



- 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

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

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

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

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

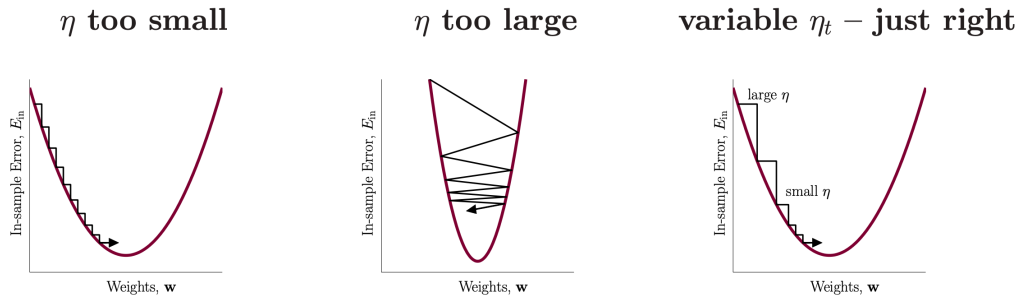
Optimizing $E_{in}(\vec{w})$: Gradient Descent

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

- $\vec{v}_t$: a unit vector, determining the direction of the update
- η_t: a 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 (GD) for Logistic Regression

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

η: learning rate
A parameter the learner can choose

We focus on fixed learning rate GD
There are other variants

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

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.

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

- Cross entropy of q related to p: $H(p,q) = \sum_{i=1}^{n} p(x_i) \log \frac{1}{q(x_i)}$
  - Distance measure between two distributions
  - Fix p, $H(p,q)$ is minimized when $q = p$ [Solve for $\nabla_q H(p,q) = 0$]

- Cross-entropy error
  - $E_{in}(\vec{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln(1 + e^{-y_n \vec{w}^T \vec{x}_n})$
    $= \frac{1}{N}\sum_{n=1}^{N} \left[\mathbb{I}[y_n = 1] \ln \frac{1}{\theta(\vec{w}^T \vec{x}_n)} + \mathbb{I}[y_n \neq 1] \ln \frac{1}{1-\theta(\vec{w}^T \vec{x}_n)} \right]$

  - 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

Computation of 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 the gradient $\nabla_{\vec{w}} E_{in}(\vec{w}) = -\frac{1}{N}\sum_{n=1}^{N} \frac{y_n \vec{x}_n}{1+e^{y_n \vec{w}^T \vec{x}_n}}$
  - 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., "vectorization"

- 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
    - $E_{in}(\vec{w}) = \frac{1}{N}\sum_{n=1}^{N} e_n(\vec{w})$
    - $\nabla_{\vec{w}} E_{in}(\vec{w}) = \frac{1}{N}\sum_{n=1}^{N} \nabla_{\vec{w}} e_n(\vec{w})$ (gradient of dataset is the average gradient of points)
  - 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 a data point 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

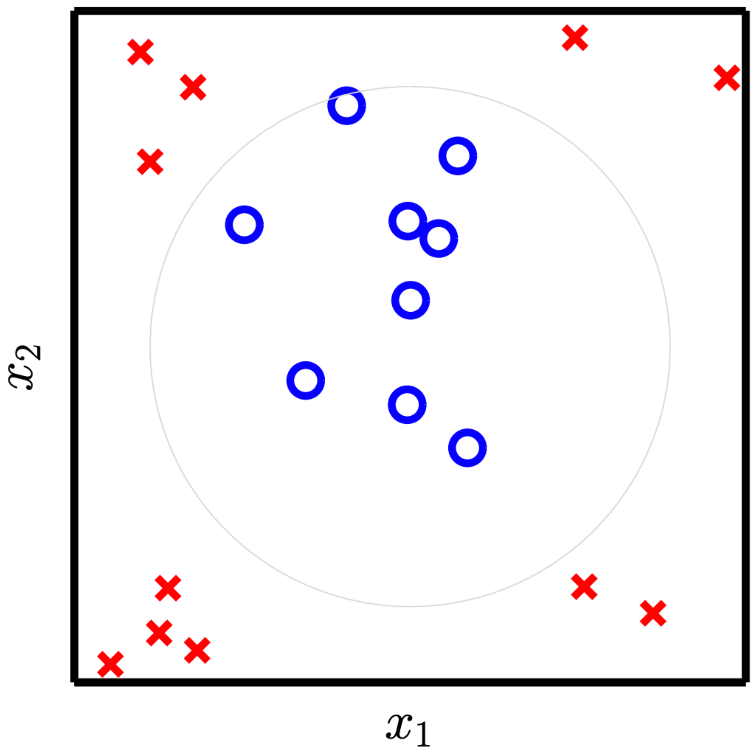
- GD: Computationally heavy, stable updates
- SGD: Computationally light, noisy updates

- 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.

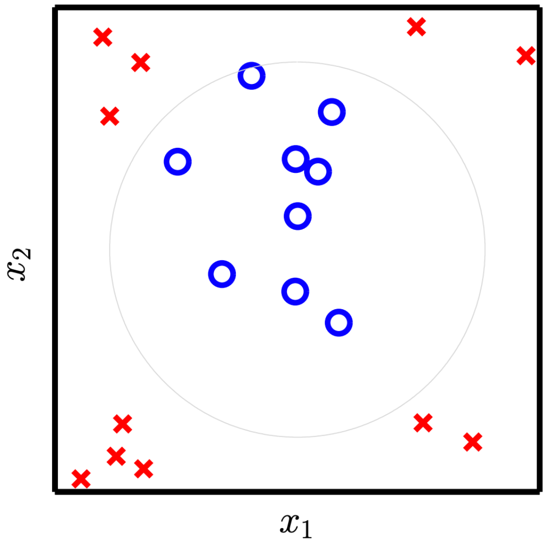
Non-Linear Transformation

Limitations of Linear Models

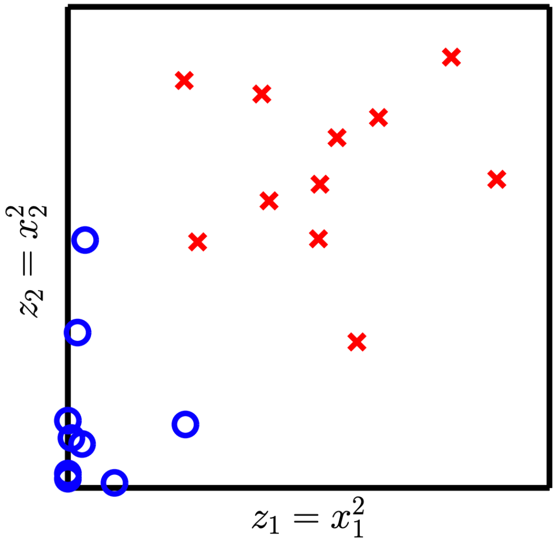


Using Non-Linear Transformations

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



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

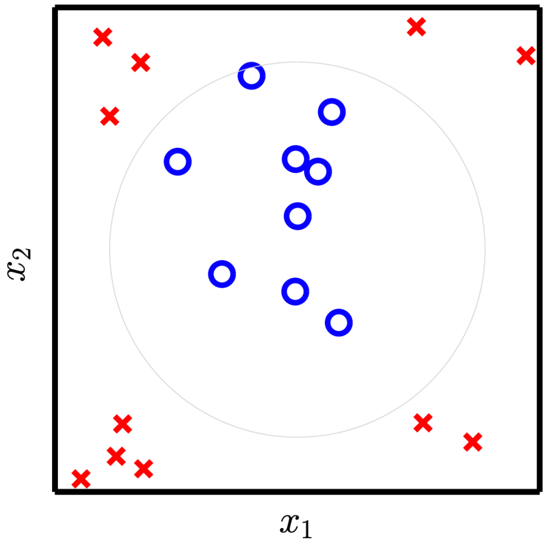


$$\vec{x} = (1, x_1, x_2)$$

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

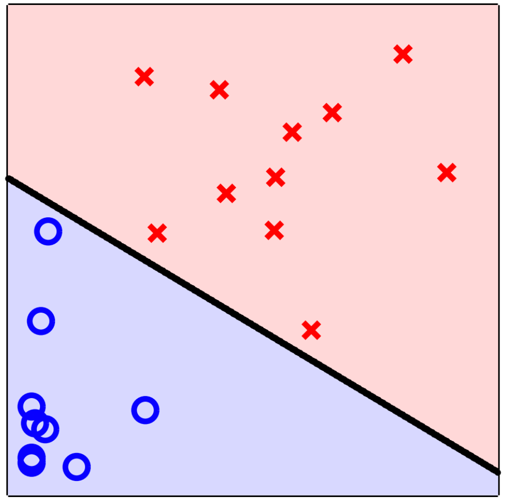
Using Non-Linear Transformations

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



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

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

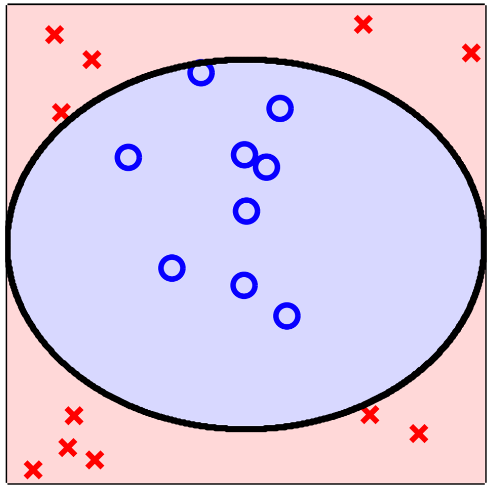


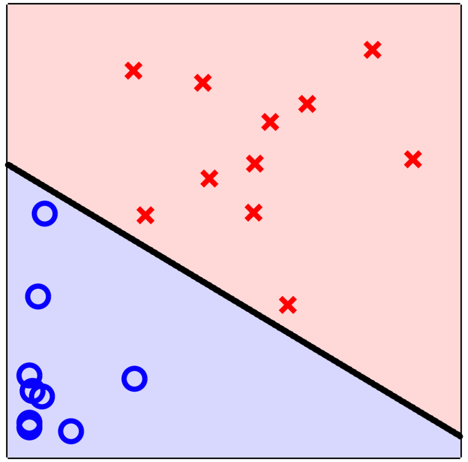
$\vec{z} = (1, x_1^2, x_2^2)$

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

Using Non-Linear Transformations

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



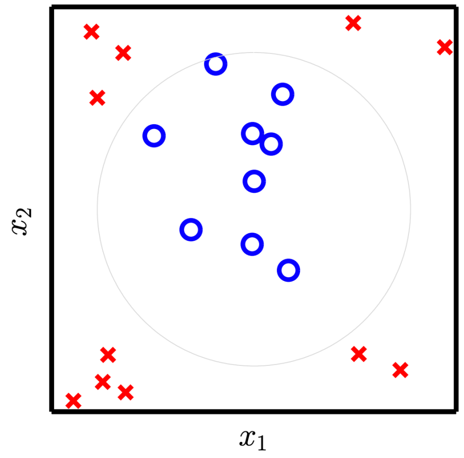


$\vec{x} = (1, x_1, x_2)$

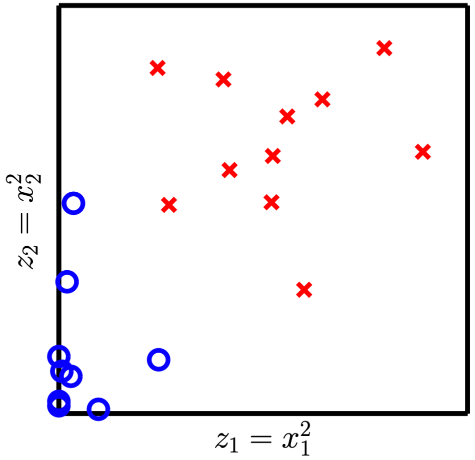
$\vec{z} = (1, x_1^2, x_2^2)$

$g(\vec{x}) = sign(-0.6 + x_1^2 + x_2^2)$ $\longleftarrow$ $g^{(z)}(\vec{z}) = sign(-0.6 + z_1 + z_2)$

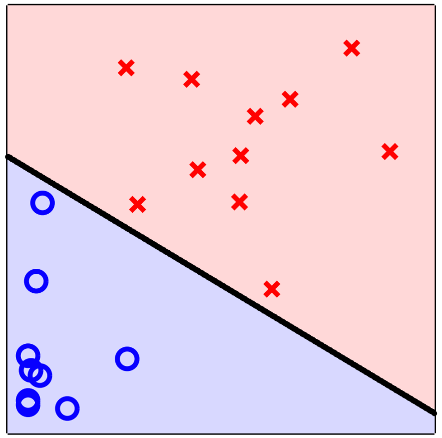
Nonlinear Transformation



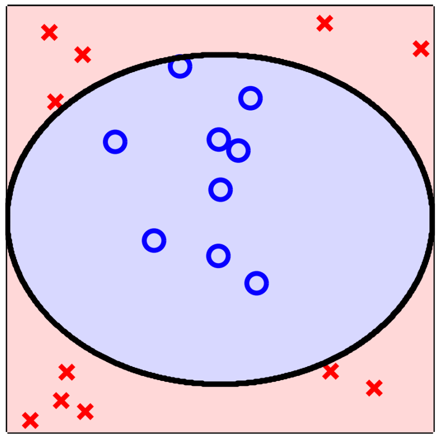
$$\vec{z} = \Phi(\vec{x})$$



$$g^{(z)}(\vec{z}) = sign({\overrightarrow{w}^{(z)}}^T \vec{z})$$

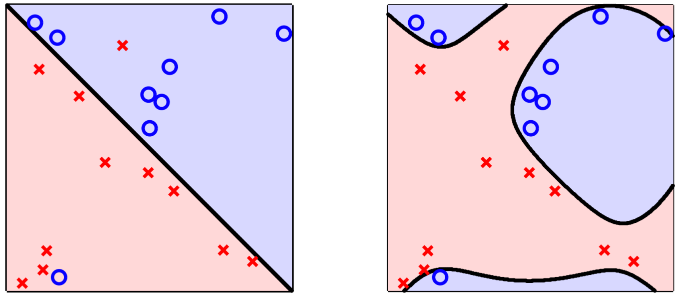


$$g(\vec{x}) = g^{(z)}(\Phi(\vec{x}))$$
$$= sign({\overrightarrow{w}^{(z)}}^T \Phi(\vec{x}))$$

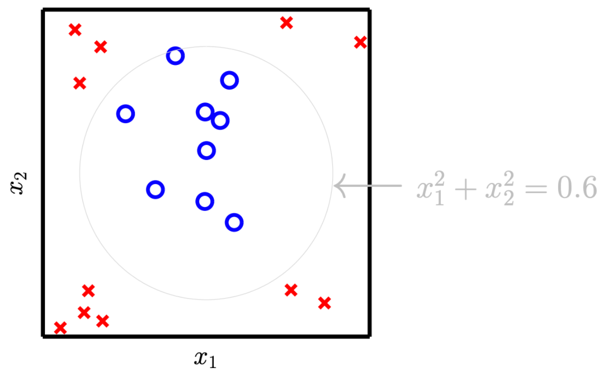


Generalization of Nonlinear Transformation

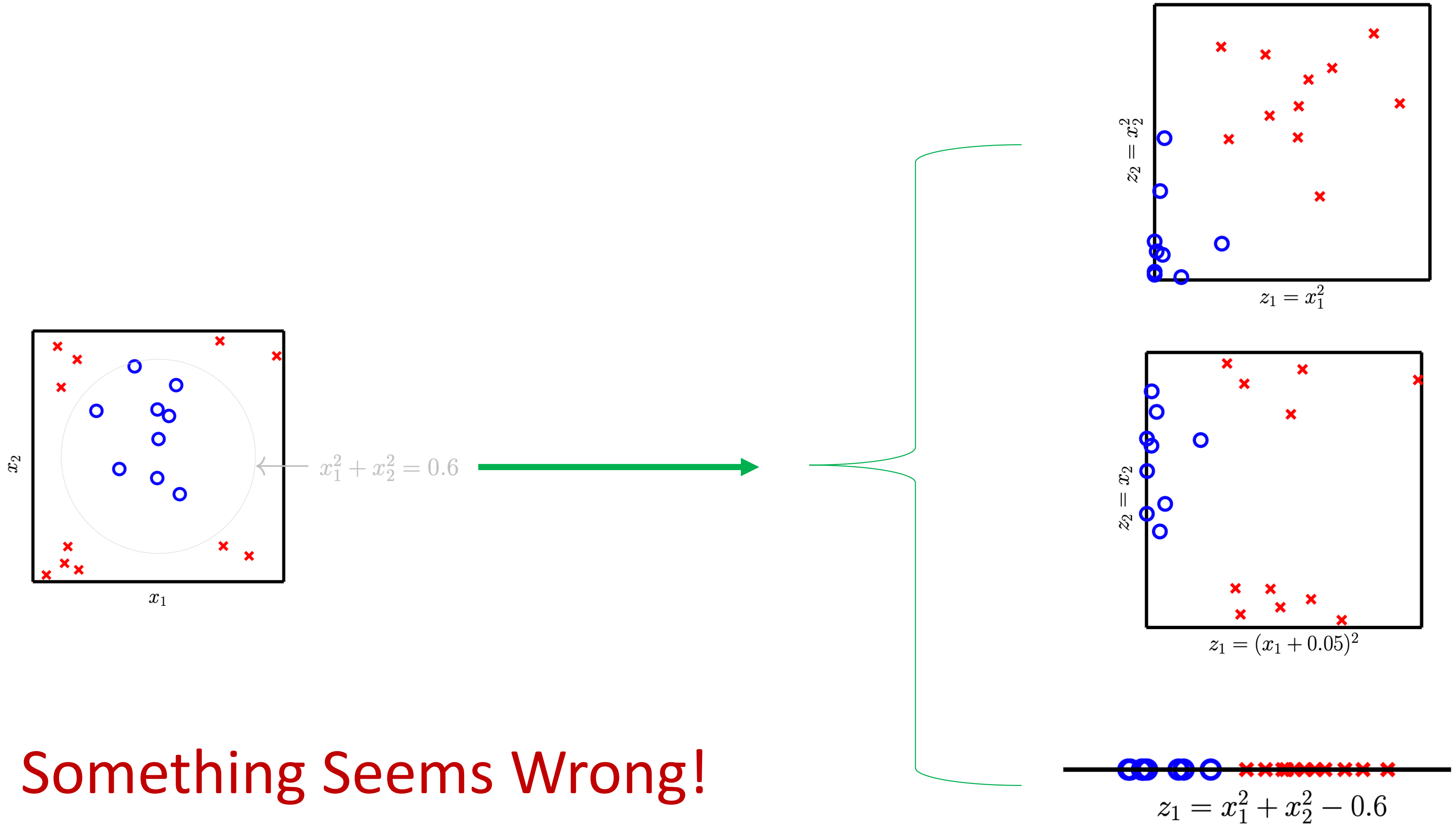
- Fact (We'll prove this later)
  - The VC Dimension of d-dim perceptron is $d + 1$
- VC dimension of perceptron on input space $\vec{x} = (x_0, .., x_d)$
  - d+1
- VC dimension of perceptron on input space $\vec{z} = (z_0, .., z_{d'})$
  - $\leq d' + 1$ (usually treated as $\approx d' + 1$)
- Careful: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



How to Choose Feature Transform Φ



How to Choose Feature Transform Φ



Something Seems Wrong!

Must choose Φ 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 Φ 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
    - $\vec{x} = (1, x_1, x_2)$
    - $\Phi_2(\vec{x}) = (1, x_1, x_2, x_1 x_2, x_1^2, x_2^2)$
    - Pros: more powerful (contains circle, ellipse, hyperbola, etc)
    - Cons: 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:
  - $\Phi_1(\vec{x}) = \vec{x}$
  - $\Phi_2(\vec{x}) = (\Phi_1(\vec{x}), x_1^2, x_1 x_2, x_1 x_3, \dots, x_d^2)$
  - …
  - $\Phi_Q(\vec{x}) = (\Phi_{Q-1}(\vec{x}), x_1^Q, x_1^{Q-1} x_2, \dots, x_d^Q)$

- Number of elements in $\Phi_Q(\vec{x})$

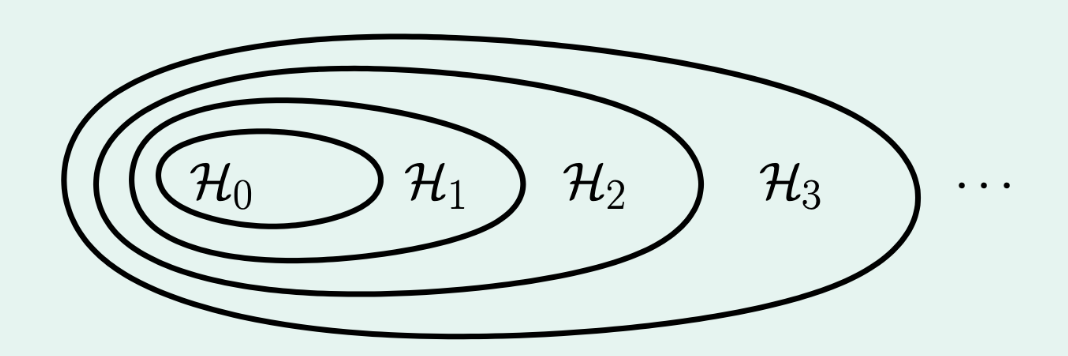
Q-th Order Polynomial Transform

- $\vec{x} = (1, x_1, \ldots, x_d)$
- From 1-st order to Q-th order polynomial transform:
  - $\Phi_1(\vec{x}) = \vec{x}$
  - $\Phi_2(\vec{x}) = (\Phi_1(\vec{x}), x_1^2, x_1 x_2, x_1 x_3, \ldots, x_d^2)$
  - …
  - $\Phi_Q(\vec{x}) = (\Phi_{Q-1}(\vec{x}), x_1^Q, x_1^{Q-1} x_2, \ldots, x_d^Q)$

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

Structural Hypothesis Sets

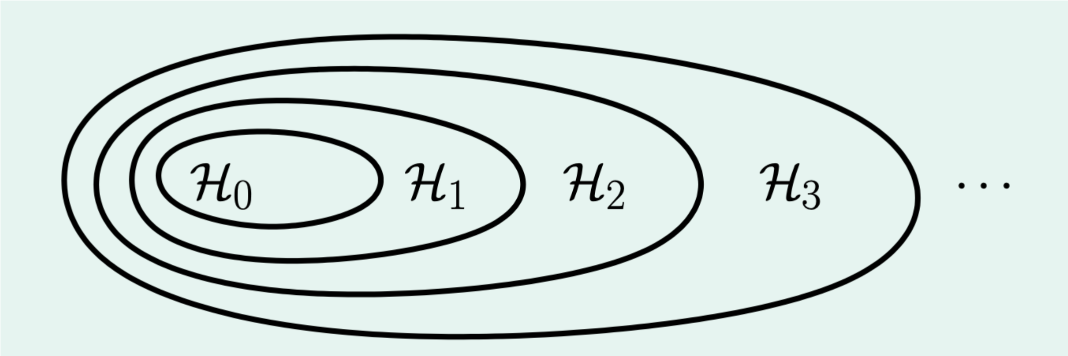
- Let H_Q be the linear model for the $\Phi_Q(\vec{x})$ space



- Let $g_Q = argmin_{h \in H_Q} E_{in}(h)$
  - $H_0 \quad H_1 \quad H_2 \ldots$
  - $d_{vc}(H_0) \quad d_{vc}(H_1) \quad d_{vc}(H_2) \ldots$
  - $E_{in}(g_0) \quad E_{in}(g_1) \quad E_{in}(g_2) \ldots$

Structural Hypothesis Sets

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



- Let $g_Q = argmin_{h \in H_Q} E_{in}(h)$
  - $H_0 \subset H_1 \subset H_2 \ldots$
  - $d_{vc}(H_0) \leq d_{vc}(H_1) \leq d_{vc}(H_2) \ldots$
  - $E_{in}(g_0) \geq E_{in}(g_1) \geq E_{in}(g_2) \ldots$