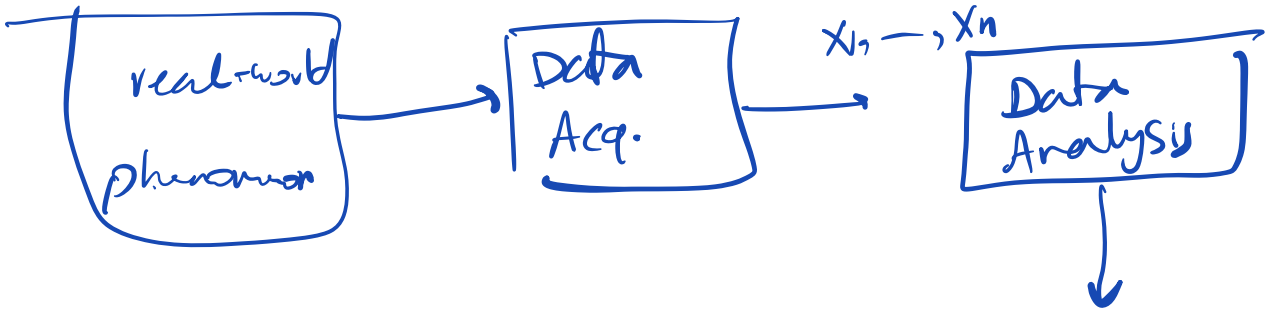
Lecture 13:



Modules 1, 2 :

- learned to estimate properties/parameters connected to the underlying phenomenon
- hypothesis testing: verify different hypotheses connected with the underlying phenomenon.

Module 3: Supervised learning: "learn" some "predictive relation" connected to the phenomenon.

Supervised learning:

We assume that each data point x_i is associated with a "label" (which is a part of the data). I.e. each data point is of the form (x_i, y_i), where x_i is the "feature vector" and "y_i" is the label associated with x_i.

Examples:

- x_i: could be an image
- y_i: could an object inside the image

- x_i: could be (the text of) an email
- y_i: could be whether/not the email is spam.

x_i: a vector representing a patients profile, $X_i = (\text{age}, \text{height}, \text{weight}, \text{MRI}, \ldots)$

y_i: cancer / not cancer

Data: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$

Goal: $y_i \approx f(x_i)$

- y_i: spam/ham (0 / 1)
- f: predictor
- x_i: email text

→ Main goal: Find the best predictor f.

Setting: Data: $\{(x_i, y_i)\}_{i=1}^{n}$

$x_i = (x_{i1}, x_{i2}, \ldots, x_{ip}) \in \mathbb{R}^p$ ←

- image: pixel representation
- profile of the i-th patient

y_i = label

Supervised learning
- Regression
  - label $y_i \in \mathbb{R}$
  - e.g. probability of having cancer
- classification
  - $y_i \in A$

where A is a discrete set

e.g. $A = \{0, 1\}$

e.g. Spam / Ham

Regression:

- Let's start with the simplest setting (for the regression problem):

- 1-d regression

Data: $\{(x_i, y_i)\}_{i=1}^{n}$

$x_i \in \mathbb{R}$

$y_i \in \mathbb{R}$ ($p = 1$)

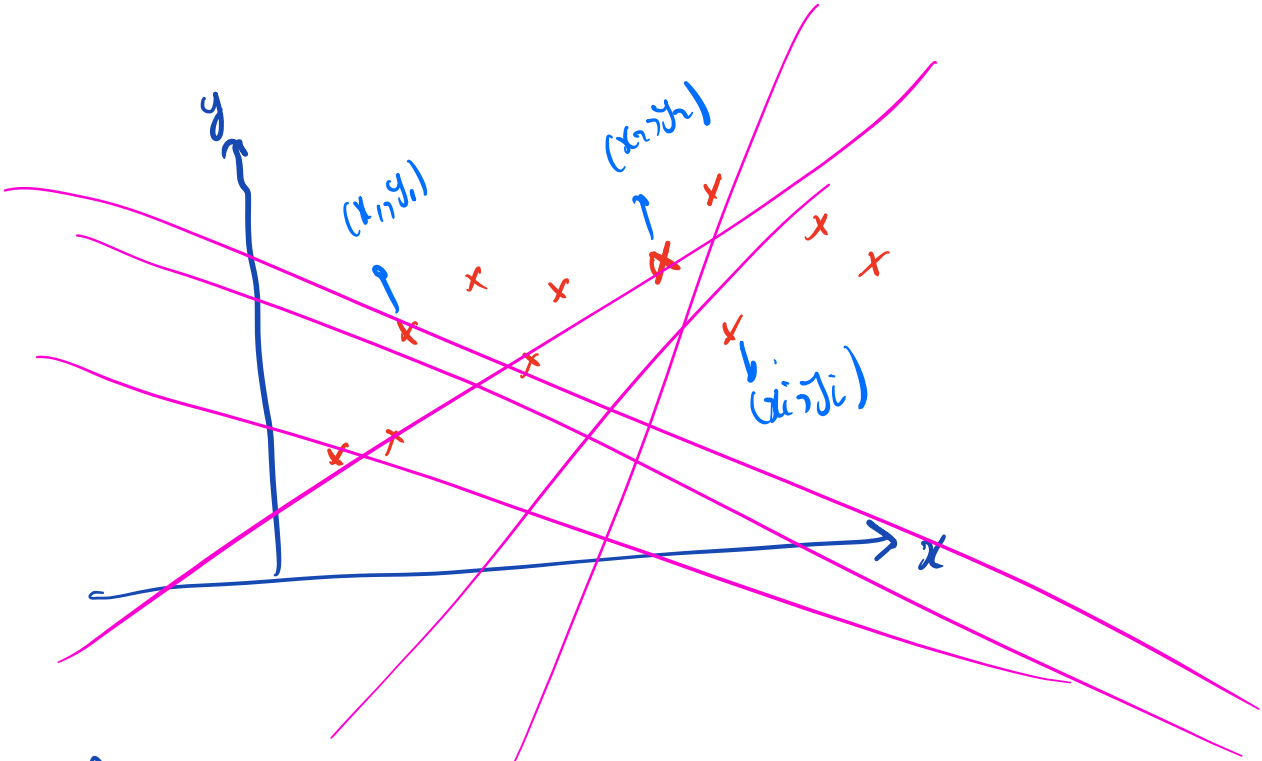
Goal: learn a "predictive" relation:

$$\hat{y} = \hat{f}(x)$$

example: x = blood sugar at the age of 20

y = probability of having Diabetes at the age of 40.

Data: $\{(x_i, y_i)\}_{i=1}^{\hat{}}$



$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x \leftarrow$$

$$\underbrace{\hat{\beta}_0 + \hat{\beta}_1 x}_{f(x)}$$

Approach:

Goal: learn $y = f(x)$

- Search for the best "candidate" for f inside a prescribed family of functions F.
- find the best f inside F
- typically F is a parametric family $\rightarrow f(x; \beta)$.

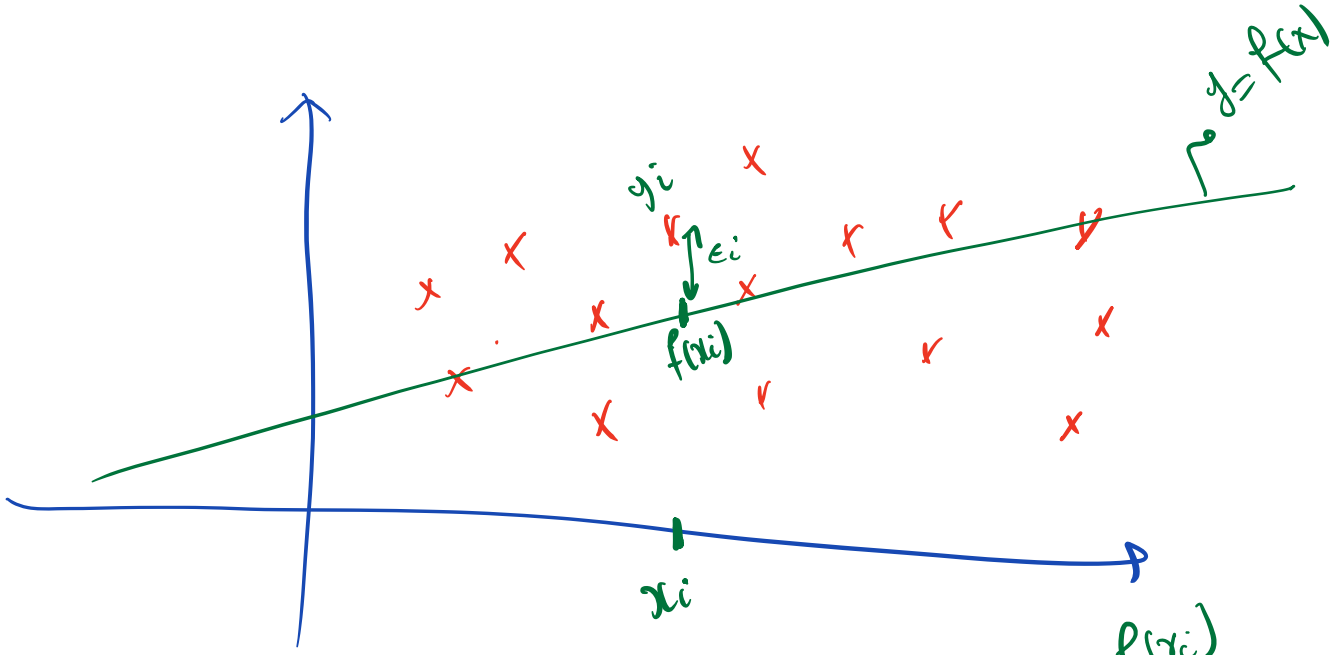
  e.g. linear functions

- Let's consider the family of linear functions as our first step. If $x \in \mathbb{R}$ then this family is described by two parameters $\hat{\beta}_0, \hat{\beta}_1$

$$f(x; \hat{\beta}_0, \hat{\beta}_1) = \hat{\beta}_0 + \hat{\beta}_1 x$$

β

Goal: find the best choice of $\hat{\beta}_0, \hat{\beta}_1$ from data.



$$y_i = f(x_i) + \varepsilon_i$$

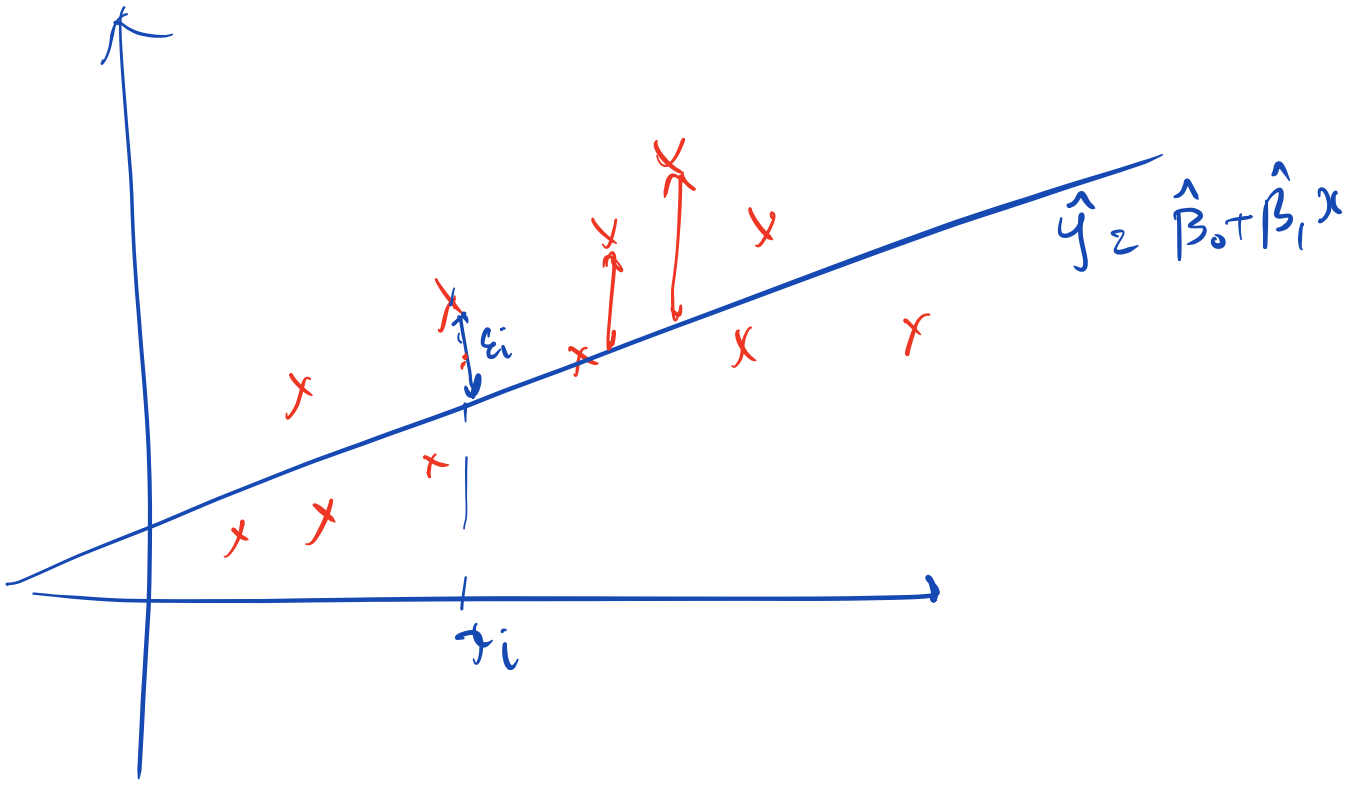
ε_i → error $= \varepsilon_i := y_i - f(x_i)$, with $f(x_i) = \hat{\beta}_0 + \hat{\beta}_1 x_i$

$$y_i = \hat{\beta}_0 + \hat{\beta}_1 x_i + \varepsilon_i$$

→ the error term models what we miss when we use this simple linear model. The true relation between the input and output is probably not linear as many other factors can be involved

(there may also be noise in the label)

Question: What is the fundamental procedure principle behind learning/choosing the parameters $\hat{\beta}_0, \hat{\beta}_1$?



Idea 1: find $\hat{\beta}_0, \hat{\beta}_1$ s.t.

$$\varepsilon_i = y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i)$$

Sum of the errors

$$\sum_{i=1}^{n} |\varepsilon_i|$$

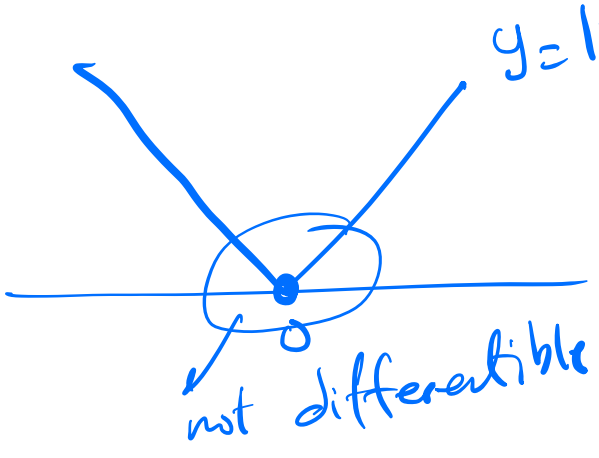
is minimized

$$\min_{\hat{\beta}_0, \hat{\beta}_1} \sum_{i=1}^{n} |y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i|$$

↓

- In order to solve the above optimization problem, we'll have to take the derivative wrt. $\hat{\beta}_0, \hat{\beta}_1$ and set it to zero.

- The problem with the above objective is that it's not differentiable.



- Idea 2: To make things differentiable let's consider the "square" of the errors:

$$\underset{\hat{\beta}_0, \hat{\beta}_1}{\text{minimize}} \sum_{i=1}^{n} \underbrace{\left(y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i)\right)^2}_{\varepsilon_i^2}$$

Lecture 14:

supervised learning → Regression → $y \in \mathbb{R}$
supervised learning → Classificatio

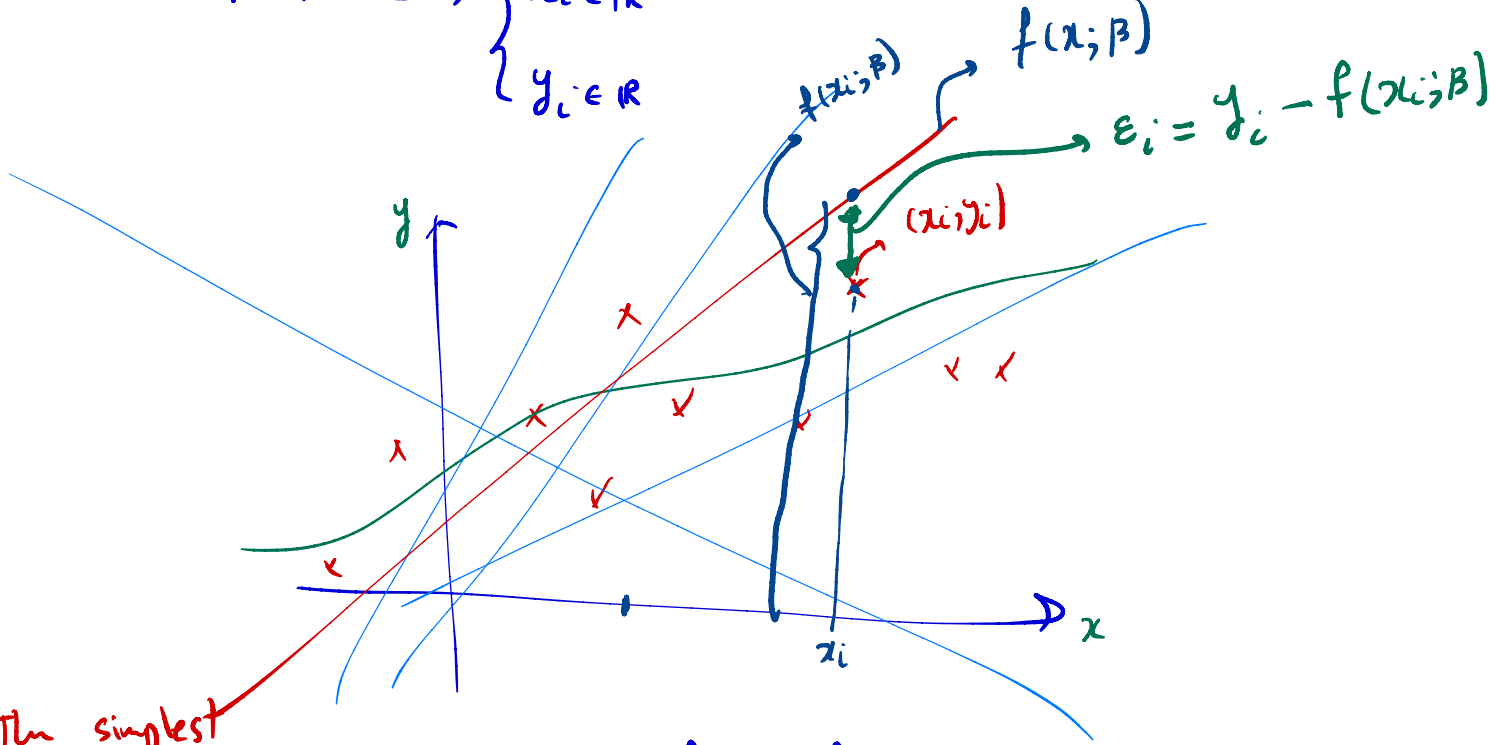
Data : $\{(x_i, y_i)\}_{i=1}^{n}$

$x_i \in \mathbb{R}^p$

$y_i \in \mathbb{R}$ → $y_i \approx f(x_i)$

f will be used for prediction on newly unseen data points

$p = 1$ → $x_i \in \mathbb{R}$, $y_i \in \mathbb{R}$



The simplest parametric family: $y = f(x;\beta) = \hat{\beta}_0 + \hat{\beta}_1 x$

our general methodology:

Come up with a parametric family of predictors

$f(x; \beta)$ ← parameters

↓

Define an objective function that quantifies how good a parameter is w.r.t the data set. Minimizing the objective will give us the best "fit" to the data set

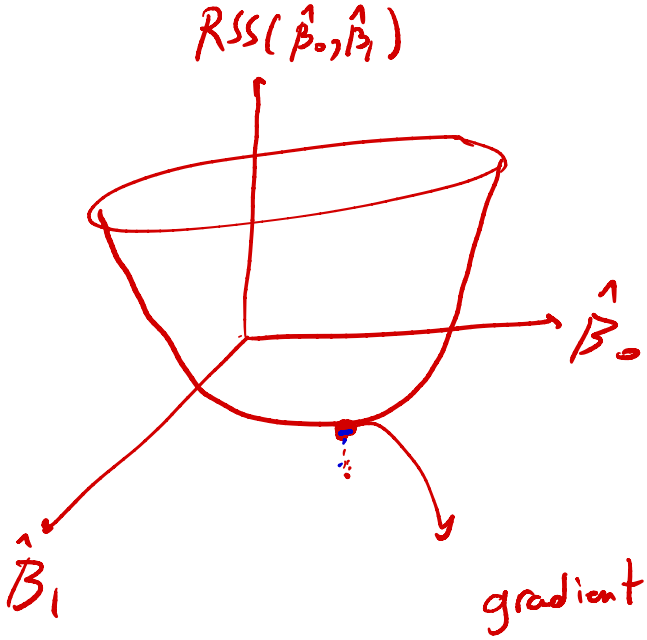
---

objective for the linear model:

$$\min_{\hat{\beta}_0, \hat{\beta}_1} \sum_{i=1}^{n} \varepsilon_i^2 \triangleq \min_{\hat{\beta}_0, \hat{\beta}_1} \sum_{i=1}^{n} \left(y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i) \right)^2$$

$$RSS(\hat{\beta}_0, \hat{\beta}_1)$$

Residual Sum of Squares



gradient of RSS at the minimizer is zero.

These two equations should hold at the minimizer.

$$\frac{d\, RSS(\hat{\beta}_0, \hat{\beta}_1)}{d\hat{\beta}_0} = 0$$

$$\frac{d}{d\hat{\beta}_1} RSS(\hat{\beta}_0, \hat{\beta}_1) = 0$$

$$RSS(\hat{\beta}_0, \hat{\beta}_1) = \sum_{i=1}^{n} \left(y_i - \hat{\beta}_0 + \hat{\beta}_1 x_i \right)^2$$

$$\frac{d}{d\hat{\beta}_0} RSS(\hat{\beta}_0, \hat{\beta}_1) \qquad (1)$$

$$= -2 \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) = 0$$

$$\frac{d}{d\hat{\beta}_1} RSS(\hat{\beta}_0, \hat{\beta}_1) \qquad (2)$$

$$= -2 \sum_{i=1}^{n} x_i (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i) = 0$$

$$(1) \longrightarrow \sum_{i=1}^{n} y_i - n\hat{\beta}_0 - \hat{\beta}_1 \sum_{i=1}^{n} x_i = 0$$

$$\hookrightarrow \div n \qquad \frac{\sum_{i=1}^{n} y_i}{n} - \hat{\beta}_0 - \hat{\beta}_1 \frac{\sum_{i=1}^{n} x_i}{n} = 0$$

$$\bar{y} \qquad\qquad \bar{x}$$

$$\boxed{\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 \bar{x}} \qquad (3)$$

(equivalent to (1))

$$(2) \Rightarrow \sum_{i=1}^{n} x_i y_i - \hat{\beta}_0 \sum_{i=1}^{n} x_i - \hat{\beta}_1 \sum_{i=1}^{n} x_i^2 = 0$$

$$\div n \Rightarrow \frac{\sum x_i y_i}{n} - \underbrace{\hat{\beta}_0 \bar{x}} - \hat{\beta}_1 \frac{\sum x_i^2}{n} = 0$$

use (3)

$$\Rightarrow \frac{\sum x_i y_i}{n} - (\bar{y} - \hat{\beta}_1 \bar{x})\bar{x} - \hat{\beta}_1 \frac{\sum x_i^2}{n} = 0$$

$$\frac{\sum x_i y_i}{n} - \bar{x}\bar{y} = \hat{\beta}_1 \left(\frac{\sum x_i^2}{n} - \bar{x}^2 \right)$$

Two relations (excercise)

$$\frac{\sum_{i=1}^{n} x_i y_i}{n} - \bar{x}\bar{y} = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) \leftarrow$$

$$\frac{\sum_{i=1}^{n} x_i^2}{n} - \bar{x}^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^2$$

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}$$

(3) $\rightarrow \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$

$$\Rightarrow \boxed{\hat{\beta}_0 = \bar{y} - \bar{x} \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}}$$

Question: How good are the above estimators $\hat{\beta}_0, \hat{\beta}_1$?

To evaluate the performance of our estimators of $\hat{\beta}_0, \hat{\beta}_1$, let's assume that the data is generated according to the following probabilistic model:

$$y = \beta_0^{true} + \beta_1^{true} x + \varepsilon$$

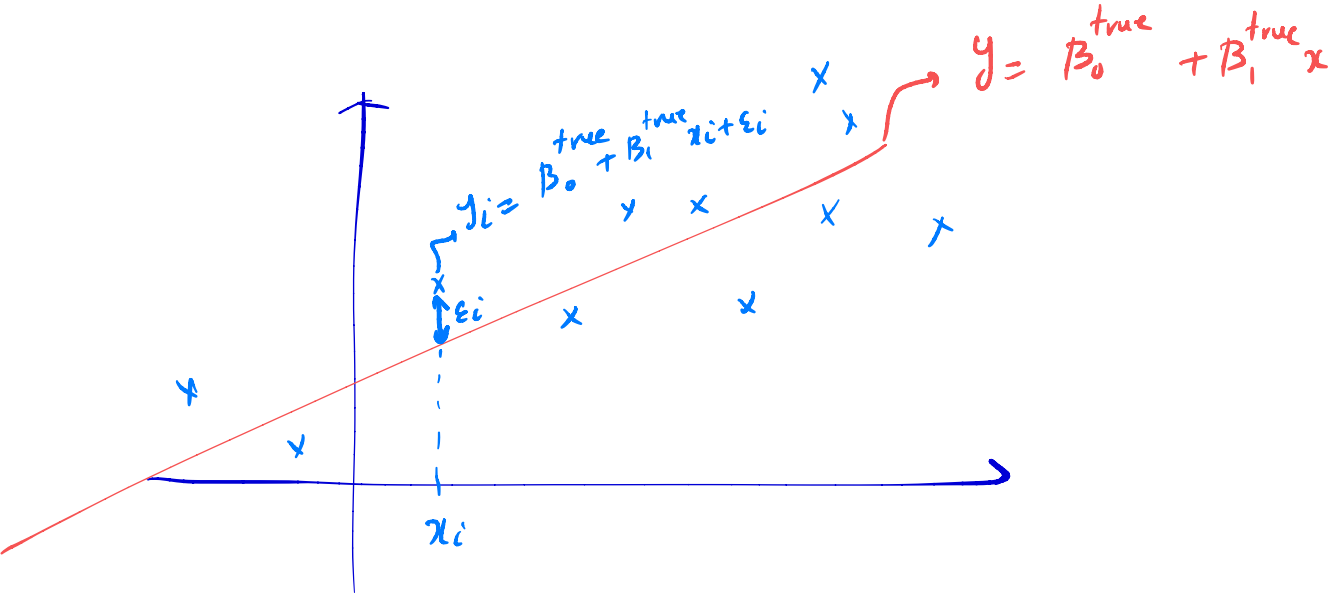
ε: noise
- zero-mean
- independent
- $\text{var}[\varepsilon^2] = \sigma^2$

Now, assume that we generate data points $\{(x_i, y_i)\}_{i=1}^{n}$ from the above model. Let $\hat{\beta}_0, \hat{\beta}_1$ be the solutions of the regression problem.

as $n \to \infty$

$\hat{\beta}_0 \to \beta_0^{true}$, $\hat{\beta}_0$ is an estimator of: β_0^{true}

$\hat{\beta}_1 \to \beta_1^{true}$, $\hat{\beta}_1$ is an estimator of: β_1^{true}



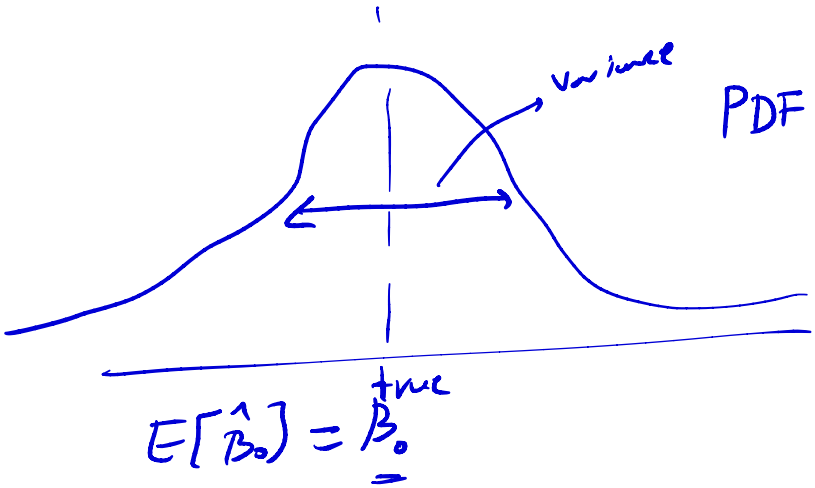
$\hat{\beta}_0 \longrightarrow \beta_0^{true}$

$\hat{\beta}_1 \longrightarrow \beta_1^{true}$

- In order to quantify the performance of the estimates $\hat{\beta}_0, \hat{\beta}_1$, we need to compute their expectation (Bias) and Variance.
- The expectation and the variance of $\hat{\beta}_0$ (and $\hat{\beta}_1$) will tell us how close $\hat{\beta}_0$ (and $\hat{\beta}_1$) are w.r.t their true value β_0^{true} (and β_1^{true})
- Note that, because of the fact that the noises ε_i's are random variables, our

data set $\{(x_i, y_i)\}$ is also random.

This means that $\hat{\beta}_0$, $\hat{\beta}_1$, which depend on the data, are random variables as well.



We'll now compute the expectation of $\hat{\beta}_0$, $\hat{\beta}_1$ and show that they are unbiased estimators of β_0^{true} and β_1^{true}, respectively.

$$E[\hat{\beta}_1] \stackrel{?}{=} \beta_1^{true}$$

$\downarrow$

$$E\left[\hat{\beta}_1\right] = E\left[\frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sum_{i=1}^{n}(x_i-\bar{x})^2}\right]$$

(note: $y_i = \beta_0^{true} + \beta_1^{true} x_i + \varepsilon_i$)

ε_i: zero-mean, independent of everything else

$\forall i$: $y_i = \beta_0^{true} + \beta_1^{true} x_i + \varepsilon_i$

$$\Downarrow$$

$$\bar{y} = \beta_0^{true} + \beta_1^{true} \bar{x} + \bar{\varepsilon} \quad (1) \qquad \left(\bar{\varepsilon} = \frac{\varepsilon_1 + \cdots + \varepsilon_n}{n}\right)$$

$$E\left[\hat{\beta}_1\right] = E\left[\frac{\sum (x_i-\bar{x})(y_i-\bar{y})}{\sum (x_i-\bar{x})^2}\right]$$

$$\overset{(1)}{=} E\left[\frac{\sum_i (x_i-\bar{x})\left(\beta_1^{true}(x_i-\bar{x})+\varepsilon_i-\bar{\varepsilon}\right)}{\sum_i (x_i-\bar{x})^2}\right]$$

$$(y_i - \bar{y})$$

$$(\beta_0^{true} + \beta_1^{true} x_i + \varepsilon_i) - (\beta_0^{true} + \beta_1^{true} \bar{x} + \bar{\varepsilon})$$

$$= \beta_1^{true} (x_i - \bar{x}) + \varepsilon_i - \bar{\varepsilon}$$

$$E[\hat{\beta}_1] = E\left[\frac{\beta_1^{true} \sum_i (x_i - \bar{x})^2 + \sum (\varepsilon_i - \bar{\varepsilon})(x_i - \bar{x})}{\sum_i (x_i - \bar{x})^2}\right]$$

$$= \beta_1^{true} + E\left[\frac{\sum_{i=1}^{n} (\varepsilon_i - \bar{\varepsilon})(x_i - \bar{x})}{\sum_i (x_i - \bar{x})^2}\right]$$

noises were independent of everything else

$$= \beta_1^{true} + \sum_{i=1}^{n} E\left[\frac{(\varepsilon_i - \bar{\varepsilon})(x_i - \bar{x})}{\sum_i (x_i - \bar{x})^2}\right]$$

$$= \beta_1^{true} + \sum_{i=1}^{n} E[\varepsilon_i - \bar{\varepsilon}] \, E\left[\frac{(x_i - \bar{x})}{\sum_i (x_i - \bar{x})^2}\right]$$

$E[\varepsilon_i] - E[\bar{\varepsilon}]$ → 0 (each)

$$\Rightarrow \quad E[\hat{\beta}_1] = \beta_1^{true}$$

In general:

(1) $E[\hat{\beta}_1] = \beta_1^{true}$

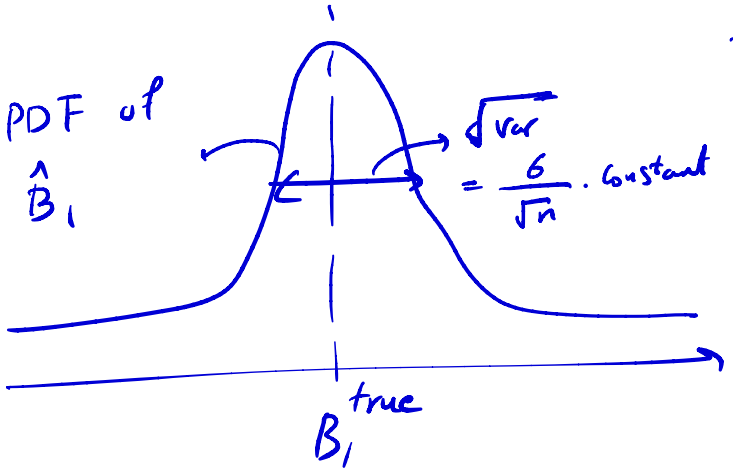
(2) $E[\hat{\beta}_0] = \beta_0^{true}$

(3) $Var(\hat{\beta}_0) = E\left[(\hat{\beta}_0 - \beta_1^{true})^2\right]$

$$\left(Var(\varepsilon_i) = \sigma^2\right) \qquad = \sigma^2\left(\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^{n}(x_i-\bar{x})^2}\right)$$

(4) $Var(\hat{\beta}_1) = \dfrac{\sigma^2}{\sum_{i=1}^{n}(x_i-\bar{x})^2} = \dfrac{\sigma^2}{n\left(\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2\right)}$

$$= \frac{\sigma^2}{n \cdot (\text{empirical variance of } \{x_i\})}$$



So far we've only been talking about the 1-dim case ($p=1$). Let's now consider the general case:

$$\{(x_i, y_i)\}_{i=1}^{n} \qquad x_i \in \mathbb{R}^p, \; y_i \in \mathbb{R}$$

$$x_i = \begin{pmatrix} x_{i1} \\ \vdots \\ x_{ip} \end{pmatrix} \in \mathbb{R}^p, \; y_i \in \mathbb{R}$$

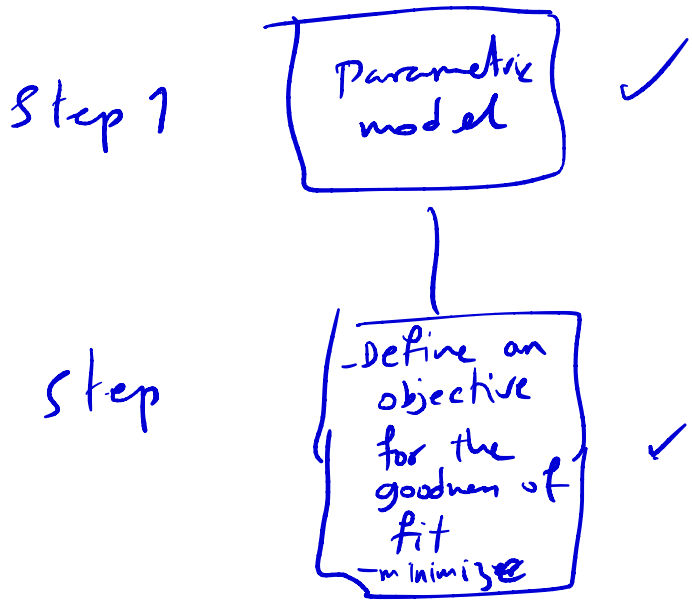
simplest parametric family (linear):

$$y_i \simeq \underbrace{\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \hat{\beta}_2 x_{i2} + \cdots + \hat{\beta}_p x_{ip}}_{f(x_i, \vec{\beta})}$$

Lecture 15

P-dimensional linear regression:

Data: $\{(x_i, y_i)\}_{i=1}^{n}$ $\quad x_i \in \mathbb{R}^p$



Step 1:

$$x_i = \begin{pmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{ip} \end{pmatrix} \in \mathbb{R}^p \longrightarrow y_i \approx \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip}$$

p+1 parameters (to be learned from data)

step 2:

$$RSS = \sum_{i=1}^{n} (\text{error}_i)^2$$

$$= \sum_{i=1}^{n} \left(y_i - \left(\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip} \right) \right)^2$$

y_i: true label; $\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip}$: prediction

to find $\{\hat{\beta}_j\}_{j=0}^{p}$ we'll have to minimize RSS.

$$RSS = \sum_{i=1}^{n} (y_i - \tilde{x}_i^T \vec{\beta})^2$$

$$\hat{\beta} = \begin{pmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_p \end{pmatrix} \in \mathbb{R}^{p+1}, \quad \tilde{x}_i = \begin{pmatrix} 1 \\ x_{i1} \\ x_{i2} \\ \vdots \\ x_{ip} \end{pmatrix} \in \mathbb{R}^{p+1}$$

$$RSS = \| \vec{Y} - X \vec{\beta} \|_2^2$$

$\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip}$

$= [1, x_{i1}, x_{i2}, \ldots, x_{ip}] \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_p \end{bmatrix}$ (data i)

- $\vec{Y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}_{n \times 1}$ $\quad \vec{X} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & & & \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix}_{n \times (p+1)}$

$$= \begin{pmatrix} \tilde{x}_1^T \\ \hline \tilde{x}_2^T \\ \hline \vdots \\ \hline \tilde{x}_n^T \end{pmatrix}$$

- $\|z\|_2^2 = \sum_{i=1}^{n} z_i^2$

$z_2 \begin{pmatrix} z_1 \\ \vdots \\ z_n \end{pmatrix}$

$$\underset{\vec{\beta}}{\text{minimize}} \; RSS(\vec{\beta})$$

$$\underset{\vec{\beta}}{\text{minimize}} \; \underbrace{\| \vec{y} - X\vec{\beta} \|_2^2}_{RSS(\vec{\beta})}$$

$$\Rightarrow \quad \underbrace{\nabla RSS(\vec{\beta}) = 0}$$

$$-2 X^T (\vec{y} - X\vec{\beta}) = 0$$

$$\Rightarrow X^T X \vec{\beta} = X^T \vec{y}$$

$$\Rightarrow \boxed{\vec{\beta} = (X^T X)^{-1} X^T \vec{y}}$$

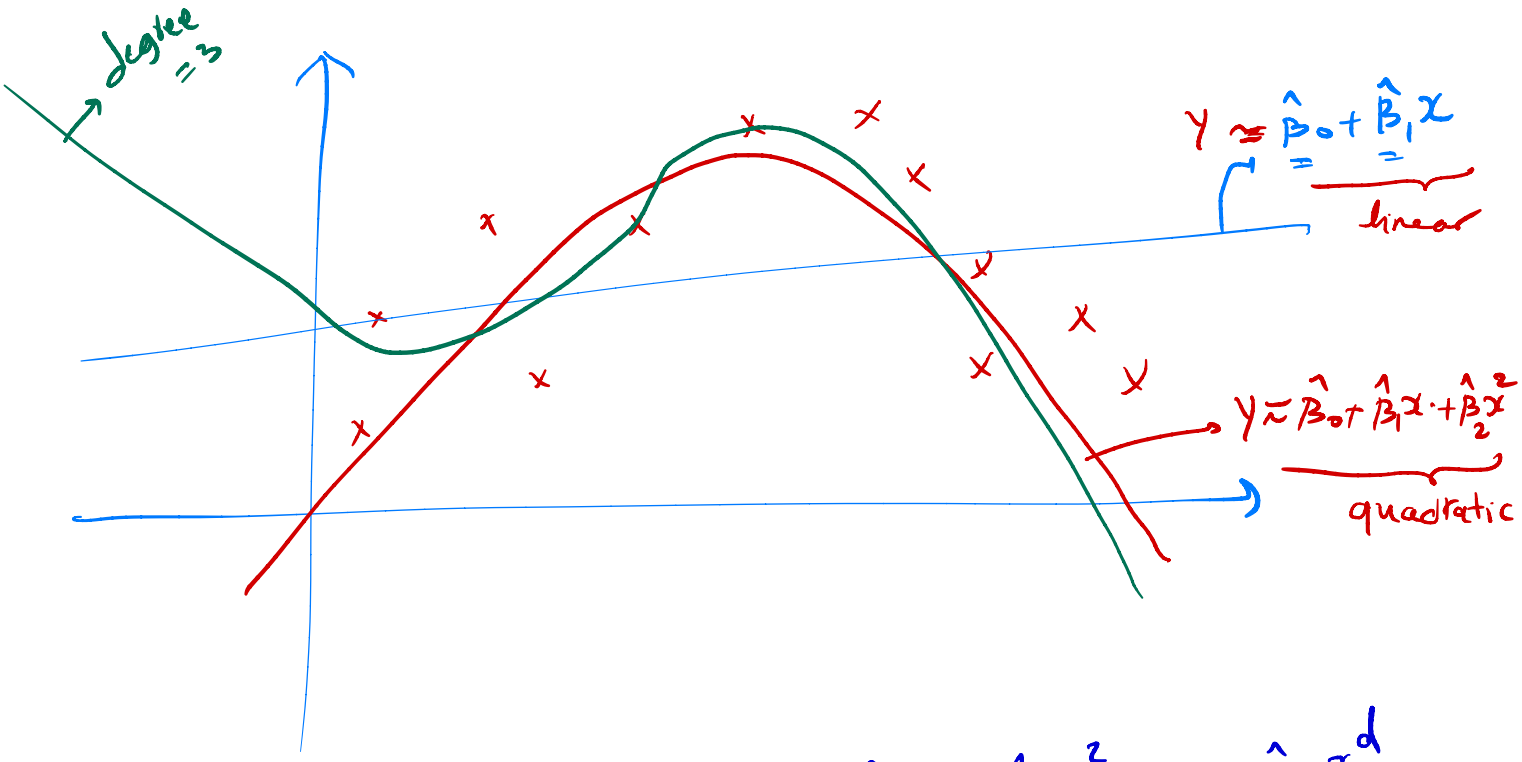
$$\begin{pmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_p \end{pmatrix}$$

$$X = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ 1 & & & \\ \vdots & & & \\ 1 & x_{i1} & \cdots & x_{ip} \\ \vdots & & & \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix} \quad \text{i-th} \rightarrow$$

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

- So far, we've only talked about linear models. Let's now see how we can construct more complex (non-linear) parametric models.

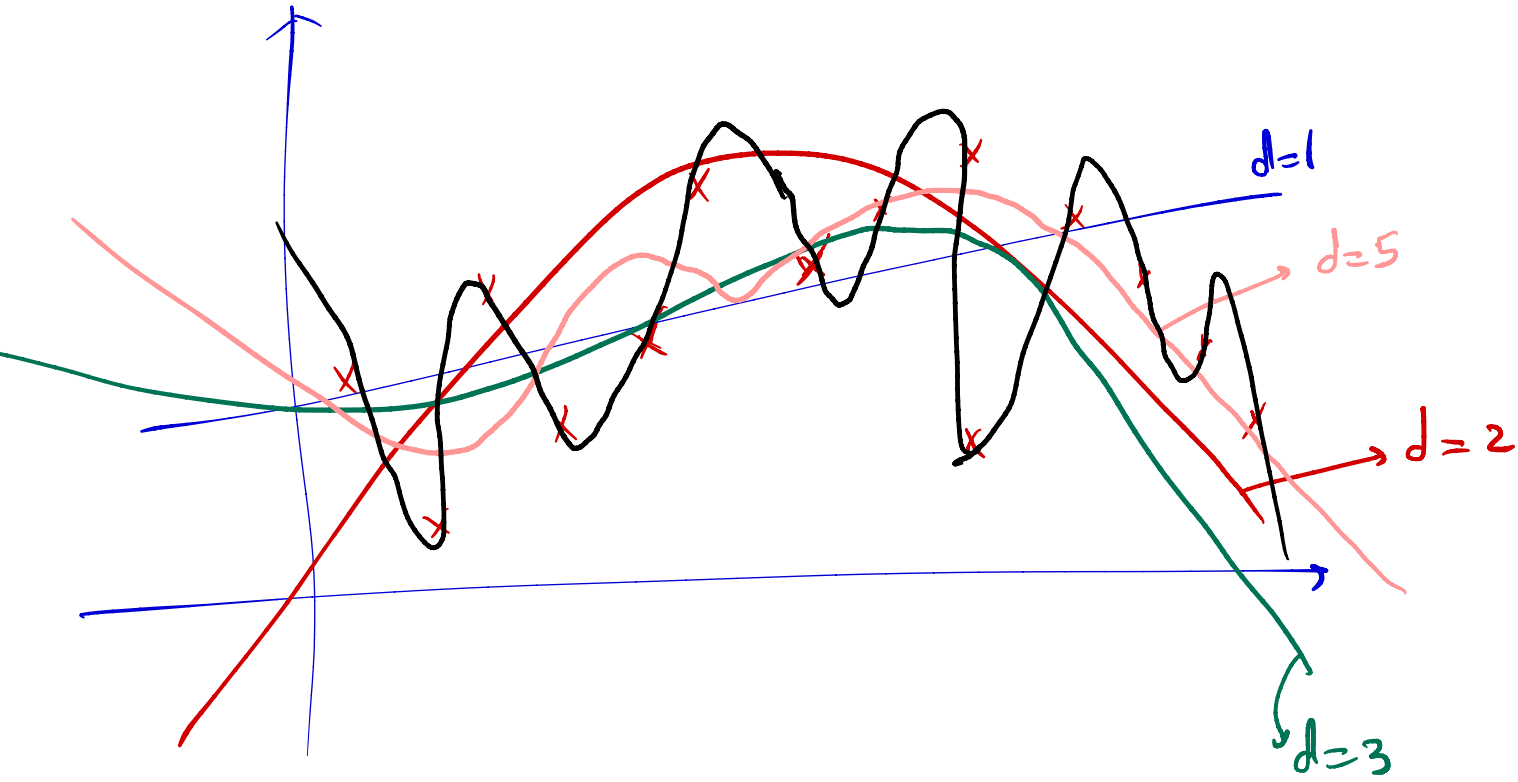
Let's assume $p=1$.



polynomial : $y = p(x) = \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2 + \cdots + \hat{\beta}_d x^d$

One way to construct non-linear parametric models is by using polynomial models:

- degree of the model : d
- parameters : $\hat{\beta}_0, \hat{\beta}_1, \cdots, \hat{\beta}_d$ ($d+1$ parameters)



d=1000

$$y \approx \hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2 + \cdots + \hat{\beta}_{1000} x^{1000}$$

Question: Given a polynomial family of degree d how do we find the best parameters (the best fit)?

Data: $\{(x_i, y_i)\}_{i=1}^{n}$ $\quad x_i \in \mathbb{R}$, $y_i \in \mathbb{R}$

Step 1: Parametric model: polynomials of degree d: $y = \hat{\beta}_0 + \hat{\beta}_1 x + \cdots + \hat{\beta}_d x^d$

Step: Define an objective for the goodness of fit:

$$\xrightarrow{\text{objective}} RSS = \sum_{i=1}^{n} \text{error}_i^2$$

$$= \sum_{i=1}^{n} \left(y_i - \left(\hat{\beta}_0 + \hat{\beta}_1 x_i + \hat{\beta}_2 x_i^2 + \cdots + \hat{\beta}_d x_i^d \right) \right)^2$$

this objective is analogous to the multidimensional linear regression:

$$\left(\begin{pmatrix} 1 \\ x_1 \\ x_1^2 \\ \vdots \\ x_1^d \end{pmatrix} \right)$$

$$x_i \in \mathbb{R} \longrightarrow \tilde{x}_i = \begin{pmatrix} 1 \\ x_i \\ x_i^2 \\ \vdots \\ x_i^d \end{pmatrix}, \quad \vec{\beta} = \begin{pmatrix} \hat{\beta}_0 \\ \vdots \\ \hat{\beta}_d \end{pmatrix}$$

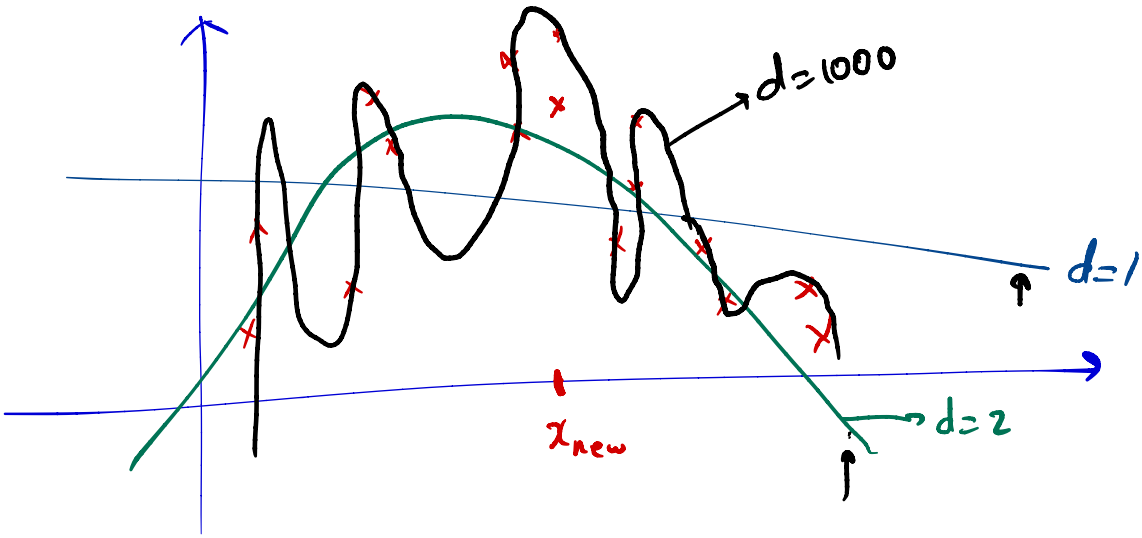
new data point #i

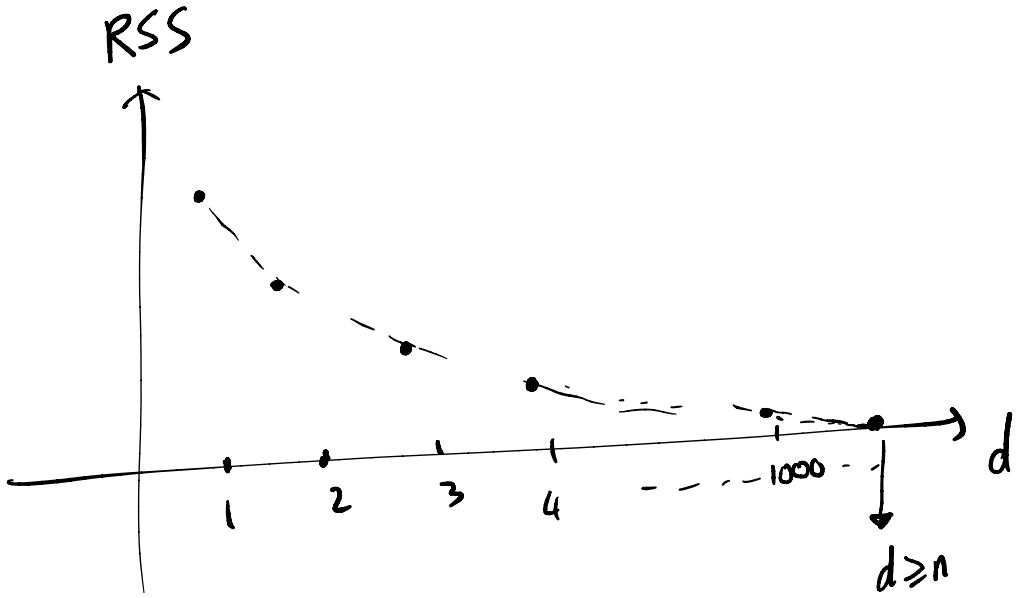
$$RSS = \sum_{i=1}^{n} \left(y_i - \tilde{x}_i^T \vec{\beta} \right)^2$$

$$\Rightarrow \vec{\beta} = (X^T X)^{-1} (X^T Y)$$

$$X = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^d \\ 1 & x_2 & x_2^2 & \cdots & x_2^d \\ \vdots & & & & \\ 1 & x_n & x_n^2 & \cdots & x_n^d \end{pmatrix}_{n \times (d+1)}$$

$$\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$$





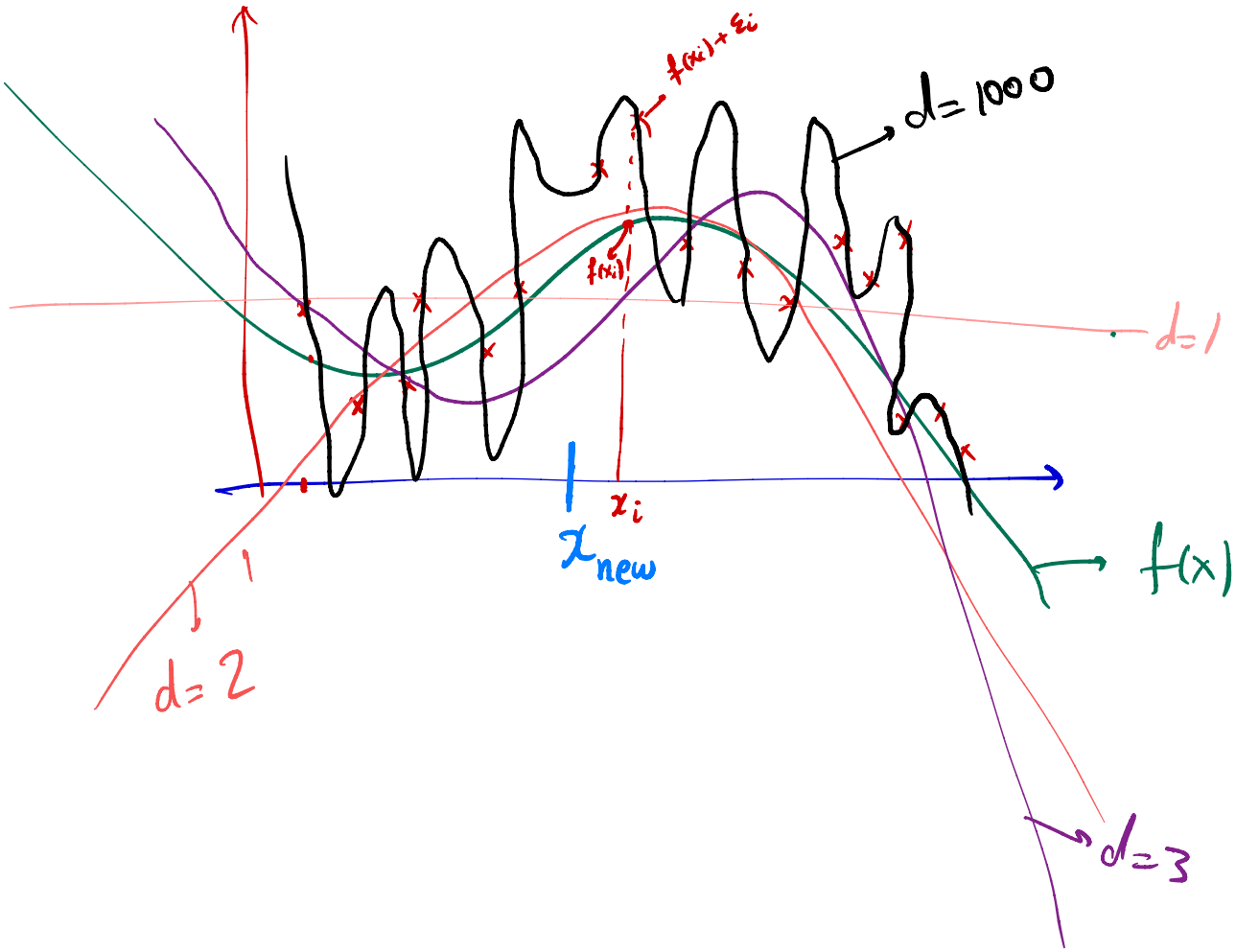
- Remember that the main goal behind learning predictive models was to be able to predict well on new and unseen data points.

- Consider the following model for data:

$$y = f(x) + \varepsilon$$

(f: how the data is generate; ε: noise)

e.g. $f(x) = x^3 + 3x^2 - 4x + 5$



- On a new data point x_{new}

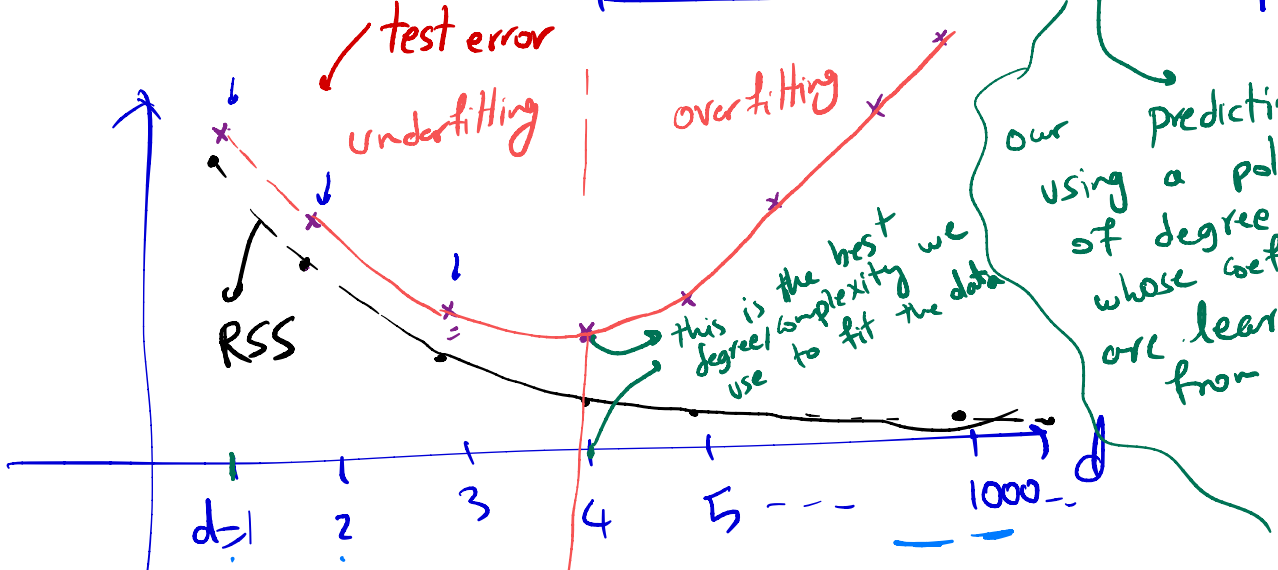
→ $\underline{\underline{y_{new} = f(x_{new}) + \varepsilon}}$

the model learned from data

w

→ Test error = $E\left[\left(y_{new} - \hat{P}_d(x_{new}, \vec{\beta}) \right)^2 \right]$

$y_{new} = f(x_{new}) + \varepsilon$



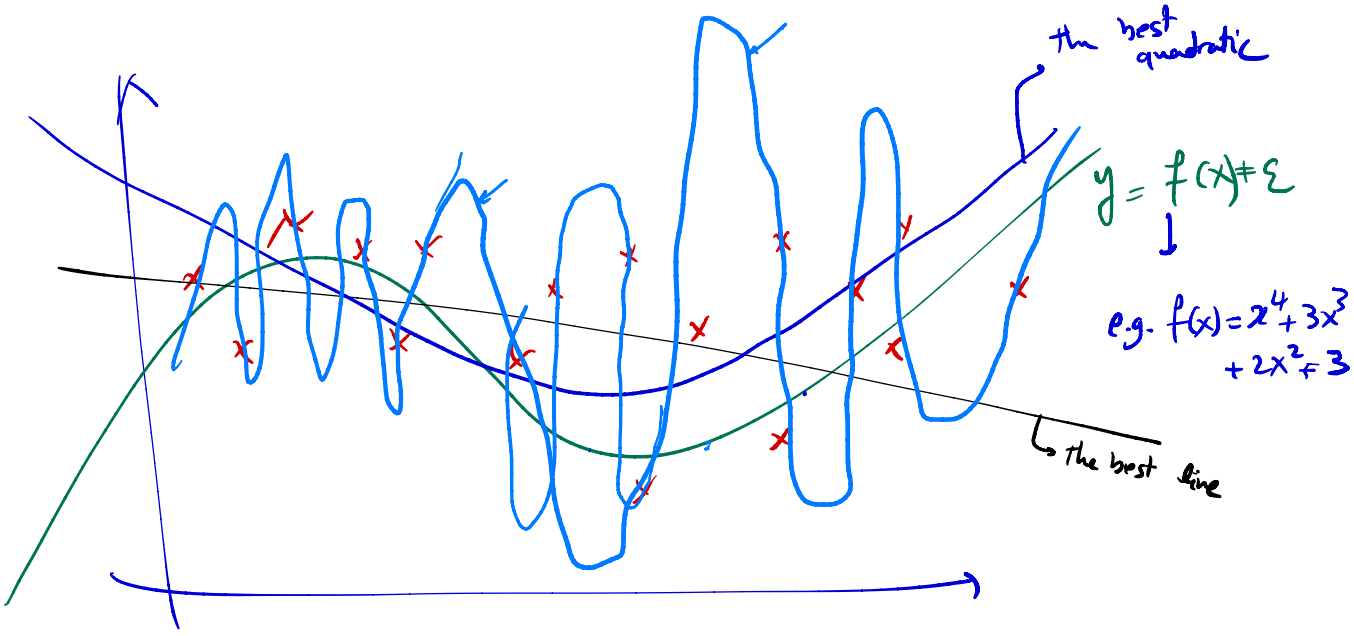
our prediction using a polynomial of degree d whose coefficients are learned from data

We have two regions depending on the complexity (degree) of the parametric model that we're fitting to the data:

- Underfitting:
  - the model is not sufficiently complex to explain the data (high bias)
  - the trained model does not change much if we add new training data points (low variance)

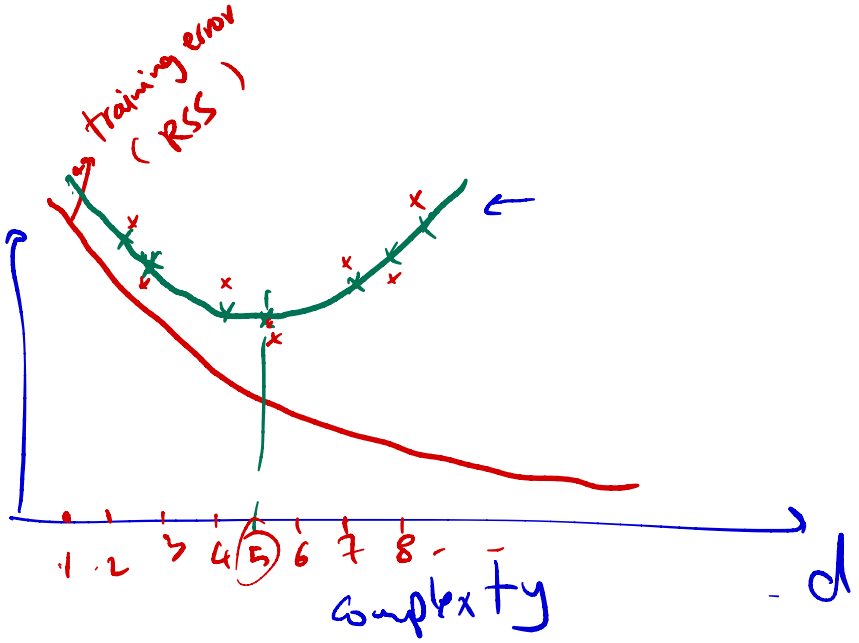
- Overfitting:
  - model is too complex for training the data (low bias)
  - the trained model will change significantly if we add new data points (high variance)

Lecture 16 :



the best fit / model with degree 100

Question : How can we estimate the the test error curve?

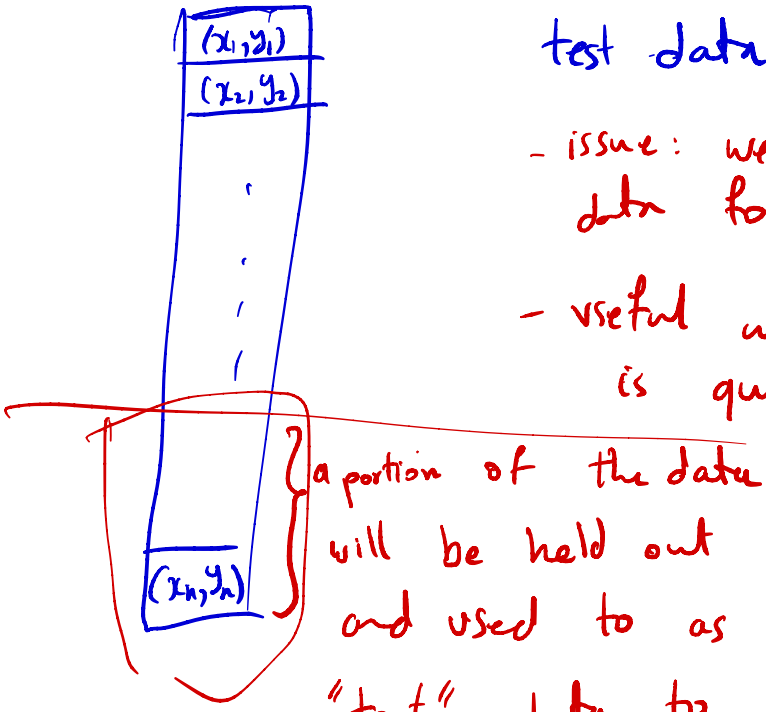


Two methods / answers:

(1) validation / hold-out data:

Randomly choose a (given) portion of the data and use it as test data. Use the rest for training.

- issue: we're not using all the available data for training
- useful when the size of the data is quite large



(2) k-fold cross-validation:

Partition/Divide the data randomly into k parts:

$$D = D_1 \cup D_2 \cup D_3 \cdots \cup D_k$$

for each $i=1,\dots,k$

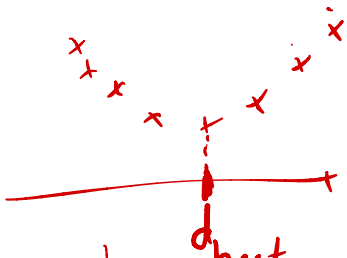
- train on $D \setminus D_i$
- test the resulting model on D_i

$\Rightarrow$ $test_i$ = test error of the trained model on D_i

$$test\text{-}error \approx \frac{1}{k}\left(test_1 + test_2 + \dots + test_k\right)$$

$\Rightarrow$ - both of the methods are used to estimate the test curve

- once the "best complexity" is found (minimum in the estimated test curve), we will use the whole data set to train a model with the "best complexity"



- Cross validation is done for each complexity to estimate the test error.

Classification:

Regression:

label $y_i \in \mathbb{R}$

classification:

label $\in$ Discrete set

$y_i \in \{0, 1\}$

$\in \{$ Spam, not-spam $\}$

$\in \{$ Sunny, Snow, rain $\}$

Example:

$\{(x_i, y_i)\}_{i=1}^{n}$

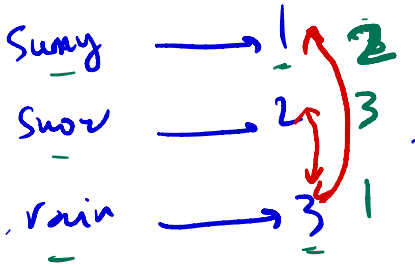
x_i = weather data for day i (clouds, ---)

$y_i = \{$ sunny, snowy, rainy $\}$

$y_i \approx f(x_i)$

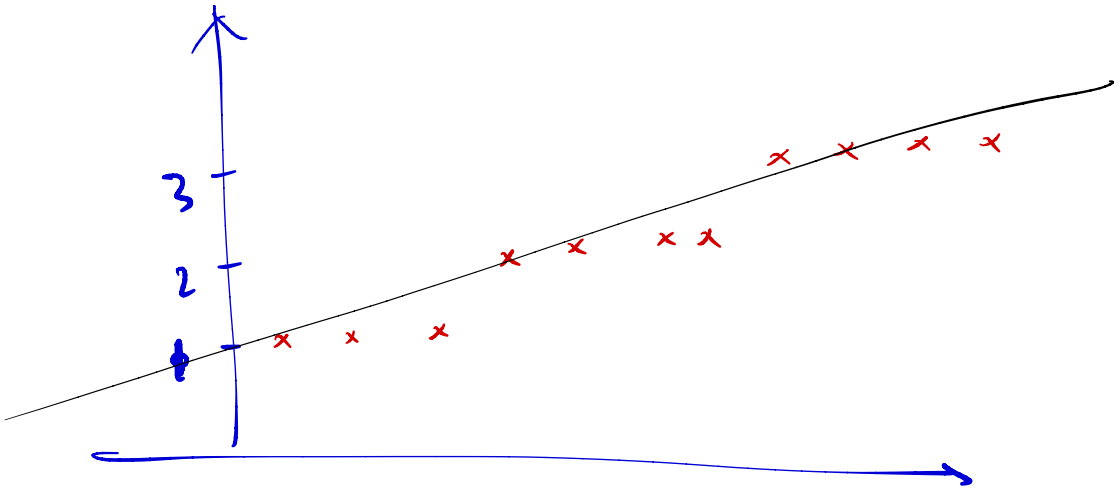
(remember, we want to be accurate at test time.

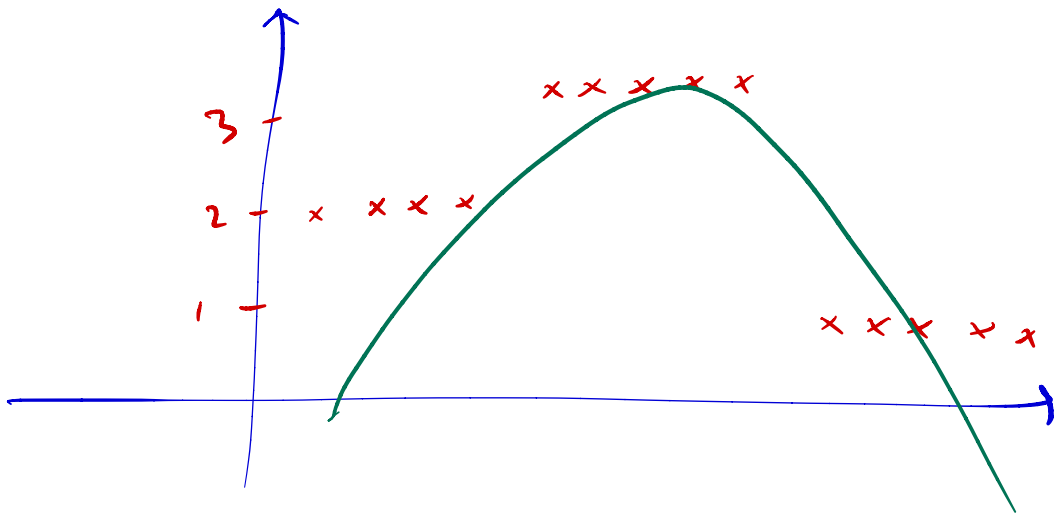
As a naive approach, let's try model/solve a classification problem using regression:



$y_i : 1, 2, 3$ → real-valued

→ let's fit a regression model





Example:

input x (input data
weather data)

$$\begin{cases} \Pr\{\text{Sunny} \mid x\} = 0.2 \\ \Pr\{\text{Snow} \mid x\} = 0.5 \\ \Pr\{\text{rain} \mid x\} = 0.3 \end{cases}$$

- what we really need to learn/estimate is the "conditional probabilities":

$$\Pr\{\text{class}_k \mid x\} \quad \text{for } k = 1, \ldots, K$$

Bayes Optimal Classifier:

Assume for the moment that all the conditional probabilities are known

$\Pr\{\text{class}_k \mid x\}$ is known for $k = 1, \ldots, K$.

How should we (optimally) predict the

class associated to input x?

Bayes optimal classifier:

$$\hat{y}_{Bayes}(x) = \text{predicted class}(x)$$

$$= \underset{k}{\text{argmax}} \; Pr\{\text{class}_k \mid x\}$$

(break ties arbitrarily)

Theorem: The Bayes optimal classifier has the smallest classification error among all the possible classifiers.

This means: for every other classifier $\hat{y}_{other}(x)$, we have:

$$Pr_{(x,y)}\{\hat{y}_{other}(x) \neq y\} \geq Pr_{(x,y)}\{\hat{y}_{Bayes}(x) \neq y\}$$

- The fundamental problem with the Bayes classifier is that it assumes the knowledge of the conditional probabilities (which not true in practice)

- All the methods for classification in ML aim at estimating the conditional probabilities.

- We'll cover several methods to estimate these conditional probabilities from data.

Logistic Regression:

For now, assume that we have two classes (the binary classification problem):

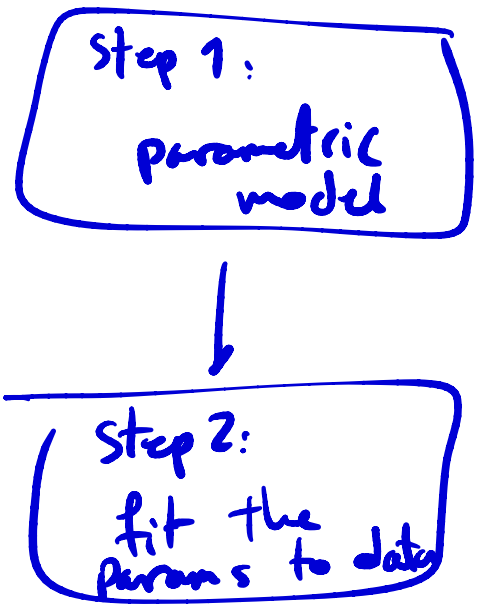
$\text{Class}_0 \longrightarrow y_i = 0$

$x_i \in \mathbb{R}$

$\text{Class}_1 \longrightarrow y_i = 1$

Goal: given $x \longrightarrow \begin{cases} \Pr\{0 \mid x\} \\ \Pr\{1 \mid x\} \end{cases}$

Data: $\{(x_i, y_i)\}_{i=1}^{n}$



Step: we're going to design a parametric model for estimation

$$\underbrace{Pr\{0|x\}}_{\in[0,1]}, \quad \underbrace{Pr\{1|x\}}_{\in[0,1]}$$

Let's begin with linear models.

What is a linear model in the binary classification setting?

$$Pr\{0|x\} = f(x; \underline{\text{parameters}})$$
$$Pr\{1|x\} = 1 - f(x; \text{parameters})$$

let's come up with simplest candidates for the parametric model.

- Perhaps the simplest model would be something like

$$\begin{cases} Pr\{0 \mid x\} = \beta_0 + \beta_1 x \\ Pr\{1 \mid x\} = 1 - (\beta_0 + \beta_1 x) \end{cases}$$

But, one can not enforce the constraint that the probabilities are between 0 and 1 using the above model (i.e. it's not possible to ensure that $\beta_0 + \beta_1 x \in [0,1]$).

Instead, we consider the following model:

$$\begin{cases} \Pr\{0|x\} = \dfrac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}} \\ \Pr\{1|x\} = 1-\Pr\{0|x\} = \dfrac{1}{1+e^{\beta_0+\beta_1 x}} \end{cases}$$

It's easy to see that $\dfrac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}} \in [0,1]$.

Lecture 17:

Step 1: Devise a parametric model:

$$Pr\{0|x\} = \frac{e^{B_0 + B_1 x}}{1 + e^{B_0 + B_1 x}} \leftarrow$$

$$Pr\{1|x\} = 1 - Pr\{0|x\}$$

Step 2:

Define an objective for the "goodness of fit" → minimize the objective → find $\hat{B}_0, \hat{B}_1$

$$Pr\{y_i | x_i, B_0, B_1\} \begin{cases} y_i = 0 \rightarrow \frac{e^{B_0 + B_1 x_i}}{1 + e^{B_0 + B_1 x_i}} \\ y_i = 1 \rightarrow \frac{1}{1 + e^{B_0 + B_1 x_i}} \end{cases}$$

$$Pr\{x_i, y_i | B_0, B_1\} = Pr\{x_i\} \cdot Pr\{y_i | x_i, B_0, B_1\}$$

$$P(\underset{x_i}{A}, \underset{y_i}{B}) = P(\underset{x_i}{A}) \cdot Pr(\underset{y_i}{B} | \underset{x_i}{A})$$

as we learned in module 1, mle can be used to estimate parameters of the distribution that generates the data

mle:

$$lik(\beta_0, \beta_1) = \prod_{i=1}^{n} Pr\{x_i, y_i \mid \beta_0, \beta_1\}$$

$$= \prod_{i=1}^{n} Pr\{x_i\} \, Pr\{y_i \mid x_i; \beta_0, \beta_1\}$$

$$= \underbrace{\prod_{i=1}^{n} Pr\{x_i\} \cdot \prod_{i=1}^{n} Pr\{y_i \mid x_i; \beta_0, \beta_1\}}_{\text{objective to be optimized}}$$

$$\underset{\beta_0, \beta_1}{\operatorname{argmax}} \; lik(\beta_0, \beta_1)$$

the same value for any choice of β_0, β_1

$$= \underset{\beta_0, \beta_1}{\operatorname{argmax}} \; \underbrace{\prod_{i=1}^{n} Pr\{x_i\}}_{C} \cdot \prod_{i=1}^{n} Pr\{y_i \mid x_i; \beta_0, \beta_1\}$$

$$= \cancel{C}' \underset{\beta_0 \beta_1}{\operatorname{argmax}} \; \prod_{i=1}^{n} Pr\{y_i \mid x_i; \beta_0, \beta_1\}$$

$$= \underset{\beta_0, \beta_1}{\operatorname{argmax}} \; \sum_{i=1}^{n} \log Pr\{y_i \mid x_i; \beta_0, \beta_1\}$$

$$= \underset{\beta_0, \beta_1}{\operatorname{argmax}} \; \sum_{i: y_i = 0} \log\left(\frac{e^{\beta_0 + \beta_1 x_i}}{1 + e^{\beta_0 + \beta_1 x_i}}\right) + \sum_{i: y_i = 1} \log\left(\frac{1}{1 + e^{\beta_0 + \beta_1 x_i}}\right)$$

$$= \underset{\beta_0, \beta_1}{\operatorname{argmax}} \underbrace{\left\{ \sum_{i: y_i = 0} (\beta_0 + \beta_1 x_i) - \sum_{i=1}^{n} \log\left(1 + e^{\beta_0 + \beta_1 x_i}\right) \right\}}_{g(\beta_0, \beta_1)}$$

↳ Convex in terms of β_0, β_1

$$\begin{cases} \dfrac{\partial g}{\partial \beta_0} = 0 \\ \dfrac{\partial g}{\partial \beta_1} = 0 \end{cases} \Longrightarrow$$

- two equations in terms of the two unknowns β_0, β_1.
- no closed-form solution.
  ⇒ we need to solve it numerically / iteratively
- the solution is unique.

$\Longrightarrow \hat{\beta}_0, \hat{\beta}_1$ ← the best fit to the data.

given a new data point x →

$$\hat{Pr}\{0|x\} = \frac{e^{\hat{\beta}_0 + \hat{\beta}_1 x}}{1 + e^{\hat{\beta}_0 + \hat{\beta}_1 x}}$$

$$\hat{Pr}\{1|x\} = \frac{1}{1 + e^{\hat{\beta}_0 + \hat{\beta}_1 x}}$$

$$\rightarrow \hat{y}_{\text{logistic\_reg}}(x) = \underset{y \in \{0,1\}}{\operatorname{argmax}} \hat{Pr}\{y|x\}$$

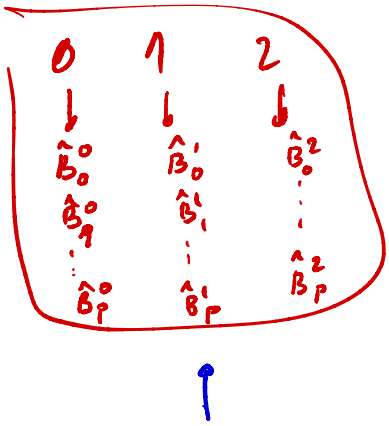
- logistic - regression for $x \in \mathbb{R}^p \rightarrow x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{pmatrix} \in \mathbb{R}^p$

- Binary labels: $y \in \{0,1\}$

$$Pr\{y=0 \mid x\} = \frac{e^{B_0 + B_1 x_1 + \cdots + B_p x_p}}{1 + e^{B_0 + B_1 x_1 + \cdots + B_p x_p}}$$

$$Pr\{y=1 \mid x\} = 1 - Pr\{y=0 \mid x\}$$

- general label set:



$y \in \{0,1,2\}$

$$Pr\{i \mid x\} = \frac{e^{\hat{B}_0^i + \hat{B}_1^i x_1 + \cdots + \hat{B}_p^i x_p}}{\sum_{j=0,1,2} e^{\hat{B}_0^j + \hat{B}_1^j x_1 + \cdots + \hat{B}_p^j x_p}}$$

logistic regression is typically used for settings with binary labels ($y \in \{0,1\}$), and it does not perform well when we have multiple classes.

- Linear / Quadratic Discriminant Analysis:

- Linear Discriminant Analysis (LDA):

$$Pr\{ y \mid x \}$$

Logistic regression directly models this conditional probabilities.

LDA/QDA try to estimate the conditional probabilities through a different route → Baye rule.

Baye's rule

$$Pr\{ y=k \mid X=x \} = \frac{Pr\{ X=x \mid y=k \} \cdot Pr\{ y=k \}}{Pr\{ X=x \}}$$

LDA/QDA aim at estimating these probabilities.

$$Pr\{ X=x \} = \sum_{k=1}^{K} Pr\{ y=k \} Pr\{ X=x \mid y=k \}$$

Assume K classes:

$f_k(x) = \text{gaussian}(\mu_k, \sigma^2)$

$$\Pr\{y=k \mid x=x\} = \frac{\Pr\{x=x \mid y=k\}\,\Pr\{y=k\}}{\Pr\{x=x\}}$$

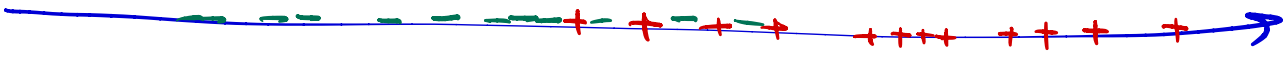
$\Pr\{y=k\}$

$k = 1, \dots, K$

$$\text{Data} = \{(x_i, y_i)\}_{i=1}^{n}$$

example $(x_i \in \mathbb{R},\ y_i \in \{-,+\})$

$\Pr\{x=x \mid y=+\}$



$$\Pr\{y=+\} \overset{\text{estimate}}{\approx} \frac{\#\{i : y_i = +\}}{n}$$

$$Pr\{y=k\} \simeq \frac{\#\{i : y_i = k\}}{n}$$

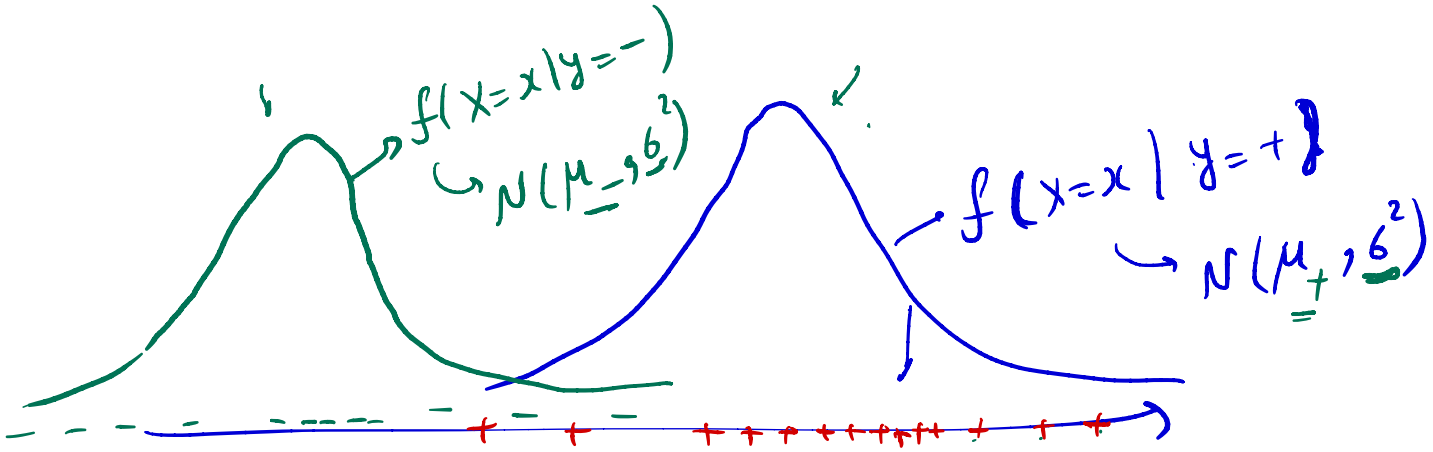
↓

Unbiased estimate of $pr\{y=k\}$

$$f\{X=x \mid Y=k\} \overset{LDA}{=} f_k(x)$$

$$= \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{(x-\mu_k)^2}{2\sigma^2}}$$

LDA "assumes" / "models" that data, given a class k, is generated according to the gaussian distribution.



In LDA, we assume that the variance of the data x is the same over all the classes:

For a gaussian, we need to estimate two parameters: μ_k, σ.

μ_k ← mean of the density

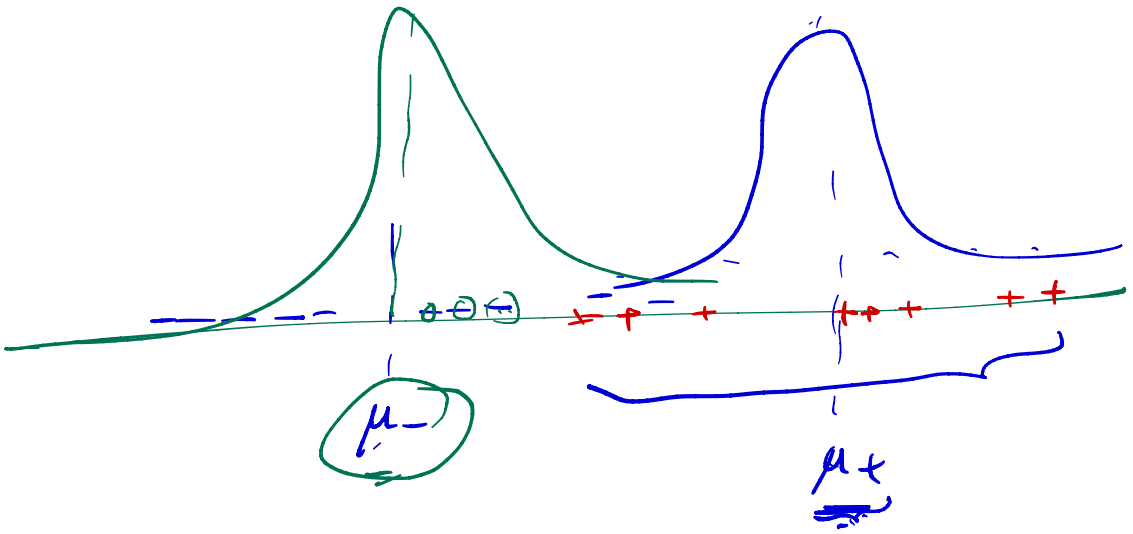
$$\hat{\mu}_k \simeq \frac{\sum_{i: y_i = k} x_i}{n_k}$$ → sample mean

$$n_k = \#\{ i : y_i = k \}$$

Let's now estimate σ:

$$\hat{\sigma}^2 = \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i: y_i = k} (x_i - \hat{\mu}_k)^2$$

$\sum_{i: y_i = k}$: variance across class k



---

* LDA uses the same Σ for each class, $\Rightarrow$ LDA requires to learn less parameters that the case where we assign different Σ's to the classes.

$\Rightarrow$ QDA → uses different Σ's for the classes.

LDA :

Lecture 18:

LDA $\longrightarrow$ $\Pr\{Y=k \mid X=x\}$

$\approx$ $\hat{f}_k(x)$ $\hat{\pi}_k$ $\Pr\{X=x\}$

$\hat{f}_k(x)$: $N(\mu_k, \tilde{\sigma}^2)$

$$\hat{\pi}_k = \frac{\#\{i: y_i = k\}}{n}$$ (an estimate for $\Pr\{y=k\}$)

$$\frac{1}{\sqrt{2\pi}\hat{\sigma}} e^{-\frac{(x-\hat{\mu}_k)^2}{2\hat{\sigma}^2}}$$ (an estimate for $\Pr\{X=x \mid y=k\}$)

At prediction time, given a new input x:

$$\hat{y}(x) = \underset{k}{\operatorname{argmax}} \Pr\{y=k \mid X=x\}$$

$$= \underset{k}{\operatorname{argmax}} \frac{\hat{f}_k(x) \cdot \hat{\pi}_k}{\sum_{l=1}^{K} \hat{f}_l(x) \cdot \hat{\pi}_l}$$

the same for every class k

$$= \underset{k}{\operatorname{argmax}} \; \hat{f}_k(x) \, \hat{\pi}_k$$

$$\frac{1}{\sqrt{2\pi}\hat{\sigma}} e^{-\frac{(x-\hat{\mu}_k)^2}{2\hat{\sigma}^2}}$$

$$= \underset{k}{\operatorname{argmax}} \left\{ \log \hat{f}_k(x) + \log \hat{\pi}_k \right\}$$

$$= \underset{k}{\operatorname{argmax}} \left\{ \log \frac{1}{\sqrt{2\pi}\hat{\sigma}} - \frac{(x-\hat{\mu}_k)^2}{2\hat{\sigma}^2} + \log \hat{\pi}_k \right\}$$

Because of the assumption that the variances were the same over all the classes, we can remove this quadratic term $\frac{x^2}{2\hat{\sigma}^2}$

$$= \underset{k}{\operatorname{argmax}} \left\{ \log \frac{1}{\sqrt{2\pi}\hat{\sigma}} + \frac{x^2}{2\hat{\sigma}^2} - \frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + \frac{x\hat{\mu}_k}{\hat{\sigma}^2} + \log \hat{\pi}_k \right\}$$

do not depend on the class k

$$= \underset{k}{\operatorname{argmax}} \left\{ -\frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + \frac{x\hat{\mu}_k}{\hat{\sigma}^2} + \log \hat{\pi}_k \right\}$$

linear in x

- Since the decision rule is linearly dependent on x, this classifier is called "linear" discriminant analysis.

Extension to the multi-dimensional setting:

$$\text{Data} = \left\{ (x_i, y_i) \right\}_{i=1}^{n} \quad \begin{cases} y_i \in \{1, \dots, K\} \\ x_i \in \mathbb{R}^p \end{cases}$$

$$\Pr\{ y = k \mid X = x \} = \frac{\overbrace{\Pr\{ X = x \mid y = k \}}^{f_k(x) \sim N(\tilde{\mu}_k, \tilde{\Sigma})} \cdot \overbrace{\Pr\{ y = k \}}^{\pi_k}}{\Pr\{ X = x \}}$$

the same over all the classes

Multi-variate Gaussian Distribution

$$\left(\mathcal{N}(\mu, \Sigma)\right)$$

$\mu \in \mathbb{R}^p$; Σ: $p \times p$ matrix

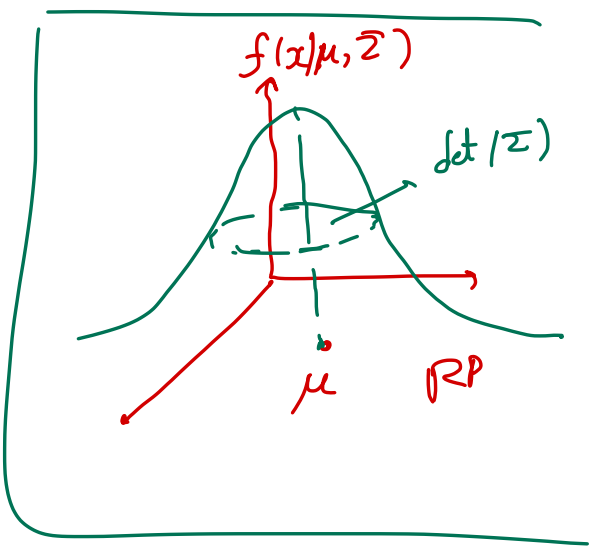
$$f(x \mid \underbrace{\mu}_{\text{mean}}, \underbrace{\Sigma}_{\text{covariance matrix}}) = \frac{1}{(2\pi)^{p/2} \cdot \det(\Sigma)^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

($(x-\mu)^T$: transpose)

properties:

$$X \sim \mathcal{N}(\mu, \Sigma)$$

$$X = \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_p \end{pmatrix} \in \mathbb{R}^p$$



$$E[X] = \mu \quad , \quad E[(X-\mu)(X-\mu)^T] = \Sigma$$

$\forall\, r,s \in \{1, \dots, p\}$

$$E[(X_r - \mu_r)(X_s - \mu_s)] = \Sigma_{r,s}$$

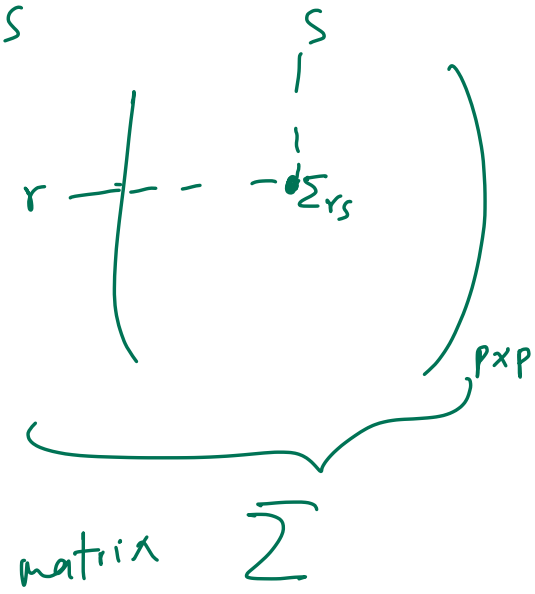
$$X = \begin{pmatrix} X_1 \\ \vdots \\ X_r \\ \vdots \\ X_s \\ \vdots \\ X_p \end{pmatrix}$$

$X_r \rightarrow E[X_r] = \mu_r$

$X_s \rightarrow E[X_s] = \mu_s$

$$E[(X_r - \mu_r)(X_s - \mu_s)]$$

$$= \Sigma_{r,s}$$



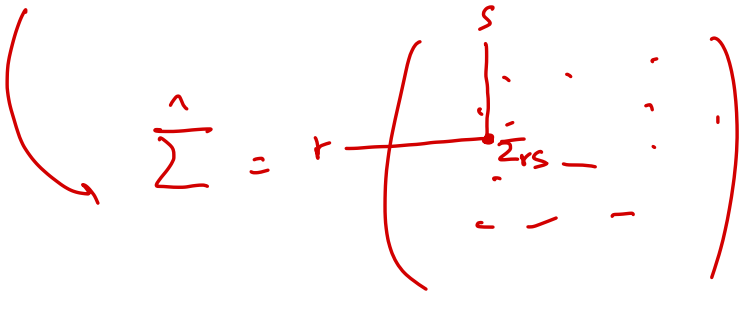
matrix Σ

$$\pi_k = Pr\{y = k\} \xrightarrow{\text{estimate}} \hat{\pi}_k = \frac{\overbrace{\#\{i : y_i = k\}}^{n_k}}{\text{total number of data points}}$$

$$f_k(x) = N(\mu_k, \Sigma)$$

$$\hat{\mu}_k = \frac{\sum_{i : y_i = k} x_i}{n_k}$$

n_k → number of data points with label k



$$\hat{\Sigma}_{rs} = \frac{\sum_{k=1}^{K} \sum_{i: y_i = k} (x_{i,r} - \hat{\mu}_{k,r})(x_{i,s} - \hat{\mu}_{k,s})}{n - K}$$

we estimate $\hat{\mu}_k$ and $\hat{\Sigma}$ and $\hat{\pi}_k$ from data

$\Rightarrow$ an estimate for the conditional probabilities

$$- \quad Pr\{ y = k \mid X = x \} = \frac{Pr\{ X = x \mid y = k \} \, Pr\{ y = k \}}{p(X = x)}$$

estimated $\Rightarrow$

$$= \frac{\hat{f}(x \mid \hat{\mu}_k, \hat{\Sigma}) \; \hat{\pi}_k}{p(X = x)}$$

ep.

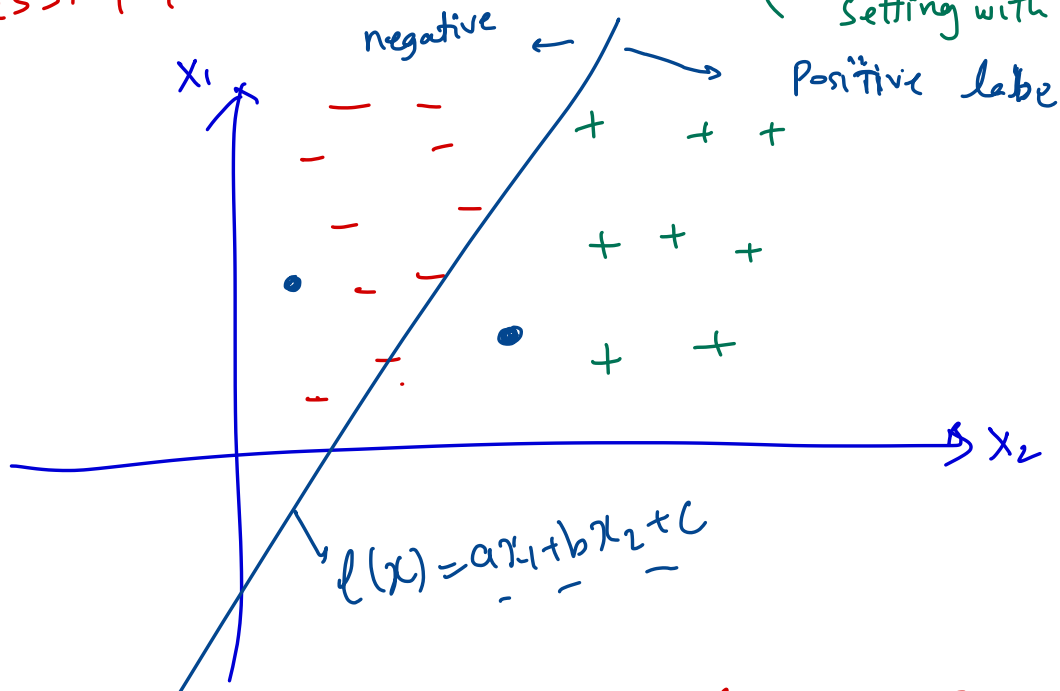
- Prediction of the label at a new data point x:

$$\hat{y}(x) = \underset{k}{\operatorname{argmax}} \left\{ \hat{f}(x \mid \hat{\mu}_k, \hat{\Sigma}) \hat{\pi}_k \right\}$$

$$= \underset{k}{\operatorname{argmax}} \left\{ \underbrace{x^T \hat{\Sigma}^{-1} \hat{\mu}_k - \frac{1}{2} \hat{\mu}_k^T \hat{\Sigma}^{-1} \hat{\mu}_k + \log \hat{\pi}_k}_{\ell_k(x)} \right\}$$

$\ell_k(x) \longrightarrow$ linear in x

This will lead to what we call linear classification boundaries. (let's consider a binary classification setting with 2-d data)



LDA: $\underset{k}{\operatorname{argmax}} \{ \ell_+(x), \ell_-(x) \}$

$$\Downarrow$$

$$\begin{cases} \text{if } \ell_+(x) > \ell_-(x) \Rightarrow \hat{y}(x) = + \\ \text{if } \ell_+(x) \leq \ell_-(x) \Rightarrow \hat{y}(x) = - \end{cases}$$

$$\Downarrow$$

$$\underbrace{\ell_+(x) - \ell_-(x)}_{\triangleq \ell(x)} \gtrless 0$$

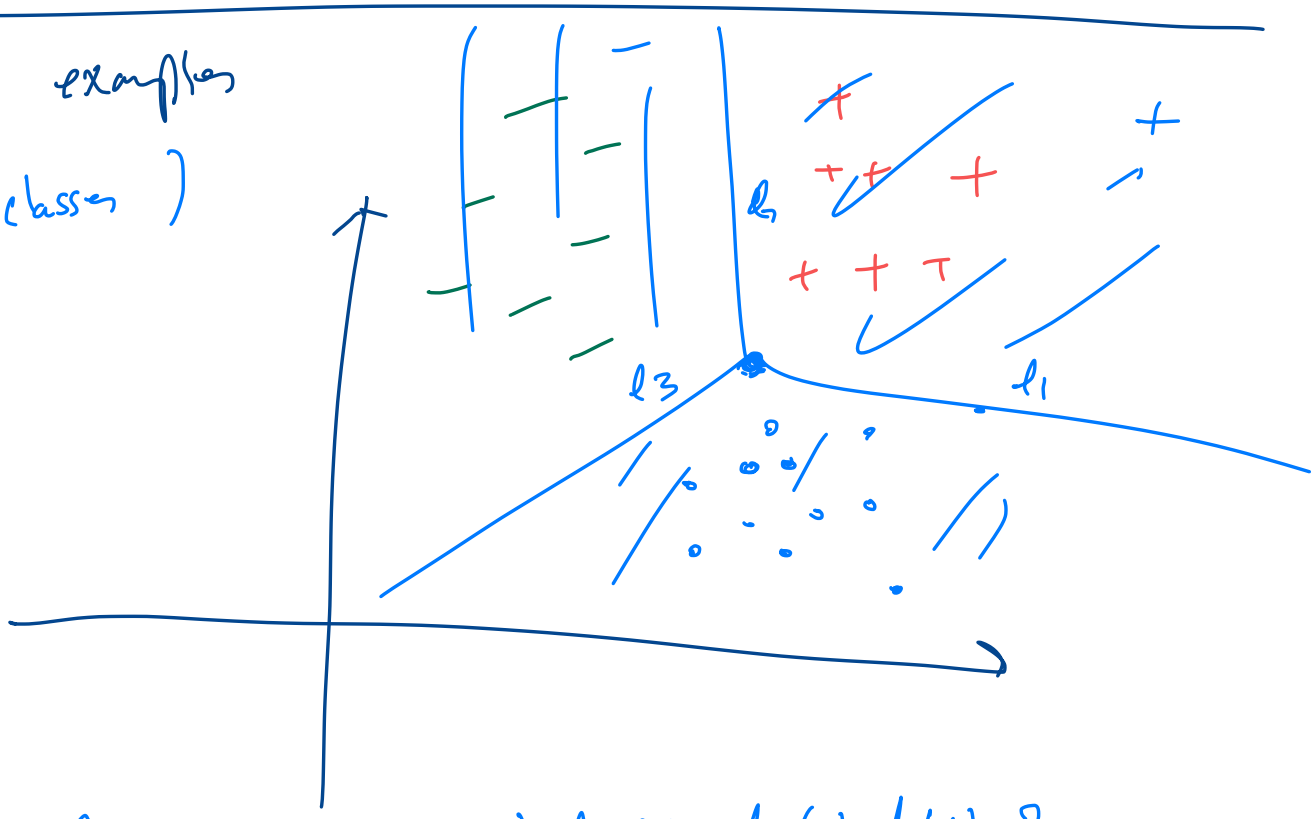
$\hat{y}(x) = +$ if $\ell(x) > 0$

$\hat{y}(x) = -$ if $\ell(x) \leq 0$

$\ell(x) = ax_1 + bx_2 + c$

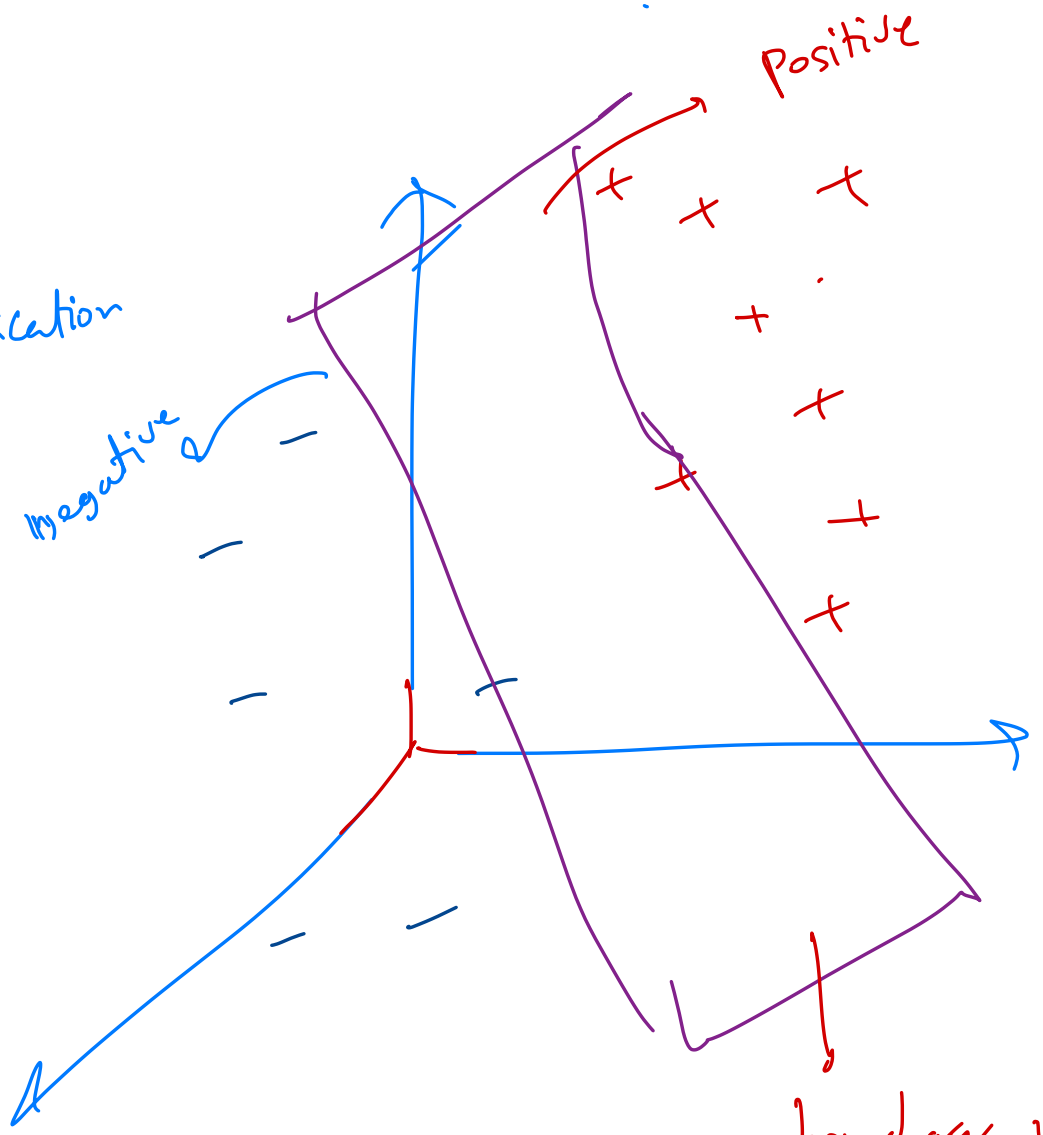
---

other examples (3 classes)



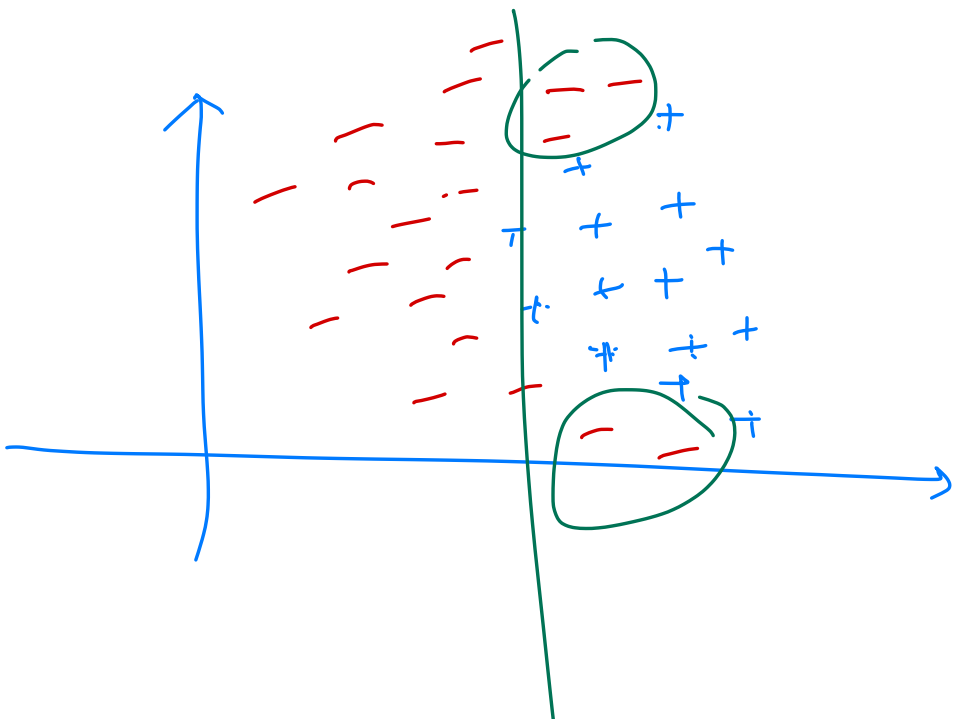
$$\hat{y}(x) = \operatorname{argmax}\{\ell_+(x), \ell_-(x), \ell_0(x)\}$$

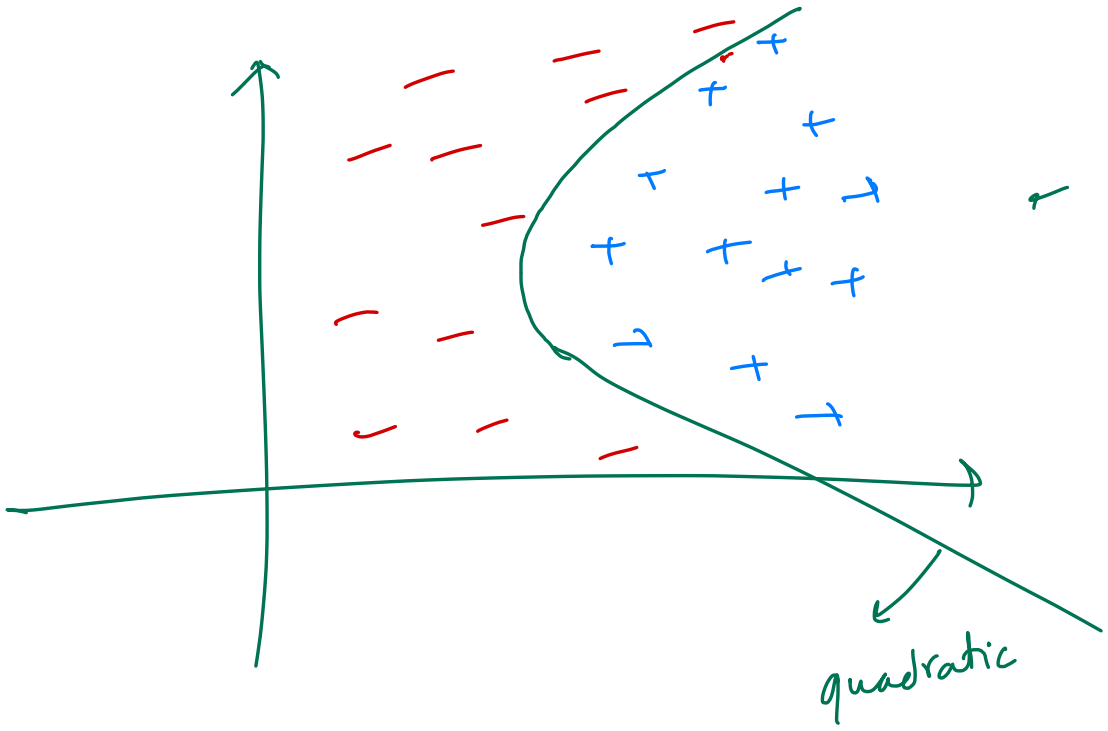
3-dim binary classification



boundary will be a hyperplane

In what cases can LDA be ineffective?





Quadratic Discriminant Analysis:

- Similar to LDA except that the variances can change across the classes.

- for each class k : $f_k(x) = N(\hat{\mu}_k, \hat{\Sigma}_k)$

learned using data from class k.

quadratic term

$$\hat{y}_{QDA}(x) = \underset{k}{\operatorname{argmax}} \left\{ -\frac{1}{2}(x-\hat{\mu}_k)^T \hat{\Sigma}_k^{-1} (x-\hat{\mu}_k) - \frac{1}{2}\log\det(\hat{\Sigma}_k) + \log\hat{\pi}_k \right\}$$

- Why QDA is a more complex class than LDA ?

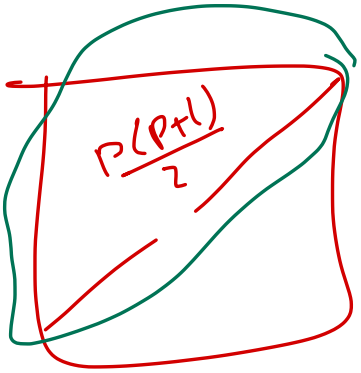
LDA

→ learn the mean vector per class + the covariance matrix.

$K \cdot p + \frac{p(p+1)}{2}$ → covariance matrix

means

→ symmetric matrix

$\Sigma =$ 

QDA

- 1 mean vector per class
- 1 covariance matrix per class

$K \cdot p + K \frac{p(p+1)}{2}$

$Kp + \frac{p(p+1)}{2} \ll Kp + \frac{Kp(p+1)}{2}$

↓

when p is large

LDA

- less parameters
- less flexibility / complexity

might underfit

QDA

might overfit

LDA is a better bet than QDA if there are relatively few training data points. In contrast, QDA is a better choice if the number of training data points is large or if the assumption of a common covariance matrix for all the k classes is off.

(in this case LDA will underfit)

Module 4 : Unsupervised learning:

Supervised learning: $\{(x_i, y_i)\}_{i=1}^{n} \longrightarrow y_i \approx \tilde{f}(x_i)$

unsupervised learning $\{x_i\}_{i=1}^{n}$

Lecture 19:

Module 4: Unsupervised learning:

- There is no label (in contrast with supervised ML)

Data: $x_1, x_2, \dots, x_n$

Goal: Discover informative patterns, structures, subgroups, etc within data.

In this course, we'll consider two specific instances of unsupervised learning:

- clustering
- Dimensionality Reduction

Clustering: Partition data into distinct subgroups such that data points within each group are similar to each other and points from different groups are "dissimilar".

$x_i \in \mathbb{R}^p$

"Similarity" will be quantified by a distance function:

$x \bullet \xleftarrow{\quad d(x,y) \quad} \bullet y$

$x = \begin{pmatrix} x_1 \\ \vdots \\ x_p \end{pmatrix} \in \mathbb{R}^p, \quad y = \begin{pmatrix} y_1 \\ \vdots \\ y_p \end{pmatrix} \in \mathbb{R}^p$

Example:

(1) $d(x,y) = \left(\sum_{i=1}^{p} (x_i - y_i)^2 \right)^{1/2} = \|x-y\|_2$

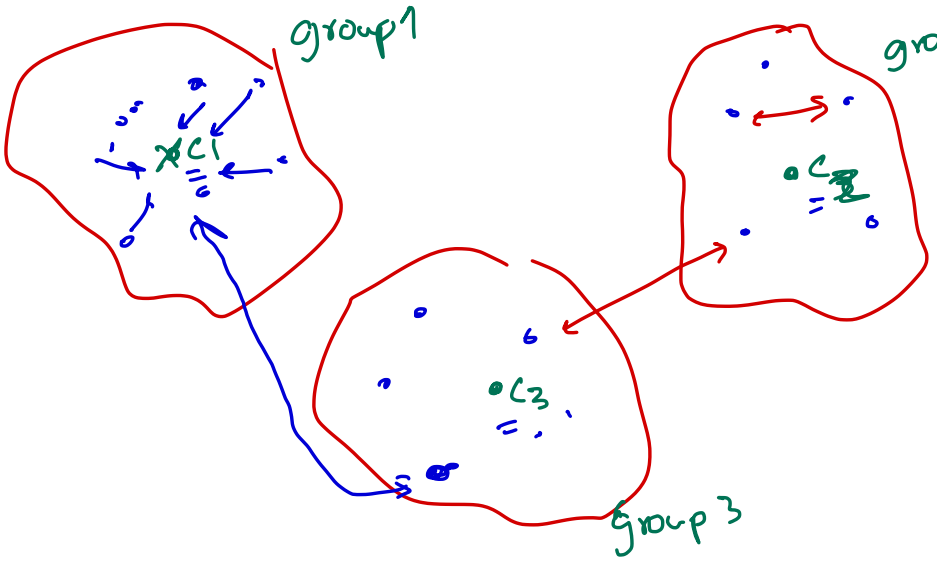
- the Euclidean distance
- the L_2 distance

(2) $d(x,y) = \sum_{i=1}^{p} |x_i - y_i| = \|x-y\|_1$

- L_1 distance
- the Manhattan distance

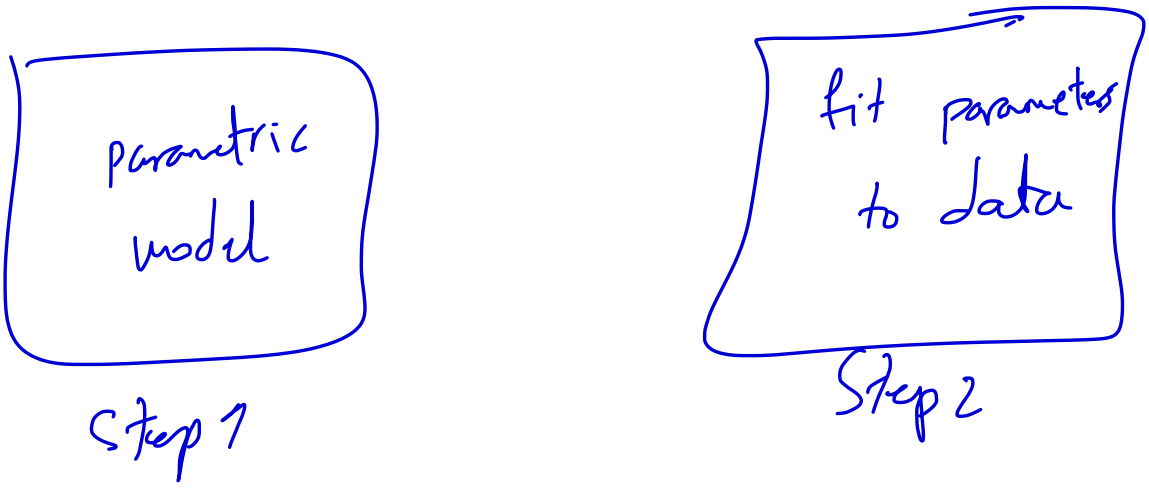
distance ↓ Similarity ↑

Clustering: partition data into K subgroups (K is given) such that points in each group are similar (i.e smaller distance) and points in different groups are dissimilar (relatively larger distance).



$K = 3$

remember our ~~opr~~ general approach:



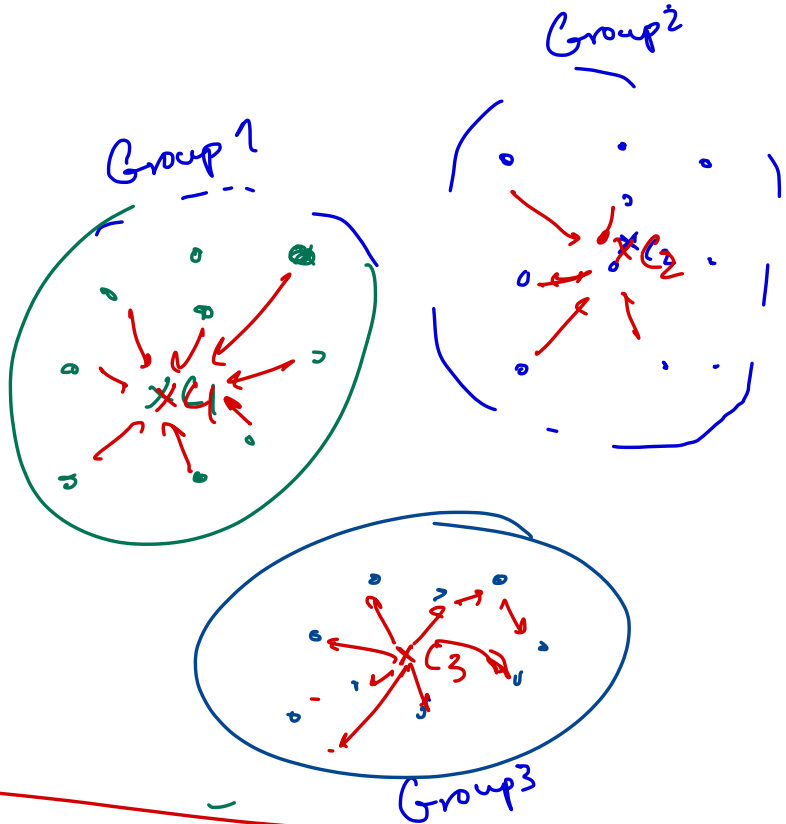
Step 1 (parametric modeling):

How do we represent a group?

- Assign to group i a "center", $c_i \in \mathbb{R}^p$, such that points in group i are close to the center c_i and points outside group i and far from c_i.

$\Rightarrow$ parameters of the clustering problem are $= c_1, c_2, \dots, c_k \in \mathbb{R}^p$

step 2:



$k = 3$

c_1, c_2, c_3

k-means objective $\leftarrow$

$$\min_{c_1, \dots, c_k} \sum_{j=1}^{k} \sum_{i \in \text{Group } j} \| x_i - c_j \|^2 \quad \text{(k-means)}$$

$\| x_i - c_j \|^2$ = internal distance of Group j

k-means clustering: solve (k-means) and find $c_1, \dots, c_k$ along with the groups.

- Difficulty: Finding the exact solution of (k-means) is very difficult (it's an NP-hard problem)

- we'll describe an algorithm that finds an approximate (but sub-optimal) solution for the k-means problem which works well in practice!

The k-means Algorithm:

the k-means algorithm is based on two simple principles:

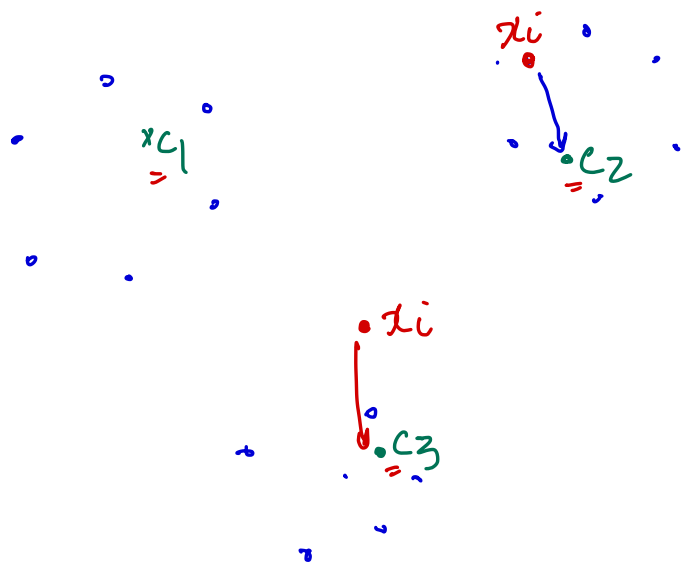
$$\sum_{j=1}^{K} \sum_{x_i \in \text{Group}_j} \|x_i - c_j\|^2$$

$$= \sum_{i=1}^{n} \|x_i - c(x_i)\|^2$$

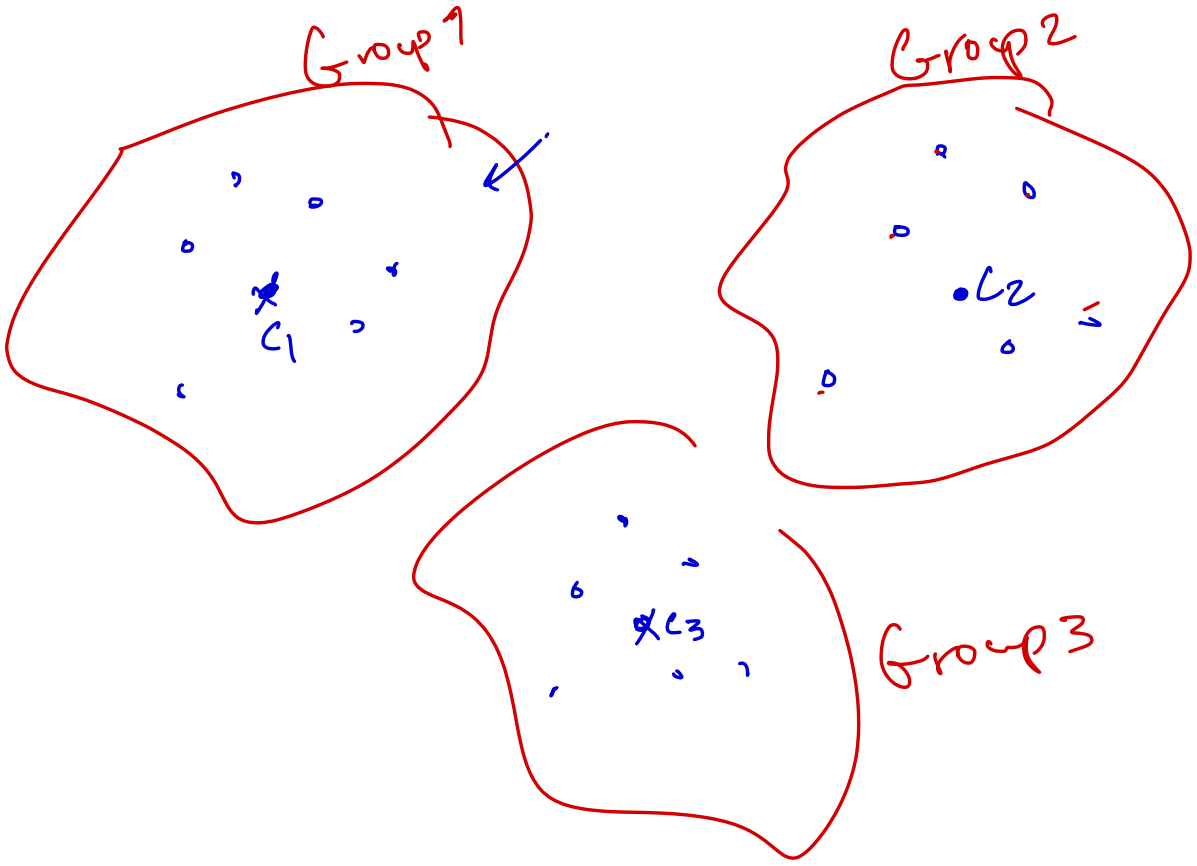
$c(x_i)$ is the center of the group that x_i belongs to.

Principle 1: Given centers $c_1, \dots, c_k$, the center (group) that we assign to data point x_i is the one that's closest to x_i

$$c(x_i) = \underset{c_j \in \{c_1, \dots, c_k\}}{\operatorname{argmin}} \|x_i - c_j\|$$



Principle 2: Let's assume that we fix the groups:



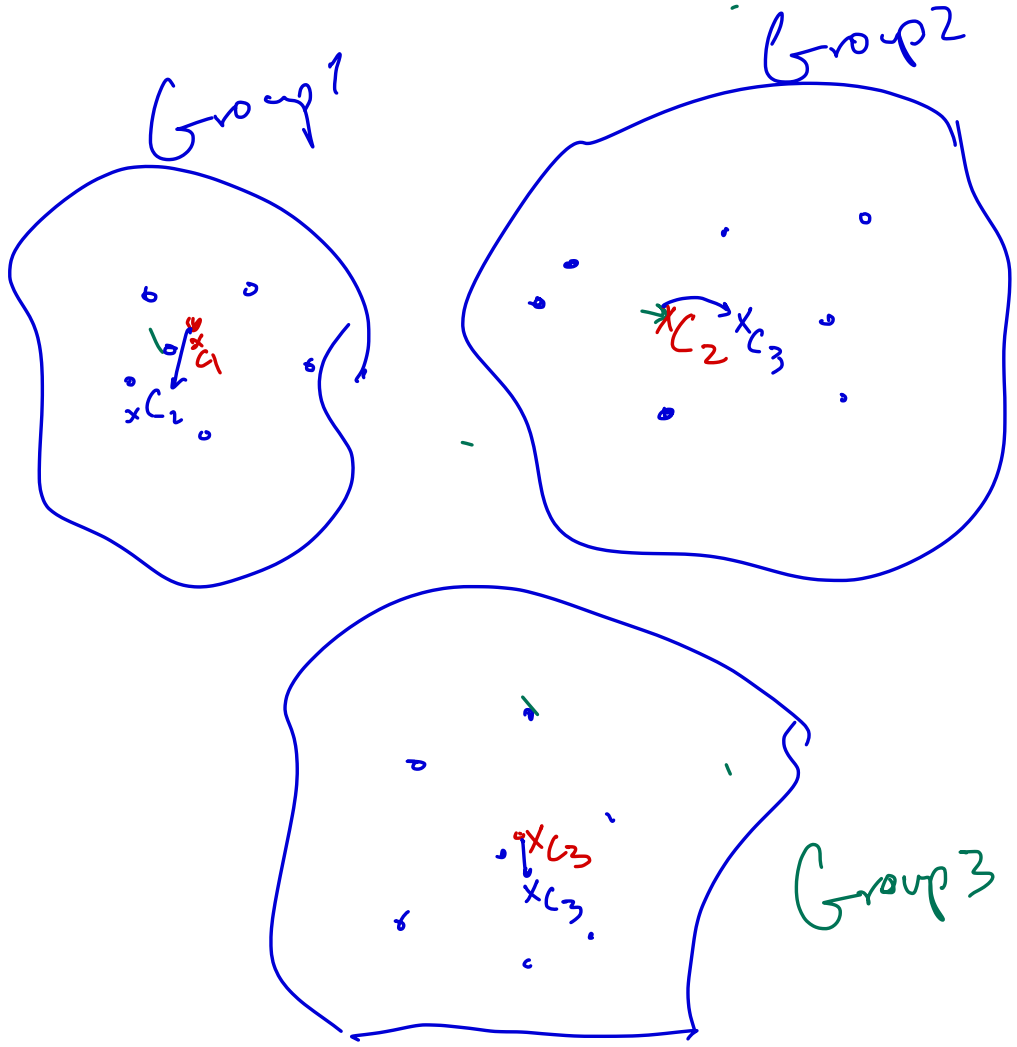
$$c_j = \frac{\sum_{x_i \in \text{Group } j} x_i}{N_j}$$

N_j = # points in group j

Let's try to solve the 1-means problem

$$\min_{c_1 \in \mathbb{R}^p} \sum_{i=1}^{n} \|x_i - c_1\|_2^2$$

$$\rightarrow c_1 = \frac{\sum x_i}{n}$$



k-means algorithm:

- choose $\{c_1^0, c_2^0, \ldots, c_k^0\}$ according to some (arbitrary) choice.

- For $t = 1, 2, 3, \ldots$

  (1) Define Group j to be the set of all the data points that are closest to c_j^{t-1}. → (principle 1)

(2) update $c_j^t = \frac{\sum_{x_i \in \text{Group } j} x_i}{N_j}$ → (principle 2)

(3) if $c_j^t = c_j^{t-1}$ for all $j \in \{1 \to k\}$

we will stop!

Some important (practical) points about the k-means algorithm:

- Let

$L_t \triangleq L(\{c_1^t, \ldots, c_k^t\}) = \sum_{i=1}^{n} \|x_i - c^t(x_i)\|^2$

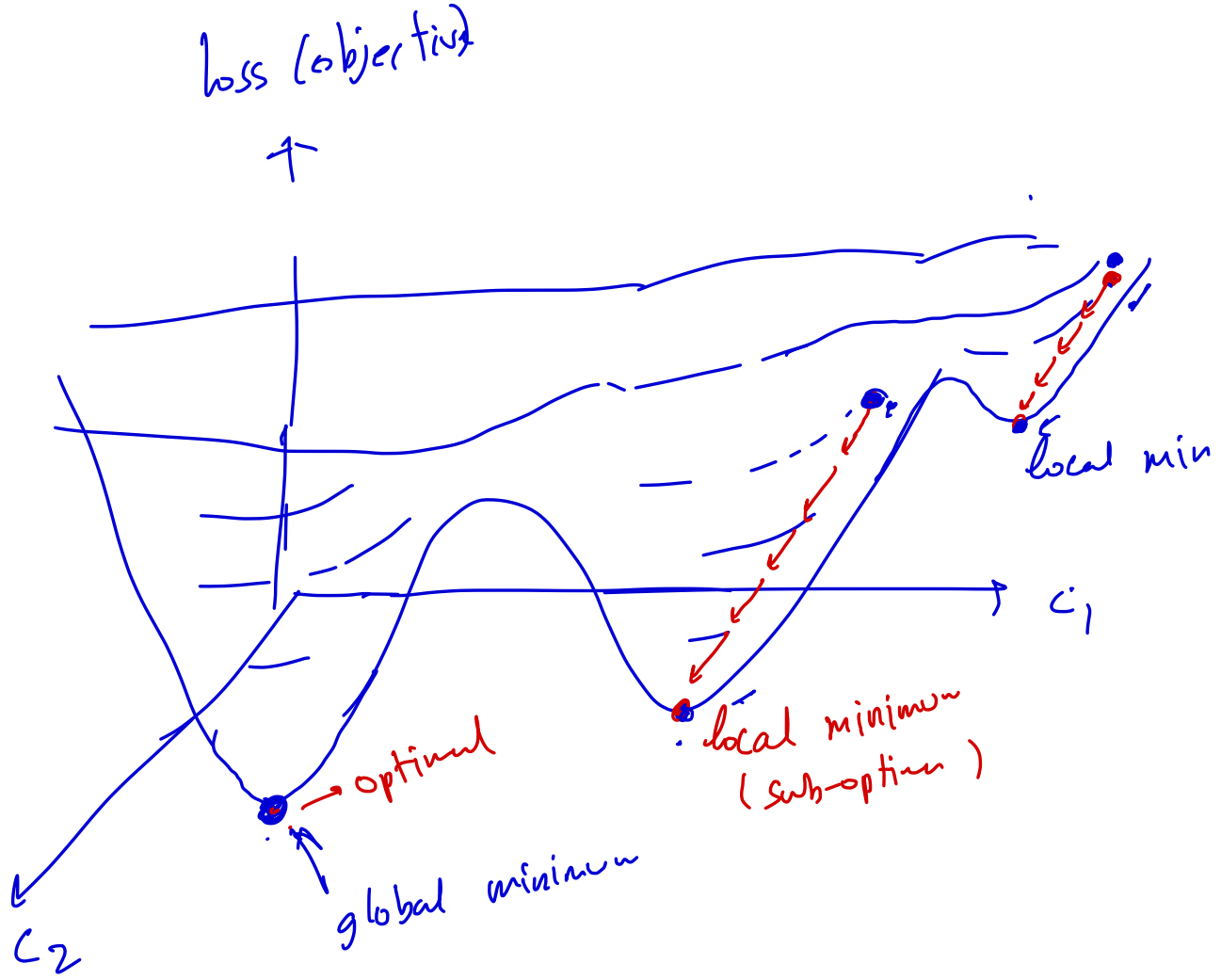
objective computed for $\{c_1^t, \ldots, c_k^t\}$

- then it can be proven that

$$L_t \leq L_{t-1}$$

- Also, it can be proven that

after a "finite" number of iterations, the k-means algorith will stop (converge.)



- The quality of the final set of centers that we obtain from the k-means algorithm depends on the initial centers.
- In practive, we often choose 10-20-50 (random) different initializations and run the k-means algorithm for each initialization, and choose the set of

centers that has the best loss.

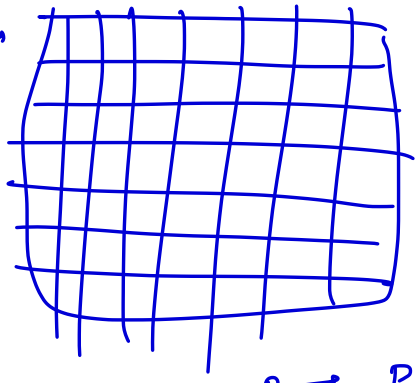
centers that has the best loss.

Lecture 20 :

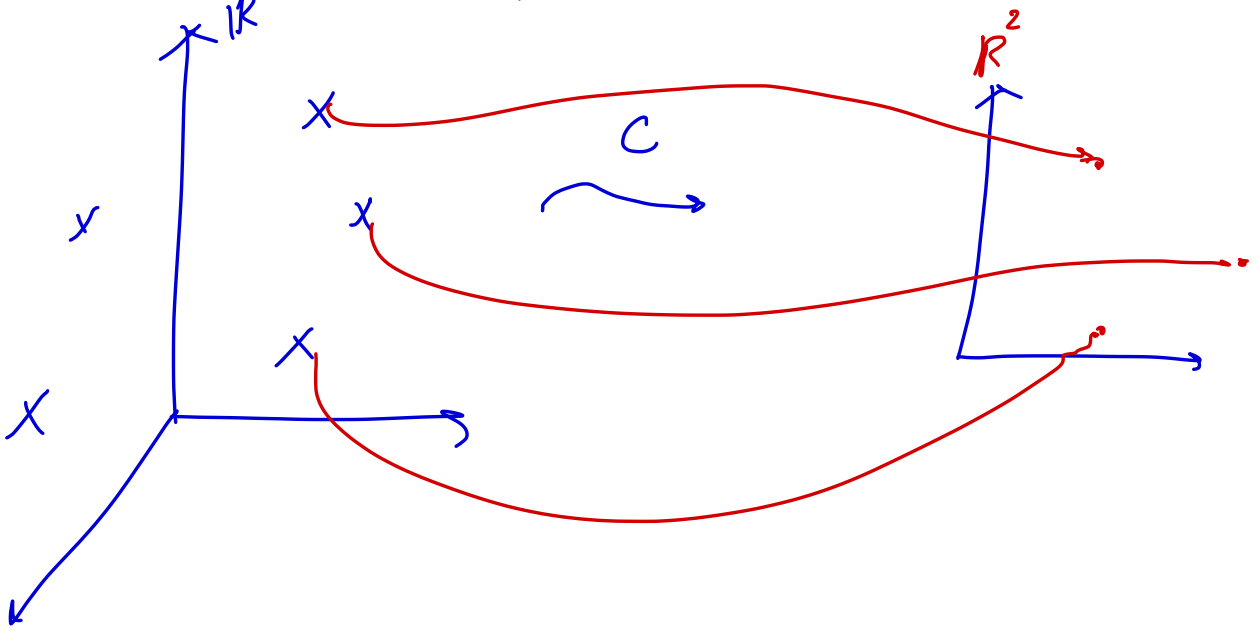
unsupervised learning
- clustering
- dimensionality reduction
  - PCA

$\{x_1, x_2, \dots, x_n\}$

image

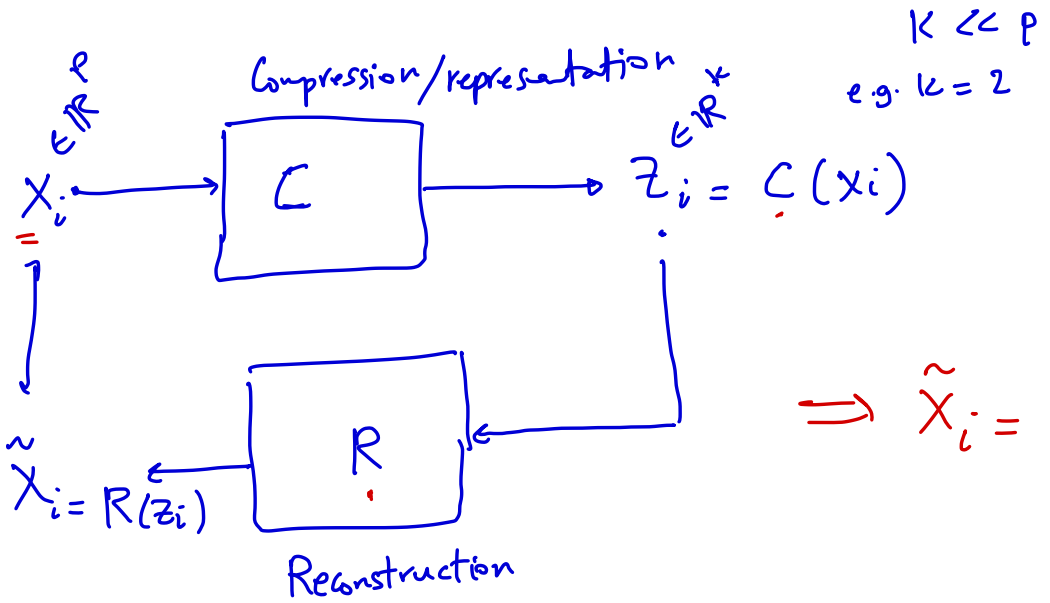


$\mathbb{R}^p \to$ P : # pixels



- $x_1, x_2, \ldots, x_n \in \mathbb{R}^p$ (p is typically large)

- we'd like to represent the data in a low-dimensional space (e.g. $\mathbb{R}^2$)



$$\Rightarrow \tilde{x}_i = R \circ C(x_i)$$

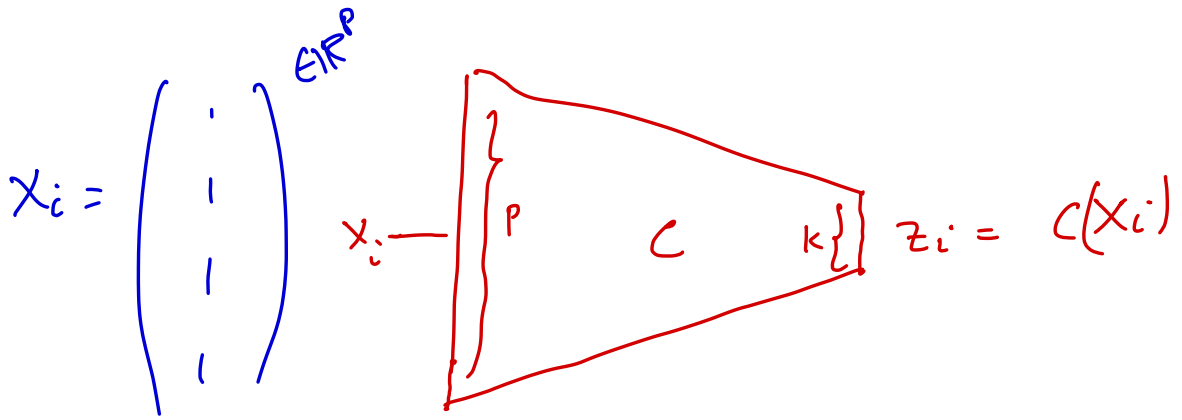
$$R(C(x_i))$$

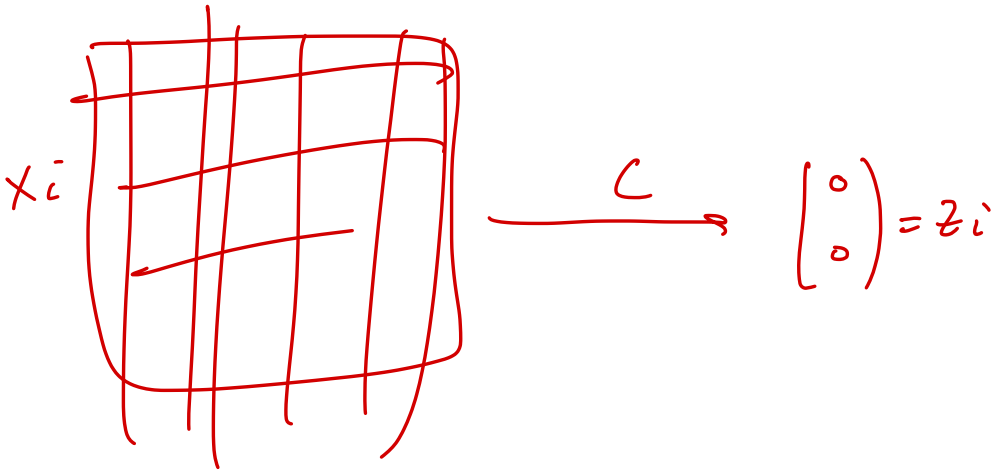
Our goal is to design C, R such that x_i and $\tilde{x}_i$ are as close as possible.

objective:

$$\underset{C,R}{\text{minimize}} \sum_{i=1}^{n} \| x_i - \tilde{x}_i \|_2^2$$

$$= \underset{C,R}{\text{minimize}} \sum_{i=1}^{n} \| x_i - R \circ C(x_i) \|_2^2$$





As usual, let's start with the simplest class of mappings which are linear mappings.

$$z_i = C(x_i) = \underbrace{C}_{\text{matrix}} x_i \qquad C = [\quad]_{k \times}$$

e.g. $k = 2 \longrightarrow C = \left[\begin{array}{c} \text{———} \\ \text{———} \end{array} \right]_{2 \times p}$

$$= \left[\begin{array}{c} U_1^T \\ \hline U_2^T \end{array} \right]$$

$$\Rightarrow \quad z_i = Cx_i = \begin{bmatrix} u_1^T x_i \\ u_2^T x_i \end{bmatrix}$$

Also, R = linear mapping

$$\tilde{x}_i = R z_i = R \cdot C \cdot x_i$$

$R \in \mathbb{R}^{p \times k}$, $C \in \mathbb{R}^{k \times p}$, $[\ ]_{p \times k}$

objective: $\min_{R, C} \sum_{i=1}^{n} \| x_i - R \cdot C \cdot x_i \|_2^2$ (PCA)

$R \in \mathbb{R}^{p \times k}$, $C \in \mathbb{R}^{k \times p}$

This problem is called the Principal Component Analysis (PCA)

$C : \mathbb{R}^p \longrightarrow \mathbb{R}^k$

$C = (\text{matrix})_{k \times p}$

In order to solve this problem, we need to review an important tool in linear algebra:

(this tool is important for many other branches of Data science)

The Singular Value Decomposition:

Theorem: Every symmetric matrix, $A_{p\times p}$, can be written as

$$A_{p\times p} = U^T_{p\times p} \Lambda_{p\times p} U_{p\times p}$$

- where $UU^T = I_{p\times p}$ ← identity matrix

- Λ = diagonal: $\begin{pmatrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ 0 & & \ddots \\ & & & \lambda_n \end{pmatrix}$

- $A = U^T \cdot \begin{pmatrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ 0 & & \ddots \\ & & & \lambda_n \end{pmatrix} U$

- $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \cdots \geq \lambda_n$

- λ_i = eigenvalues of A.

- Each row of U is an "eigenvector" of the matrix A.

Example:

Assume A is 2×2 matrix.

$$A = \begin{pmatrix} a & c \\ c & b \end{pmatrix}$$

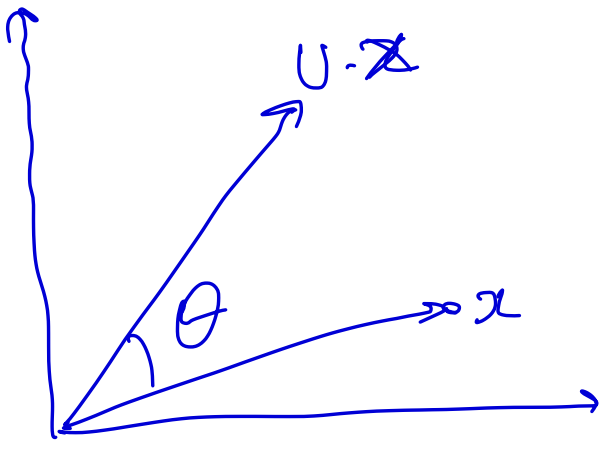
rescaling

$$A = U \cdot \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \cdot U^T$$

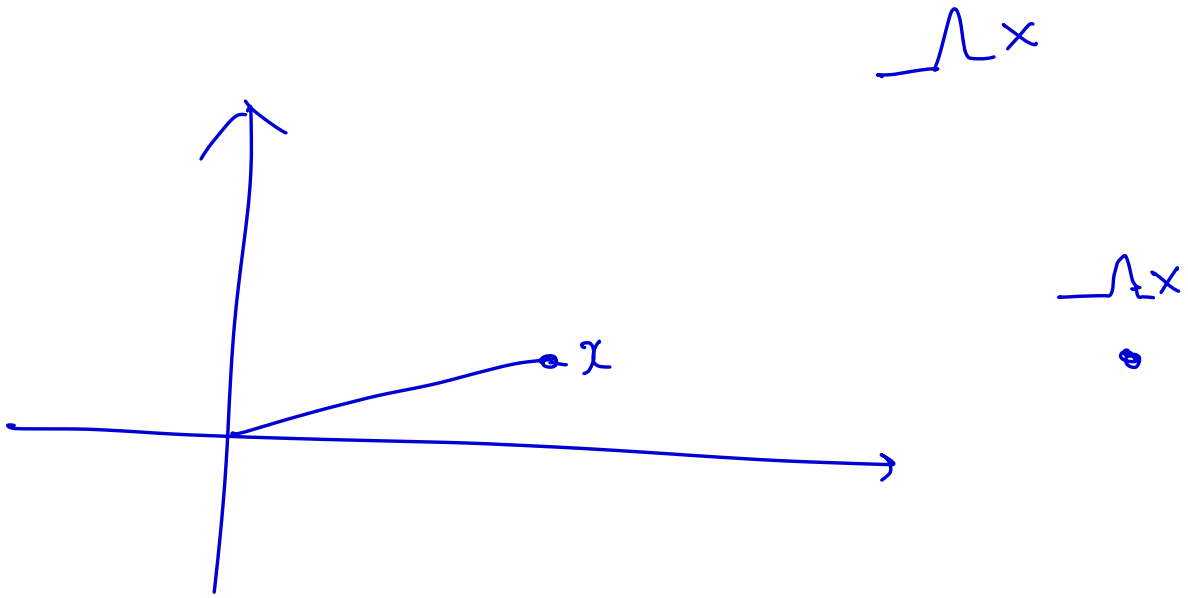
U: Rotation(θ)

U^T: Rotation($-\theta$)

$$U \cdot U^T = I \longrightarrow U = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$



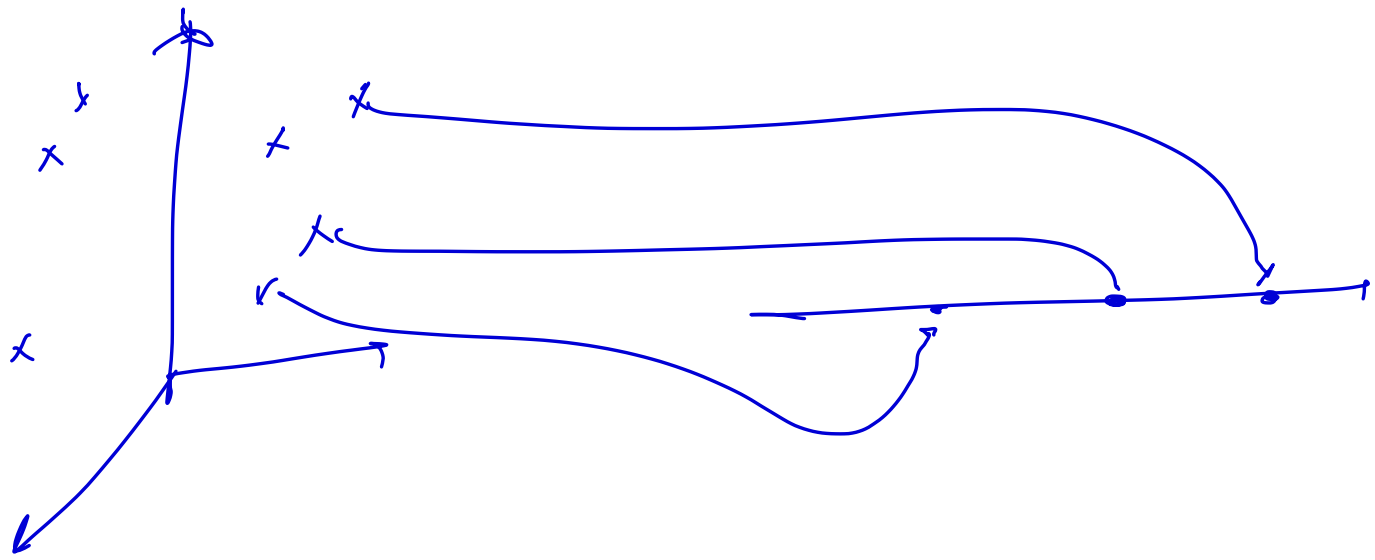
$$\Lambda = \begin{pmatrix} 4 = \lambda_1 \geq 0 & 0 \\ 0 & 1 = \lambda_2 \geq 0 \end{pmatrix}$$ ← Rescaling



$$A = \text{Rotation}(-\theta) \cdot \text{Rescaling} \cdot \text{Rotation}(\theta)$$

Let's go back to our problem.

Let's assume for simplicity that $k=1$; we're looking for the best 1-dimensional representation of the data.



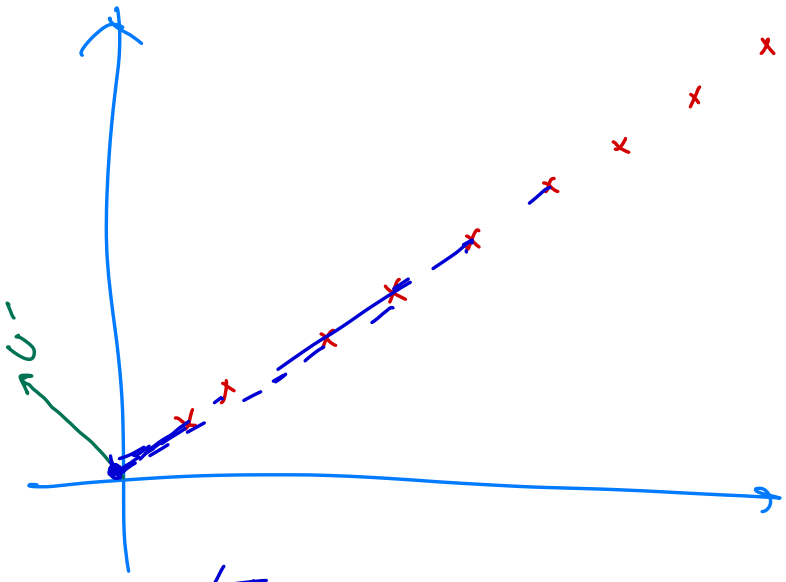
$$C = [\quad\quad]_{k \times p} \longrightarrow C = U^T$$

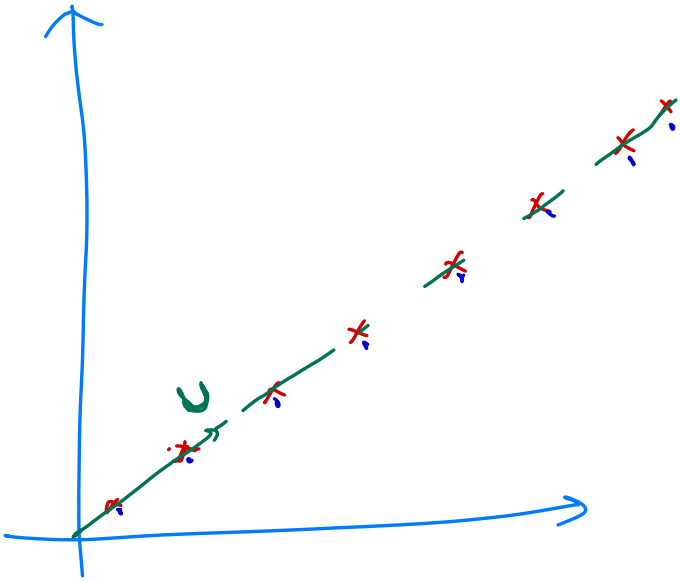
$$C \cdot X_i = \underset{\downarrow}{U^T} X_i$$

vector

When $k=1$, our goal is to find a single vector U^T such that the representation $z_i = U^T X_i$ is "the most informative" representation.

Example:



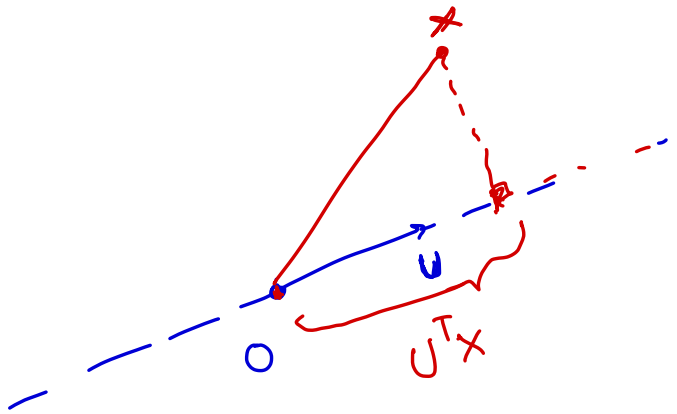


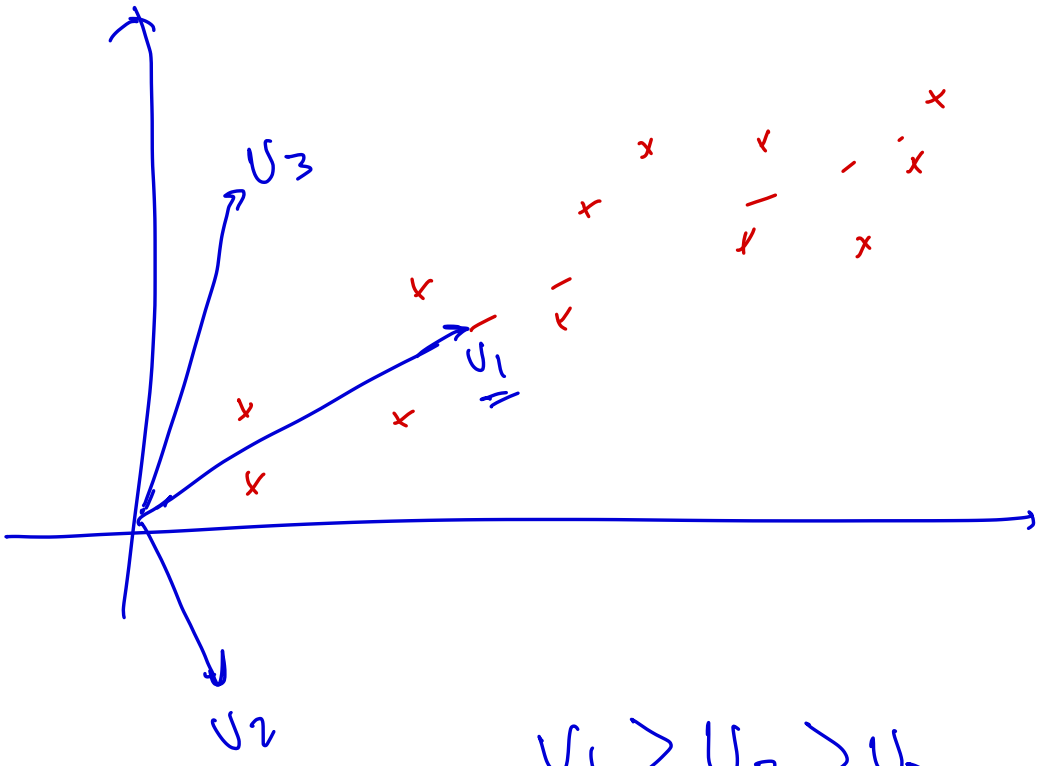
$U'^{T} X_i \approx 0$

$U^{T} X_i$

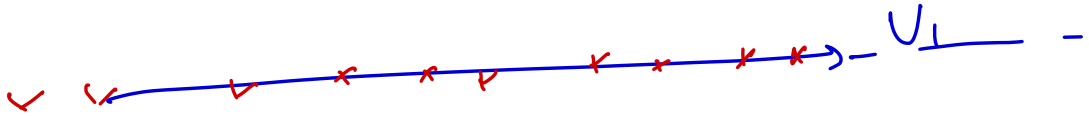
U is a better direction than U' in terms of capture more information about the data

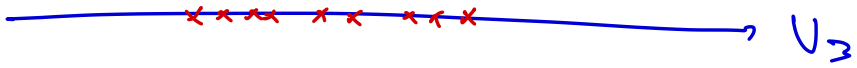
Assume $\|U\|_2 = 1$





$$u_1 > u_3 > u_2$$





We're looking for a direction u along which data has the most "variation".

- Mathematically, we're looking for a direction u such the "variance" of $u^T \cdot X$ is the most.

- Assume that the data Cons from a a distribution $X \sim P(X)$. Also assume for simplicity that $E[X] = 0$.

$$\max_{\substack{u^T \in \mathbb{R}^{1\times p} \\ \|u\|_2 = 1}} \mathrm{Var}(u^T \cdot X)$$

$$= \max_{u : \|u\|_2 = 1} E\left[(u^T X)^2 \right]$$

$$= \max_{u : \|u\|_2 = 1} E\left[u^T X^T X u \right]$$

$$= \max_{u : \|u\|_2 = 1} u^T \underbrace{E[X X^T]}_{\text{covariance matrix of the data}} u$$

$\left(X \in \mathbb{R}^p,\ E[XX^T] = \Sigma_{p\times p} \right)$

Let $\Sigma \triangleq E[XX^T]$

$$= \max_{u:\ \|u\|_2=1} u^T \Sigma u$$

The solution of the problem above can be found from the singular value decomposition of Σ.

$$\Lambda = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & \lambda_p \end{pmatrix}$$

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p$$

$$\text{SVD} \longrightarrow \Sigma = U \Lambda U^T$$

It can be shown that the first row of the matrix U, is the maximizer of the following problem:

$$\max_{u:\|u\|=} u^T \Sigma u$$

$$U = \begin{pmatrix} u_1^T \\ u_2^T \\ \vdots \\ u_p^T \end{pmatrix}_{p \times p}$$

- u_1^T is the solution to the problem above.
- u_1^T is the principal component of the data

- v_1^T is the vector along which the data has the most "variation".

$$\Sigma = E[XX^T]$$

- We know that the data

$$X_1, X_2, \dots, X_n \overset{\text{iid}}{\sim} \text{distribution } X \sim P(X)$$

in practice:

- We cannot compute the matrix Σ because we do not know the distribution of the data

$$\Sigma \approx \underbrace{\frac{1}{n} \sum_{i=1}^{n} X_i X_i^T}_{\text{unbiased estimator}}$$

Algorithm (PCA): ($k=1$)

- Input: $X_1, X_2, \dots, X_n \in \mathbb{R}^p$

- Compute the empirical covariance matrix of data: $\hat{\Sigma} = \frac{1}{n} \sum_{i=1}^{n} X_i X_i^T$

- Compute the Singular value decomposition of $\hat{\Sigma}$:

$$\hat{\Sigma} = U^T \Lambda U$$

- use the first row of U, u_1, as the principal component of the data:

$$z_i = u_1^T x_i .$$

---

* let's now find the best 2-dimensional representation of the data.

$$\rightarrow \max_{u_1, u_2} \; u_1^T \Sigma u_1 + u_2^T \Sigma u_2$$

$$\|u_1\|, \|u_2\| = 1$$

(orthogonal) $u_1^T u_2 = 0$

It can be shown that u_1, u_2 are the first two rows of the matrix U.

* In general if we want to find a matrix $C = \begin{bmatrix} u_1^T \\ u_2^T \\ \vdots \\ u_k^T \end{bmatrix}_{k \times n}$ then

- we need to solve the following optimization problem:

$$\sum_{i=1}^{k} u_i^T \Sigma u_i$$

s.t. $\|u_i\| = 1$

and $u_i^T u_j = 0$ for $i \neq j$

Cumulative variance of the data along the directions $u_1, \dots, u_k$

* it can be shown that the solution to the above optimization problem is the first k rows of the matrix U.

Algorithm PCA for general k:

- Input: $x_1, \dots, x_n \in \mathbb{R}^p$

- Compute: $\hat{\Sigma} = \frac{1}{n} \sum_{i=1}^{n} x_i x_i^T$

- Compute: (SVD) $\hat{\Sigma} = U^T \Lambda U$

- $C = \begin{bmatrix} u_1^T \\ u_2^T \\ \vdots \\ u_k^T \end{bmatrix}$ where $u_1, \dots, u_k$ are the first k rows of the matrix U.

It can be proven that PCA gives the optimal linear representation of the data (it solves equation (PCA) exactly).

Lecture 21 :

Module 5: An Introduction to statistical Learning Theory:

We consider the problem of supervised learning in this module, but all the concept generalize to other forms of learng (e.g. unsupervised learning).

A formal framework for learning theory:

(1) Data is assumed to be generated iid according to a distribution $(x, y) \sim D(x,y)$. The input domain is denoted by X ($x \in X$, e.g. $X = \mathbb{R}^p$), the label domain is denoted by Y (i.e. $y \in Y$, e.g. $Y \in \{0, 1\}$), and

input / features x ; label y

D is a distribution over $X \times Y$.

2: The learner output: the learner's goal is to find a predictive relation $h: X \longrightarrow Y$ which has low error/ high accuracy. Formally speaking, for any function $h: X \longrightarrow Y$ we define its error as:

error / loss — h: a function from X to Y

$$L_D(h) = \Pr_{(x,y)\sim D}\{ h(x) \neq y \}$$

D = distribution of the data

$$= E_{(x,y)\sim D}[\mathbb{1}\{h(x) \neq y\}]$$

$$\mathbb{1}\{A\} = \begin{cases} 1 & \text{if } A \text{ is true} \\ 0 & \text{if } A \text{ is false} \end{cases}$$

Ideally, we would like to find a predictor h that has the smallest error; i.e.

our "gold standard" is to solve the following problem:

$$\text{Minimize}_{h} \quad \underbrace{L_D(h)}_{P_{(x,y)\sim D}\{h(x)\neq y\}} \qquad (1)$$

$$= E_{(x,y)\sim D}[\mathbb{1}\{h(x)\neq y\}]$$

However, this task is impossible since D is unknown.

(3) The learner's input: Training Data!

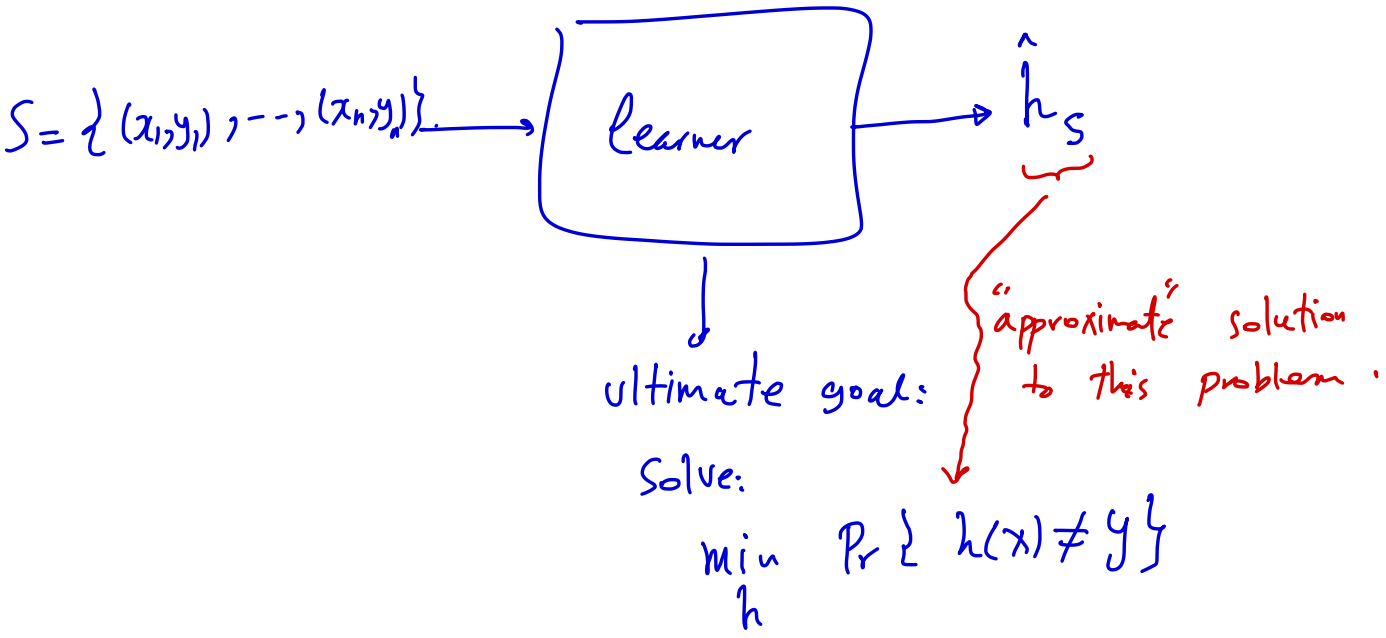
The only information that the learner has about the data distribution is a set of training samples:

$$S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

Where each (x_i, y_i) is generated i.i.d.

from the distribution D. Hence, the learner has has to find "approximate" solutions to problem (1) using the training data.

Data distribution D



$S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ $\quad \min_h \underbrace{E_{(x,y)\sim D}[\mathbb{1}\{h(x) \neq y\}]}$

↳ estimate: $\frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{h(x_i) \neq y_i\}$

(4) Empirical Risk Minimization (ERM):

true error $\longrightarrow L_D(h) = E_{(x,y)\sim D}\left[\mathbb{1}\{h(x) \neq y\}\right]$

unbiased estimate

training error $\longrightarrow L_S(h) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{h(x_i) \neq y_i\}$

ERM is the task of finding a predictor that minimizes the training error:

$$\min_h L_S(h)$$

---

instead of solving $\rightarrow \min_h L_D(h)$

ERM solves $\rightarrow \min_h L_S(h)$

claim: $\min_h L_S(h) = 0$

overfitting

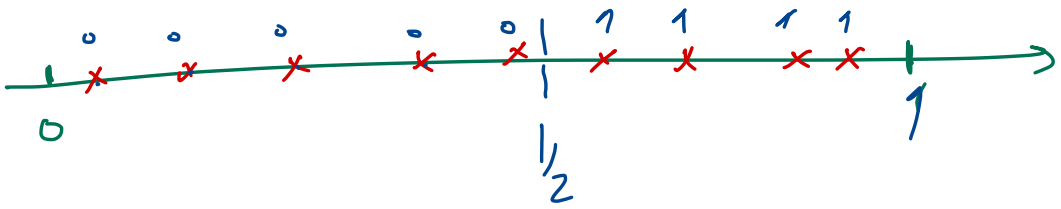
Example: Consider the following distribution on data:

$$X = [0,1]$$

$$Y = \{0,1\}$$

$\longrightarrow$ $x \sim$ uniform $[0,1]$

$$y = \begin{cases} 0 & \text{if } x \le 1/2 \\ 1 & \text{if } x > 1/2 \end{cases}$$



$$\min_h L_D(h) = \min_h \Pr\{h(x) \neq y\}$$

$$\min_h L_S(h) =$$

$$\frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{h(x_i) \neq y_i\}$$

$$\hat{h}_S = \begin{cases} y_i & \text{if } x = x_i \\ 0 & x \neq \{x_1, \ldots, x_n\} \end{cases}$$

Proof of the claim:

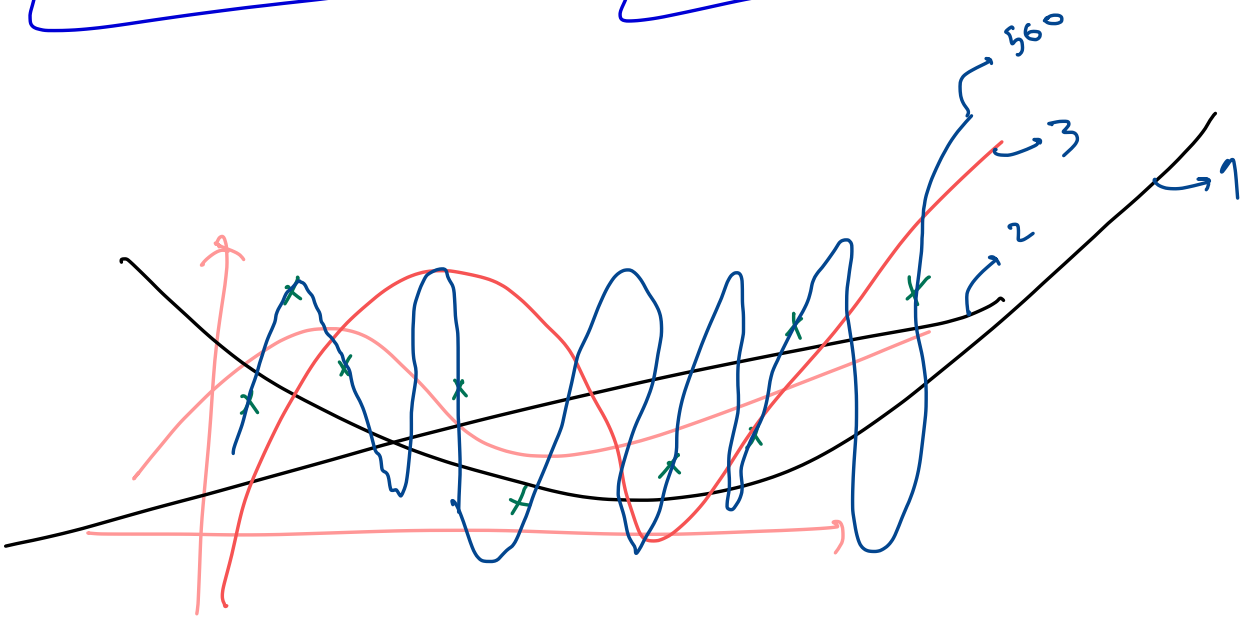
Define $\hat{h}_S(x) = \begin{cases} y_i & \text{if } x = x_i \\ 0 & \text{if } x \notin \{x_1, \dots, x_n\} \end{cases}$

$$L_S(\hat{h}_S) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{\hat{h}_S(x_i) \neq y_i\}$$

$$= 0$$

$\min_h L_D(h)$ → $\min_h L_S(h)$ → not good overfitting

Recall (Regression)

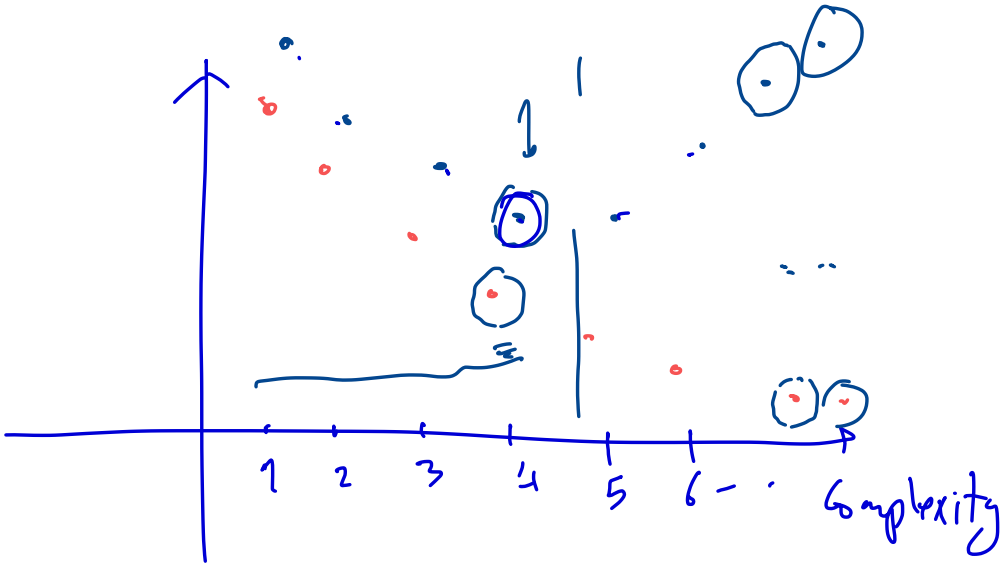


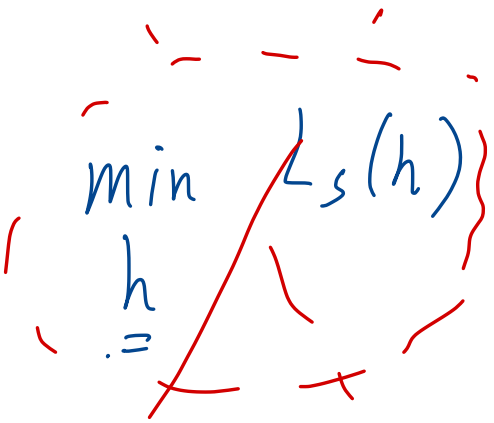
(5) Although ERM is natural, it can miserably if we are not careful: It can overfit easily: the minimum of

$$\min_h L_S(h)$$

is always zero (we always overfit).

(6) We need to search for conditions under which there is a guarantee that ERM does <u>not</u> overfit; namely, conditions under which when ERM has good performance on training data, it also has good performance over the underlying distribution.





$$\min_h L_S(h) \longrightarrow \min_{h \in \mathcal{H}} L_S(h)$$

restrict the class of functions/predictors (restrict the complexity)

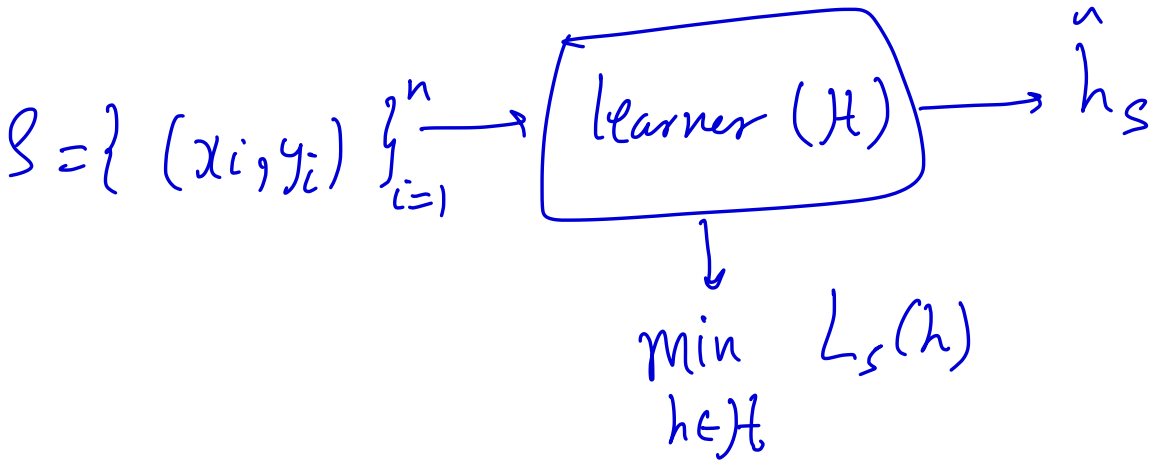
$\Downarrow$

prevents overfitting

The above problem is denoted by

$ERM_{\mathcal{H}}$

$\mathcal{H}$ is the specific function class that we use to fit the data.


$S = \{(x_i, y_i)\}_{i=1}^{n}$
learner ($\mathcal{H}$)
$\hat{h}_S$
$\min_{h \in \mathcal{H}} L_S(h)$

Module 5: An Introduction to the Statistical Learning theory

we will consider the supervised learning problem, but the framework can also be extended to unsupervised learning.

A formal framework for learning theory:

(1) Data is assumed to be generated according to a distribution $(x, y) \sim D(x, y)$. (x: vector of features; y: label) The input domain is denoted by X ($x \in X$), the label domain is denoted Y (i.e $y \in Y$), and D is a distribution over $X \times Y$.

(2) The learner's output: We are looking for predictive relations $h: X \longrightarrow Y$ with low prediction error. Formally speaking, for each function $h: X \longrightarrow Y$ we define its error as:

prediction error » generalization error » true error

$$L_D(h) = \Pr_{(x,y)\sim D}\{h(x) \neq y\}$$

D: distribution of data

$$= \mathbb{E}_{(x,y)\sim D}\left[\mathbb{1}\{h(x)\neq y\}\right]$$

Ideally, we'd like to find a predictor h that has the

Smallest true error, i.e our gold standard is to solve

$$\text{Minimize}_{h} \; \mathcal{L}_D(h) \qquad (1)$$

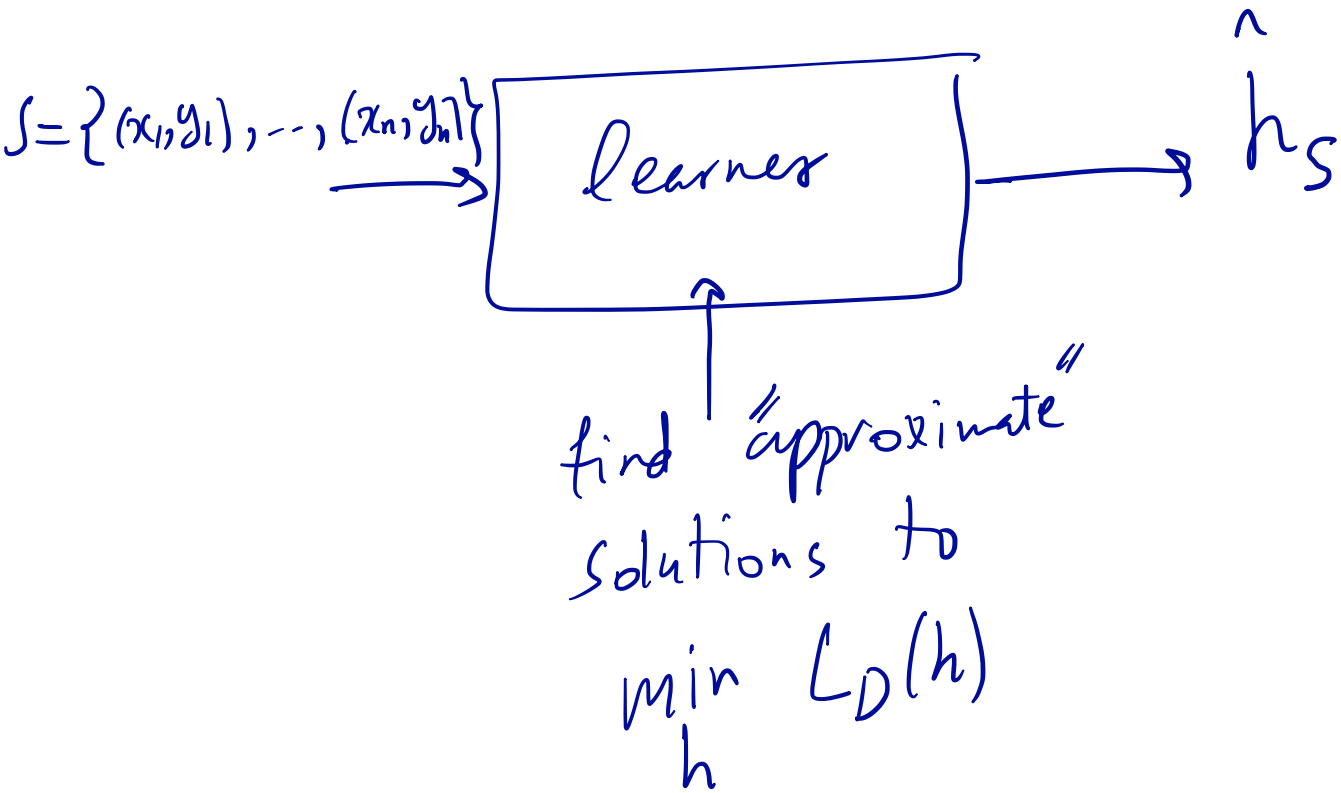
$\hookrightarrow \mathcal{L}_D(h) = E_{(x,y)\sim D}[\#\{h(x) \neq y\}]$

However, this task is impossible since D is unknown.

(3) the learner's input: Training data! The only information that the learner has about the data distribution is a set of training samples (data points), $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$

where each (x_i, y_i) is

Generated i.i.d. according to the distribution D. Hence, the learner has to find "approximate" solutions to problem (1) using the training data set.



$(x_i, y_i) \overset{iid}{\sim} D$

4) Empirical Risk Minimization (ERM):

$L_D(h)$ ⟶ find an unbiased estimator

↑ Can't compute exactly

$$L_D(h) = \mathop{\mathbb{E}}_{(x,y)\sim D}\left[\mathbb{1}\{h(x) \neq y\}\right]$$

unbiased estimate

$$L_S(h) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{h(x_i) \neq y_i\}$$

Note: $\mathbb{E}_S[L_S(h)] = L_D(h)$

Empirical risk minimization (ERM) is the task of finding a predictor that minimizes the training error:

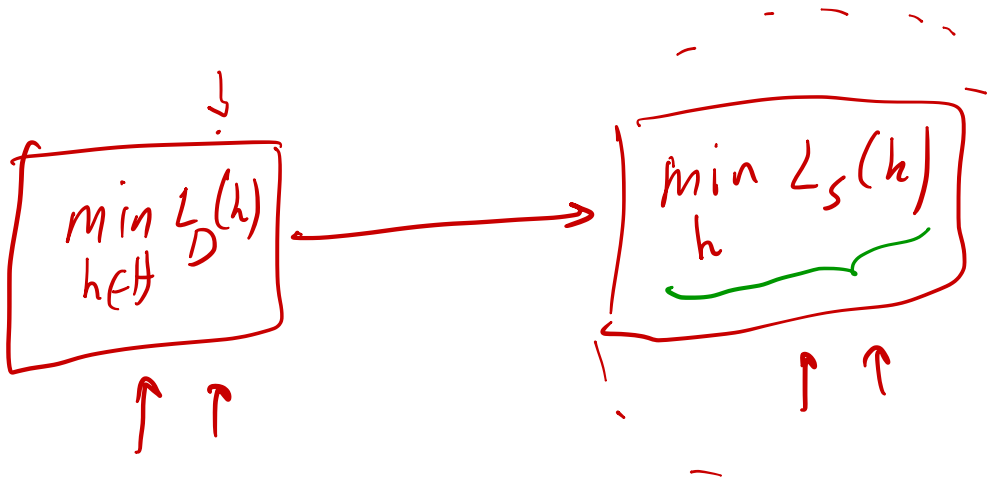
$$\min_h L_S(h) \qquad (2)$$

$\min_h L_D(h)$ $\xrightarrow{\text{instead}}$ $\min_h L_S(h)$

($\min_h L_D(h)$: we can't do)

$$L_D(h) \xrightarrow{\text{instead}} L_S(h) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{h(x_i) \neq y_i\}$$

Lecture 23:



Claim:

$$\min_h L_S(h) = 0$$

$$L_S(h) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{h(x_i) \neq y_i\}$$

Define $\tilde{h} : X \longrightarrow Y$ as follows:

$$\tilde{h}(x) : \begin{cases} y_i & \text{if } x = x_i \\ 0 & \text{if } x \in \{x_1, \ldots, x_n\} \end{cases}$$

It's easy to see that

$$L_S(\tilde{h}) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{\tilde{h}(x_i) \neq y_i\}$$

$$= 0$$

$L_S(\tilde{h}) = 0 \longrightarrow$

$$\min_h L_S(h) = 0$$

5) Although ERM is natural, it can fail miserably if we are not careful: It can "overfit" easily.

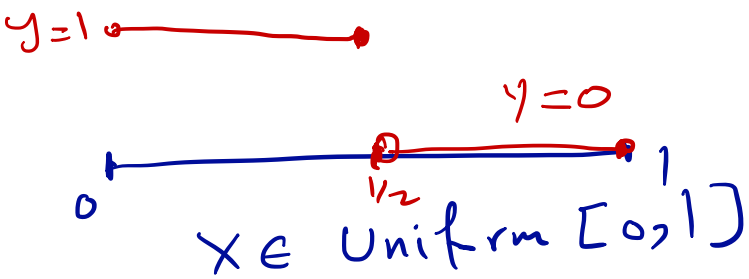
The minimizer of

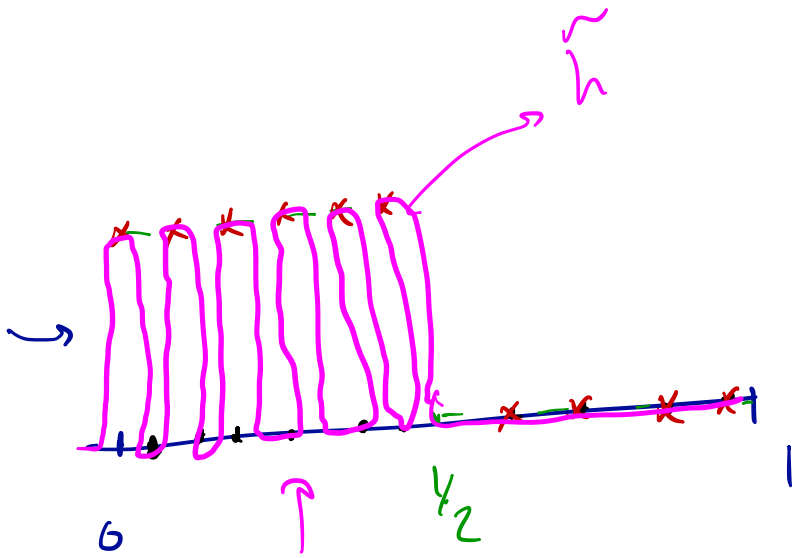
$$\min_h L_S(h) \qquad (2)$$

is always zero.

Example:

Distribution of data:





prediction on unseen data here is very bad

$$\min_h L_S(h)$$

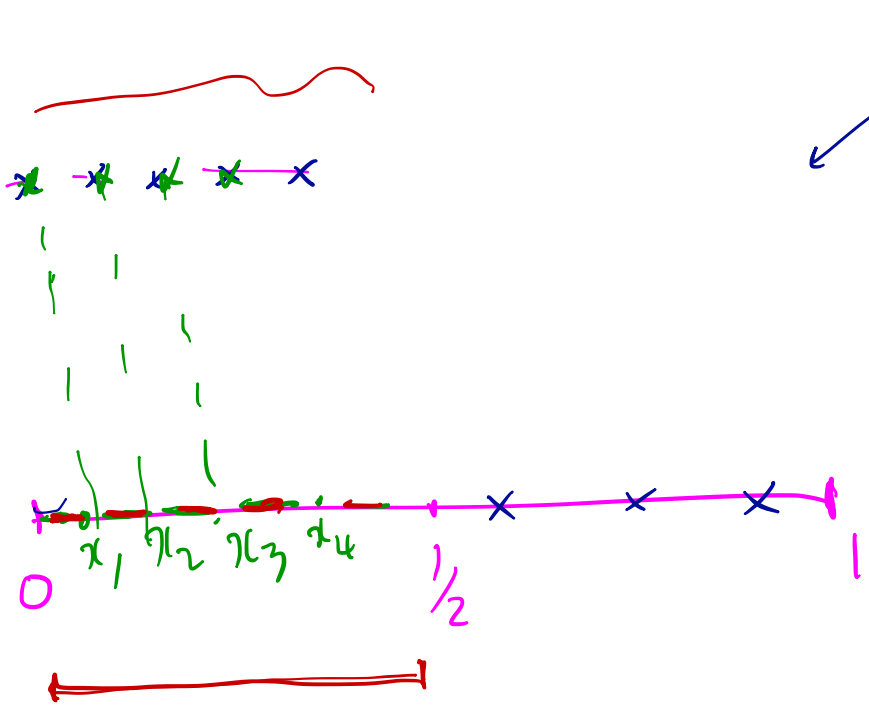
h

$$\left\{ \begin{array}{l} L_S(\tilde{h}) = 0 \\ L_D(\tilde{h}) \cong 0.5 \end{array} \right.$$

$$\min_h L_D(h)$$

$$\min_h L_S(h) \to 0$$

$h \uparrow \tilde{h}$

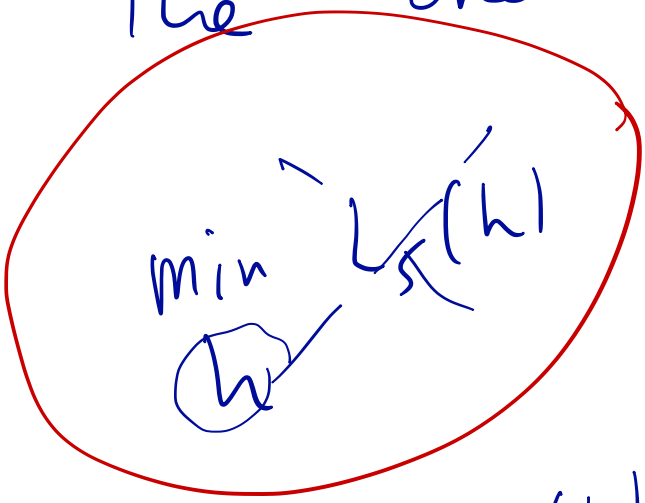


$$\tilde{h} = \begin{cases} h(x_i) = y_i \\ h(x) = 0 \end{cases} \qquad x \in \{x_1, \dots x_n\}$$

$$\begin{cases} L_S(\tilde{h}) = 0 \\ L_{(P,f)}(\tilde{h}) = \underset{(x,y)\sim D}{\mathbb{P}}[h(x) \neq y] \end{cases}$$

$$= 1/2$$

6) We need to search for conditions under which there is a guarantee that ERM does not overfit, namely conditions under which when the ERM predictor has a good performance with respect to the training data, it is also likely to perform well over the underlying (true) distribution



$\min_{h} L_S(h)$ → $\min_{h \in \mathcal{H}} L_S(h)$

$\min_{h} L_D(h)$

restrict the class of functions (complexity)

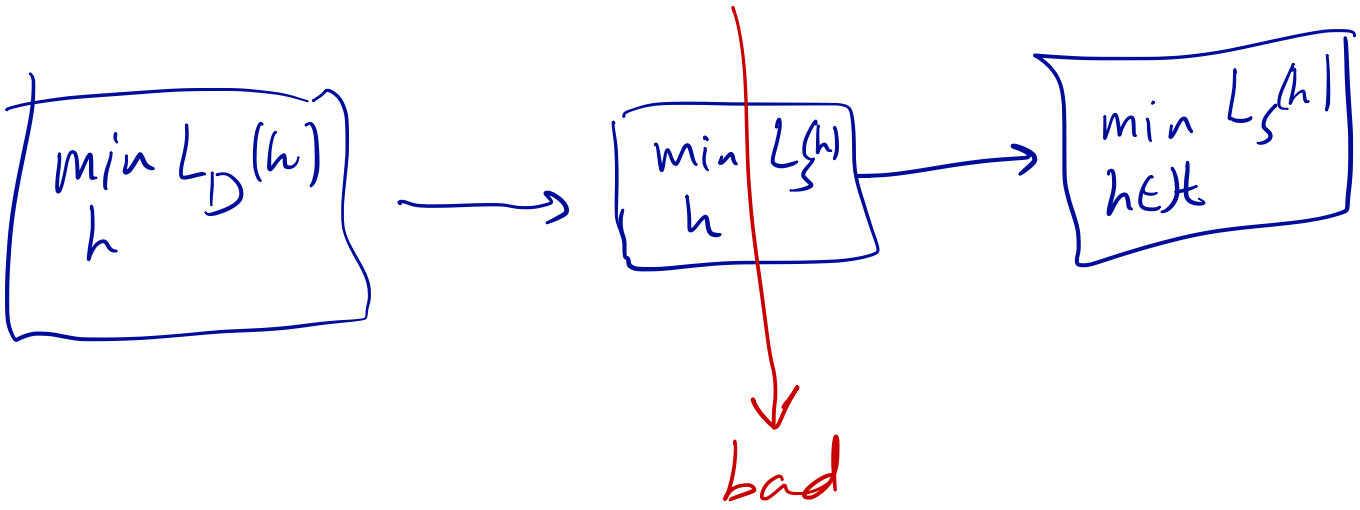
↓

prevent overfitting

A common solution is to apply ERM over a restricted set of functions (classifiers). Formally the learner should choose in advance (before seeing the data) a set of functions (classifiers). This set is called "the predictor class" or "the function class" or "the hypothesis class" and is denoted by $\mathcal{H}$. Each $h \in \mathcal{H}$ is a function from $\mathcal{X}$ to $\mathcal{Y}$.

And the restricted ERM problem ($ERM_{\mathcal{H}}$) is:

minimize $L_S(h)$.
$h \in \mathcal{H}$



So we are biasing the learner towards a particular set of predictors.

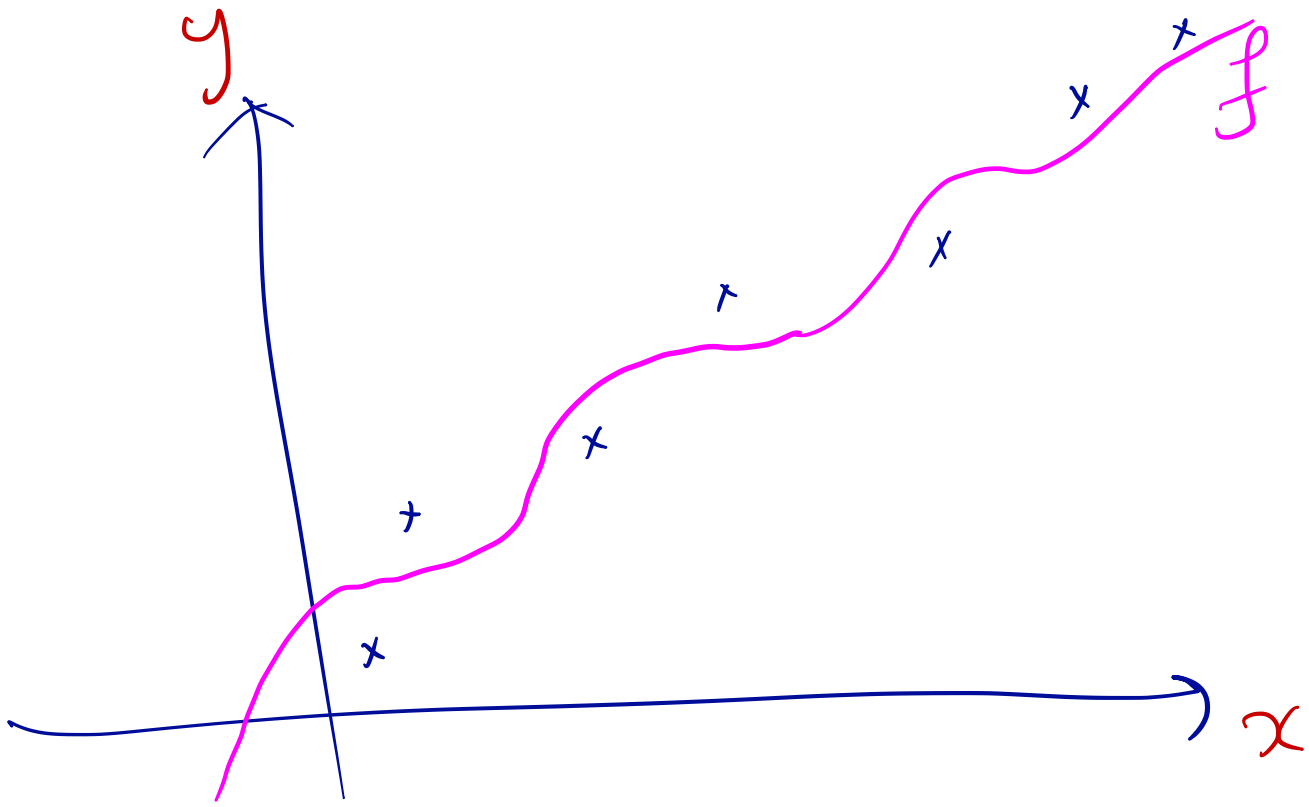
To explain things further, let's look at an example about regression (even though it is not classification, it will help us understand the concepts better).

$$y = f(x) + \epsilon$$

ϵ: independent gaussian $N(0, \sigma^2)$

We don't know f and we'd like to estimate the predictive relation between x and y using training data.

$$S = \{(x_i, y_i)\}_{i=1}^{n}$$



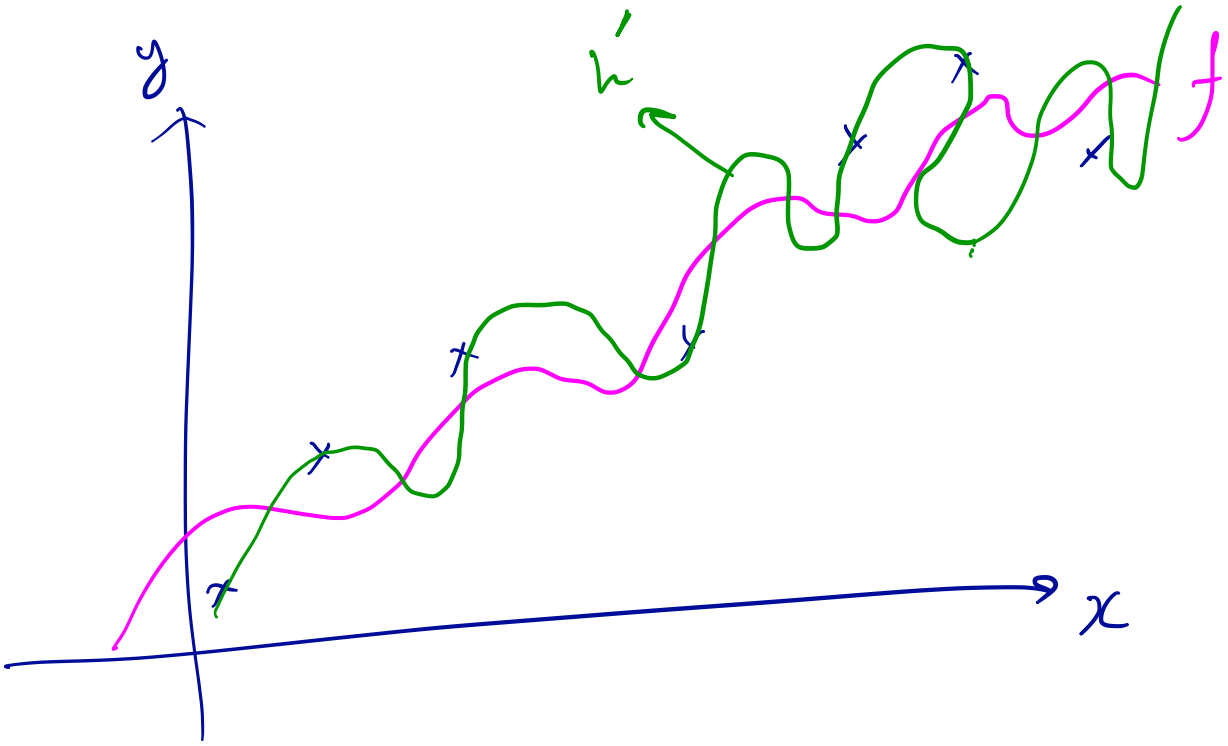
for any $h : X \longrightarrow Y$

$$L_D(h) = \underset{(x,y)\sim D}{\mathbb{E}}\left[(h(x) - y)^2 \right]$$

$$\min_h L_D(h) = \sigma^2$$

$\min_h$ ↳ minimizer $\longrightarrow f$

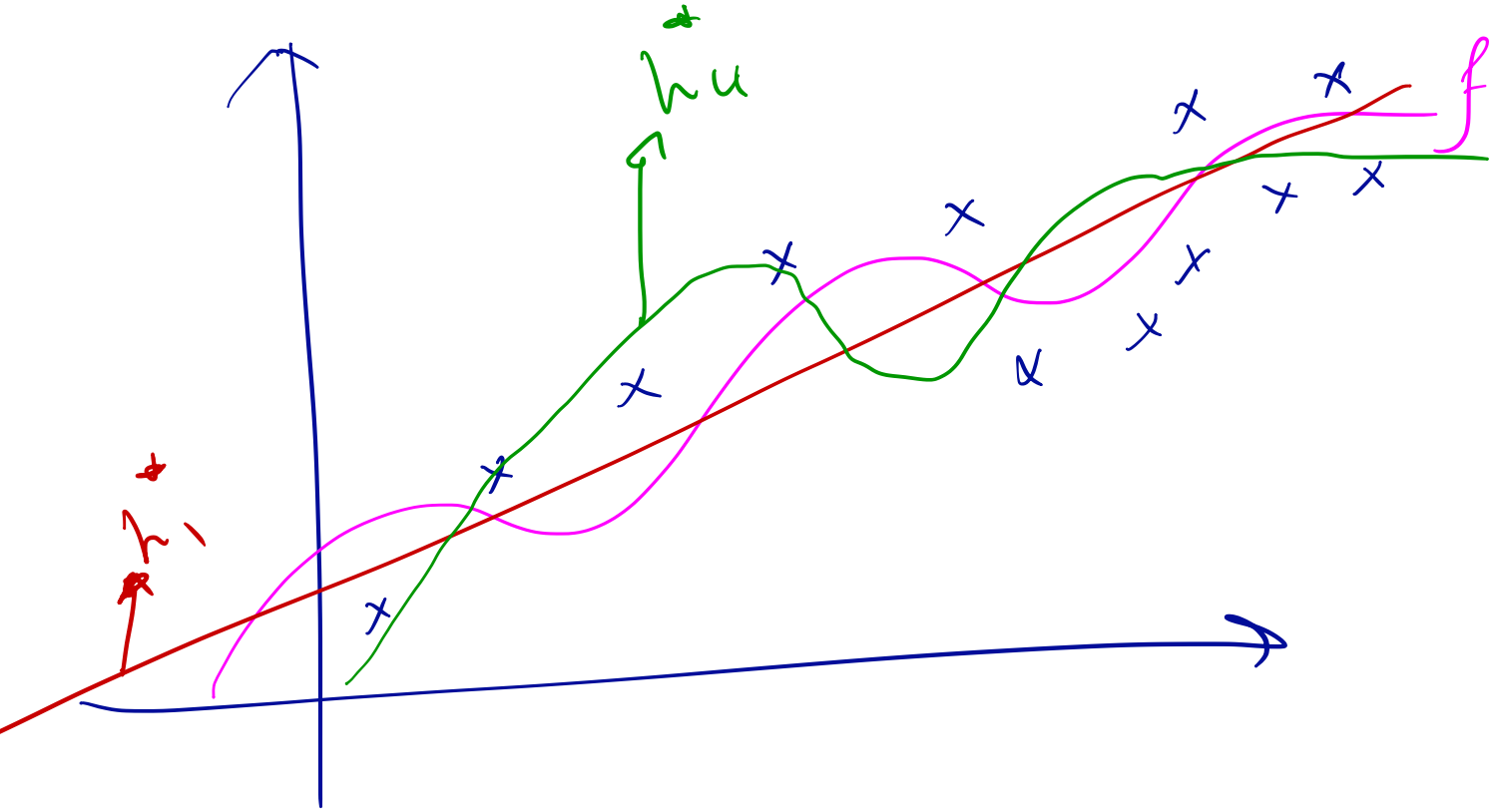
$$L_D(f) = \underbrace{\mathbb{E}\left[(y - f(x))^2 \right]}_{\epsilon^2} = \sigma^2$$



$$\begin{cases} \min_h L_D(h) = 6^2 \\ \min_h L_S(h) = 0 \end{cases} \longrightarrow L_S(h') = 0$$

Let's now consider restricted ERM objectives:

$$\min_{h \in \mathcal{H}} L_S(h)$$



$ERM_{\mathcal{H}} \longrightarrow \min_{h \in \mathcal{H}} L_S(h)$

e.g. when $\mathcal{H}_1 = \{$ polynomials of degree 1 $\}$

$\hookrightarrow \min_{h \in \mathcal{H}_1} L_S(h) \longrightarrow h_1^*$

$\mathcal{H}_4 = \{$ polynomials of degree 4 $\}$

$\min_{h \in \mathcal{H}_4} L_S(h) \longrightarrow h_4^*$

There are two important points:

(1) since we are restricting our function class, our best bet would be to achieve

$$\min_{h \in \mathcal{H}} L_D(h)$$

which is larger than

$$\min_{h \in \mathcal{H}} L_D(h) \geq \min_{h} L_D(h)$$

$$\min_{h} L_D(h) \neq 0$$

Hence, if the class $\mathcal{H}$ is not rich enough we may have

(underfitting)

$$\min_{h \in \mathcal{H}} L_D(h) \gg \min_h L_D(h)$$

$\Rightarrow$ $\mathcal{H}$ should be sufficiently complex

(2) The minimizer of

$$\min_{h \in \mathcal{H}} L_S(h)$$

is not the same as the minimizer of

$$\min_{h \in \mathcal{H}} L_D(h)$$

In other words if we denote

$$\rightarrow h_S^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}}\ L_S(h)$$

↳ this is computable

then it is possible that

$$L_D(h_S^*) \gg \min_{h \in \mathcal{H}} L_D(h)$$

↑ overfitting

To avoid overfitting in $\mathcal{H}$ we should make sure that we have sufficiently many training data points.

h_S^* → minimizer of $L_S(h)$ within the class $\mathcal{H}$.

since $h^* \in \mathcal{H} \Rightarrow L_D(h^*) \geq \min_{h \in \mathcal{H}} L_D(h)$

$$L_D(h^*) >> \min_{h \in \mathcal{H}} L_D(h)$$

↑ overfitting

---

two potential problems:

(1) $\min_{h \in \mathcal{H}} L_D(h) >> \min_h L_D(h)$

↑ underfitting

we need to increase the complexity of $\mathcal{H}$

→ solution of $ERM_{\mathcal{H}}$

(2) $L_D(h_S^*) >> \min_{h \in \mathcal{H}} L_D(h)$

we need to increase the # of train data points

↓ overfitting

Lecture 24:

$$\min_h L_D(h) \longrightarrow \min_h L_S(h)$$

$L_S(h) \to 0$

$$\tilde{h} = \begin{cases} y_i & x = x_i \\ 0 & x \notin \{x_1, \dots, x_n\} \end{cases}$$

$L_S(\tilde{h}) = 0$

$L_D(\tilde{h})$ very large

$$\min_{h \in \mathcal{H}} L_S(h)$$

1. Since we're restricting our search to $\mathcal{H}$

$$\min_{h \in \mathcal{H}} L_D(h) \geq \min_h L_D(h)$$

2. what we're actually optimizing is

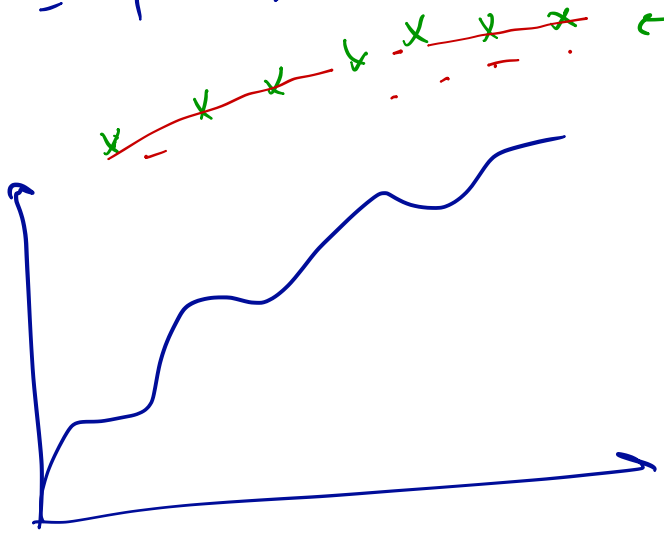
$$\min_{h \in \mathcal{H}} L_S(h) \Longrightarrow$$ Let's denote the minimizer by h_S^*.

$$L_D(h_S^*) \geq \min_{h \in \mathcal{H}} L_D(h)$$

↳ if $L_D(h_S^*)$ is much larger than $\min_{h \in \mathcal{H}} L_D(h)$, then this is called overfitting.

3. All the guarantees and bounds are probabilistic (they'll hold with high probability excluding the unlikely scenarios).

$$y_i = f(x_i) + \epsilon_i$$

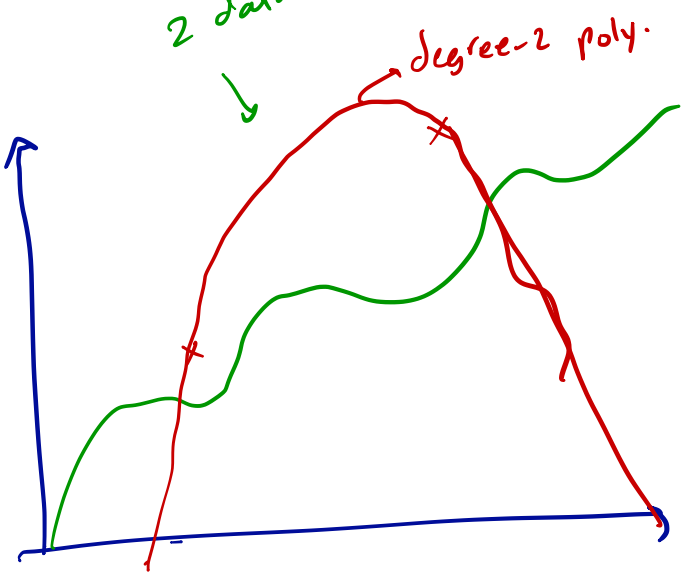


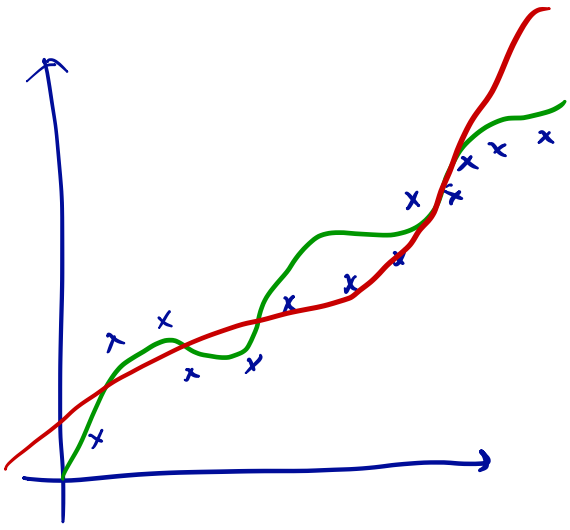
$L_D \rightarrow E_D$

$L_S \rightarrow \frac{1}{n} \sum_i$

$L_S \rightarrow L_D$

$\min_{h \in \mathcal{H}} L_S(h) \rightarrow$ in order to ensure that overfitting does not take place, we need to have a sufficient number of training data points.





$y_i = f(x_i) + \epsilon_i$

$\mathcal{H} = \{$ polynomials of degree 3 $\}$

$\Rightarrow$ overfitting can be avoided provided that we have sufficiently many traing data points.

Hence, the main question is:

How many training data points do we need to make sure that overfitting does not happen?

↳ # of data points that we need will depends on:

$\mathcal{H}$, ϵ , δ

what does "avoiding overfitting" mean?

$h_S^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}}\, L_S(h)$:

what we can do computationally

$\min_{h \in \mathcal{H}} L_D(h)$

the best error that we can hope for.

overfitting is avoided when: $L_D(h_S^*)$ is close to $\min_{h \in \mathcal{H}} L_D(h)$

Overfitting is avoided when

small value

$$\min_{h \in \mathcal{H}} L_D(h) \leq L_D(h_S^*) \leq \min_{h \in \mathcal{H}} L_D(h) + \epsilon$$

With probability $1-\delta$ we have:

$$L_D(h_S^*) - \min_{h \in \mathcal{H}} L_D(h) \leq \epsilon$$

provided that sufficiently many data points are available.

---

if we have more than n_ϵ data points, then with high probability the following is true:

with probability at least $1-\delta$

$$L_D(h_S^*) - \min_{h \in \mathcal{H}} L_D(h) \leq \epsilon$$

n_ϵ depends on $\epsilon, \delta, \mathcal{H}$.

PAC Learning:

PAC = Probably Approximately Correct

Definition (PAC): A function class $\mathcal{H}$ is called PAC learnable if for every $\epsilon, \delta \in (0,1)$, there exists a number $n_0(\epsilon, \delta)$ such that the following holds:

There exist a learning algorithm that for any distribution D over $X \times Y$, by using a set S of $n_0(\epsilon, \delta)$ data points, we obtain a predictor function h_S^* such that with probability $1-\delta$ we have

$$\min_{h \in \mathcal{H}} L_D(h) \leq L_D(h_S^*) \leq \min_{h \in \mathcal{H}} L_D(h) + \epsilon.$$

$\mathcal{H} \longrightarrow \begin{cases} \text{Algorithm} \rightarrow h_S^* \\ n_0(\epsilon, \delta) \end{cases} \Rightarrow \mathcal{H}$ is PAC learnable

---

Let's start with the simplest possible hypothesis class and see what PAC-learning means with that class.

$$\mathcal{H} = \{h_1, h_2, \ldots, h_m\}$$

$\min_{h \in \mathcal{H}} L_D(h)$ $\longrightarrow$ $\min_{h \in \mathcal{H}} L_S(h)$

$\Downarrow$

h_S^* is one of the functions in $\mathcal{H}$ with smallest empirical error.

$\min_{h \in \mathcal{H}} L_S(h) = \min\{L_S(h_1), \ldots, L_S(h_m)\}$

What we'd like to find in this case is a number $n_0(\epsilon, \delta)$ such that for any distribution D over data we have:

with probability $1-\delta$:

$$L_D(h_S^*) \leq \min_{h \in \mathcal{H}} L_D(h) + \epsilon$$

In other words we're asking how large the number of training data points should be s.t. we have w.p. $1-\delta$ that $L_D(h_S^*) \leq \min_{h \in \mathcal{H}} L_D(h) + \epsilon$.

In order to find $n_0(\epsilon, \delta)$ we need to answer two main questions.

(1) Given a fixed function $h: X \to Y$, how many traing data point should we have such that:

with probability $1-\delta$:

$$|L_S(h) - L_D(h)| < \varepsilon$$

$$\left| \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{h(x_i) \neq y_i\} - E_D[\mathbb{1}\{h(x) \neq y\}] \right| < \epsilon$$

what should be $|S| = n_0(\epsilon, \delta)$?

To answer this question we'll use a very important probabilistic tool which is call the Hoeffding's inequality. This inequality has very useful in a variety of applications in data science (both in terms of theory

and algorithm design).

Hoeffding's Inequality:

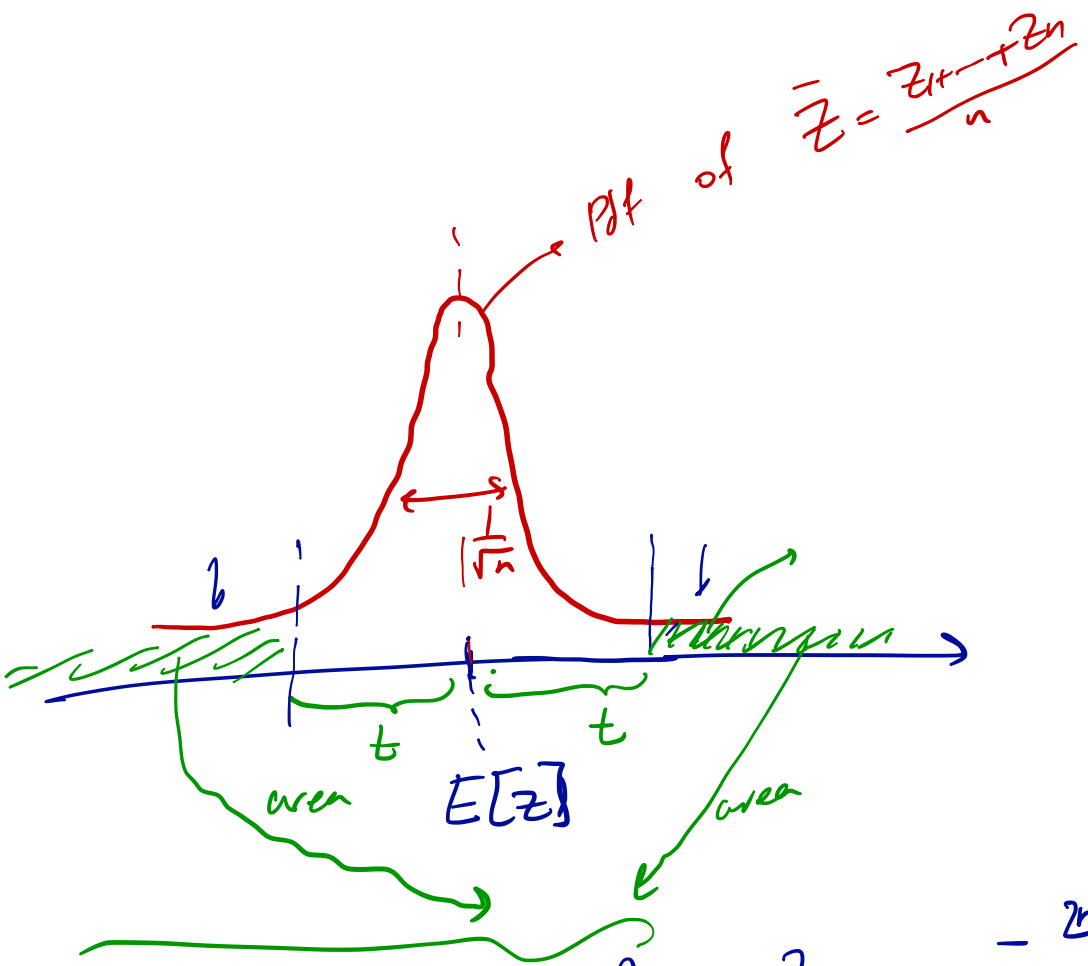
Let $Z_1, Z_2, \ldots, Z_n$ be n iid random variables that take value in the unit interval (i.e. $Z_i \in [0,1]$).

Then:

$$\Pr\left\{ \left| \frac{\sum_{i=1}^{n} Z_i}{n} - E[Z_i] \right| \geq t \right\} \leq 2e^{-2nt^2}$$

$Z_1, Z_2, \ldots, Z_n$ iid $Z \longrightarrow \sigma^2 = \mathrm{Var}(Z)$

$$\frac{Z_1 + Z_2 + \cdots + Z_n}{n} \sim E[Z] + \frac{1}{\sqrt{n}} N(0, \sigma^2)$$



$$Pr\{|\bar{Z} - E[Z]| > t\} \leq 2e^{-2nt^2}$$

e.g. $n = 1000$, $t = 0.5$

$$e^{-2 \cdot 1000 \cdot (0.5)^2} = e^{-500} \longrightarrow 0$$

Lecture 25:

$\mathcal{H}$ → PAC learnable:

(1) $n_0(\epsilon, \delta)$

(2) learning algorithm that takes as input a set of f training data points S, and outputs h^*_S.

for any data distribution D, as long as $|S| \geq n_0(\epsilon, \delta)$ then with probability $1-\delta$ we have:

$$L_D(h^*_S) \leq \min_{h \in \mathcal{H}} L_D(h) + \epsilon$$

Example:

$\mathcal{H} = \{h_1, \dots, h_m\}$

(1) specify what $n_0(\epsilon, \delta)$ is.

(2) Algorithm: ERM $\mathcal{H}$:

$$h_S^* = \operatorname*{argmin}_{h \in \mathcal{H}} L_S(h)$$

what is $n_0(\epsilon, \delta)$, s.t.

w.p. $1-\delta$

$$L_D(h_S^*) \leq \min_{h \in \mathcal{H}} L_D(h) + \epsilon$$

Two questions:

(1) Given a fixed function h, how many data points, S, do we need such that

w.p. $1-\delta$

$$|L_S(h) - L_D(h)| < \varepsilon$$

$$L_S(h) = \frac{1}{|S|} \sum_{i=1}^{|S|} \mathbb{1}\{h(x_i) \neq y_i\}$$

$\Updownarrow$ equivalent

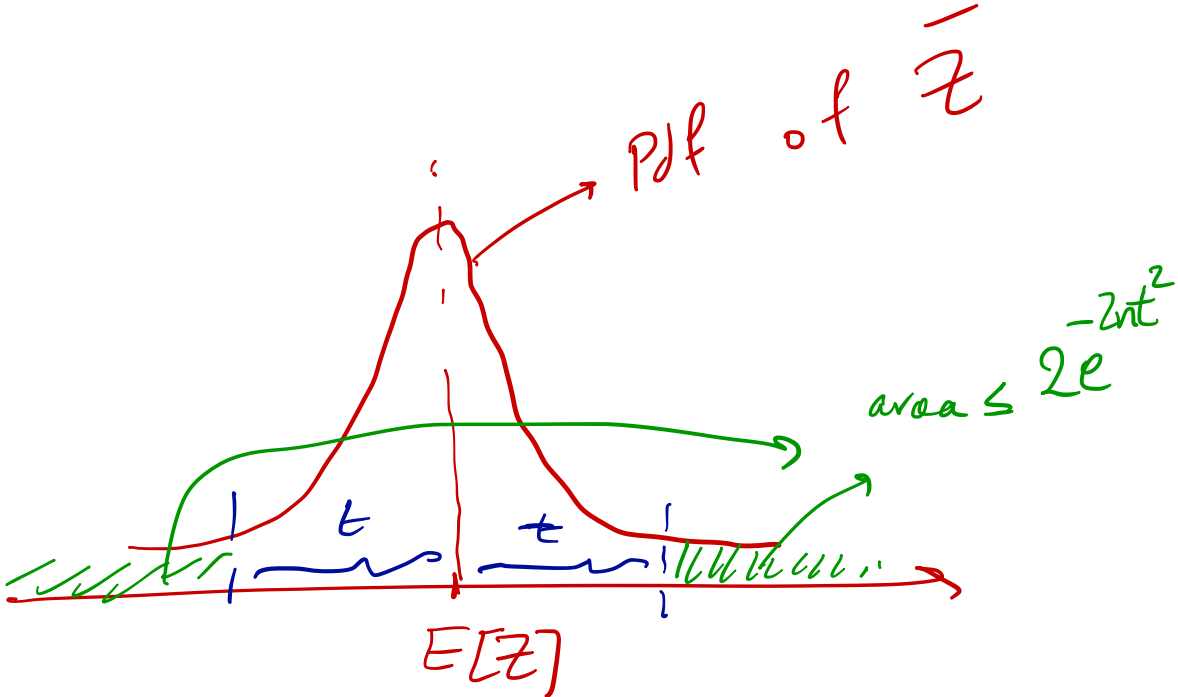
$$\Pr\{|L_S(h) - L_D(h)| \geq \varepsilon\} \leq \delta$$

Recall that Hoeffding's inequality was given as follows:

For any iid random variables $Z_1, Z_2, \dots, Z_n$, s.t. $Z_i \in [0,1]$,

we have

$$\Pr\left\{ \left| \overline{\frac{1}{n}\sum_{i=1}^{n} Z_i} - \overline{E[Z]} \right| > t \right\} \leq 2e^{-2nt^2}$$



$$L_S(h) = \frac{1}{n} \sum_{i=1}^{n} \underbrace{\mathbb{1}\{h(x_i) \neq y_i\}}_{Z_i}$$

$(|S| = n)$

$$L_D(h) = \underbrace{\mathop{E}_{(x,y)\sim D}\left[\mathbb{1}\{h(x) \neq y\}\right]}_{E[Z]}$$

$$L_S(h) = \frac{1}{n} \sum_{i=1}^{n} Z_i = \bar{Z}$$

$$Z_i = \begin{cases} 1 & \text{if } h(x_i) \neq y_i \\ 0 & \text{o.w.} \end{cases} \rightarrow Z_i \in [0,1]$$

$$E[Z_i] = \mathop{E}_{(x_i,y_i)\sim D}\left[\mathbb{1}\{h(x_i) \neq y_i\}\right]$$

$$= L_D(h)$$

Hoeffding:

$$P_r\left\{\left|\underbrace{L_S(h)}_{\frac{1}{n}\sum Z_i} - \underbrace{L_D(h)}_{E[Z]}\right| > \epsilon\right\} \quad \text{for any } \epsilon > 0 \qquad (1)$$

$$\leq 2e^{-2n\epsilon^2}$$

Recall that we're looking for the smallest n s.t. (2)

$$P_r\{|L_S(h) - L_D(h)| > \epsilon\} \leq \delta$$

using (1), and in order to guarantee (2), we can let:

$$2e^{-2n\epsilon^2} \leq \delta \qquad \left(\log\frac{2}{\delta} = -\log\frac{\delta}{2}\right)$$

$$\Rightarrow \boxed{n \geq \frac{1}{2\epsilon^2}\log\frac{2}{\delta}}$$

So the final statement is:

Given any ϵ, δ, as long as the number of training data points is larger than $n_1 = \frac{1}{2\epsilon^2} \log \frac{2}{\delta}$, we have

$$\Pr_{(x,y)\sim D}\left\{ |L_S(h) - L_D(h)| > \epsilon \right\} \leq \delta.$$

(for a fixed function h)

(2) Let's now assume that we have m functions $h_1, h_2, \ldots, h_m$. What is the smallest value $n_0(\epsilon, \delta)$ such that

with probability $1-\delta$: (3)

$$\forall\, i \in \{1, \ldots, m\} \;:\; |L_S(h_i) - L_D(h_i)| < \epsilon .$$

To answer this question, we're going to write an equivalent formulation of relation (3):

Let A_i be the event that

$$|L_S(h_i) - L_D(h_i)| > \epsilon .$$

Then (3) is equivalent to:

$$Pr\{A_1 \cup A_2 \cup A_3 \cdots \cup A_m\} \leq \delta . \quad (4)$$

Remember that we are searching for the smallest value of n such that (4) holds.

$$Pr\{A_1 \cup A_2 \cup \cdots \cup A_m\} \leq \delta$$

To answer this, we're going to use the Union bound:

$$Pr\{A \cup B\} \leq Pr\{A\} + Pr\{B\}$$

$$Pr\{A_1 \cup A_2 \cup \cdots \cup A_m\} \leq Pr\{A_1\} + Pr\{A_2\} + \cdots + Pr\{A_m\}.$$

---

bad event #i : A_i :

$$|L_S(h_i) - L_D(h_i)| > \epsilon$$

We'd like to make sure that none of the bad events, A_i, would take place.

So in order to guarantee

$$Pr\{A_1 \cup A_2 \cup \cdots \cup A_m\} \leq \delta \quad (5)$$

it is sufficient to guarantee that

$$Pr\{A_1\} + Pr\{A_2\} + \cdots + Pr\{A_m\} \leq \delta \quad (6)$$

Note that if (6) holds, then (5) would also hold as a result of the union bound.

Now, to guarantee (6), it is sufficient to choose n large enough s.t. for every h_i

We have

$$Pr\{A_i\} \leq \frac{\delta}{m} \qquad (7)$$

$$\Downarrow$$

$$Pr\{A_1\} + Pr\{A_2\} + \cdots + Pr\{A_m\} \leq \delta$$

$$\Downarrow$$

$$Pr\{A_1 \cup A_2 \cup \cdots \cup A_m\} \leq \delta$$

Now given our answer to question 1, in order to guarantee that $Pr\{A_i\} \leq \frac{\delta}{m}$ we need to

Choose:

if $n \geq n_1(\epsilon, \frac{\delta}{m}) = \underbrace{\frac{1}{2\epsilon^2} \log \frac{2m}{\delta}}_{n_0(\epsilon,\delta)}$

then

$$Pr\{A_i\}$$

$$= Pr\{ |L_S(h_i) - L_D(h_i)| > \epsilon \} \leq \frac{\delta}{m}$$

$\left(n_0(\epsilon,\delta) = n_1(\epsilon, \frac{\delta}{m}) \right)$

Hence,

Statement: If the number of training data points, $|S|$, is larger than $n_0(\epsilon,\delta) = \frac{1}{2\epsilon^2} \log \frac{2m}{\delta}$, then

with probability $1-\delta$ we have

$\forall i \in \{1,\dots,m\}$: $|L_S(h_i) - L_D(h_i)| < \epsilon$.

What we've shown is that we need

$n_0(\epsilon, \delta) = \frac{1}{2\epsilon^2} \log \frac{2m}{\delta}$ data points to guarantee that for all the m functions $h \in \mathcal{H}$ the value of $L_S(h)$ is close to the value of $L_D(h)$:

w.p $1-\delta$:

$$\forall h \in \mathcal{H} : |L_S(h) - L_D(h)| < \epsilon.$$

Now, Let h^*_S be the minimizer of ERM over the class $\mathcal{H}$:

$$h^*_S : \operatorname{argmin}_{h \in \mathcal{H}} L_S(h)$$

($L_S(h^*_S)$ is the smallest among $\mathcal{H}$)

Let $h^*_{true} : \operatorname{argmin}_{h \in \mathcal{H}} L_D(h)$

We will show that if the number of training data points is at least $n_0(\epsilon, \delta)$ then with $1-\delta$ we have

$$L_D(h^*_S) - \min_{h \in \mathcal{H}} L_D(h) \leq 2\epsilon.$$

Hence, $\mathcal{H}$ is PAC-learnable with $n_0(\epsilon, \delta)$ data points.

Let's see why (8) holds.

$$L_D(h^*_S) - L_D(h^*_{true})$$

$$= \underbrace{L_D(h^*_S) - L_S(h^*_S)}_{\leq \epsilon} + \underbrace{L_S(h^*_S) - L_S(h_{true})}_{\leq 0} + L_S(h^*_{true}) - L_D(h^*_{true})$$

(h^*_S is the minimizer of $L_S(\cdot)$)

$$\leq \epsilon + 0 + \epsilon \leq 2\epsilon.$$

Hence, if the number of training data points is at least $n_0(\epsilon, \delta) = \frac{1}{2\epsilon^2} \log \frac{2m}{\delta}$

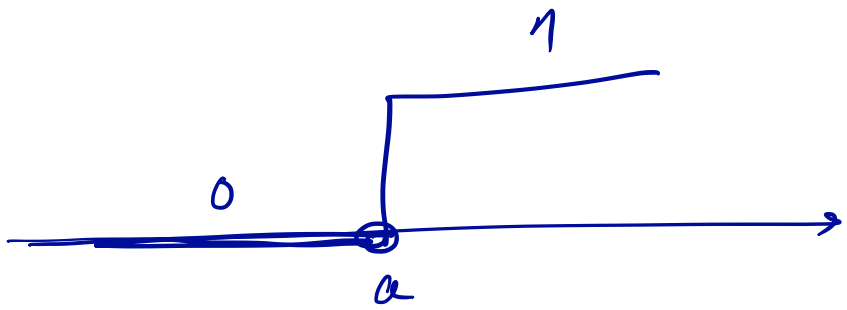
then w.p. $1-\delta$ we have

$$0 \leq L_D(h_S^*) - \min_{h \in \mathcal{H}} L_D(h) \leq 2\epsilon.$$

h_S^*: minimizer of ERM over $\mathcal{H}$.

When $\mathcal{H} = \{h_1, \dots, h_m\}$.

$h_a(x) =$



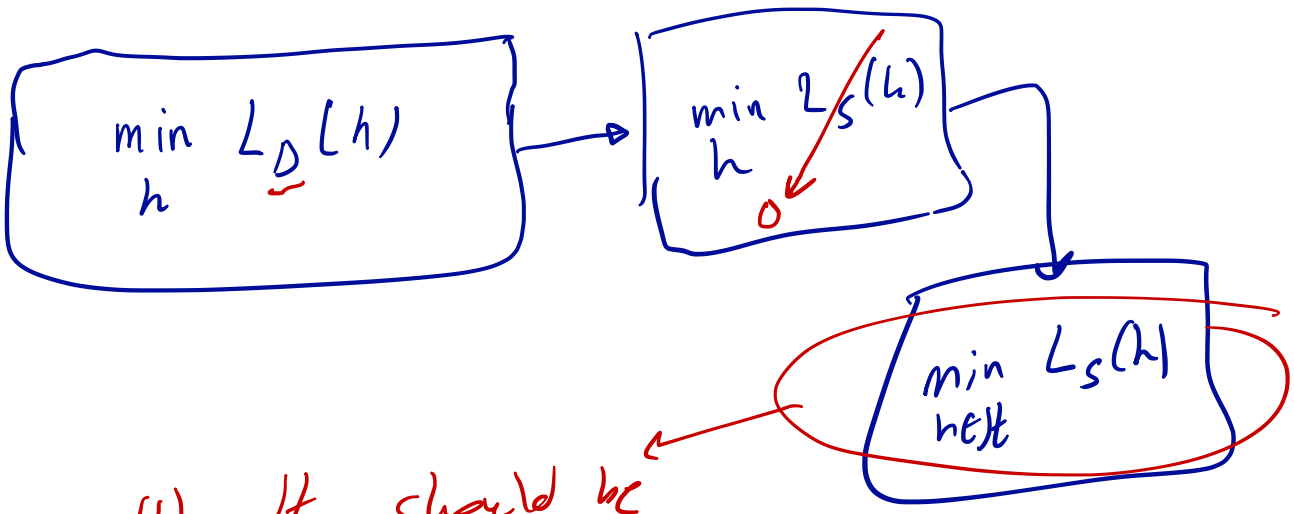
$$H = \{ h_a(x) \;,\; a \in (-\infty, \infty) \}$$

infinitely many functions

One important consequece of PAC-learnebility is that it's sufficient to work with a finite data set and we don't lose any thing in terms of generalization

Lecture 26:

- $(x, y) \sim D$ →



(1) H should be sufficiently complex

(2) Avoid overfitting: have sufficiently many data points

PAC: H is PAC learnable:

(1) $n_0(\epsilon, \delta) \in \mathbb{N}$

(2) learning alg. $S \longrightarrow h_S^* \in H$ (S: training data)

If $|S| \geq n_0(\epsilon, \delta)$:

w.p. $1-\delta$:

$$L_D(h_S^*) \leq \min_{h \in H} L_D(h) + \epsilon$$

$H = \{h_1, \dots, h_m\}$

- $n_0(\epsilon, \delta) = \frac{1}{\epsilon^2} \log \frac{2m}{\delta} \approx$ (up to constants) $\frac{\log \frac{1}{\delta} + \log m}{\epsilon^2}$

- learning algorithm: $h_S^* = \underset{i=1,\dots,m}{\operatorname{argmin}} L_S(h_i)$

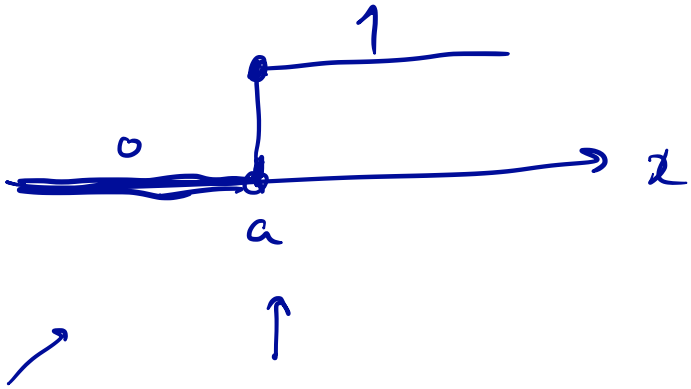
H is PAC-learnable with $n_0(\epsilon, \delta)$ samples and ERM_H as the learning algorithm.

---

what about infinite H?

$H = \{ h_a(x) : \mathbb{R} \to \{0,1\}, a \in \mathbb{R} \}$

$h_a(x) \to$



The fundamental theorem of learning thorg:

For any function class $\mathcal{H}$, if the number of training data point is larger than $n_0(\epsilon, \delta) = \frac{\log 1/\delta + \overbrace{VC\text{-}dim(\mathcal{H})}}{\epsilon^2}$, then

w.p. $1-\delta$:

$$L_D(h_S^*) \leq \min_{h \in \mathcal{H}} L_D(h) + \epsilon.$$

where $h_S^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}}\ L_S(h)$.

→ $VC\text{-}dim(\mathcal{H})$ = Complexity of the class

→ if $VC\text{-}dim(\mathcal{H})$ is finite, then

$\mathcal{H}$ is PAC-learnable with

$n_0(\epsilon, \delta) = \frac{\log 1/\delta + VC\text{-}dim(\mathcal{H})}{\epsilon^2}$ and

ERM as the learning Algorithm.

Important implication:

We can learn from finite data sets.

$$L_D(h_S^*) \leq \underbrace{\min_{h \in \mathcal{H}} L_D(h)}_{\text{the best that we could do}} + \epsilon$$

h_S^*: tractable

$\min_{h \in \mathcal{H}} L_D(h)$: intractable

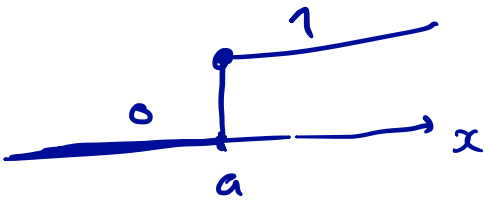
As long as we have sufficiently many data points we'll be fine with solving the "tractable" ERM-H problem and we lose at most a small value ϵ with respect to the optimal accuracy.

VC dimension:

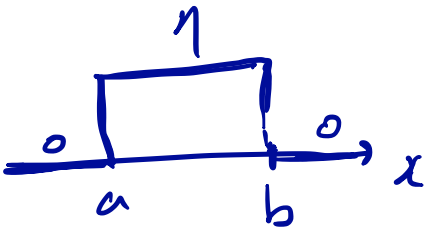
Vapnik-Chernovskii

Consider the following 4 function classes:

$$H_1 = \{ h_a(x),\ a \in \mathbb{R} \} \rightarrow$$

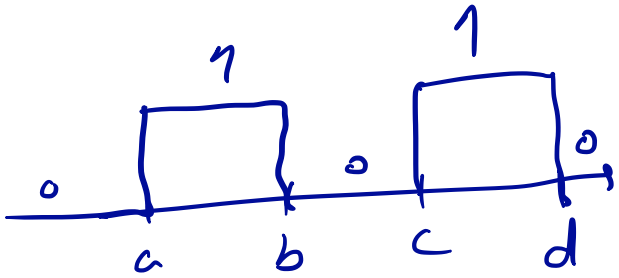


$$H_2 = \{ h_{ab}(x),\ a,b \in \mathbb{R} \} \rightarrow$$



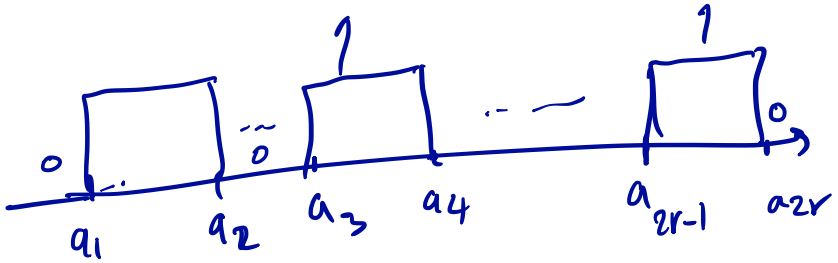
$$H_3 = \{ h_{abcd}(x),\ a,b,c,d \in \mathbb{R} \}$$

$h_{abcd}(x)$:



$$H_4 = \{ h_{a_1 a_2 - a_{2r}}(x),\ a_1, \dots, a_{2r} \in \mathbb{R} \}$$

$h_{a_1, a_2, \dots, a_{2r}}(x)$:



$H_1 < H_2 < H_3 < H_4 \rightarrow$ complexity

Definition: (restriction):

Let $\mathcal{H}$ be a class of functions from X to $\{0,1\}$ $(\forall h \in \mathcal{H},\ h: X \to \{0,1\})$.

Let $C = \{x_1, x_2, \ldots, x_k\} \subseteq X$.

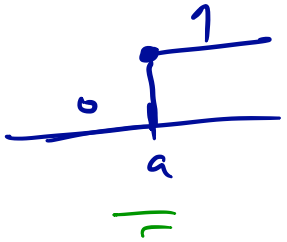
The restriction of $\mathcal{H}$ to C is the set of all the k-tuples

$$\mathcal{H}_C = \{ (h(x_1), h(x_2), \ldots, h(x_k)) ,\ h \in \mathcal{H} \}.$$

Example.

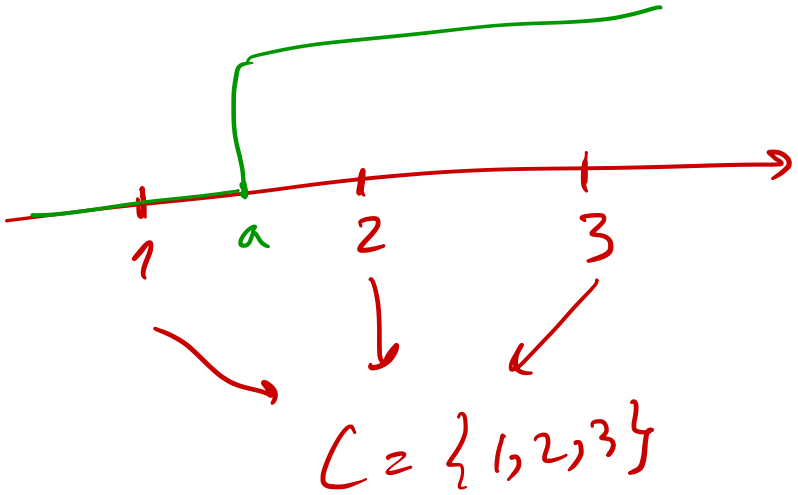
e.g. if $\mathcal{H} = \{ h_a(x) ,\ a \in \mathbb{R} \}$

$h_a(x) =$



Let $C = \{1, 2, 3\}$.

$$\mathcal{H}_C = \{ (h_a(1), h_a(2), h_a(3)) ,\ h_a \in \mathcal{H},\ a \in \mathbb{R} \}$$



$C = \{1, 2, 3\}$

$a < 1 : (h_a(1), h_a(2), h_a(3)) = (1, 1, 1)$

$a \in [1, 2) :$ " $= (0, 1, 1)$

$a \in [2, 3)$ " $= (0, 0, 1)$

$a \geq 3$ " $= (0, 0, 0)$

$$H_C = \{(1,1,1), (0,1,1), (0,0,1), (0,0,0)\}$$

$$|H_C| = 4 < 2^{|C|} = 2^3 = 8$$

↳ It is not sufficiently complex to produce all the possible binary 3-tuples on C.

Note:

If $C = \{x_1, \ldots, x_k\}$

$$|H_C| \leq 2^{|C|}$$

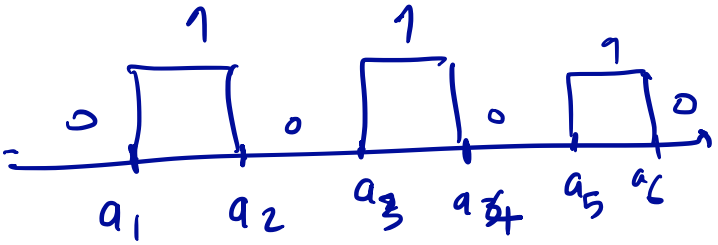
$$H_C = \underbrace{\{(0,0,\ldots,0), (1,0,\ldots,0), (0,1,0,\ldots,0), \ldots, (1,1,\ldots,1)\}}_{2^k}$$

Definition (Shattering): We say that a function class H shatters a set C if $|H_C| = 2^{|C|}$

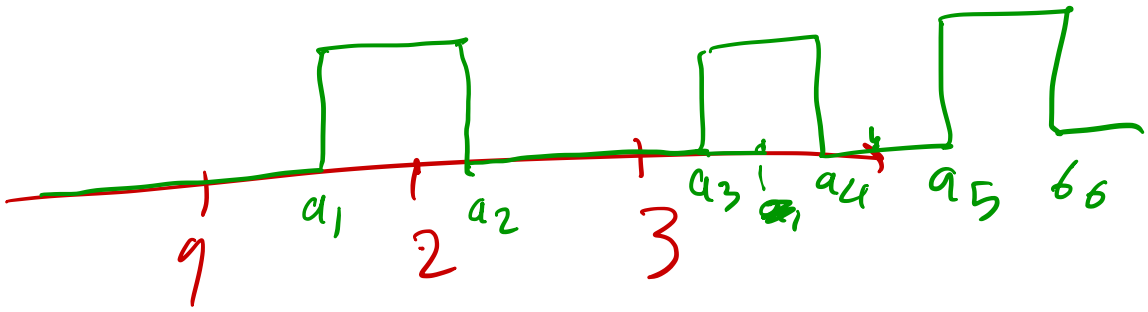
Example:

$$\mathcal{H} = \{ h_{a_1 a_2 a_3 a_4 a_5 a_6}(x) \;,\; a_1 \to a_6 \in \mathbb{R} \}$$

$h_{a_1 \to a_6}(x)$ =



$$C = \{1, 2, 3\}$$



$$\mathcal{H}_C = \{ (0,0,0), (1,0,0), (0,1,0), (0,0,1), \ldots, (1,1,1) \}$$

↳ $\mathcal{H}_C$ is all the 8 possible binary 3-tuples on C.

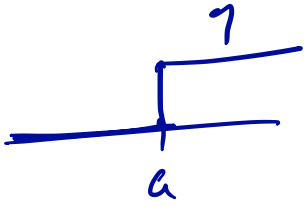
↳ It shatters C.

Definition (VC-dimension):

The VC-dimension of a function class $\mathcal{H}$ is the largest number k such that exists a set C of size k which is shattered by $\mathcal{H}$.

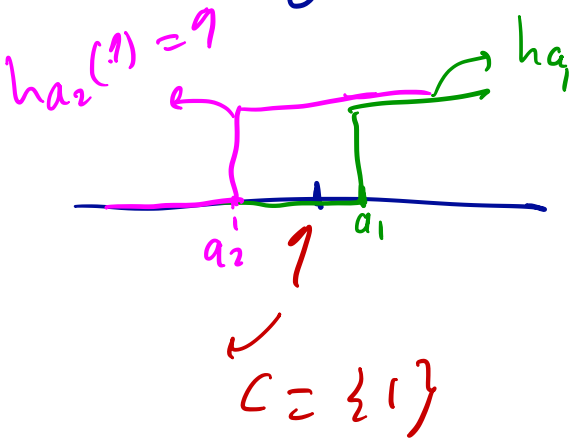
Examples:

$\mathcal{H} = \{ h_a(x), \; a \in \mathbb{R} \}$ →



VC-dim($\mathcal{H}$) =

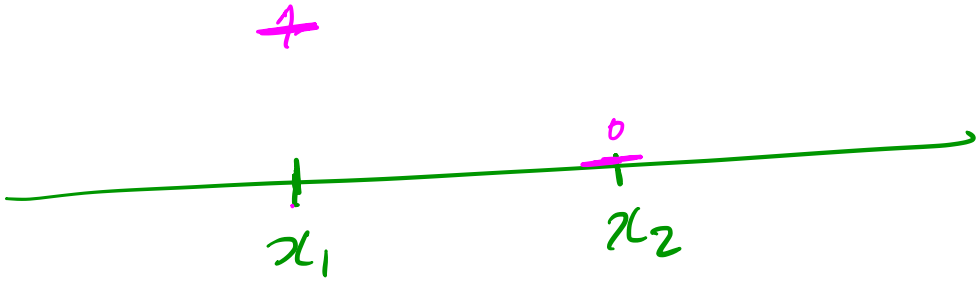
Is there a set of size 1 that's shattered by $\mathcal{H}$? YES.



$\rightsquigarrow H_C = \{0, 1\}$

↳ $\mathcal{H}$ shatters C.

Is there a set of size 2 which is shattered by $\mathcal{H}$? No



$C = \{x_1, x_2\}$ (assume $x_1 < x_2$)

↳ $\mathcal{H}_C = \{(-,-), (+,-), (-,+), (+,+)\}$

↳ there is no $h_a(x)$ that produces this binary 2-tuple

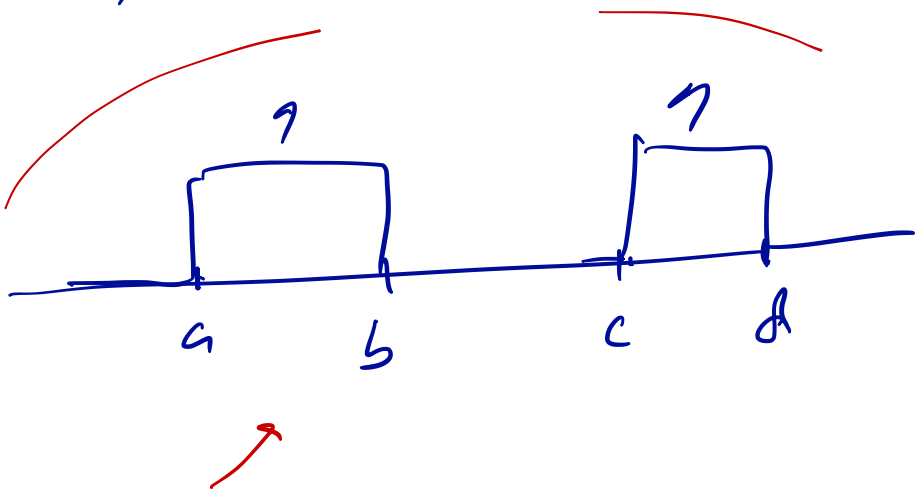
$VC\text{-}dim(\mathcal{H}) = 1$

⇓

there exists no set C of size 2 which is shattered by $\mathcal{H}$

Example:

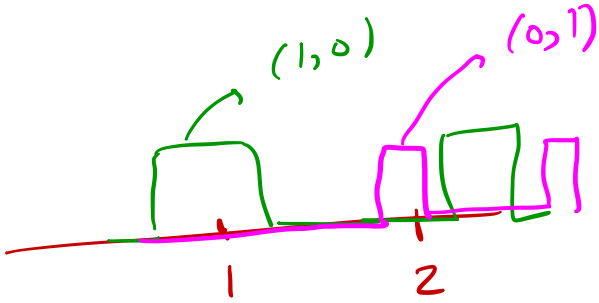
$$\mathcal{H} = \{ h_{a,b,c,d}(x) : a,b,c,d \in \mathbb{R} \}$$



is there a set C of size k which is shattered by $\mathcal{H}$?

$k \geq 1 \longrightarrow$ YES

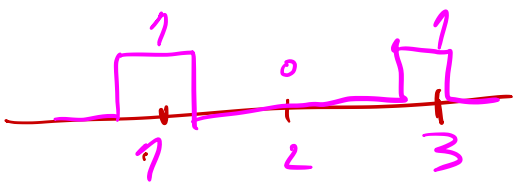
$k = 2 \longrightarrow$ YES



$k = 3 \longrightarrow$ YES

$k = 4 \longrightarrow$ YES

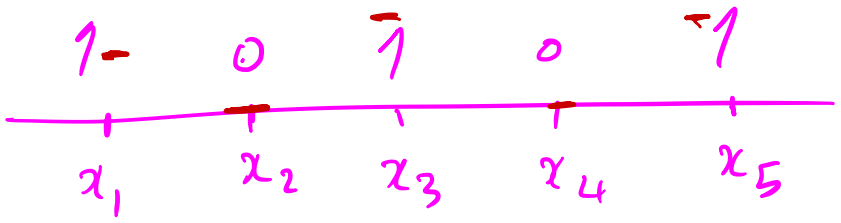
$k = 5 \longrightarrow$ NO



$$\mathcal{H}_C = \{ (0,0,0), (1,0,0), (0,1,0), (0,0,1), (1,0,1), (1,1,0), (0,1,1), (1,1,1) \}$$

$K = 5$

h should have 3 jumps from 0 to 1.



Lecture 23 :

$S = \{(x_i, y_i)\}_{i=1}^{n}$ → learner →

↓

ultimate goal:

$$\min_h \Pr_{(x,y)\sim D}\{h(x) \neq y\}$$

↓

the first relaxation:

$$\min_h \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{h(x_i) \neq y_i\} \longrightarrow \text{bad}$$

$= 0$

↓

restricted ERM_H

$$\min_{h \in H} \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{h(x_i) \neq y_i\}$$

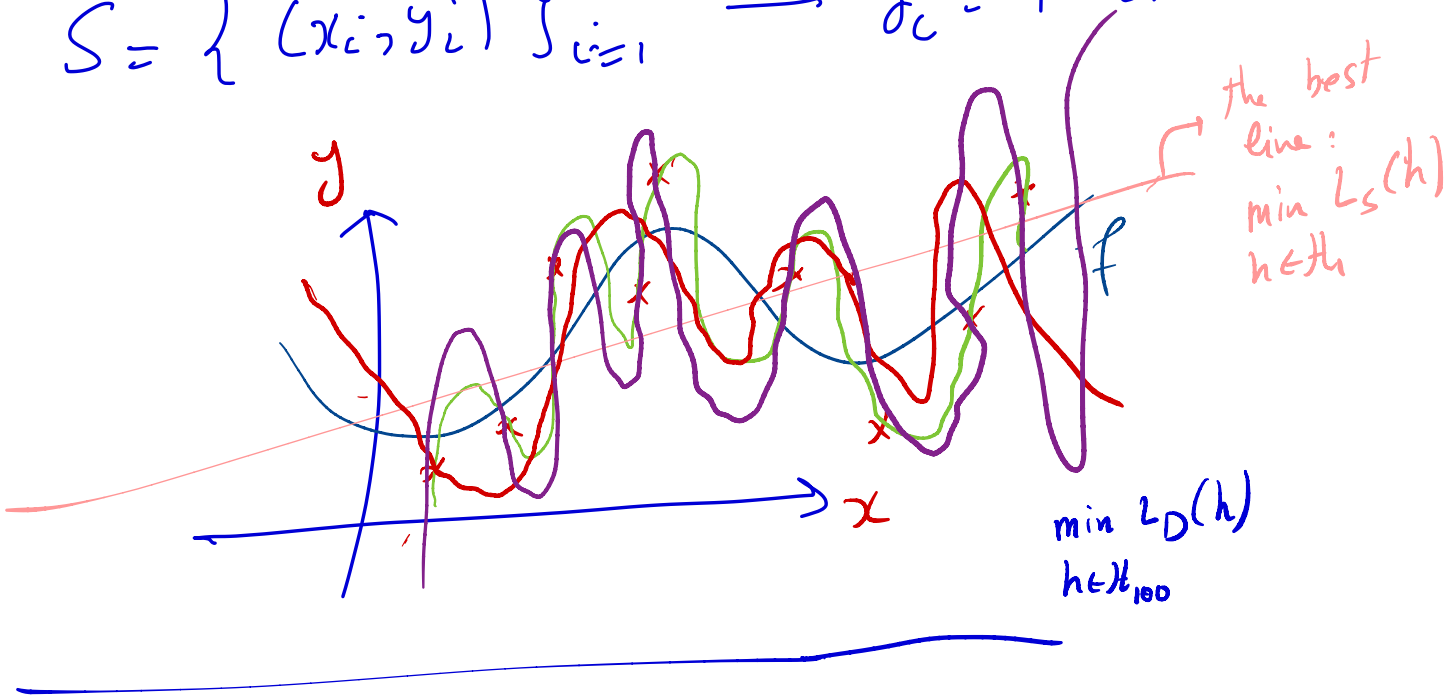
↙

the predictor is inside the class H.

Example:

$$y = f(x) + \varepsilon \qquad E[\varepsilon] = 0, \quad \mathrm{var}[\varepsilon] = \sigma^2$$

$$S = \{(x_i, y_i)\}_{i=1}^{n} \longrightarrow y_i = f(x_i) + \varepsilon_i$$



$$L_D(h) = \underset{(x,y)\sim D}{E}\left[(h(x) - y)^2 \right]$$

$$\min_h L_D(h) \Longrightarrow \text{the minimizer is } f$$

$$\rightarrow \min_{h \in \mathcal{H}} L_D(h) = E\left[(y - f(x))^2\right]$$

$$= E[\varepsilon^2] = \sigma^2$$

training error

$$\min_h L_S(h) = 0$$

$$\min_h L_D(h) = 6^2$$

$$\min_h L_S(h) = 0$$

$$\min_{h \in \mathcal{H}} L_S(h)$$

examples for $\mathcal{H}$:

$\mathcal{H}_1 = \{$ all the linear functions
i.e. polynomials of degree 1 $\}$

↳ since $\mathcal{H}_1$ is not rich enough, we will have underfitting

⇒ Hence the function class $\mathcal{H}$ that we're going to use, should be sufficiently rich / complex.

$H_6 = \{$ all the polynomials of degree 6 $\}$

$\Rightarrow$ the problem with using rich/complex function classes is: overfitting

In summary: we have 2 problems:

Problem 1:

if H is not rich enough, it'll underfit:

$$\min_{h \in H} L_D(h) \gg \min_{h} L_D(h)$$

Solution: use a class H that's sufficiantly rich.

Problem 2:

if H is complex, then it may suffer from overfitting:

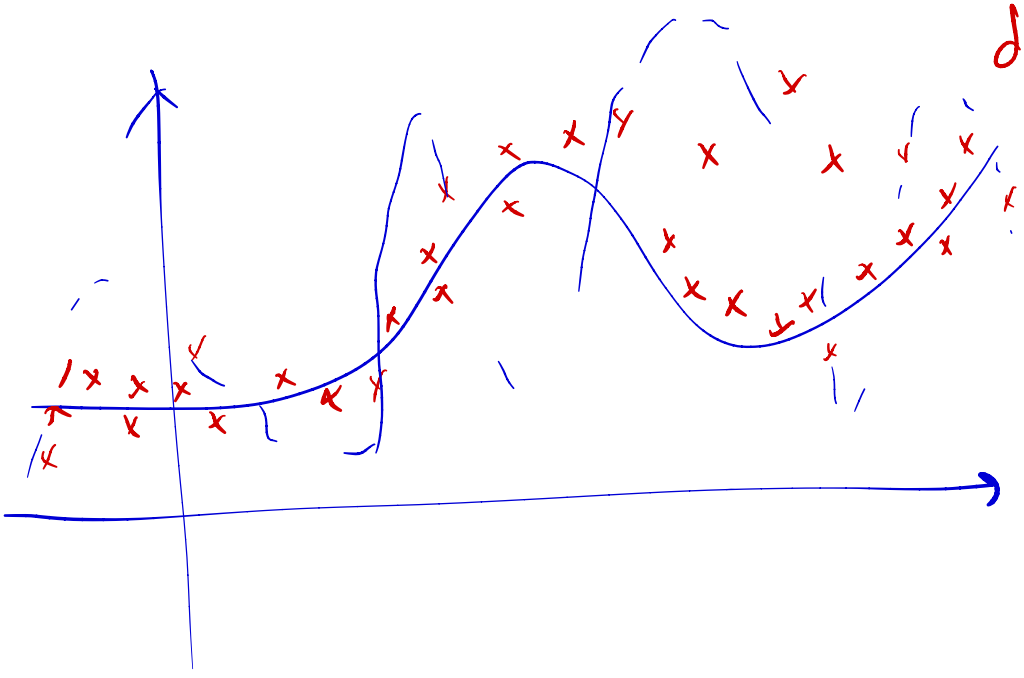
$$\min_{h \in H} L_D(h) \approx \min_{h} L_D(h)$$

$$\min_{h \in H} \underbrace{L_S(h)}_{\text{traing error}} \longrightarrow h_S^* \longrightarrow L_S(h_S^*) \approx 0$$

H: very rich/complex

$$\min_h L_D(h) \ll L_D(h_S^*)$$

true error of what we learned from fitting the function class H to the training data S.



$$L_S(h) \longrightarrow L_D(h)$$

$$\min_{h \in H} L_S(h) \longrightarrow \min_{h \in H} L_D(h)$$

$\Longrightarrow$ to avoid overfitting we should have a sufficient number of training data points.

for every: $h \in H \longrightarrow L_S(h) \approx L_D(h) \pm \varepsilon$

$$\min_{h \in H} L_S(h) \approx \min_{h \in H} L_D(h) \pm \varepsilon$$

It seems that to avoid overfitting the number of training data points should be sufficiently large

↳ - depends on H

- depends on the accuracy ε

PAC learning (PAC = Probably Approximately Correct)

We say that a function class $\mathcal{H}$ is PAC-learnable if for any $\epsilon, \delta > 0$, there exists a number $n_0(\epsilon, \delta)$, such that for any distribution of data, D, the following holds: If $n \geq n_0(\epsilon, \delta)$, then with probability $1-\delta$ we have that:

$$\forall h \in \mathcal{H} : \left| L_D(h) - L_S(h) \right| \leq \epsilon.$$

$$\min_{h \in \mathcal{H}} L_S(h) \approx \min_{h \in \mathcal{H}} L_D(h)$$

PAC Learning:

PAC = Probably Approximately Correct

Definition (PAC): A function class $\mathcal{H}$ is called PAC learnable if for every $\epsilon, \delta \in (0,1)$, there exists a number $n_0(\epsilon, \delta)$ such that the following holds:

There exist a learning algorithm that for any distribution D over $X \times Y$, by using a set S of $n_0(\epsilon, \delta)$ data points, we obtain a predictor function h_S^* such that with probability $1-\delta$ we have

$$\min_{h \in \mathcal{H}} L_D(h) \leq L_D(h_S^*) \leq \min_{h \in \mathcal{H}} L_D(h) + \epsilon.$$

$\mathcal{H} \longrightarrow \begin{cases} \text{Algorithm} \rightarrow h_S^* \\ n_0(\epsilon, \delta) \end{cases} \Rightarrow \mathcal{H}$ is PAC learnable

---

Let's start with the simplest possible hypothesis class and see what PAC-learning means with that class.

$$\mathcal{H} = \{h_1, h_2, \ldots, h_m\}$$

$\min_{h \in \mathcal{H}} L_D(h)$ $\longrightarrow$ $\min_{h \in \mathcal{H}} L_S(h)$

$\Downarrow$

h_S^* is one of the functions in $\mathcal{H}$ with smallest empirical error.

$\min_{h \in \mathcal{H}} L_S(h) = \min\{L_S(h_1), \ldots, L_S(h_m)\}$

What we'd like to find in this case is a number $n_0(\epsilon, \delta)$ such that for any distribution D over data we have:

with probability $1-\delta$:

$$L_D(h_S^*) \leq \min_{h \in \mathcal{H}} L_D(h) + \epsilon$$

In other words we're asking how large the number of training data points should be s.t. we have w.p. $1-\delta$ that $L_D(h_S^*) \leq \min_{h \in \mathcal{H}} L_D(h) + \epsilon$.

In order to find $n_0(\epsilon, \delta)$ we need to answer two main questions.

(1) Given a fixed function $h: X \to Y$, how many traing data point should we have such that:

with probability $1-\delta$:

$$|L_S(h) - L_D(h)| < \varepsilon$$

$$\left| \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{h(x_i) \neq y_i\} - E_D[\mathbb{1}\{h(x) \neq y\}] \right| < \epsilon$$

what should be $|S| = n_0(\epsilon, \delta)$?

To answer this question we'll use a very important probabilistic tool which is call the Hoeffding's inequality. This inequality has very useful in a variety of applications in data science (both in terms of theory

and algorithm design).

Hoeffding's Inequality:

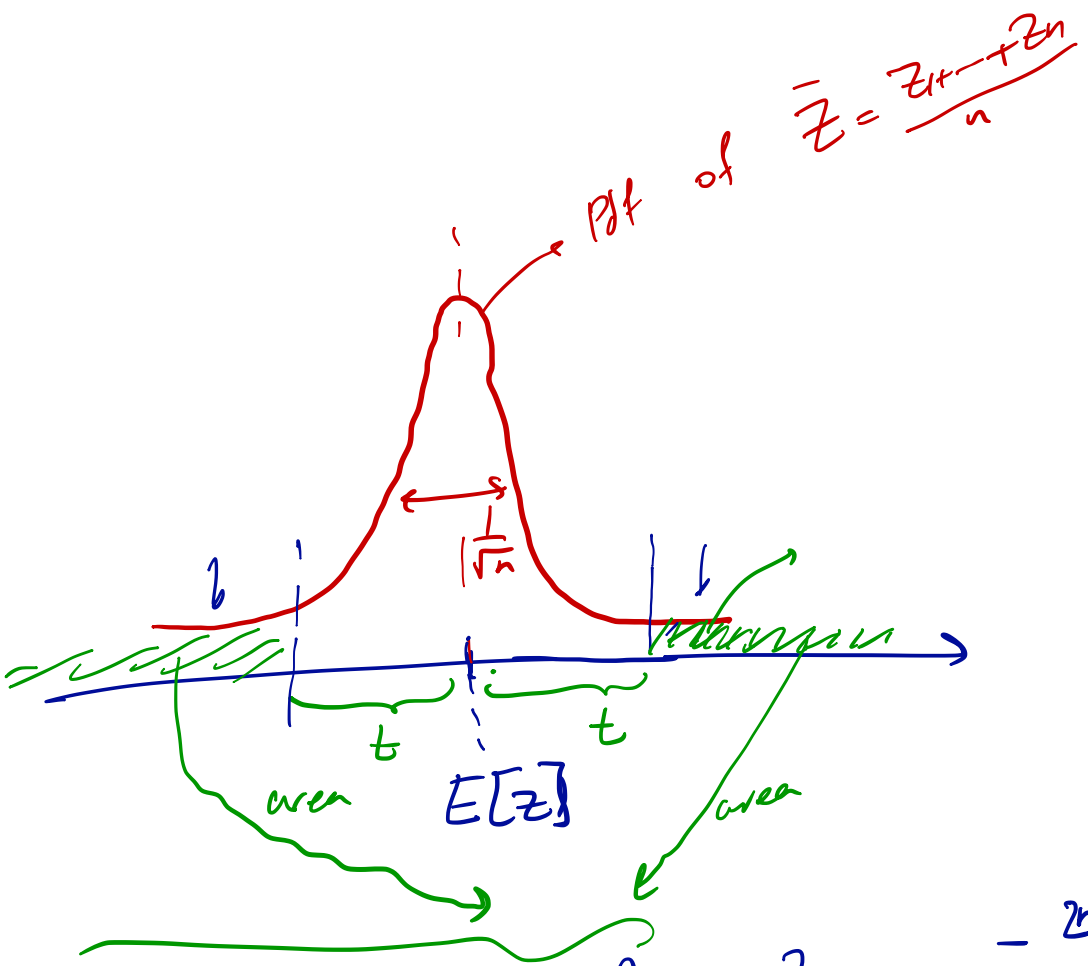
Let $Z_1, Z_2, \ldots, Z_n$ be n iid random variables that take value in the unit interval (i.e. $Z_i \in [0,1]$).

Then:

$$\Pr\left\{ \left| \frac{\sum_{i=1}^{n} Z_i}{n} - E[Z_i] \right| \geq t \right\} \leq 2e^{-2nt^2}$$

$Z_1, Z_2, \ldots, Z_n$ iid $Z \longrightarrow \sigma^2 = \mathrm{Var}(Z)$

$$\frac{Z_1 + Z_2 + \cdots + Z_n}{n} \sim E[Z] + \frac{1}{\sqrt{n}} N(0, \sigma^2)$$



$$Pr\{|\bar{Z} - E[Z]| > t\} \leq 2e^{-2nt^2}$$

e.g. $n = 1000$, $t = 0.5$

$$e^{-2 \cdot 1000 \cdot (0.5)^2} = e^{-500} \longrightarrow 0$$

$H \longrightarrow$ PAC learnable:

(1) $n_0(\epsilon, \delta)$

(2) learning algorithm that takes as input a set of training data points S, and outputs h_S^*.

for any data distribution D, as long as $|S| \geq n_0(\epsilon, \delta)$ then with probability $1-\delta$ we have:

$$L_D(h_S^*) \leq \min_{h \in H} L_D(h) + \epsilon$$

Example:

$\mathcal{H} = \{h_1, \dots, h_m\}$

(1) specify what $n_0(\epsilon, \delta)$ is.

(2) Algorithm: ERM $\mathcal{H}$:

$$h_S^* = \operatorname*{argmin}_{h \in \mathcal{H}} L_S(h)$$

what is $n_0(\epsilon, \delta)$, s.t.

w.p. $1-\delta$

$$L_D(h_S^*) \leq \min_{h \in \mathcal{H}} L_D(h) + \epsilon$$

Two questions:

(1) Given a fixed function h, how many data points, S, do we need such that

w.p. $1-\delta$

$$|L_S(h) - L_D(h)| < \varepsilon$$

$$L_S(h) = \frac{1}{|S|} \sum_{i=1}^{|S|} \mathbb{1}\{h(x_i) \neq y_i\}$$

$\Updownarrow$ equivalent

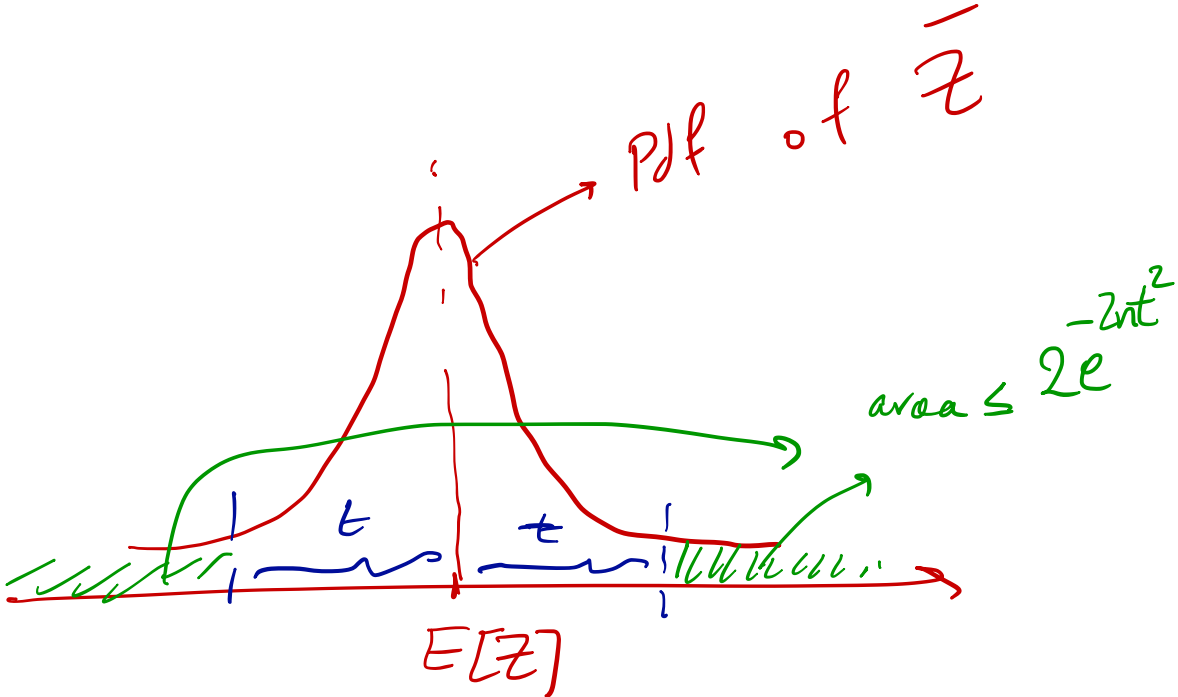
$$\Pr\{|L_S(h) - L_D(h)| \geq \varepsilon\} \leq \delta$$

Recall that Hoeffding's inequality was given as follows:

For any iid random variables $Z_1, Z_2, \dots, Z_n$, s.t. $Z_i \in [0,1]$,

we have

$$\Pr\left\{ \left| \overline{\frac{1}{n}\sum_{i=1}^{n} Z_i} - \overline{E[Z]} \right| > t \right\} \leq 2e^{-2nt^2}$$



$$L_S(h) = \frac{1}{n} \sum_{i=1}^{n} \underbrace{\mathbb{1}\{h(x_i) \neq y_i\}}_{Z_i}$$

$(|S| = n)$

$$L_D(h) = \underbrace{\mathop{E}_{(x,y)\sim D}\left[\mathbb{1}\{h(x) \neq y\}\right]}_{E[Z]}$$

$$L_S(h) = \frac{1}{n} \sum_{i=1}^{n} Z_i = \bar{Z}$$

$$Z_i = \begin{cases} 1 & \text{if } h(x_i) \neq y_i \\ 0 & \text{o.w.} \end{cases} \rightarrow Z_i \in [0,1]$$

$$E[Z_i] = \mathop{E}_{(x_i,y_i)\sim D}\left[\mathbb{1}\{h(x_i) \neq y_i\}\right]$$

$$= L_D(h)$$

Hoeffding:

$$P_r\left\{\left|\underbrace{L_S(h)}_{\frac{1}{n}\sum Z_i} - \underbrace{L_D(h)}_{E[Z]}\right| > \epsilon\right\} \quad \text{for any } \epsilon > 0 \qquad (1)$$

$$\leq 2e^{-2n\epsilon^2}$$

Recall that we're looking for the smallest n s.t. (2)

$$P_r\{|L_S(h) - L_D(h)| > \epsilon\} \leq \delta$$

using (1), and in order to guarantee (2), we can let:

$$2e^{-2n\epsilon^2} \leq \delta \qquad \left(\log\frac{2}{\delta} = -\log\frac{\delta}{2}\right)$$

$$\Rightarrow \boxed{n \geq \frac{1}{2\epsilon^2}\log\frac{2}{\delta}}$$

So the final statement is:

Given any ϵ, δ, as long as the number of training data points is larger than $n_1 = \frac{1}{2\epsilon^2} \log \frac{2}{\delta}$, we have

$$\Pr_{(x,y)\sim D}\left\{ |L_S(h) - L_D(h)| > \epsilon \right\} \leq \delta.$$

(for a fixed function h)

(2) Let's now assume that we have m functions $h_1, h_2, \ldots, h_m$. What is the smallest value $n_0(\epsilon, \delta)$ such that

with probability $1-\delta$: (3)

$$\forall\, i \in \{1, \ldots, m\} \;:\; |L_S(h_i) - L_D(h_i)| < \epsilon .$$

To answer this question, we're going to write an equivalent formulation of relation (3):

Let A_i be the event that

$$|L_S(h_i) - L_D(h_i)| > \epsilon .$$

Then (3) is equivalent to:

$$Pr\{A_1 \cup A_2 \cup A_3 \cdots \cup A_m\} \leq \delta . \quad (4)$$

Remember that we are searching for the smallest value of n such that (4) holds.

$$Pr\{A_1 \cup A_2 \cup \cdots \cup A_m\} \leq \delta$$

To answer this, we're going to use the Union bound:

$$Pr\{A \cup B\} \leq Pr\{A\} + Pr\{B\}$$

$$Pr\{A_1 \cup A_2 \cup \cdots \cup A_m\} \leq Pr\{A_1\} + Pr\{A_2\} + \cdots + Pr\{A_m\}.$$

---

bad event #i : A_i :

$$|L_S(h_i) - L_D(h_i)| > \epsilon$$

We'd like to make sure that none of the bad events, A_i, would take place.

So in order to guarantee

$$Pr\{A_1 \cup A_2 \cup \cdots \cup A_m\} \leq \delta \quad (5)$$

it is sufficient to guarantee that

$$Pr\{A_1\} + Pr\{A_2\} + \cdots + Pr\{A_m\} \leq \delta \quad (6)$$

Note that if (6) holds, then (5) would also hold as a result of the union bound.

Now, to guarantee (6), it is sufficient to choose n large enough s.t. for every h_i

We have

$$Pr\{A_i\} \leq \frac{\delta}{m} \qquad (7)$$

$$\Downarrow$$

$$Pr\{A_1\} + Pr\{A_2\} + \cdots + Pr\{A_m\} \leq \delta$$

$$\Downarrow$$

$$Pr\{A_1 \cup A_2 \cup \cdots \cup A_m\} \leq \delta$$

Now given our answer to question 1, in order to guarantee that $Pr\{A_i\} \leq \frac{\delta}{m}$ we need to

Choose:

if $n \geq n_1(\epsilon, \frac{\delta}{m}) = \underbrace{\frac{1}{2\epsilon^2} \log \frac{2m}{\delta}}_{n_0(\epsilon,\delta)}$

then

$$Pr\{A_i\}$$

$$= Pr\{ |L_S(h_i) - L_D(h_i)| > \epsilon \} \leq \frac{\delta}{m}$$

$\left(n_0(\epsilon,\delta) = n_1(\epsilon, \frac{\delta}{m}) \right)$

Hence,

Statement: If the number of training data points, $|S|$, is larger than $n_0(\epsilon,\delta) = \frac{1}{2\epsilon^2} \log \frac{2m}{\delta}$, then

with probability $1-\delta$ we have

$\forall i \in \{1,\dots,m\}$: $|L_S(h_i) - L_D(h_i)| < \epsilon$.

What we've shown is that we need

$n_0(\epsilon, \delta) = \frac{1}{2\epsilon^2} \log \frac{2m}{\delta}$ data points to guarantee that for all the m functions $h \in \mathcal{H}$ the value of $L_S(h)$ is close to the value of $L_D(h)$:

w.p $1-\delta$:

$$\forall h \in \mathcal{H} : |L_S(h) - L_D(h)| < \epsilon.$$

Now, Let h^*_S be the minimizer of ERM over the class $\mathcal{H}$:

$$h^*_S : \operatorname{argmin}_{h \in \mathcal{H}} L_S(h)$$

($L_S(h^*_S)$ is the smallest among $\mathcal{H}$)

Let $h^*_{true} : \operatorname{argmin}_{h \in \mathcal{H}} L_D(h)$

We will show that if the number of training data points is at least $n_0(\epsilon, \delta)$ then with $1-\delta$ we have

$$L_D(h^*_S) - \min_{h \in \mathcal{H}} L_D(h) \leq 2\epsilon.$$

Hence, $\mathcal{H}$ is PAC-learnable with $n_0(\epsilon, \delta)$ data points.

Let's see why (8) holds.

$$L_D(h^*_S) - L_D(h^*_{true})$$

$$= \underbrace{L_D(h^*_S) - L_S(h^*_S)}_{\leq \epsilon} + \underbrace{L_S(h^*_S) - L_S(h_{true})}_{\leq 0} + L_S(h^*_{true}) - L_D(h^*_{true})$$

(h^*_S is the minimizer of $L_S(\cdot)$)

$$\leq \epsilon + 0 + \epsilon \leq 2\epsilon.$$

Hence, if the number of training data points is at least $n_0(\epsilon, \delta) = \frac{1}{2\epsilon^2} \log \frac{2m}{\delta}$

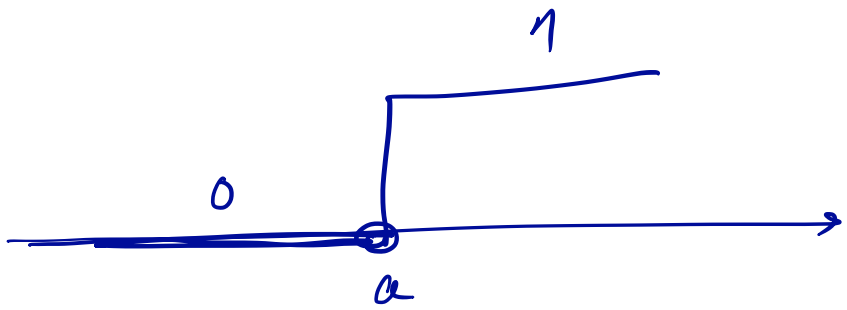
then w.p. $1-\delta$ we have

$$0 \leq L_D(h_S^*) - \min_{h \in \mathcal{H}} L_D(h) \leq 2\epsilon.$$

h_S^*: minimizer of ERM over $\mathcal{H}$.

When $\mathcal{H} = \{h_1, \dots, h_m\}$.

$h_a(x) =$



$$H = \{ h_a(x) \;,\; a \in (-\infty, \infty) \}$$

infinitely many functions

One important consequece of PAC-learnebility is that it's sufficient to work with a finite data set and we don't lose any thing in terms of generalization

Lecture 24:

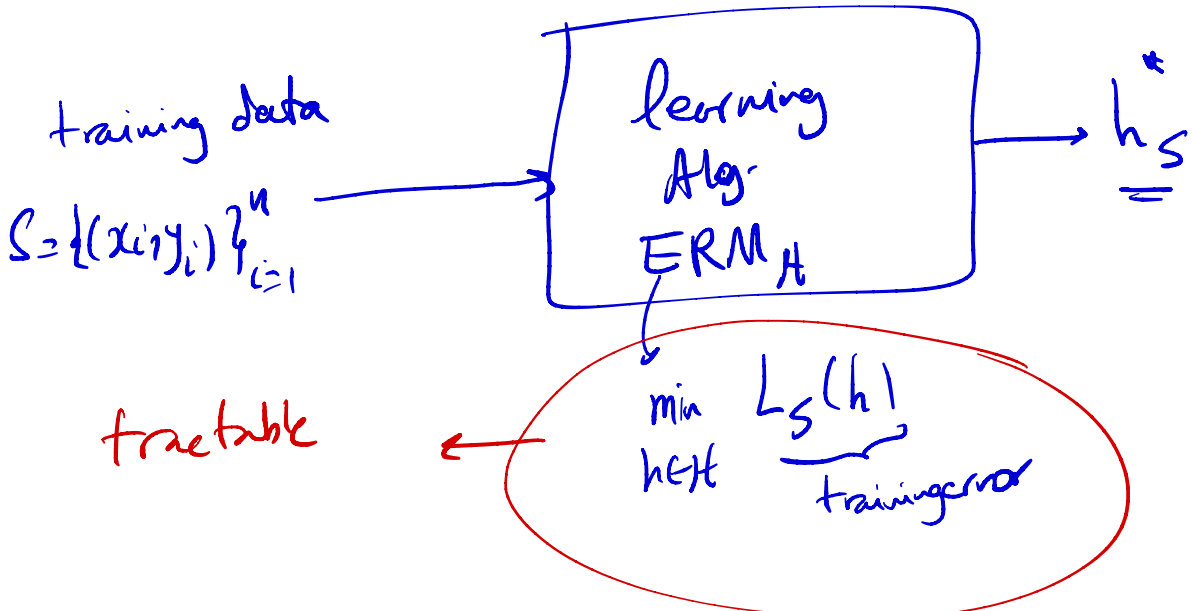
$$\mathcal{H} = \{ h_1, \ldots, h_m \}$$

PAC learning:

$$n \geq n_0(\epsilon, \delta) = \frac{2}{\epsilon^2} \log \frac{2m}{\delta}$$

Thm: For any data distribution, with probability $1-\delta$, the outcome of $ERM_{\mathcal{H}}$, which we denote by h_S^*, satisfies:

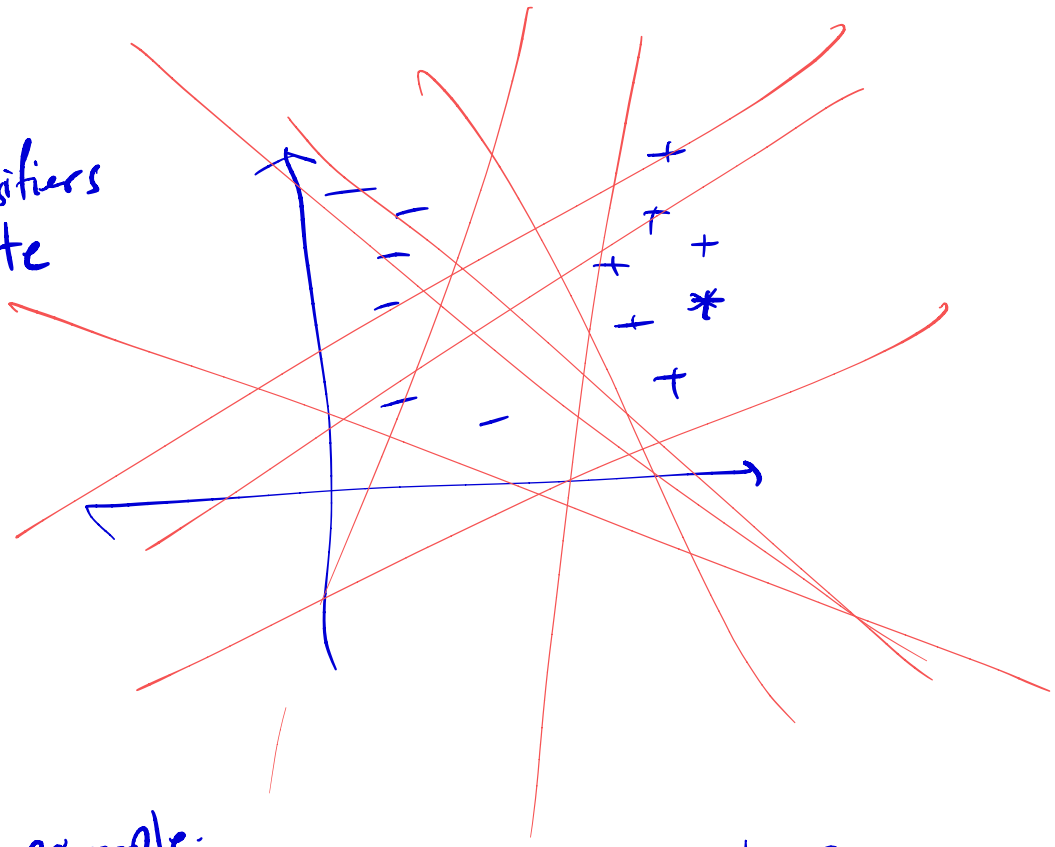
$$\min_{h \in \mathcal{H}} L_D(h) \leq L_D(h_S^*) \leq \min_{h \in \mathcal{H}} L_D(h) + \epsilon$$



So far, we've show that classes H with finitely many functions are PAC learnable.

But ~~in~~ in practice, all the function classes that we use are not finite.

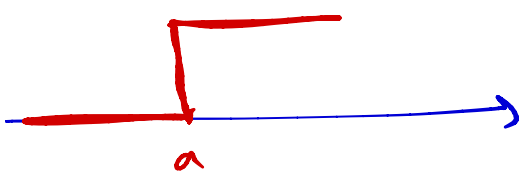
examples:
linear classifiers
~~inf~~ infinite



another example:
Consider the following simple class

$$H = \{ h_a(x), \; a \in \mathbb{R} \}$$

$$h_a(x) = \begin{cases} 1 & \text{if } x > a \\ 0 & \text{if } x \leq a \end{cases}$$



The fundamental Theorem of Learning Theory:

For any function class $\mathcal{H}$, if the number of training data points is larger than $n_0(\epsilon, \delta) = 4 \cdot \frac{\log \frac{1}{\delta} + (VC\text{-}dim(\mathcal{H}))}{\epsilon^2}$, then

(VC-dim(H)) → quantifies the "complexity" of H

$\mathcal{H}$ is PAC-learnable: For any distribution D on data, with $1-\delta$:

$$L_D(h_S^*) \leq \underbrace{\min_{h \in \mathcal{H}} L_D(h)}_{\text{intractable}} + \epsilon$$

where $h_S^* = \underbrace{\operatorname{argmin}_{h \in \mathcal{H}} L_S(h)}_{\text{tractable}}$.

As long as we have sufficiently many data points, we'll be fine with solving the "tractable" $ERM_{\mathcal{H}}$ problem and we lose at most a small value ϵ with respect to the best attainable accuracy.

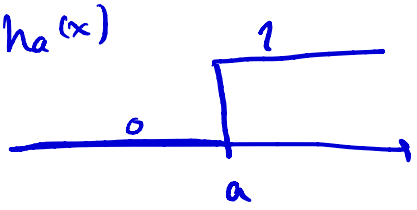
$\Rightarrow$ Machine learning is possible as long as we have sufficient data!

VC - Dimension :

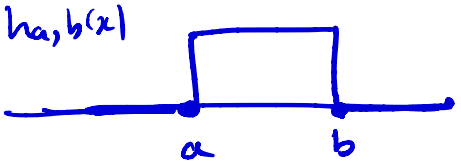
Vapnik Chernovskii

Example: Consider the following function classes:

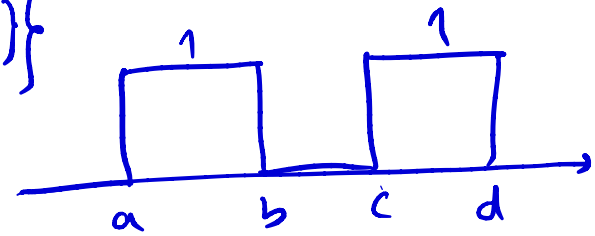
$H_1 = \{ h_a(x), a \in \mathbb{R} \} \rightarrow$



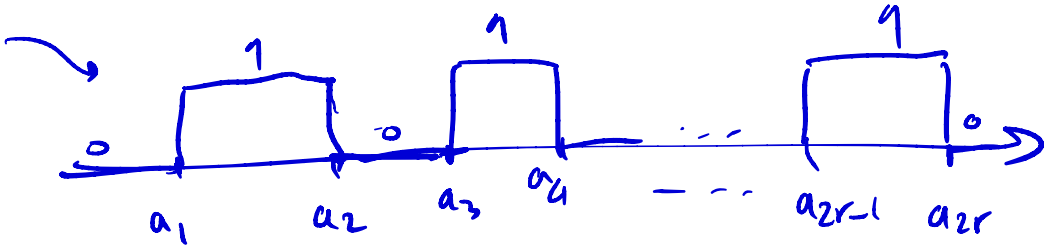
$H_2 = \{ h_{a,b}(x), a, b \in \mathbb{R} \} \rightarrow$



$H_3 = \{ h_{abcd}(x), a, b, c, d \in \mathbb{R} \}$



$H_4 = \{ h_{a_1, \ldots, a_{2r}}(x), a_1, \ldots, a_{2r} \in \mathbb{R} \}$



Intuitively, in terms of complexity we should have

$$H_1 < H_2 < H_3 < H_4$$

VC-dim is a precise mathematical notion to quantify "complexity".

In order to define VC-dim we need two other definitions (restriction, shattering).

Definition (restriction):

Let H be a class of functions from a domain X to $\{0,1\}$ ($\forall h \in H : h: X \longrightarrow \{0,1\}$).

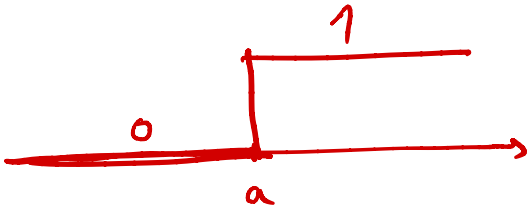
Let $C = \{x_1, \dots, x_k\} \subseteq X$. Then, the restriction of H to C is the set of all the k-tuples of the following form:

$$H_C = \{ \underbrace{(h(x_1), h(x_2), \dots, h(x_k))}_{k\text{-tuple}} ; h \in H \}$$

Example:

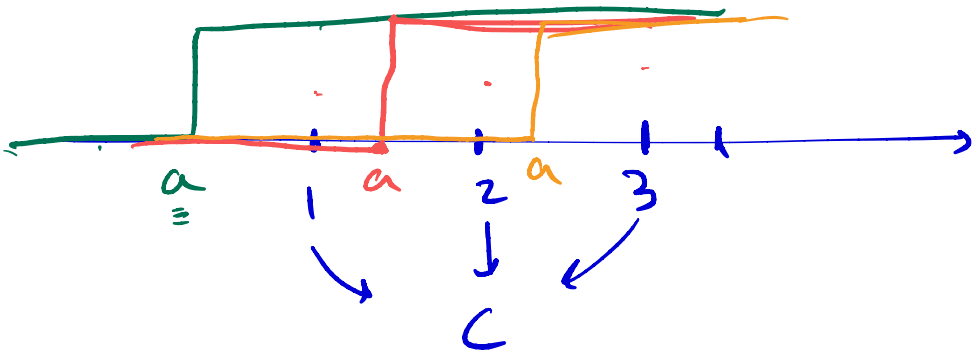
$$H = \{ h_a(x) ; a \in \mathbb{R} \}$$

$h_a(x)$



Let $C = \{1, 2, 3\}$. What is H_C?

To find H_C we need to go over all the functions $h \in H$.



if $a \leq 1 \longrightarrow (h_a(1), h_a(2), h_a(3))$

$= (1, 1, 1)$

if $1 < a \leq 2 \longrightarrow (h_a(1), h_a(2), h_a(3))$

$= (0, 1, 1)$

if $2 < a \leq 3 \quad = (h_a(1), h_a(2), h_a(3))$

$= (0, 0, 1)$

if $a > 3 \longrightarrow (0, 0, 0)$

$$\mathcal{H}_C = \{(1,1,1), (0,1,1), (0,0,1), (0,0,0)\}.$$

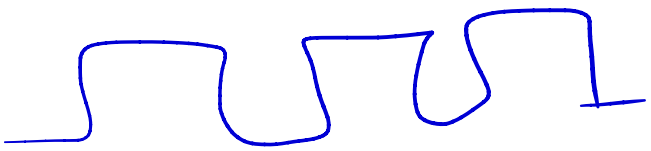
We note that if $C = \{x_1, \ldots, x_k\}$

then $|\mathcal{H}_C| \leq \underbrace{2^k}$

↳ recall that there are 2^k binary k-tuples.

$$1 \leq |\mathcal{H}_C| \leq 2^k$$

complexity ↑



Definition (shattering): We say that a function class $\mathcal{H}$ shatters a set C of size k if $|\mathcal{H}_C| = 2^k$.

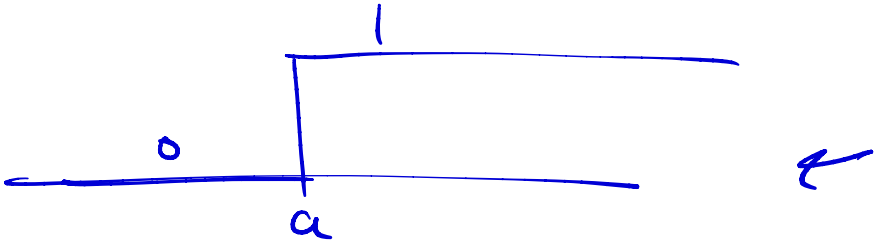
Definition (VC-dimension):

The VC-dimension of a function class $\mathcal{H}$ is the largest k such that there exists a set C of size k which is shattered by $\mathcal{H}$.

Example:

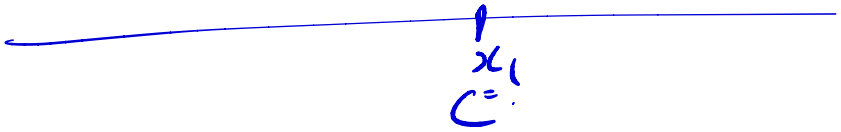
$$\mathcal{H} = \{ h_a(x) , \quad a \in \mathbb{R} \}$$

$h_a =$



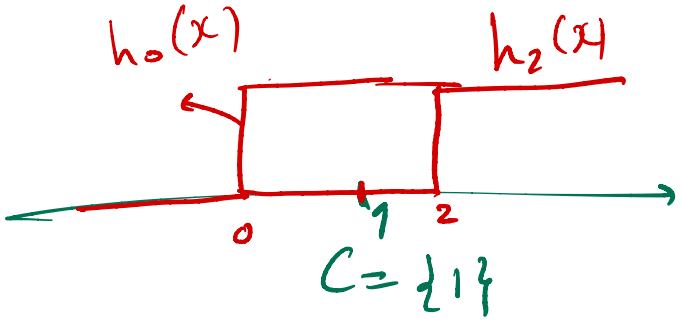
In order to find the VC-dim($\mathcal{H}$) we need to find the largest k for which $\mathcal{H}$ shatters a set C of size k:

$k=1 \longrightarrow$ Is there a set C, of size k, that is shattered by $\mathcal{H}$?



$|C|=1$, $H_C = \{ (h(x_1)) \; ; \; h \in H \}$

$C = \{x_1\}$ $= \{ 0, 1 \}$

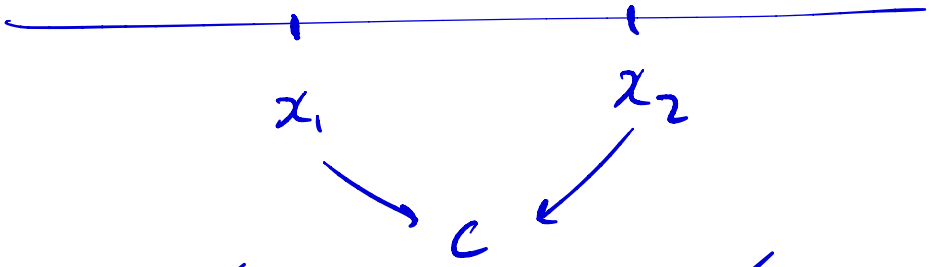


$\begin{cases} h_0(1) = 1 \\ h_2(1) = 0 \end{cases} \rightarrow H_C = \{0, 1\}$

$\Rightarrow C = \{1\}$ is shattered by H.

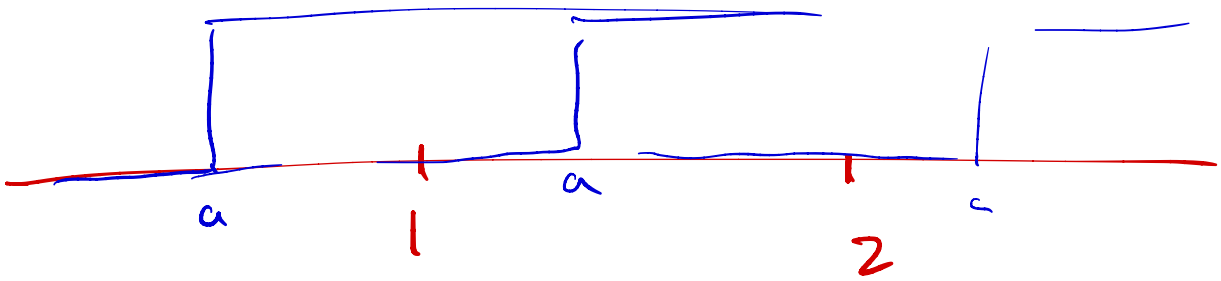
Now consider $k = 2$.

Is there a set C of size 2 that is shattered by H?



$H_c \stackrel{?}{=} \{(0,0), (0,1), (1,0), (1,1)\}$

let's take $x_1 = 1$ and $x_2 = 2$



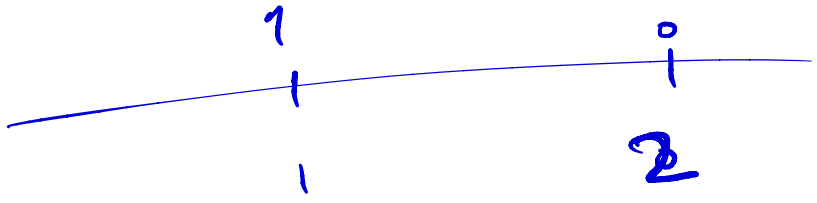
if $a > 2 \Rightarrow (h_a(1), h_a(2)) = (0,0)$

if $a < 1 \Rightarrow (h_a(1), h_a(2)) = (1,1)$

if $1 < a \leq 2 \Rightarrow (h_a(1), h_a(2)) = (0,1)$

is there any function that generates $(1,0)$?

→ No since all the functions in H are increasing.



We can show that for any set $C = \{x_1, x_2\}$, $x_1 < x_2$, the 2-tuple $(1,0)$ can not be generated by any $h \in H$.

$\Rightarrow \nexists$ C of size 2 which is shattered by H.

$k = 1$ ✓

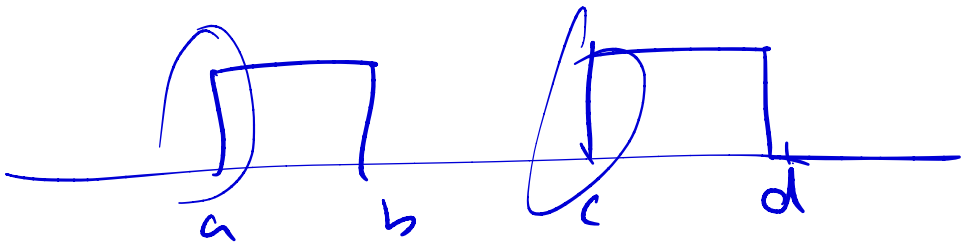
$k = 2$ ✗

$k = 3$ ✗

$\rightarrow$ $V_C\text{-dim}(H) = 1$.

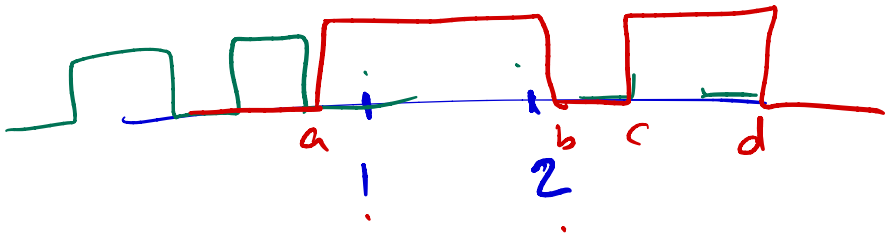
Example:

$$\mathcal{H} = \{ h_{a,b,c,d}(x) : a,b,c,d \in \mathbb{R} \}$$



$k=1 \rightarrow$ ✓

$k=2 \rightarrow$ is there a set of size 2 that's shattered by $\mathcal{H}$?

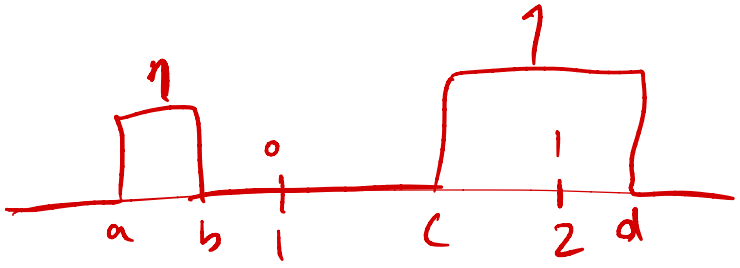


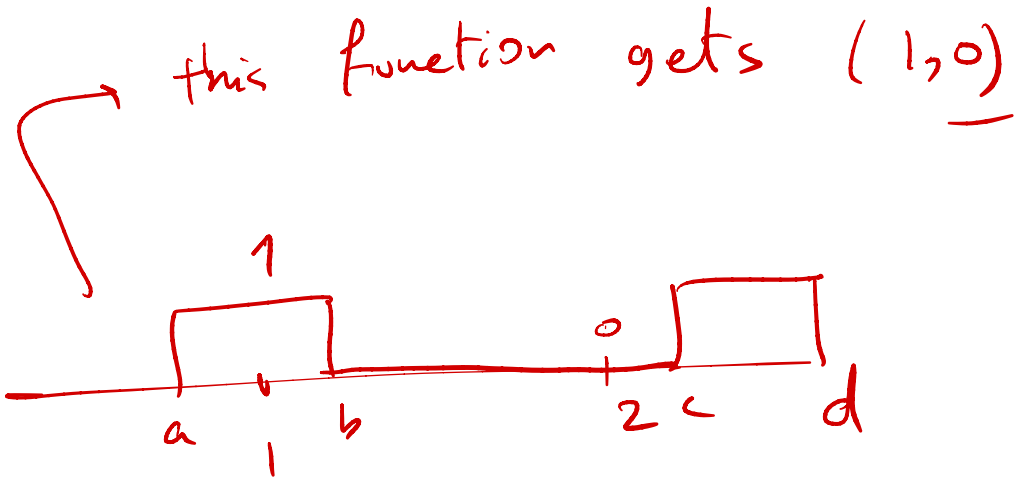
$H_C = \{(0,0), (0,1), (1,0), (1,1)\}$

if $d < 1$ then we get $(0,0)$

if $a < 1 < 2 < b$ then we get $(1,1)$

$a, b < 1$ and $c < 2 < 3 \rightarrow (0,1)$





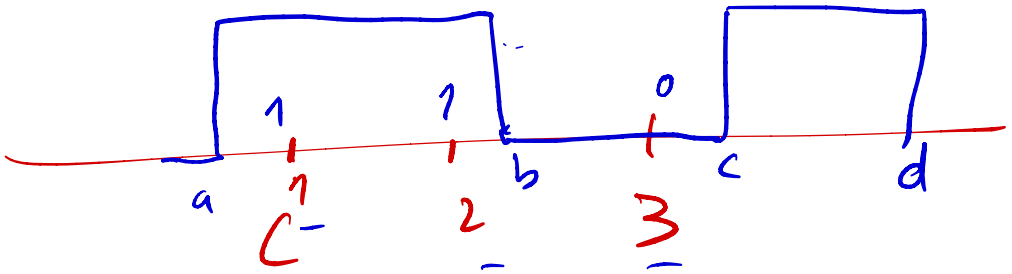
It shatters the set $C = \{1, 2\}$.

$k = 3$ ✓

$C = \{1, 2, 3\}$ will be shattered by H.

$H_C = \{(0,0,0), (0,0,1), (0,1,0), (0,1,1)$
$(1,0,0), (1,0,1), (1,1,0), (1,1,1)\}$

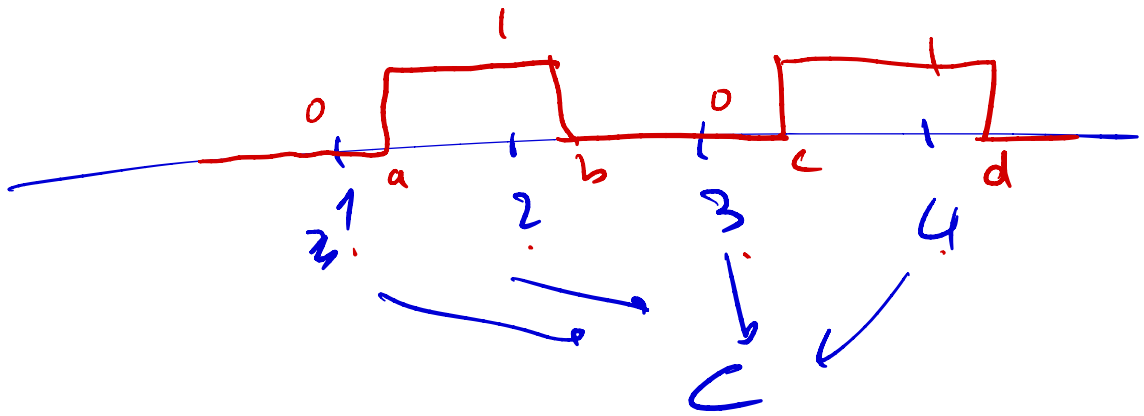
e.g. $(1, 1, 0)$



$k = 4$, the set $C = \{1, 2, 3, 4\}$ is shattered by $\mathcal{H}$.

$\mathcal{H}_C = \{$ all the 16 ~~$\in$~~ binary 4-tuples $\}$

e.g. $(0, 1, 0, 1)$



It shatters a set C of size 1 ✓

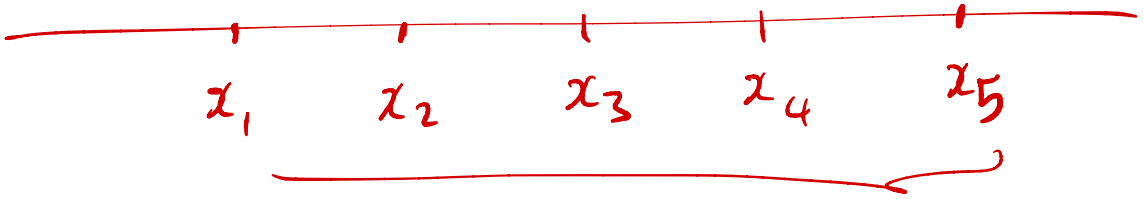
2 ✓

3 ✓

4 ✓

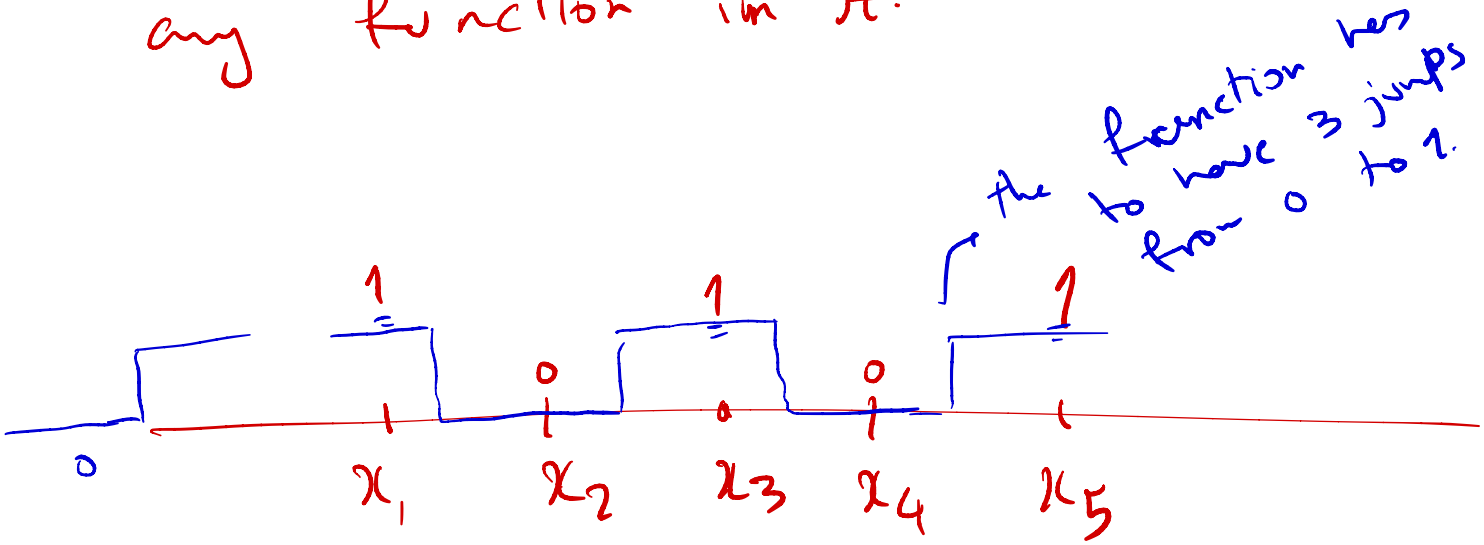
5 ? ✗

$k = 5$ does $\mathcal{H}$ shatter a set C of size 5?



C

We claim that the 5-tuple $(1,0,1,0,1)$ ~~is~~ can not be ~~to~~ generated by any function in $\mathcal{H}$.



$\Rightarrow$ VC-dim $(\mathcal{H}) = 4$.