4/2 수업

M2608.001300
Machine Learning Fundamentals & Applications

[3: Linear Classification and Regression]

Electrical and Computer Engineering
Seoul National University

Outline

Readings

- *Learning from Data* by Abu-Mostafa, Magdon-Ismail, and Lin
  - Chapter 3: The Linear Model (Sections 3.1 & 3.2)

references on linear algebra (optional):

- *Linear Algebra (MIT 18.06)* Link
  - a well-known OCW (undergraduate level)

- *Introduction to Linear Dynamical Systems (Stanford EE263)* Link
  - Lecture 3: Linear Algebra Review
  - Lecture 5: Least-Squares

The linear model

- set of lines
- often a good first choice
  - small VC dimension (more on this later) : simple
  - $\Rightarrow$ generalize well from E_{in} to $\mathrm{E}_{\mathrm{out}}$

하이퍼 파라미터 같은 존재. (조절할수 있는것 적다)

Three important problems

1. classification (supervised learning)
2. regression (〃)
3. probability estimation (*aka* logistic regression)
   ↳ 0과 1사이 값 return

   - come with different but related algorithms

딥러닝

※ 3.은 1도, 2도 될수.
(연속적인 값을 return하는 3을 기준에 따라
자르면 1도 될수있고,
2로 사용할수도 있다.

Outline

The linear model for binary classification

- uses a hypothesis set of linear classifiers
  - each h has the form

$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^{\text{T}}\mathbf{x})$$

binary output

    - ▷ $\mathbf{w} \in \mathbb{R}^{d+1}$: weight column vector
    - ▷ d : input dimension
    - ▷ $x_0 = 1$: corresponds to bias w_0

- note
  - we use h and $\mathbf{w}$ interchangeably to refer to the hypothesis

w: 파라미터, weight
(learning 대상)

Recall: two basic criteria for learning

1. can we make sure $\mathrm{E}_{\mathrm{out}}(g) \approx \mathrm{E}_{\mathrm{in}}(g)$? ○
   - what we have learned in sample will generalize out of sample
2. can we make $\mathrm{E}_{\mathrm{in}}(g) \approx 0$? ○
   - what we have learned in sample is a good hypothesis

- will these two criteria be fulfilled by a linear model? ○

Computational learning theory

- a quick summary (more details will follow in later lectures):
  - VC dimension d_{VC} of linear model: only $d+1$
  - VC generalization bound: for any tolerance $\delta > 0$

$$\mathrm{E}_{\text{out}} \leq \mathrm{E}_{\text{in}} + \sqrt{\frac{8}{N} \ln \frac{4 m_{\mathcal{H}}(2N)}{\delta}} \tag{1}$$

with probability $\geq 1-\delta$

$m_{\mathcal{H}}$: hypothesis set의 크기
m : effectively 한 M

  - the bound on the growth function

$$m_{\mathcal{H}}(N) \leq N^{d_{\text{VC}}} + 1 \tag{2}$$

M무한대라도 m을 bound가능해 : learning 가능

First criterion: $\mathrm{E}_{\mathrm{out}} \approx \mathrm{E}_{\mathrm{in}}$

- based on the above: with high probability

$$\mathrm{E}_{\mathrm{out}}(g) = \mathrm{E}_{\mathrm{in}}(g) + O\left(\sqrt{\frac{d}{N}\ln N}\right) \tag{3}$$

- for sufficiently large N
  - $\mathrm{E}_{\mathrm{out}} \approx \mathrm{E}_{\mathrm{in}} \Rightarrow$ the first criterion: fulfilled
- side note: a recurring pattern
  - generalization error typically contains $\left(\frac{d_{vc}}{N}\right)$ term (*i.e.* effective DOF over # samples)

Second criterion: $\mathrm{E}_{\mathrm{in}} \approx 0$

- linearly separable data
  - there is always some hypothesis $\mathbf{w}^*$ with $\mathrm{E}_{\mathrm{in}}(\mathbf{w}^*) = 0$

  e.g. perceptron learning algorithm (PLA)

> 1. start with an arbitrary weight vector $\mathbf{w}(0)$
> 2. then, at every time step $t \geq 0$
>
> 2a. select *any* misclassified data point $(\mathbf{x}(t), y(t))$
>
> 2b. update $\mathbf{w}$:
>
> $$\mathbf{w}(t+1) = \mathbf{w}(t) + y(t)\mathbf{x}(t)$$

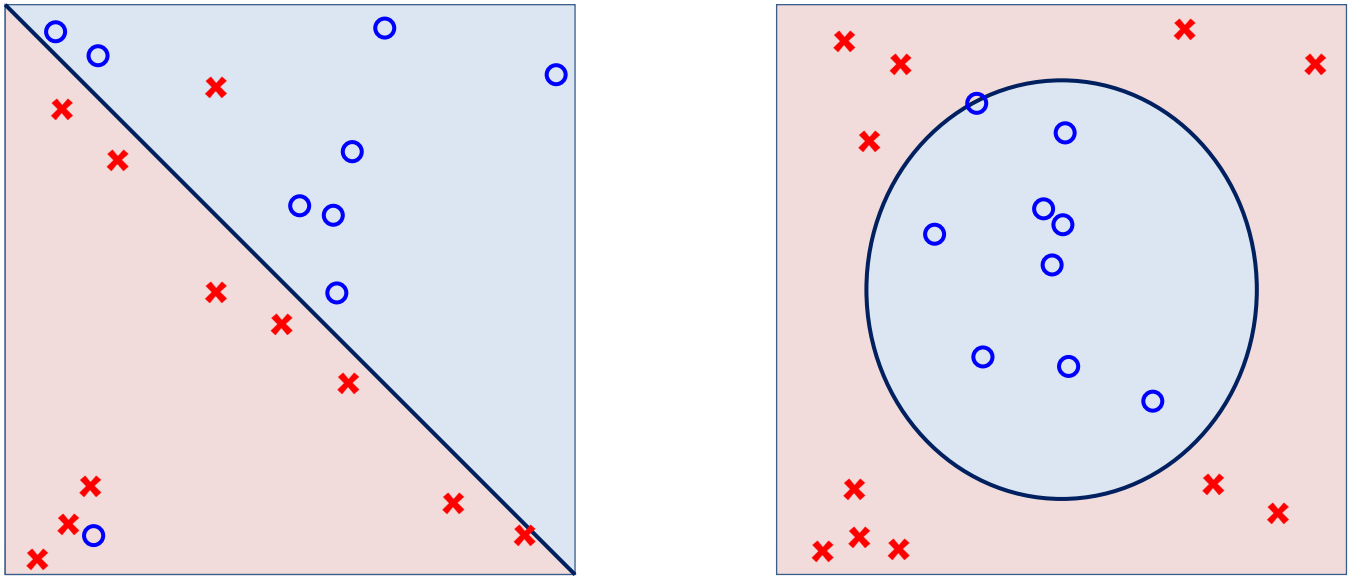
- linearly inseparable data
  - we can often make E_{in} very small

  e.g. pocket algorithm, support vector machine (SVM)

Outline

Data sets that are not linearly separable

과거시험 (a)와(b)



(a) a few noisy points

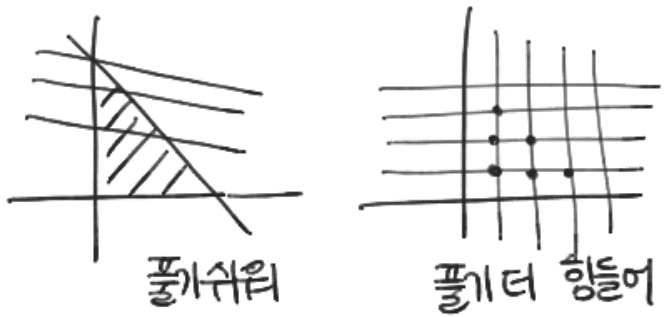
(b) nonlinearly separable

Figure 1: two types of linearly inseparable data

(a) data becomes linearly separable after removing just 2 examples (noisy examples or outliers)

(b) data can be separated by a circle rather than a line

Applicability of linear model



- Figure 1(a)
  - it seems appropriate to stick with a line
  - but to somehow tolerate noise
  - and output a hypothesis with a small E_{in} (not necessarily 0)

세로운 integer 문제가 floating point 문제보다 어려워

- Figure 1(b)
  - the linear model does not seem to be correct
  - may need a technique called nonlinear *transformation*

Hardness

- the situation in Figure 1(a): encountered very often in practice
  - a linear classifier seems appropriate
  - but data may not be linearly separable due to outliers/noise

- to find a hypothesis with the minimum E_{in}
  - need to solve combinatorial optimization

$$\min_{\mathbf{w}\in\mathbb{R}^{d+1}} \underbrace{\frac{1}{N}\sum [\![\, \mathrm{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n) \neq y_n \,]\!]}_{\mathrm{E}_{\mathrm{in}}(\mathbf{w})} \tag{4}$$

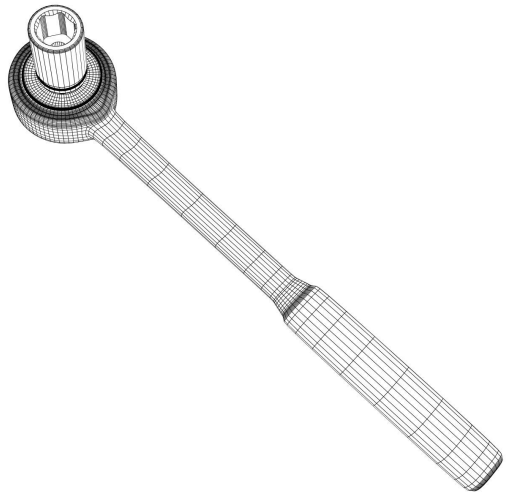
이산적(discrete) than continuous

- Eq. (4): difficult to solve (in general, NP-hard)
  - due to the discrete nature of both $\mathrm{sign}(\cdot)$ and $[\![\cdot]\!]$

$\Rightarrow$ approximately minimize E_{in}

Approximately minimizing E_{in}

- one approach:
  - extend PLA through a simple modification
  - *pocket algorithm* with ratchet (Gallant, 1990)



- in essence
  - the pocket algorithm keeps 'in its pocket' the best $\mathbf{w}$ encountered up to iteration t in PLA
  - at the end, the best $\mathbf{w}$ will be reported as the final hypothesis

The pocket algorithm

1: set the pocket weight vector $\hat{\mathbf{w}}$ to $\mathbf{w}(0)$ of PLA;
2: **for** $t = 0, \ldots, T-1$ **do**
3: run PLA for one update to obtain $\mathbf{w}(t+1)$;
4: evaluate $\mathrm{E}_{\mathrm{in}}(\mathbf{w}(t+1))$; /* time consuming */
5: **if** $\mathbf{w}(t+1)$ is better than $\hat{\mathbf{w}}$ in terms of E_{in}
6: **then** set $\hat{\mathbf{w}}$ to $\mathbf{w}(t+1)$;
7: return $\hat{\mathbf{w}}$;

모든 example에 대해 다 본다.

Comparison

in each iteration:

- PLA
  - only checks *some* examples using $\mathbf{w}(t)$ to identify $(\mathbf{x}(t), y(t))$
- pocket algorithm
  - needs an additional step (line 4)
  - evaluates all examples using $\mathbf{w}(t+1)$ to get $\mathrm{E}_{\mathrm{in}}(\mathbf{w}(t+1))$

  $\Rightarrow$ much slower than PLA

  $\Rightarrow$ no guarantee for how fast it can converge to a good E_{in}

Outline

Handwritten digit recognition

- data: sampled digits from the US Postal Service zip code DB
  - each image has 16×16 pixels

- goal: recognize each digit
  - non-trivial task (even human $\mathrm{E}_{\mathrm{out}}$ is about 2.5%)
  - common confusion: digits $\{4, 9\}$ and $\{2, 7\}$

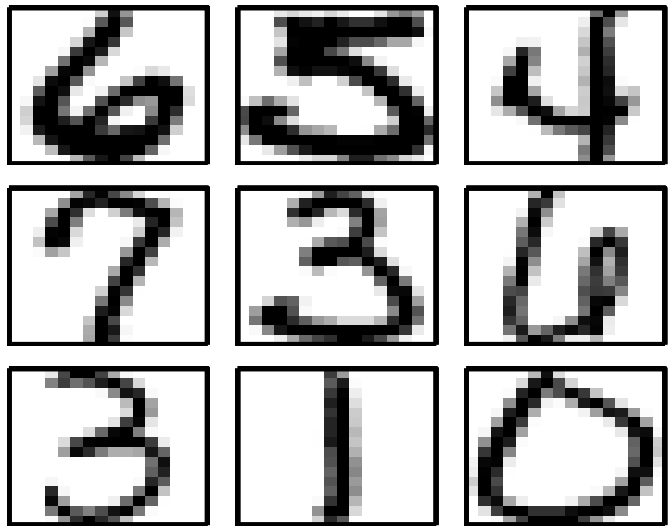


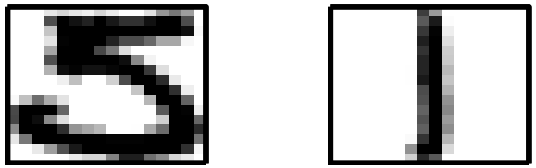
Figure 2: digit images

Solution components

1. decomposition from *multiclass* to *binary* classification
   - a commonly used approach for multiclass setup
2. feature extraction
   - a human approach to determining the digit of an image: to look at the shape (or other properties) of black pixels
   - rather than carrying all the info in 256 pixels, summarize the info contained in the image into a few features.
   - again a very common/essential approach in machine learning

Image features

- two important features here: intensity and symmetry
  - digit 5 usually occupies more black pixels than digit 1
  - digit 1 is symmetric while digit 5 is not
  - these two features will define the *feature space*



- definitions
  - asymmetry: the average absolute difference between an image and its flipped version
  - symmetry: the negation of asymmetry

Input representation

- in 'raw' input space $\mathbb{R}^{256+1}$: $\mathbf{x} = (x_0, x_1, x_2, \ldots, x_{256})$
  - linear model: $\mathbf{w} = (w_0, w_1, w_2, \ldots, w_{256})$
- in 'feature' input space $\mathbb{R}^{2+1}$: $\mathbf{x} = (x_0, x_1, x_2)$
  - linear model: $\mathbf{w} = (w_0, w_1, w_2)$
- introducing feature space (preprocessing 에서 차원 줄이는것 매우 중요)
  - reduces the number of dimensionality (*e.g.* from 256 to 2)

  $\Rightarrow$ allows us to better handle the curse of dimensionality
  - but can cause loss of information

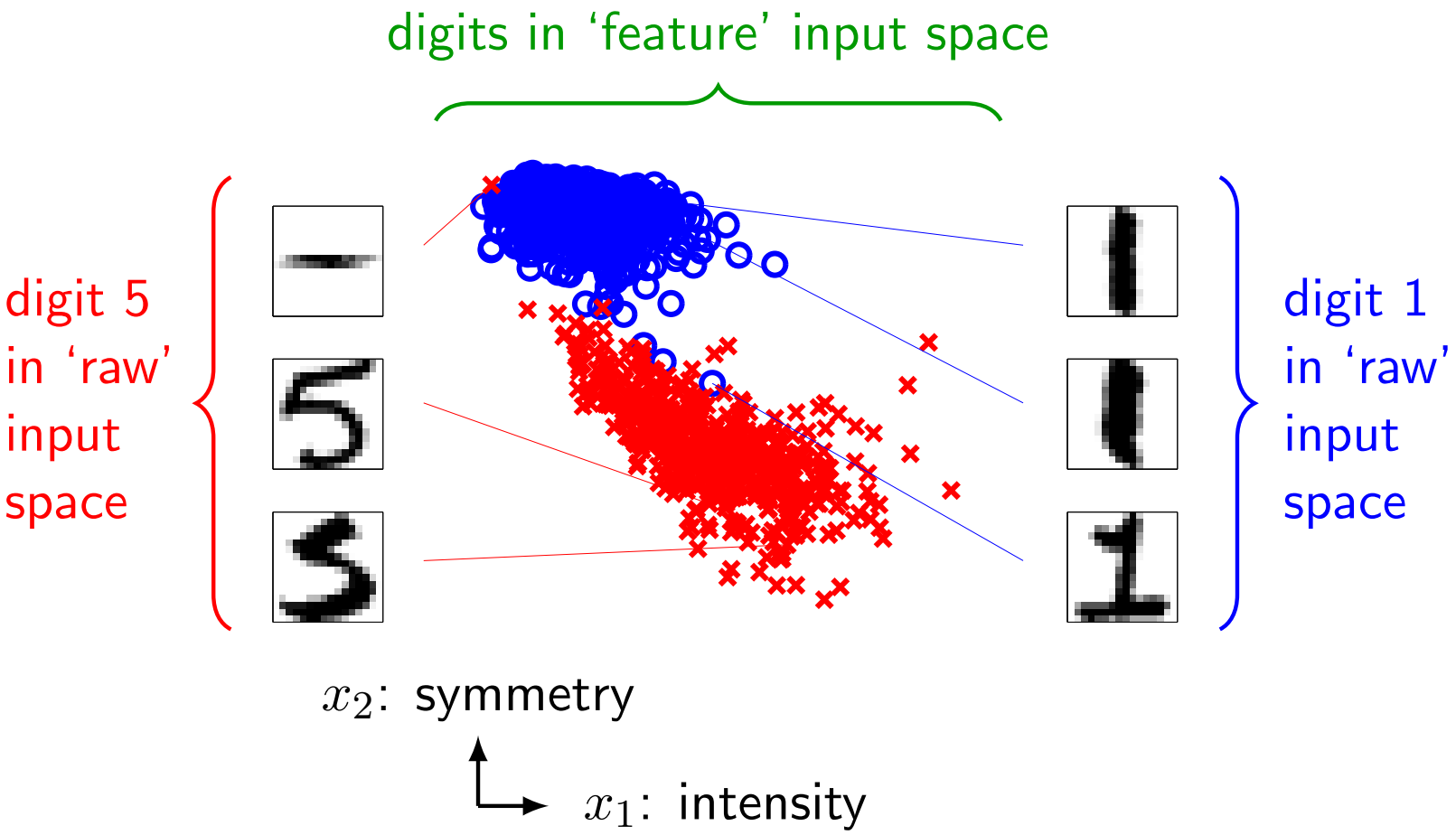
① 차원↑이면 고려해야하는 경우의수 너무많아 (2의 지수로 기하급수↑)

② 0 0.5 1 → 50%

1 1 → 25% ↓ 줄어 전체에서 우리가 아는부분 비율 기하급수↓

Example

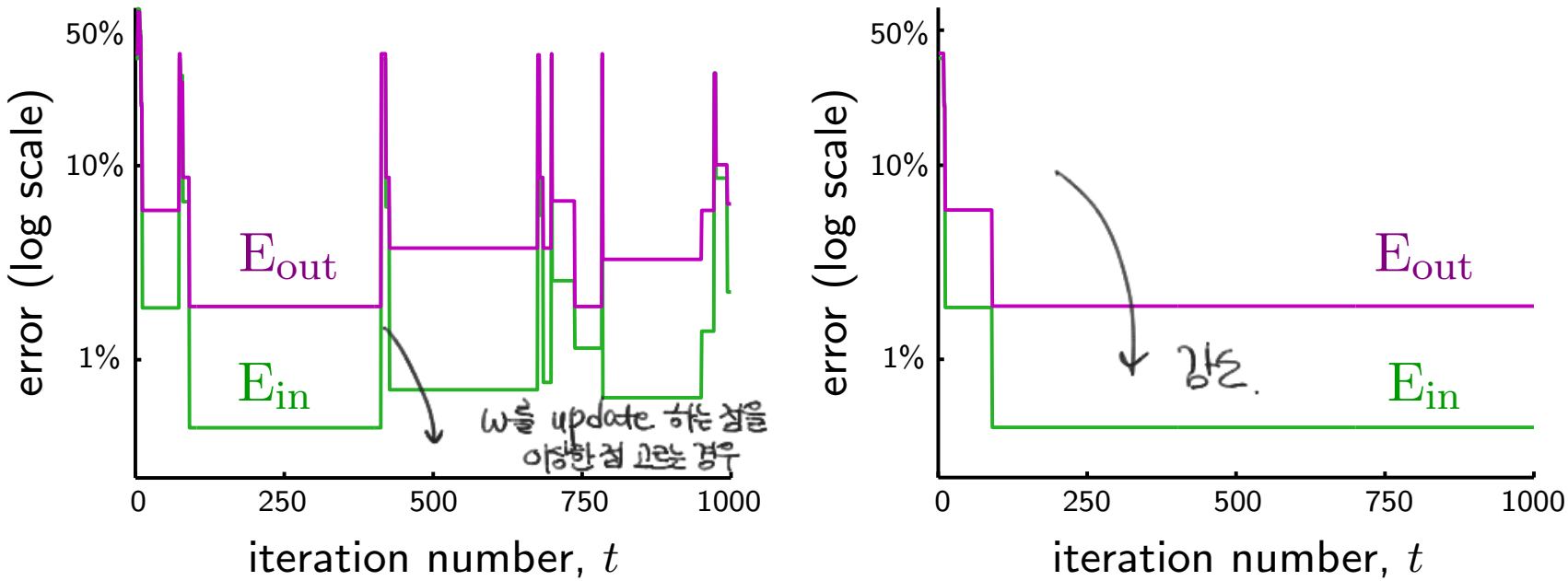
- a scatter plot for these intensity and symmetry features:
  - $\mathbf{x} = (x_0, x_1, x_2)$



Classification in (intensity, symmetry) space

- digits can be roughly separated by a line in the feature plane
  - but poorly written digits prevent a perfect line separation
  - *e.g.* the '5' depicted in the top-left corner
- we now run PLA & pocket algorithm
  - with data points represented in the feature space

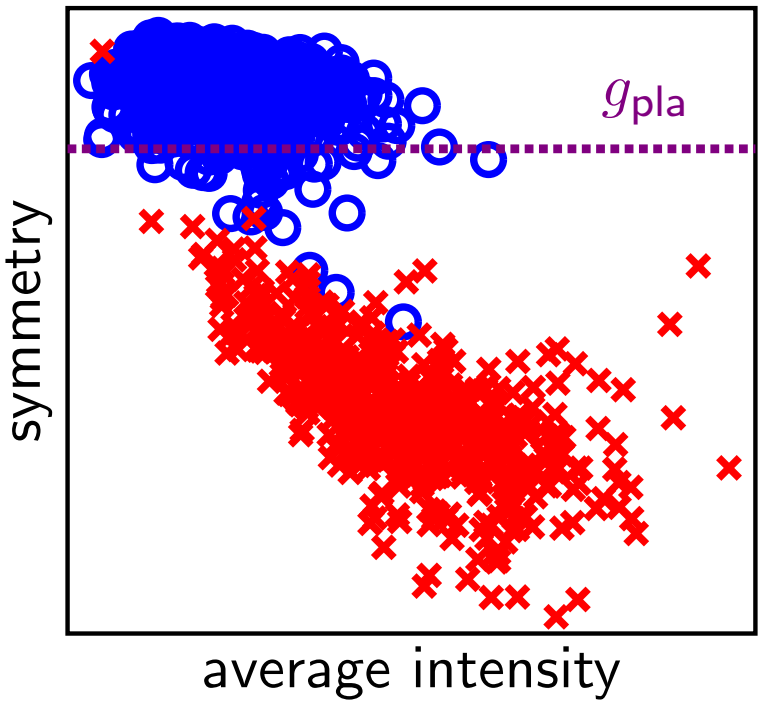
- error versus iteration number
  - PLA: $\mathrm{E_{in}} = 2.24\%$ and $\mathrm{E_{out}} = 6.37\%$ (quit at iteration 1000)
  - pocket: $\mathrm{E_{in}} = 0.45\%$ and $\mathrm{E_{out}} = 1.89\%$



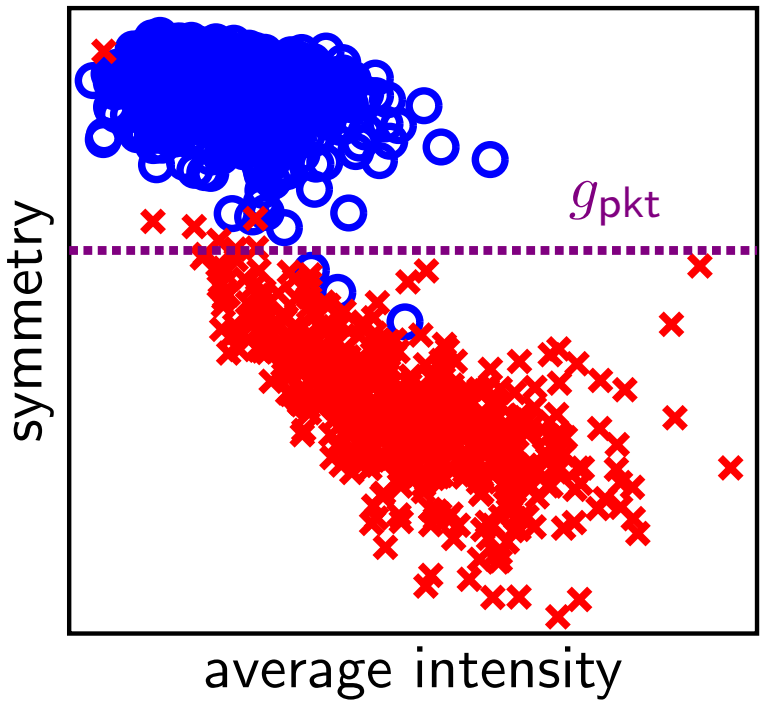
(a) PLA (b) pocket algorithm

Figure 3: comparison of two linear algorithms for separating digits 1 and 5

- learned hypothesis g



(a) PLA

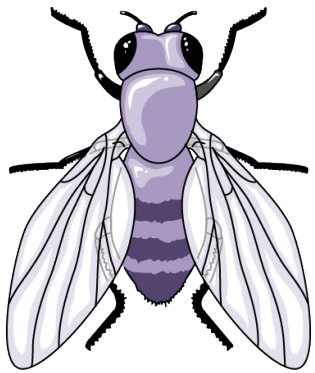


(b) pocket algorithm

MNIST dataset

Link

- 70k handwritten digits
  - 28×28 pixels
  - training set: 60k
  - test set: 10k
- best performing classifiers
  - linear: 7.6% error
  - Gaussian SVM: 1.4%
  - CNN: 0.23%
- "drosophila of machine learning"
  - Geoffrey Hinton



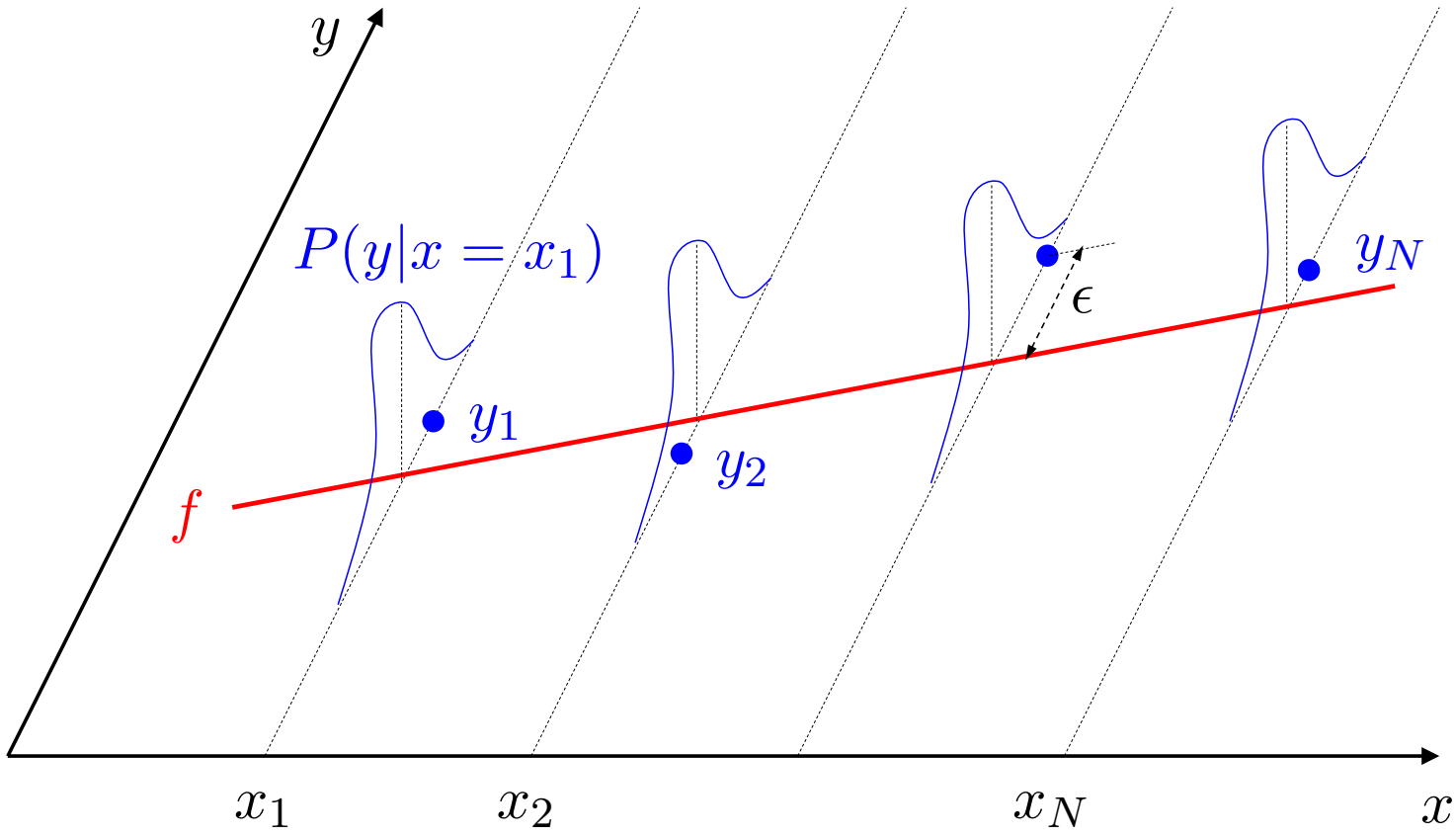
Outline

Regression

- a statistical method to study relationship between $\mathbf{x}$ and y
  - $\mathbf{x}$ covariate/predictor variable/independent variable/feature
  - y response/dependent variable
- training data $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$
  - noise ϵ is added to target: $y_n = f(\mathbf{x}_n) + \epsilon$

  $\Rightarrow$ $y \sim P(y|\mathbf{x})$ instead of $y = f(\mathbf{x})$
- our goal
  - find a model $g(\mathbf{x})$ that approximates y_n

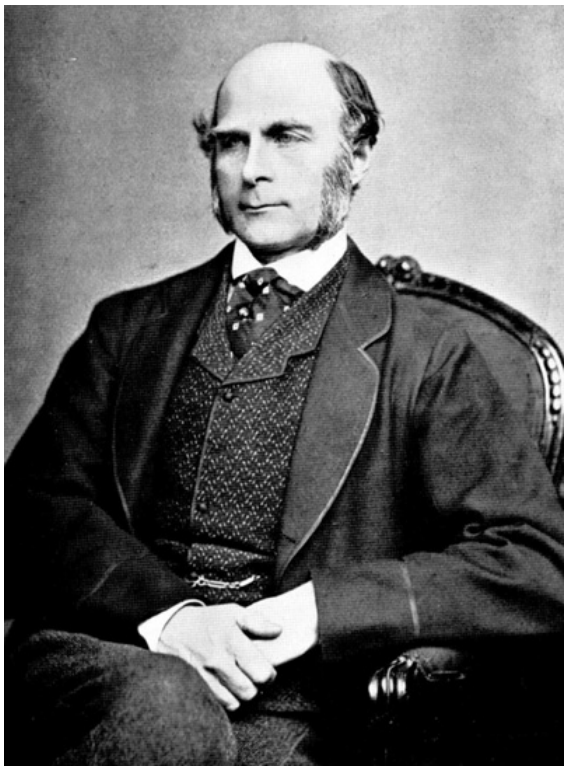
- illustration (simple linear regression):

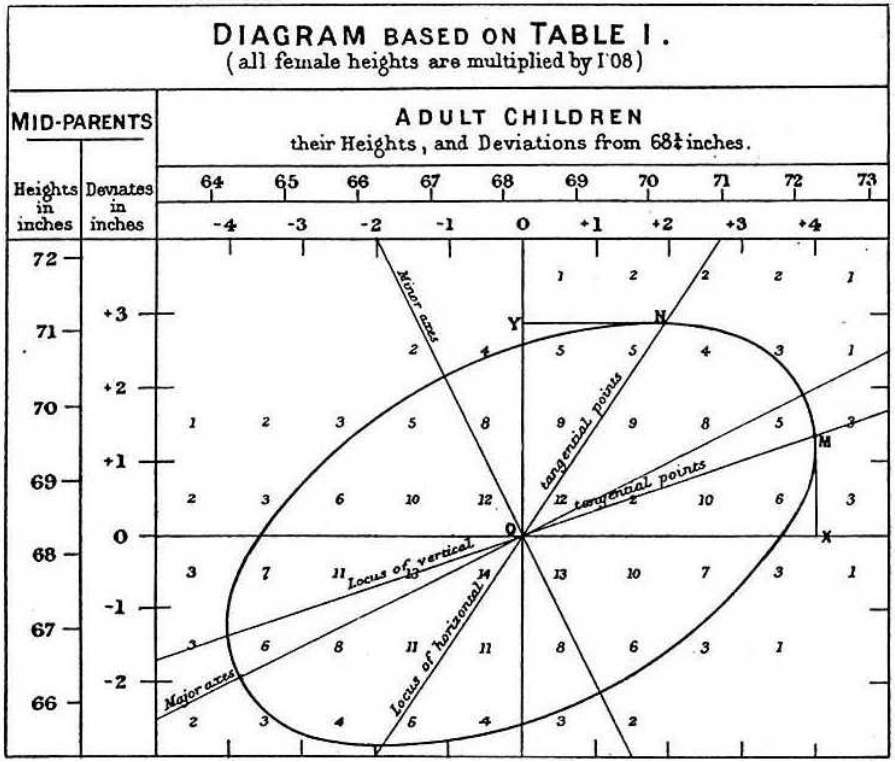


assumption: homoscedasticity (homogeneity of variance; the same finite variance for each value of x)

- etymology: Sir Francis Galton (1822-1911)
  - re (back)+gression (going): going back from data to formula
  - regression towards the mean: "tall and short men tend to have sons with heights closer to the mean"

data에서 fomular을 찾다.





Linear regression

$y \in$ continuous

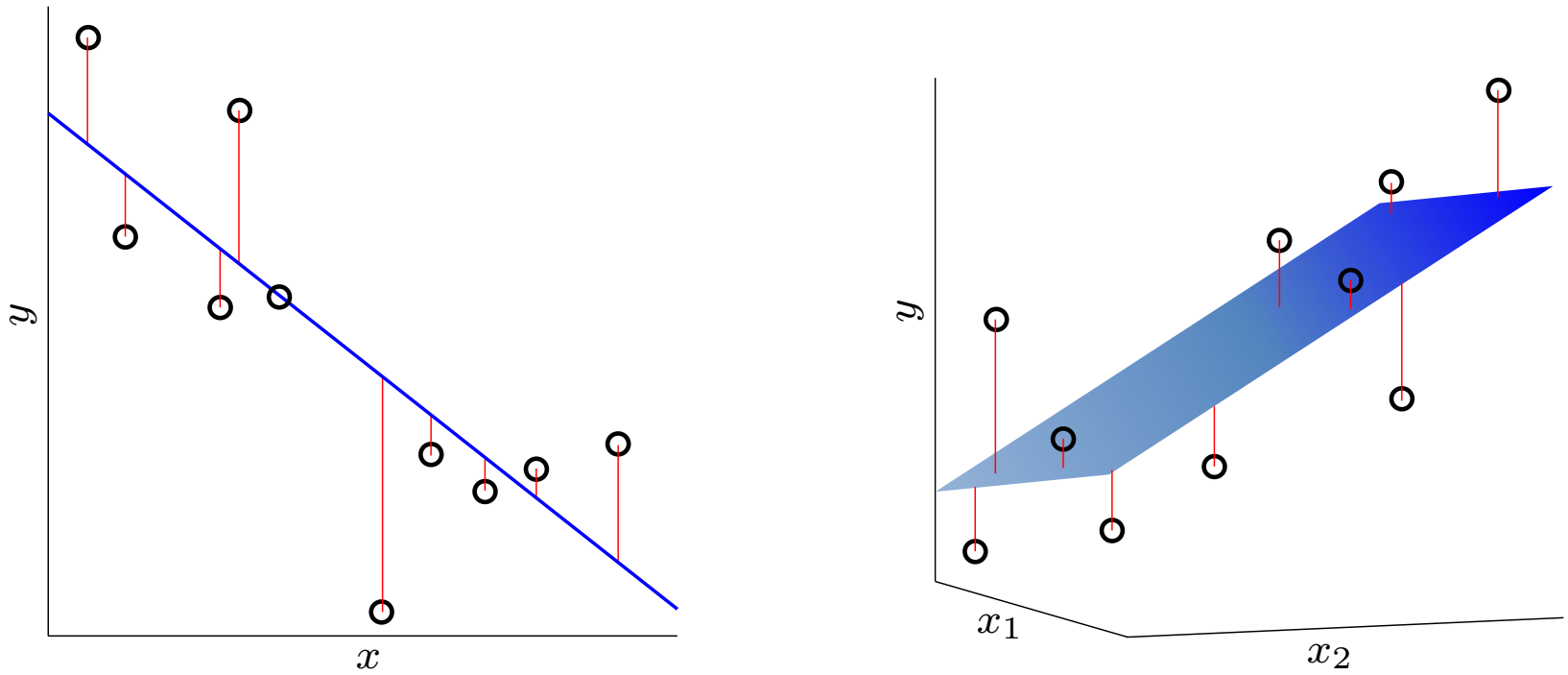
- popular linear model for predicting a quantitative response
  - applies to real-valued target functions (실수)
  - long history in statistics, social and behavioral sciences
  - "regression" means y is real-valued (inherited from statistics)
- simple versus multiple
  - $d = 1$: simple linear regression (one predictor)
  - $d \geq 2$: multiple linear regression (multiple predictors)
- we discuss "simple linear regression" from a learning perspective

$d=1$

Parameter estimation for linear regression

- least squares
  - ordinary least squares (OLS)
    - minimizes the sum of squared residuals
    - leads to a closed-form expression
  - generalized least squares, iteratively reweighted least squares
- maximum likelihood
  - ridge (Tikhonov regularization)/lasso regression: regularized
  - least absolute deviation regression
- other techniques
  - Bayesian linear regression, principal component regression

Linear regression in 1D and 2D



(a) 1D (line)

(b) 2D (hyperplane)

Figure 4: the solution hypothesis (in blue) of the linear regression algorithm in 1D and 2D. in OLS, the sum of squared error is minimized.

Credit approval example revisited

실수 값을 예측 하는것.

- consider a regression problem (rather than classification)
  - *e.g.* set a credit limit for each customer

- the bank uses historical records to build a data set $\mathcal{D}$
  - $\mathcal{D}$: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)$
  - $\mathbf{x}_n$: customer information
  - y_n: credit limit (set by human experts)

- y_n is now a real number
  - *e.g.* \$123,000 for Alice and \$57,500 for Bob

- the bank wants to use learning to find a hypothesis g
  - g replicates how human experts determine credit limits

- our target:
  - not a deterministic function $y = f(\mathbf{x})$
    - there is more than one human expert
    - each expert may not be perfectly consistent
  - instead: noisy target
    - formalized as a distribution of random variable y

Target distribution rather than a function

- regression label y_n comes from some distribution $P(y|\mathbf{x})$
  - instead of a deterministic function $f(\mathbf{x})$
- nonetheless, the nature of the problem is not changed
  - given: *unknown* distribution $P(\mathbf{x}, y)$ generating each $(\mathbf{x}_n, y_n)$
  - objective: find a hypothesis g that minimizes the error between $g(\mathbf{x})$ and y wrt the distribution P

Outline

The linear regression algorithm

- based on minimizing the squared error between $h(\mathbf{x})$ and y

$$\mathrm{E}_{\mathrm{out}}(h) = \mathbb{E}\left[(h(\mathbf{x}) - y)^2\right]$$

  - expected value: taken wrt joint pdf $P(\mathbf{x}, y)$

- goal
  - find a hypothesis h that achieves a smallest $E_{out}(h)$

- issue: $\mathrm{E}_{\mathrm{out}}$ cannot be computed

  $\because$ $P(\mathbf{x}, y)$ is unknown

- thus, we resort to in-sample error instead:

$$\mathrm{E}_{\mathrm{in}}(h) = \frac{1}{N}\sum_{n=1}^{N}(h(\mathbf{x}_n) - y_n)^2$$ (정의 OLS)

- in *linear* regression, h takes the form of
  - a linear combination of the components of $\mathbf{x}$:

$$h(\mathbf{x}) = \sum_{i=0}^{d} w_i x_i = \mathbf{w}^{\mathrm{T}}\mathbf{x}$$

  - $\mathbf{w}^{\mathrm{T}}\mathbf{x}$: also called "signal"

Matrix representation: data

- data matrix $X \in \mathbb{R}^{N \times (d+1)}$
  - rows: inputs $\mathbf{x}_n$ as row vectors
- target vector $\mathbf{y} \in \mathbb{R}^N$
  - components: target values y_n
- example: $d = 1$, $N = 4$

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ 1 & x_4 \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

Matrix representation: in-sample error

- in-sample error is a function of $\mathbf{w}$ and data $X, \mathbf{y}$:

$$\mathrm{E}_{\mathrm{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n - y_n)^2$$

$$= \frac{1}{N}\left\|\begin{bmatrix} \mathbf{x}_1^{\mathrm{T}}\mathbf{w} - y_1 \\ \mathbf{x}_2^{\mathrm{T}}\mathbf{w} - y_2 \\ \vdots \\ \mathbf{x}_N^{\mathrm{T}}\mathbf{w} - y_N \end{bmatrix}\right\|^2 \tag{5}$$

$$= \frac{1}{N}\| X\mathbf{w} - \mathbf{y} \|^2 \qquad (\because \|x\|^2 = x^{\mathrm{T}}x) \tag{6}$$

$$= \frac{1}{N}(\mathbf{w}^{\mathrm{T}}X^{\mathrm{T}}X\mathbf{w} - 2\mathbf{w}^{\mathrm{T}}X^{\mathrm{T}}\mathbf{y} + \mathbf{y}^{\mathrm{T}}\mathbf{y}) \tag{7}$$

- $\|\cdot\|$: Euclidean norm of a vector
- scalar $\mathbf{y}^{\mathrm{T}}X\mathbf{w} = (\mathbf{w}^{\mathrm{T}}X^{\mathrm{T}}\mathbf{y})^{\mathrm{T}} = \mathbf{w}^{\mathrm{T}}X^{\mathrm{T}}\mathbf{y}$

- example: $d = 1,\ N = 4$

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ 1 & x_4 \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

$$\mathrm{E_{in}}(\mathbf{w}) = \frac{1}{4}\sum_{i=1}^{4}(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n - y_n)^2$$

E_{in}을 최소화 하는 W찾자.

$$= \frac{1}{4}\left\| \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ 1 & x_4 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \right\|^2$$

Getting the solution $\mathbf{w}_{\text{lin}}$

- $\mathbf{w}_{\text{lin}}$: the solution to linear regression : E_{in}을 최소화하는 W 찾자.
  - derived by minimizing $\mathrm{E}_{\text{in}}(\mathbf{w})$ over all possible $\mathbf{w} \in \mathbb{R}^{d+1}$
    E_{in}을 최소화하는 arg W 찾기.

$$\mathbf{w}_{\text{lin}} = \underset{\mathbf{w}\in\mathbb{R}^{d+1}}{\operatorname{argmin}} \mathrm{E}_{\text{in}}(\mathbf{w}) \tag{8}$$

$$= \underset{\mathbf{w}\in\mathbb{R}^{d+1}}{\operatorname{argmin}} \frac{1}{N}||X\mathbf{w} - \mathbf{y}||^2 \tag{9}$$

Minimization of $\mathrm{E}_{\mathrm{in}}(\mathbf{w})$

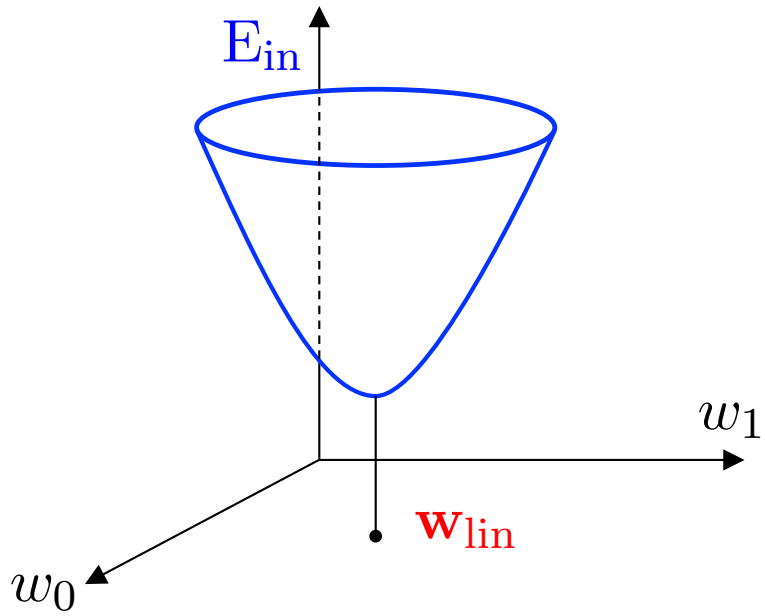
- Eq. (7) implies
  - $\mathrm{E}_{\mathrm{in}}(\mathbf{w})$ is continuous, differentiable, and convex → 이세다.

  $\Rightarrow$ we can use standard matrix calculus to find $\mathbf{w}$ that minimizes $\mathrm{E}_{\mathrm{in}}(\mathbf{w})$ by requiring

$$\nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w}) = 0$$

E_{in}(objective fn)을 최소로 하는 W 찾자.

- general optimization techniques
  - *e.g.* gradient descent



- recall: gradient identities[1]

$$\nabla_{\mathbf{w}}(\mathbf{w}^{\mathrm{T}} A\mathbf{w}) = (A + A^{\mathrm{T}})\mathbf{w}$$
$$\nabla_{\mathbf{w}}(\mathbf{w}^{\mathrm{T}}\mathbf{b}) = \mathbf{b}$$

- scalar w

$$\mathrm{E}_{\mathrm{in}}(w) = aw^2 - 2bw + c$$
$$\nabla \frac{\partial}{\partial w}\mathrm{E}_{\mathrm{in}}(w) = 2aw - 2b$$

- vector $\mathbf{w}$

$$\mathrm{E}_{\mathrm{in}}(\mathbf{w}) = \mathbf{w}^{\mathrm{T}} A\mathbf{w} - 2\mathbf{w}^{\mathrm{T}}\mathbf{b} + c$$
$$\nabla E_{\mathrm{in}}(\mathbf{w}) = (A + A^{\mathrm{T}})\mathbf{w} - 2\mathbf{b}$$

[1] matrix analog of ordinary differentiation of quadratic and linear functions

The solution

- from Eq. (7)

$$\nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w}) = \frac{2}{N}(X^{\mathrm{T}}X\mathbf{w} - X^{\mathrm{T}}\mathbf{y})$$

앞에있던 식 미분 ∇하면

  - both $\mathbf{w}$ and $\nabla \mathrm{E}_{\mathrm{in}}(\mathbf{w})$ are column vectors

- finally, one should solve for $\mathbf{w}$ that satisfies the *normal equations*:

$$X^{\mathrm{T}}X\mathbf{w} = X^{\mathrm{T}}\mathbf{y}$$

X^{T}가 invertible : $X\mathbf{w}=\mathbf{y}$ 풀면. $\forall \varepsilon = 0$

Two scenarios

$(X^TX)w = X^Ty$: normal eq-

1. if X^TX is invertible, $w = X^+y$: w의 값 1개로 결정가능
   - $X^\dagger = (X^TX)^{-1}X^T$ is pseudo-inverse of X
   - resulting $\mathbf{w}$ is the unique optimal solution to (8)

   inverse가 매우 불안정해.
   실제로는 X^TX가 invertible 해도 ↙이 식을 사용해
2. if X^TX is not invertible
   - a pseudo-inverse can still be defined, but no unique solution
   - there will be many solutions for $\mathbf{w}$ that minimizes E_{in}
   - *e.g.* $\mathbf{w}_{\text{lin}} = V\Gamma^{-1}U^T\mathbf{y}$ is a solution to (8)[2]

   에러 증폭↓

[2] $X = U\Gamma V^T$ is the singular value decomposition (SVD) of X (see page 55)

In practice

- $X^{\mathrm{T}}X$ is invertible in most cases
  - since N is often much bigger than $d+1$

  $\Rightarrow$ there will likely be $d+1$ linearly independent vectors $\mathbf{x}_n$

- when $X^{\mathrm{T}}X$ is (almost) singular
  - use a well-implemented $X^{\dagger}$ routine instead of $(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}$
  - this is for numerical stability $\longrightarrow$ 에러 amplify ↓

    에러 적도록 디자인한 $X^{\dagger}$를 믿고 사용해.
- we have derived the following *linear regression algorithm*

The linear regression algorithm

1. construct matrix X and vector $\mathbf{y}$ as follows:
   - use data set $(\mathbf{x}_1, y1), \ldots, (\mathbf{x}_N, y_N)$
   - each $\mathbf{x}$ includes $x_0 = 1$ bias coordinate

$$\underbrace{X = \begin{bmatrix} — \ \mathbf{x}_1^{\mathrm{T}} \ — \\ — \ \mathbf{x}_2^{\mathrm{T}} \ — \\ \vdots \\ — \ \mathbf{x}_N^{\mathrm{T}} \ — \end{bmatrix}}_{\text{input data matrix}} \qquad \underbrace{\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\text{target vector}}$$

2. compute pseudo-inverse $X^\dagger$ of X; if $X^{\mathrm{T}}X$ is invertible,

$$X^\dagger = (X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}$$

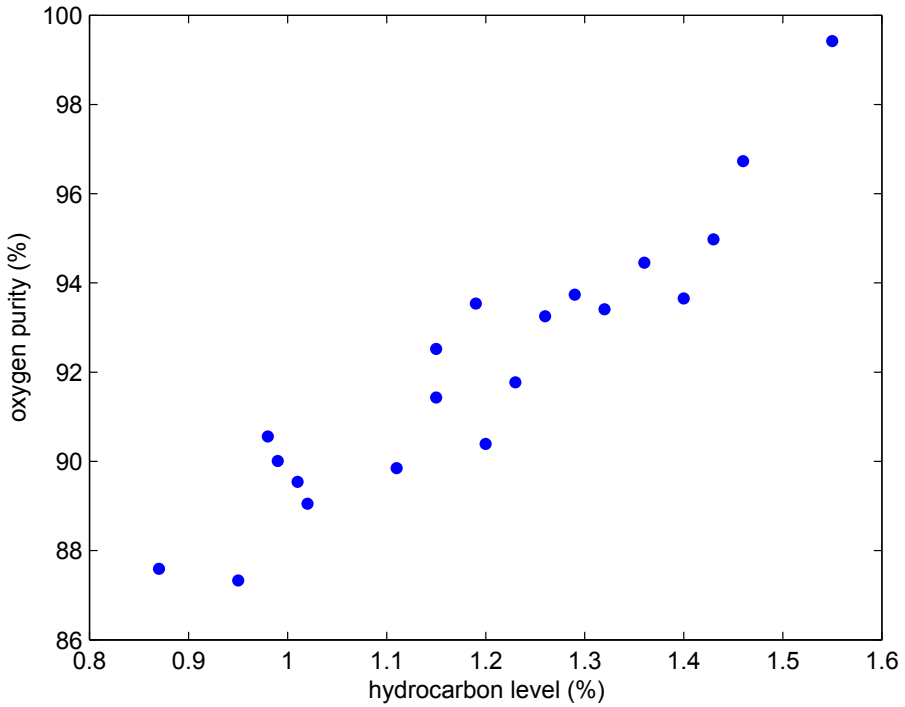
invertible 안하면 SVD등 다양한 방법이 존재한다.

3. return $\mathbf{w}_{\mathrm{lin}} = X^\dagger \mathbf{y}$

Example: oxygen and hydrocarbon levels

no.	x	y
1	0.99	90.01
2	1.02	89.05
3	1.15	91.43
4	1.29	93.74
5	1.46	96.73
6	1.36	94.45
7	0.87	87.59
8	1.23	91.77
9	1.55	99.42
10	1.40	93.65
11	1.19	93.54
12	1.15	92.52
13	0.98	90.56
14	1.01	89.54
15	1.11	89.85
16	1.20	90.39
17	1.26	93.25
18	1.32	93.41
19	1.43	94.98
20	0.95	87.33

- y: purity (%) of oxygen produced in a chemical distillation process
- x: hydrocarbons (%) that are present in the main condenser of the distillation unit



- training data:

$$\text{data matrix } X = \begin{bmatrix} 1 & 0.99 \\ 1 & 1.02 \\ 1 & 1.15 \\ 1 & 1.29 \\ 1 & 1.46 \\ 1 & 1.36 \\ 1 & 0.87 \\ 1 & 1.23 \\ 1 & 1.55 \\ 1 & 1.40 \\ 1 & 1.19 \\ 1 & 1.15 \\ 1 & 0.98 \\ 1 & 1.01 \\ 1 & 1.11 \\ 1 & 1.20 \\ 1 & 1.26 \\ 1 & 1.32 \\ 1 & 1.43 \\ 1 & 0.95 \end{bmatrix}, \text{ target vector } \mathbf{y} = \begin{bmatrix} 90.01 \\ 89.05 \\ 91.43 \\ 93.74 \\ 96.73 \\ 94.45 \\ 87.59 \\ 91.77 \\ 99.42 \\ 93.65 \\ 93.54 \\ 92.52 \\ 90.56 \\ 89.54 \\ 89.85 \\ 90.39 \\ 93.25 \\ 93.41 \\ 94.98 \\ 87.33 \end{bmatrix}$$

19개

- $X^{\mathrm{T}}X$ is invertible

$$X^{\mathrm{T}}X = \begin{bmatrix} 20.00 & 23.92 \\ 23.92 & 29.29 \end{bmatrix} \Rightarrow (X^{\mathrm{T}}X)^{-1} = \begin{bmatrix} 2.15 & -1.76 \\ -1.76 & 1.47 \end{bmatrix}$$

- $(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}\mathbf{y}$ yields

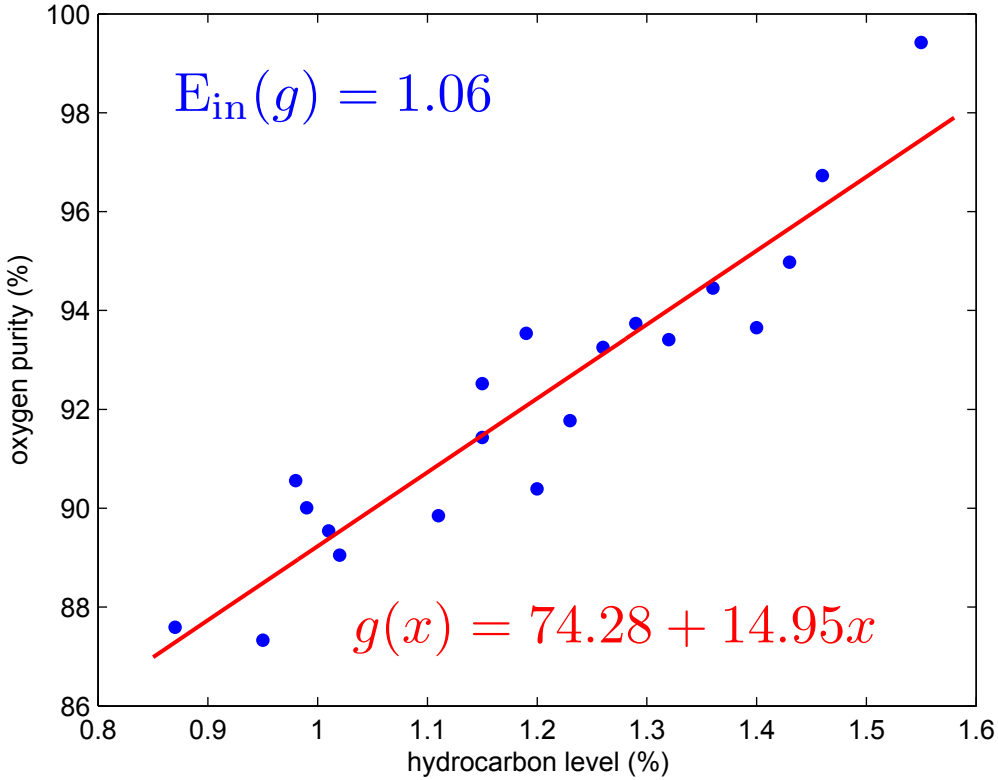
$$\mathbf{w}_{\mathrm{lin}} = \begin{bmatrix} 74.28 \\ 14.95 \end{bmatrix}$$

- the learned model:

$$\begin{aligned} g(x) &= \mathbf{w}_{\mathrm{lin}}^{\mathrm{T}}\mathbf{x} \\ &= 74.28 + 14.95x \end{aligned}$$

- error (see Eq. 3):

$$\mathrm{E}_{\mathrm{in}}(g) = 1.06$$
$$\mathrm{E}_{\mathrm{out}}(g) \approx 1.45$$



Remarks

- the linear regression algorithm on page 50
  - also called *ordinary least squares* (OLS)
  - provides the best linear unbiased estimator (BLUE)

- linear regression does not really look like learning
  - compared with PLA
  - hypothesis $\mathbf{w}_{\text{lin}}$ comes from an analytic solution (matrix inversion/multiplications) rather than iterative learning steps

- but learning *has* occurred ("one-step" learning)
  - as long as hypothesis $\mathbf{w}_{\text{lin}}$ has a decent E_{out}

포켓 등과 다르게 한번의 iteration으로 찾을수 있다는 것

- linear regression: a rare case
  - analytic formula for learning that is easy to evaluate

  $\Rightarrow$ "very popular"

- there are methods for computing the pseudo-inverse $X^\dagger$ directly
  - without inverting a matrix
  - numerically more stable than matrix inversion

- assume: the SVD of input matrix X is $X = U\Gamma V^{\mathrm{T}}$
    - $U \in \mathbb{R}^{N\times\rho}$ satisfies $U^{\mathrm{T}}U = I_\rho$,
    - $V \in \mathbb{R}^{(d+1)\times\rho}$ satisfies $V^{\mathrm{T}}V = I_\rho$,
    - $\Gamma \in \rho \times \rho$ is a positive diagonal matrix with $\rho = \mathrm{rank}(X)$
  - then $\mathbf{w}_{\mathrm{lin}} = V\Gamma^{-1}U^{\mathrm{T}}\mathbf{y}$ satisfies $X^{\mathrm{T}}X\mathbf{w}_{\mathrm{lin}} = X^{\mathrm{T}}\mathbf{y}$

Outline

Hat matrix H (aka projection matrix)

- H relates to in-sample and out-of-sample errors
  - one of analysis tools developed in statistics

- H maps the vector of observed values to the vector of fitted values
  - if $\mathbf{y}$ and $\hat{\mathbf{y}}$ denote observed and fitted values, respectively

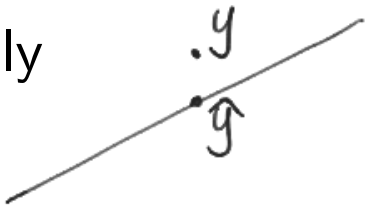
$$\hat{\mathbf{y}} = H\mathbf{y} \tag{10}$$

  - H: observed values $\mapsto$ fitted values
  - matrix H 'puts a hat' on $\mathbf{y}$, hence the name

Hat matrix in linear regression

- the linear regression weight vector $\mathbf{w}_{\text{lin}}$: 빗살무늬토기 가장작은점
  - an attempt to map inputs X to outputs $\mathbf{y}$

- however, $\mathbf{w}_{\text{lin}}$ does not produce $\mathbf{y}$ directly
  - but produces an estimate



$$\hat{\mathbf{y}} = X\mathbf{w}_{\text{lin}}$$

which differs from $\mathbf{y}$ due to in-sample error

- substituting the expression for $\mathbf{w}_{\text{lin}}$
  - we get (assuming $X^{\mathrm{T}}X$ is invertible)

$$\hat{\mathbf{y}} = \underbrace{X(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}}_{H \in \mathbb{R}^{N\times N}}\mathbf{y} = X\mathbf{w}_{\text{lin}} = H\mathbf{y}$$

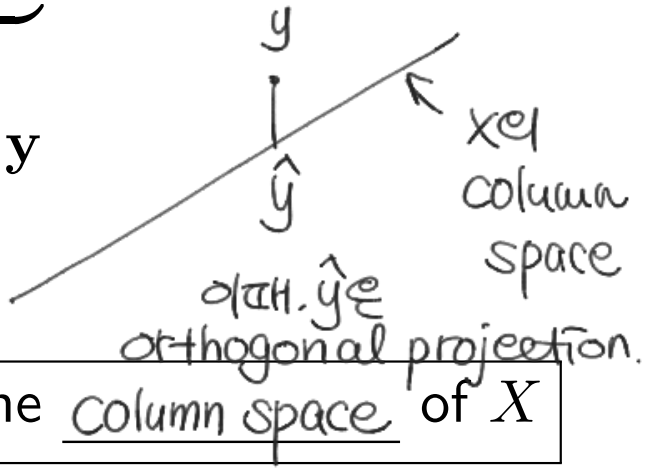
- therefore, estimate $\hat{\mathbf{y}}$ is a linear transformation of actual $\mathbf{y}$
  - through matrix multiplication with H where

$$H = X(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}} \tag{11}$$

projection : XX^{T} 인데 가운데는 normalize 하는것.

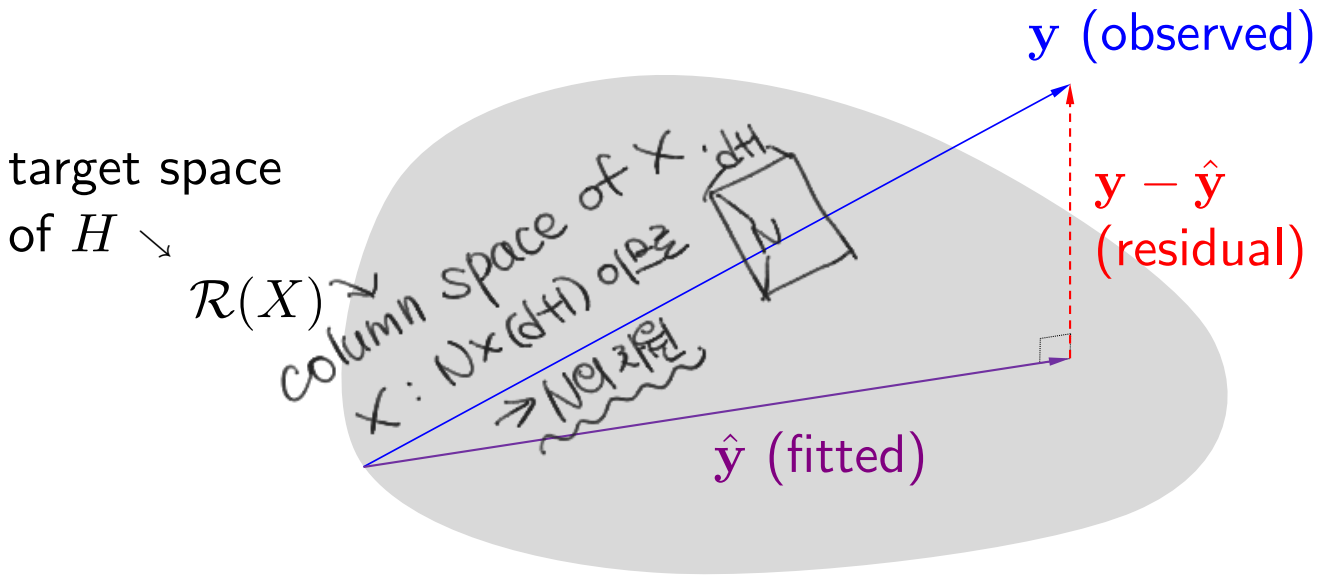
- bottom line:

$$\begin{aligned}\hat{\mathbf{y}} &= X\mathbf{w}_{\mathrm{lin}} \\ &= X\underbrace{(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}\mathbf{y}}_{\mathbf{w}_{\mathrm{lin}}} \\ &= \underbrace{X(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}}_{H}\mathbf{y}\end{aligned}$$



$\therefore$ $\hat{\mathbf{y}}$: orthogonal projection of $\mathbf{y}$ onto the column space of X

Geometric interpretation



to minimize $||\mathbf{y} - \hat{\mathbf{y}}||$, $\hat{\mathbf{y}}$ should be the orthogonal projection of $\mathbf{y}$ onto $\mathcal{R}(X)$

$$\hat{\mathbf{y}} = X\mathbf{w}_{\text{lin}} = H\mathbf{y}$$

$$H = X(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}$$

$\mathcal{R}(X) =$ range (column span) of X

$$\mathbf{y}, \hat{\mathbf{y}}, \mathcal{R}(X) \in \mathbb{R}^N$$

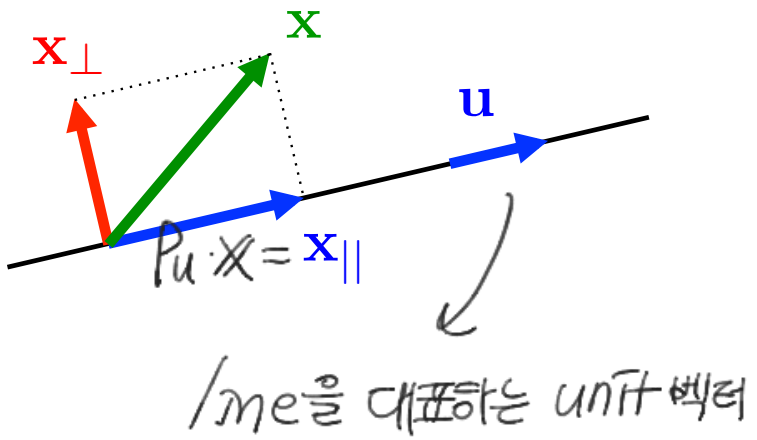
$$H \in \mathbb{R}^{N \times N}$$

$$X \in \mathbb{R}^{N \times (d+1)}$$

$$\mathbf{w}_{\text{lin}} \in \mathbb{R}^{d+1}$$

Intuitive/informal explanation of projection

- a simple case: orthogonal projection onto a line
  - let $\mathbf{u}$ denote a unit vector on the line
  - the projection matrix is then given by $P_u = \mathbf{u}\mathbf{u}^T$
- to see why, decompose an arbitrary $\mathbf{x}$ into $\mathbf{x} = \mathbf{x}_{||} + \mathbf{x}_{\perp}$
  - $\mathbf{x}_{||}$: component of $\mathbf{x}$ on the line
  - $\mathbf{x}_{\perp}$: component of $\mathbf{x}$ perpendicular to the line



- applying projection to $\mathbf{x}$ gives

$$P_{\mathbf{u}}\mathbf{x} = \mathbf{u}\mathbf{u}^{\mathrm{T}}\mathbf{x}_{||} + \mathbf{u}\mathbf{u}^{\mathrm{T}}\mathbf{x}_{\perp}$$
$$= \mathbf{u}|\mathbf{x}_{||}| + \mathbf{u}0$$
$$= \mathbf{x}_{||}$$

$P_{\mathbf{u}}\mathbf{x}$ = $\mathbf{x}$ 성분 추출.

which confirms that $P_{\mathbf{u}}$ works

- generalization (orthonormal basis):
  - let $\mathbf{u}_1, \ldots, \mathbf{u}_k$ be an orthonormal basis of subspace U
  - let A denote the matrix whose columns are $\mathbf{u}_1, \ldots, \mathbf{u}_k$
  - then the projection matrix onto U is $P_A = AA^T$

vector 여러개로.

- generalization (non-orthonormal basis):
  - let $\mathbf{u}_1, \ldots, \mathbf{u}_k$ be a (not necessarily orthonormal) basis of U
  - let A denote the matrix whose columns are $\mathbf{u}_1, \ldots, \mathbf{u}_k$
  - then the projection matrix onto U is $\boxed{P_A = A(A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}}}$
  - note that $(A^{\mathrm{T}}A)^{-1}$ is a "normalizing factor"

$$P_A = A\overbrace{(A^{\mathrm{T}}A)^{-1}}A^{\mathrm{T}}$$

- for vector $\mathbf{y}$, $P_A\mathbf{y} = A(A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}}\mathbf{y} = \hat{y}$
  - projects $\mathbf{y}$ orthogonally onto $\mathcal{R}(A)$, the column span of A

Properties of projection/hat matrices

let $H \in \mathbb{R}^{N \times N}$ (*i.e.*, H is a square matrix):

- H is called a projection/hat matrix iff $H^2 = H$
- two basic properties of hat matrix H
  - symmetric: $H^{\mathrm{T}} = H$
  - idempotent: $H^2 = H$ (no effect on vectors already on space)
- for hat matrix H and identity matrix $I \in \mathbb{R}^{N \times N}$
  - $I - H$ is also a hat matrix:

$$(I - H)^2 = I - 2H + H^2 = I - H$$

  - $H^{\mathrm{T}}(I - H) = 0 \Rightarrow$ target spaces are orthogonal

- for hat matrix $H = X(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}$ where $X \in \mathbb{R}^{N\times(d+1)}$
  - H : orthogonal projection onto the column space of X
  - range $\mathcal{R}(X)$ is thus called the target space of H
  - $\mathrm{trace}(H) =$ $d+1$

$$\begin{aligned}\mathrm{trace}(H) &= \mathrm{trace}(X(X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}) \\ &= \mathrm{trace}(X^{\mathrm{T}}X(X^{\mathrm{T}}X)^{-1}) \\ &= \mathrm{trace}(I) \\ &= d+1\end{aligned}$$

    - ▷ note: $X^{\mathrm{T}}X \in \mathbb{R}^{(d+1)\times(d+1)}$ and thus $I \in \mathbb{R}^{(d+1)\times(d+1)}$

- recall: $\mathrm{trace}(H)$ is sum of diagonal elements in H
  - property: $\mathrm{trace}(AB) = \mathrm{trace}(BA)$ for two matrices A and B

Generalization of linear regression

정리하면 : input X로 부터 y를 예측한다.
이때. best X 하이퍼 플레인 만들자.
우리는 error를 OLS로 한다.

↝ 그래서 이게 머신 러닝에서 쓸수 있는가?

- linear regression looks for optimal $\mathbf{w}$ in terms of E_{in}
  - does this guarantee decent E_{out}? yes!

1) 원샷방법 :
$w_{lin} = (X^T X)^{-1} X^T y$.
2) H : hat matrix 방법 :
투영과 관찰 둘의 차이가 최소 되도록

- theoretical analysis shows
  - generalization bound of linear regression:

이 역시 VC 차원을 따른다.

$$\mathrm{E}_{\text{out}}(g) = \mathrm{E}_{\text{in}}(g) + O\left(\frac{d}{N}\right) \quad (2)$$

→ 결국 같은데 아닌가요?

마치 앞의 classification 처럼

$\Rightarrow$ comparable to classification (3): $\mathrm{E}_{\text{out}}(g) = \mathrm{E}_{\text{in}}(g) + O\left(\sqrt{\frac{d}{N}\ln N}\right)$

- we can even derive exact formulas for E_{in} & E_{out} (see textbook)

assumption 이용하면

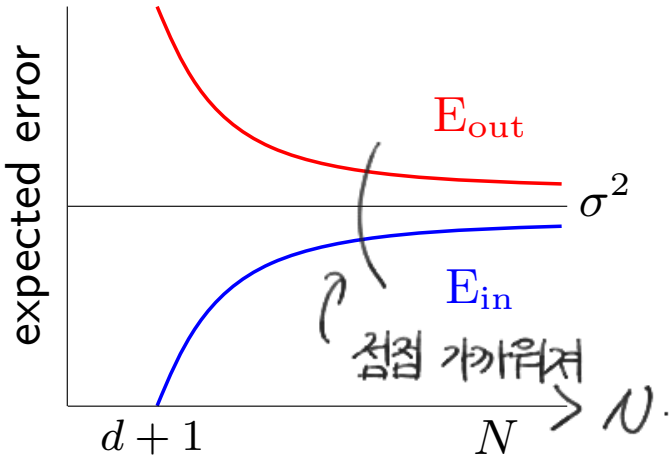
  - assume: $y = f + \epsilon$ (noise ϵ has zero mean and σ^2 variance)

ϵ은 0을 평균.
ϵ의 power를 σ로 나타냄.

- the best possible linear fit has the expected error σ^2 (noise level)

$$\mathrm{E}_{\text{in}} = \sigma^2 \left(1 - \frac{d+1}{N}\right) < \sigma^2$$

$$\mathrm{E}_{\text{out}} = \sigma^2 \left(1 + \frac{d+1}{N}\right) > \sigma^2$$



  - both converge to σ^2 as $N \to \infty$
  - $\mathrm{E}_{\text{in}} < \sigma^2$ because regression fits in-sample noise as well
  - $\mathrm{E}_{\text{out}} > \sigma^2$ reflects drift in $\mathbf{w}_{\text{lin}}$ due to fitting in-sample noise

- expected generalization error: $O\left(\frac{2(d+1)}{N}\right)$
  - learning has 'happened' by linear regression

Outline

Summary

- linear models
  - used for classification, regression, probability estimation
  - have small d_{VC} and generalize well $\Rightarrow$ first practical choice
  - use the 'signal': $\sum_{i=0}^{d} w_i x_i = \mathbf{w}^{\mathrm{T}}\mathbf{x}$

- linear classification

$$\mathcal{Y} = \{-1, +1\}$$
$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x})$$
$$\text{e}(\hat{y}, y) = [\![\hat{y} \neq y]\!]$$

  - NP-hard in general

- linear regression

$$\mathcal{Y} = \mathbb{R}$$
$$h(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}$$
$$\text{e}(\hat{y}, y) = (\hat{y} - y)^2$$

  - efficient closed-form solution

- linear regression algorithm (1-step learning): $\boxed{\mathbf{w} = (X^{\mathrm{T}}X)^{-1}X^{\mathrm{T}}\mathbf{y}}$