{已上传至github ML notes}

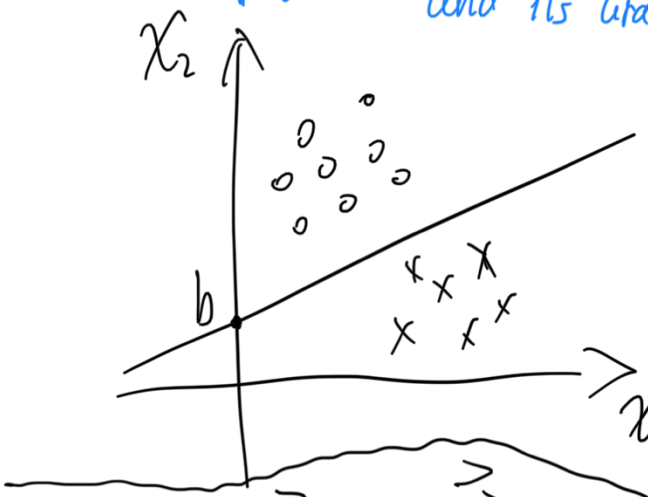
$\nabla = \left[\frac{\partial}{\partial x}\ \frac{\partial}{\partial y}\ \frac{\partial}{\partial z}\right]^T$

/nabla/

Linear Discriminant Function: $g(x)$

判别函数

and its Gradient Descent Optimization.



slope

$x_2 = mx_1 + b$

In LDF, we assu

$g(x)$ is linear f

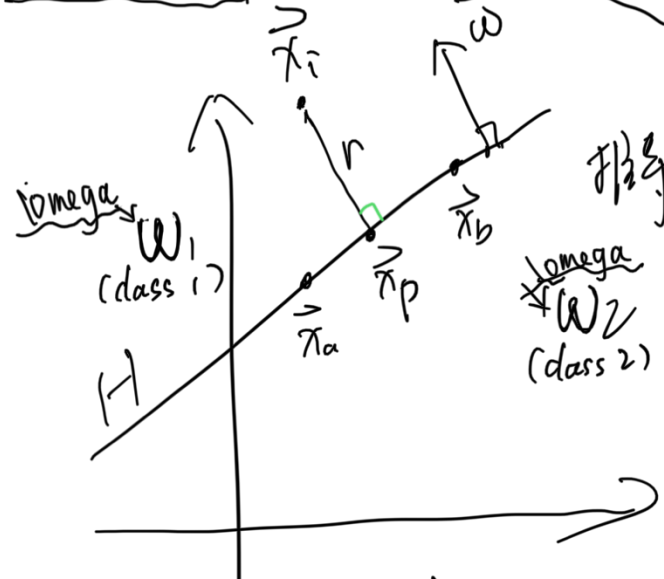
$\Rightarrow$ general form: $w_0 + w_1x_1 + w_2x_2 = 0$

$\Rightarrow$ vector form: $w^T\vec{x} + w_0 = 0$

where: $w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$, $\vec{x} = \begin{bmatrix} x \\ x_2 \end{bmatrix}$

$\vec{x} = [x_1\ x$

normal.



1) $g(x_a) = w^Tx_a + w_0 = 0$

$g(x_b) = w^Tx_b + w_0 = 0$

$\therefore w^Tx_a + w_0 = w^Tx_b + w_0$

$\therefore w^T(\vec{x}_a - \vec{x}_b) = 0$

$\therefore w \perp (\vec{x}_a - \vec{x}_b) \Rightarrow w \perp H$

Distance to/from the boundary?

2) $\vec{x}_i = \vec{x}_p + r\frac{\vec{w}}{\|w\|_2}$

$\frac{\vec{w}}{\|w\|_2}$ normalized, called unit vector

$\vec{x}_i - \vec{x}_p = r\frac{\vec{w}}{\|w\|}$

3) $\because g(x_p) = w^Tx_p + w_0 = 0$. ($\because x_p$ 在 H 上)

$$\therefore\ w^T\left(x_i - r\frac{w}{\|w\|}\right) + w_0 = 0$$

$$\therefore\ w^T x_i + w_0 = r\cdot\frac{w^T w}{\|w\|} \quad (w^T w \to \|w\|^2)$$

$$= r\|w\|$$

$$\therefore\ g(x_i) = r\|w\|$$

$$\therefore\ \boxed{r = \frac{g(x_i)}{\|w\|}}$$

r: distance of x_i to the decision boundary $g(x)$ or H.

More confident if x is further away from decision boundary

Ex: $w^T = [1, 2]$, $w_0 = 1$

① H (decision boundary)?

H: $g(x) = w^T x + w_0 = [1\ \ 2]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + 1 = x_1 + 2x_2 + 1 = 0$

$\therefore$ boundary H: $x_2 = -\frac{1}{2}x_1 - \frac{1}{2}$

② Test two samples:

$$g(x) \underset{w_2}{\overset{w_1}{\gtrless}} 0$$

(ω: omega is class. w: double-u is weight)

$$\vec{x_1} = \begin{bmatrix} 0 \\ 1 \end{bmatrix},\quad \vec{x_2} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

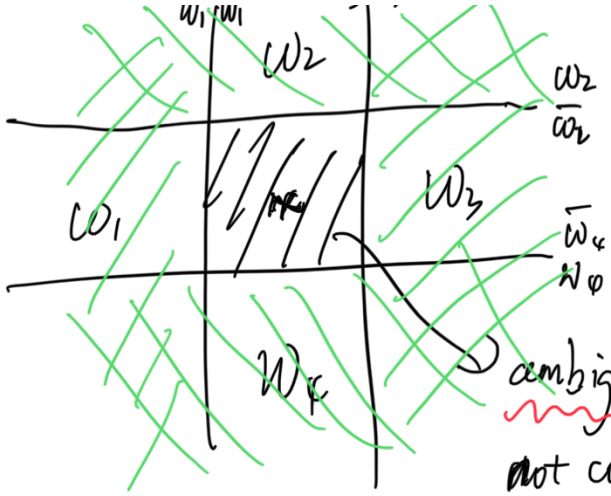
$g(\vec{x_1}) = 0 + 2\times 1 + 1 = 3 > 0 \Rightarrow \vec{x_1} \in \omega_1$

$g(\vec{x_2}) = 0 + 2\times(-1) + 1 = -1 < 0 \Rightarrow \vec{x_2} \in \omega_2$

The system is more confident of $\vec{x_1}$ belonging to ω_1 than $\vec{x_2}$ belonging to ω_2 because $|g(\vec{x_1})| > |g(\vec{x_2})|$

Multi-category LDF.

$\omega_1, \omega_2, \omega_3$



ambiguity region (caused by not covering the whole space

⇓

one solution of this issue is by Bayesian Decision Theory:

★ Assign x to ω_i if $g_i(x) > g_j(x), \forall j \neq i$

Generalized LDF

◉ Linear DF:

$$g(x) = w_0 + \sum_{i=1}^{d} w_i x_i$$

◉ Higher order forms

- Quadratic DF.

$$g(x) = w_0 + \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d} \sum_{j=1}^{d} w_{ij} x_i x_j$$

- Polynomial DF (cubic or higher)

$$g(x) = w_0 + \sum_{i} w_i x_i + \sum_{ij} w_{ij} x_i x_j + \sum w_{ijk} x_i x_j x_k + \cdots$$

i, j, k

So, how to find H, i.e. estimate decision boundary parameter

– $w_0, w_1, \ldots, w_d$: free params of H.

e.g. 2 features, 2 classes.

$$g(\vec{x}) = w_1 x_1 + w_2 x_2 + w_0 = [w_0\ w_1\ w_2] \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

[augmenting]: $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$ (升了一维) ← 对应 augmenting

$\vec{x}$ → $\vec{y}$

$\vec{a} = \begin{bmatrix} w_0 \\ \vec{w} \end{bmatrix}$, $\vec{y} = \begin{bmatrix} 1 \\ \vec{x} \end{bmatrix}$

标准写法: $g(\vec{y}) = \vec{a}^T \vec{y}$ (变量增维) BTW: 这个是把 unknown 写在后

★ One of the good reasons to use augmenting is that:

※ $\vec{a}$ is always locked to origin i.e. $(0,0,0) \to (0)$

$\because (0\ 0\ 0) \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix} = 0$

so the task now is to rotate $\vec{a}$ via origin to find the best boundary curve.

⇒ Estimate $\vec{a}$. or rotate $\vec{a}$.

+ We need good training sample set (Labeled).

If they are linearly separable, we can find the ideal $\vec{a}$, which is also called [solution vector].

BTW: now the goal is training not classification.

+ We need to integrate label information into an uniform function.

※ Normalization

$$\hat{y} = \begin{cases} y_i, & y_i \in \omega_1 \\ -y_i, & y_i \in \omega_2 \end{cases}$$

① suppose for ω_1 class: $g(x) > 0$
for ω_2 class: $g(x) < 0$

$g(x) = a^T y_i < 0 \Rightarrow a^T \hat{y}_i > 0$.

※ After normalization,
find the optimal $\vec{a}$, s.t. $a^T \hat{y} > 0, \forall \hat{y}_i$ ($\forall x_i$)

? What if we get $g(\hat{y}_i) < 0$ for a training data ⇒ 说明这个 classification da
不对也即 label 不对.

⇒ After the previous two steps, i.e. augmenting and normalization
we can now find the optimal $\vec{a}$.

Before move on, we need to ask two questions:
① What are you optimizing? → J
② how do you prove it is optimal? → global minimu

Find $\vec{a}^*$ → Find criterion $J(\vec{a})$.

$$\vec{a}^* = \arg\min_{\vec{a}} J(\vec{a})$$

↓

Solution vector. if → differentiable → $\nabla J(\vec{a}) = 0$.
continuous

Options of $J(a)$

+ Count # of misclassifications (errors)

★ + Error Correction methods — perceptron algorithm
+ cule measurable errors of only from misclassified sample.
if linearly separable.

$$J_p(a) = \sum_{y \in Y} (-a^T y)$$

metrit → $\frac{\partial J_p(\vec{a})}{\partial \vec{a}} : \nabla_a J_p = \sum_{y \in Y} (-y$

Y ← misclassified samples

★ + Squared Error methods.
+ measure errors from all training samples; i.e. misclassified samples and correctly classified samples.

Perceptron $J_p(a)$
+ no samples misclassified: $J_p(a) \neq 0$. <0
+ if y is misclassified, $a^T y \leq 0$ ∴ $J_p(a) \geq 0$.

$$\nabla_{\vec{a}} J_p = \sum_{y \in Y} (-y)$$

step① Initialization: random $\vec{a}(0)$
step② use. gradient descent algorithm iteratively:

$$\vec{a}(k+1) = \vec{a}(k) - \eta(k) \nabla J(\vec{a}(k))$$

($\eta(k)$: learning rate)

step③ stop: $|\nabla J(\vec{a}(k))| < \theta$

Ex:

x_1	x_2	ω_1/ω_2 (class)
0.1	1.1	1
6.8	7.1	1
7.1	4.2	2
-1.4	-4.3	2

$\eta(k) = 0.1$

tolerance $\theta = 0$

Goal: $a = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$

s.t. $\hat{y}_i \cdot a > 0, \forall i$.

① Augmentation

y 矩阵应该是 $N \times (d+1)$, $g(y) = Y \cdot a$

$$y = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0.1 & 6.8 & 7.1 & -1.4 \\ 1.1 & 7.1 & 4.2 & -4.3 \end{bmatrix}^T$$ (a 为 column vector)

$N \times (d+1)$

② Normalization (即把 ω_2 类 in samples 按 origin 对称过去)

$$\hat{y} = \begin{bmatrix} 1 & 1 & -1 & -1 \\ 0.1 & 6.8 & -7.1 & 1.4 \\ 1.1 & 7.1 & -4.2 & 4.3 \end{bmatrix}^T$$

$N \times (d+1)$

now, $y = \hat{y}$.

class 2 ($\hat{y} = -y_i$)

Alg: ① Initialization:

$$a(0) = [0\ 0\ 0]^T.$$

② 1st iteration: $a(0)^T y = [0\ 0\ 0] \begin{bmatrix} \\ \\ \end{bmatrix}$

$= [0\ 0\ 0\ 0]$ all classified

$$\nabla J_p(a) = \sum_i (-y_i) = \begin{bmatrix} 0 \\ -1.2 \\ -8.3 \end{bmatrix}$$ (y_i: column vector)

$$a(1) = a(0) - \eta \nabla J_p(a(0))$$

$$= 0 - 0.1 \times \begin{bmatrix} 0 \\ -1.2 \\ -8.3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.12 \\ 0.83 \end{bmatrix}$$

2nd iteration:

$>0 \quad >0 \quad >0$

$a^T(1)\,y = \cdots = [\,0.0\ldots,\ [illegible],\ (0.4\ldots),\ \ldots\,]$

↑ < 0 : misclassified

只对错分的计算

$\therefore \nabla J_p(a(k=1)) = -y_3 = \begin{bmatrix} 1 \\ 7.1 \\ 4.2 \end{bmatrix}$

$\therefore a(2) = a(k=1) - 0.01 \times \begin{bmatrix} 1 \\ 7.1 \\ 4.2 \end{bmatrix} = \begin{bmatrix} -0.01 \\ -0.06 \\ 0.04 \end{bmatrix}$

3rd iteration $(k=2)$:

step 1 : classification

$a^T(2)\,y = \cdots = [\,0.028,\ -0.1201,\ 0.25,\ 0.103\ \ldots\,]$

↑ < 0 : misclassified

只对错分的计算.

step 2: $\nabla J_p(a(2)) = -y_2 = \begin{bmatrix} -1 \\ -6.8 \\ -7.1 \end{bmatrix}$

step 3 : $a(3) = a(2) - 0.01 \times \begin{bmatrix} -1 \\ -6.8 \\ -7.1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.008 \\ 0.111 \end{bmatrix}$

4th iteration.

5th iteration.

$H: g(x) = -0.01 - 0.063x_a + 0.069x_b = 0$

(BW: but is this the best solution vector? e.g. equal margin to each class)

★ Problem formulated as that of minimizing or cost function.

⇑ Gradient descend & perceptron criterion.

⇒ Batch and single sample perceptron criterion.

Step 1 LDF: Augmentation

- LDF: $g(x) = w^T x + w_0$
- Rewrite it: $g(x) = [w_0 \; w^t] \begin{bmatrix} 1 \\ x \end{bmatrix} = a^t y = g(y)$

  new weight vector a ; new feature vector y
- y is the Augmented Feature Vector $\Rightarrow (d+1) \times 1$

Step 2 LDF: Normalization (i.e. switch the y_i sign)

线性可分 $\begin{cases} a^T y_i > 0 & \forall y_i \in C_1 \\ a^T y_i < 0 & \forall y_i \in C_2 \end{cases} \Rightarrow a^T(-y_i) > 0 \quad \forall y_i \in C_2$

$\Rightarrow a^T y_i > 0 \quad \forall y_i$ 都在线的一边

If such a exists, it is called a solution vector.
(→ linearly seperable)

【Step 3】 LDF : criterion $J(a)$ → always be a scalar.

Particularly, we use perceptron criterion $J_p(a)$
optimization method : gradient descent with $J_p(a), \nabla J_p$

$$J_p(a) = \sum_{y \in Y} (-a^T y), \quad \nabla J_p(a) = \sum_{y \in Y} (-y)$$

↑ error samples ; 这种方法也叫 Error Correcti methods.

※ Relaxation procedures (用 squared error 来 smoothen the error change)
- $J_q(a)$: squared Error
- $J_r(a)$: Normalized squared error

$J_p(a)$ - abrupt change (cons)

Basic Gradient Descent Abogorithm

begin initialize a, threshold θ, $\eta(\cdot)$, $k \leftarrow 0$
do $k \leftarrow k+1$
$a \leftarrow a - \eta(k) \nabla J(a)$
until $|\eta(k) \nabla J(a)| < \theta$
return a.
end

$J(a) = \begin{cases} J_p(a) \\ J_q(a) \\ J_r(a) \end{cases}$

Choose one of the ab
when designing an algorith

Relaxation Procedures

* Squared error criterion:

$$J_q(a) = \sum_{y \in Y} (a^T y)^2$$

- It is the generalization of perceptron criterion.
- Error surface is continuous and smooth.

$$\nabla J_q = 2 \cdot \sum_{y \in Y} (a^T y) \cdot \vec{y} \qquad \nabla J_p = \sum (-\vec{y})$$

($a^T y$: 1×1)

↳ dominated by the ~~longest~~ largest sample vectors. Can use GD alg

($\|y\|$)

* Normalized squared-error criterion

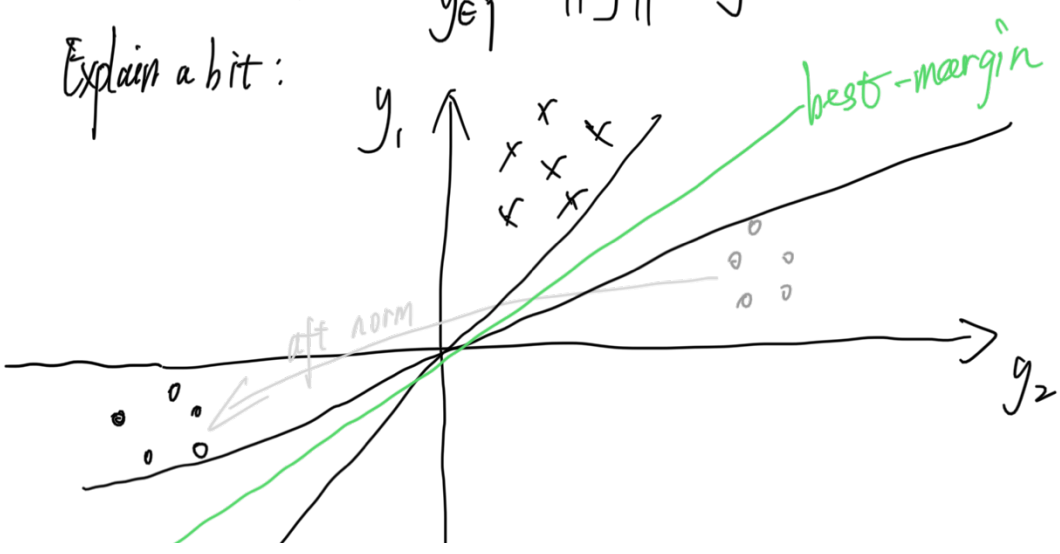
$$J_r(a) = \frac{1}{2} \sum_{y \in Y} \frac{(a^T y - b)^2}{\|y\|^2}$$

b: margin ← maximize as well

Distance is normalized with margin.

$$\nabla J_r(a) = \sum_{y \in Y} \frac{a^T y - b}{\|y\|^2} \vec{y}$$

Explain a bit:



J_q, J_r — batch / single sample mode

① Test misclassification

② Update of $\vec{a}$ (if any misclassification) Y: misclf samples

batch $\begin{cases} J_q: & a \leftarrow a - \eta(\cdot) \sum_{y \in Y} (a^T y)\vec{y} \\ J_r: & a \leftarrow a - \eta(\cdot) \sum_{y \in Y} \frac{b - a^T y}{\|y\|^2} \cdot \vec{y} \end{cases}$

simple sample $\begin{cases} J_q: a \leftarrow a - \eta(\cdot)(a^T y_k)\vec{y}_k \\ J_r: a \leftarrow a - \eta(\cdot) \frac{b - a^T y_k}{\|y_k\|^2} \vec{y}_k \end{cases}$

All is error-correction.

If Not linearly separable,

* Error correcting procedures not so effective.
* Why?
  - modify weight only when error is encounted bu there is always error.

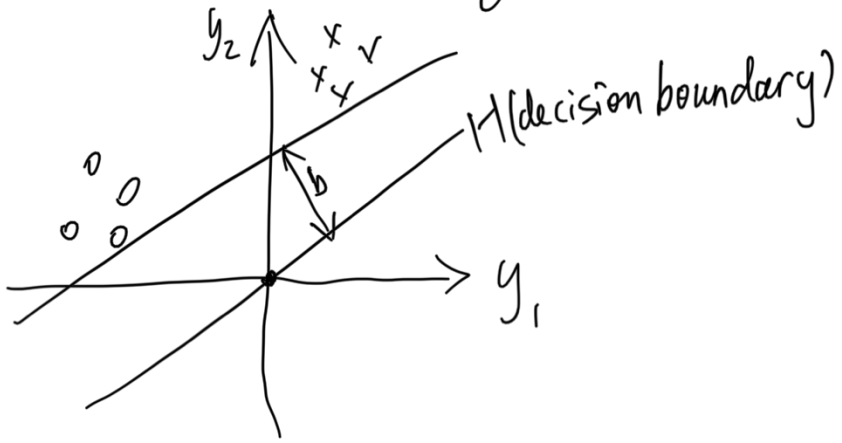
---

learning rate $\eta(k)$ is a variable as well.

Now it's not error correction any more since we're considering misclassfied and correctly classified

samples, i.e. all sample are considered".

Minimum Squared-Error (MSE) Procedures



$$a^T y_i = b_i > 0$$

$(1\times(d+1))\ ((d+1)\times 1)$

formulate the problem

margin

$$\underset{N\ \text{samples}}{\begin{bmatrix} y_{10} & y_{11} & \cdots & y_{1d} \\ y_{20} & y_{21} & \cdots & y_{2d} \\ \vdots & & & \\ y_{N0} & y_{N1} & & y_{Nd} \end{bmatrix}} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_d \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix}$$

$N\times(d+1)$ training samples

$(d+1)\times 1$

$N\times 1$ ↖ preset

abbr. $\mathbf{Y}\vec{a} = \vec{b}$

column vector

can we find such a

If $\mathbf{Y}$ is (square matrix and) nonsingular, $\vec{a} = \mathbf{Y}^{-1}\vec{b}$.

But in practice, $N \gg d+1$, $\mathbf{Y}$ is a tall matrix, $\vec{a} = ?$

⇒ Answer: no solution of $\vec{a}$.

(BW: nonsingular means a regular matrix that has a

matrix inverse.)

*pseudoinverse 擬逆

$$\Rightarrow J_s(\vec{a}) = \|Y\vec{a} - \vec{b}\|^2 = (Y\vec{a} - \vec{b})^T (Y\vec{a} - \vec{b})$$

$$\therefore \vec{a} = \arg\min_{\vec{a}} J_s(\vec{a})$$

$$\nabla J_s(\vec{a}) = \sum_{i=1}^{N} 2(\vec{a}^T y_i - b_i) y_i$$

$$= 2Y^T(Y\vec{a} - \vec{b}) = 2(Y^T Y\vec{a} - Y^T\vec{b})$$

Let $\nabla J_s(\vec{a}) = 0$, then

$$Y^T Y\vec{a} = Y^T\vec{b}.$$

$(d+1)\times N$ $\quad N\times(d+1)$

$(d\times 1)\times(d\times 1)$

If nonsingular which most likely is true because all samples are independent to each other (covariance matrix), then

$$\vec{a} = \underbrace{(Y^T Y)^{-1} Y^T}_{(d+1)\times N} \vec{b} = \underbrace{Y^{\dagger}}_{(d+1)\times N} \underbrace{\vec{b}}_{N\times 1}$$

called pseudoinverse: $Y^{\dagger}$ $\quad (d+1)\times N$

Example:

$$X = \begin{bmatrix} 1 & 2 & 2 & 3 \\ 2 & 0 & 3 & 1 \end{bmatrix}^T \xrightarrow{\text{Aug.}} Y = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 3 \\ 2 & 0 & 3 & 1 \end{bmatrix}^T \xrightarrow{\text{norm}} \begin{bmatrix} 1 & 1 & -1 \\ 1 & 2 & -2 \\ 2 & 0 & -3 \end{bmatrix}$$

$N\times(d+1)$

$N\times d$

assume $b = [\ 1\ 1\ 1\ 1\ ]^T$ ← It doesn't matter how large

assume $b = [1 \ 1 \ 1 \ 1]^T$ (doesn't matter how many you assume.)

$$\therefore Y^TY = \begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & 2 & -2 & -3 \\ 2 & 0 & -3 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 0 \\ -1 & -2 & -3 \\ -1 & -3 & -1 \end{bmatrix} = \begin{bmatrix} 4 & 8 & 6 \\ 8 & 18 & 11 \\ 6 & 11 & 14 \end{bmatrix} \rightarrow |Y^TY| =$$

$$\therefore (Y^TY)^{-1} = \frac{1}{36} \begin{bmatrix} 131 & -46 & -22 \\ -46 & 20 & 8 \\ -20 & 8 & 8 \end{bmatrix}$$

$$\therefore Y^\dagger = (Y^TY)^{-1} \cdot Y^T = \cdots$$

$$\therefore a = Y^\dagger b = \begin{pmatrix} 11/3 \\ -4/3 \\ 2/3 \end{pmatrix}$$

$\therefore$ Decision boundary H : $\frac{11}{3} - \frac{4}{3}x_1 - \frac{2}{3}x_2 = 0$.

☆ This pseudoinverse problem is a convex problem, which means we do have a global optimal solution.

□ If it's not batch procedure, that is, for single sample, this pseudoinverse method is call Least Mean Squared-error (LMS) algorithm. (又称为 single-sample MSE procedure using gradient descent method).

SVM : kernels

SVM is fit for small amount of data, not good for deep learning.

Summary

$J_p(\vec{a})$ → two important steps. (why?)

$J_q(\vec{a})$

$J_r(\vec{a})$

$J_s(\vec{a})$ → Pseudoinverse, LMS (single sample)

Margin r, b