Perceptrons

1. ANS [b]

Which of the following set of $\mathbf{x} \in \mathbb{R}^3$ can be shattered by the 3D perceptron hypothesis set? The set contains all hyperplanes of the form with our usual notation of $x_0 = 1$:

$$h_{\mathbf{w}}(\mathbf{x}) = \text{sign}\left(\sum_{i=0}^{3} w_i x_i\right).$$

Choose the correct answer; explain your choice.

此為 3D perceptron，所以為 binary degrees of freedom，可視為每個 xi 對應的 wi 可能為-1 或+1 兩種選擇。

[a] $\{(2,3,4),(4,3,2),(3,3,3)\}$

令線方程式為a$x_1 + bx_2 + cx_3 = k$：

帶入 A(2,3,4)：$2\text{a} + 3b + 4c = k \ldots..(A)$

帶入 B(4,3,2)：$4\text{a} + 3b + 2c = k \ldots..(B)$

(A)-(B)：

$$-2\text{a} + 2c = 0$$
$$\text{a} = c$$

帶回(B)：

$$6\text{a} + 3b = 0$$
$$\text{a}: b = -1: 2$$
$$\text{a}: b: c = -1: 2: -1$$
$$x_1 - 2bx_2 + x_3 = k$$

帶入 A 點(2,3,4)：

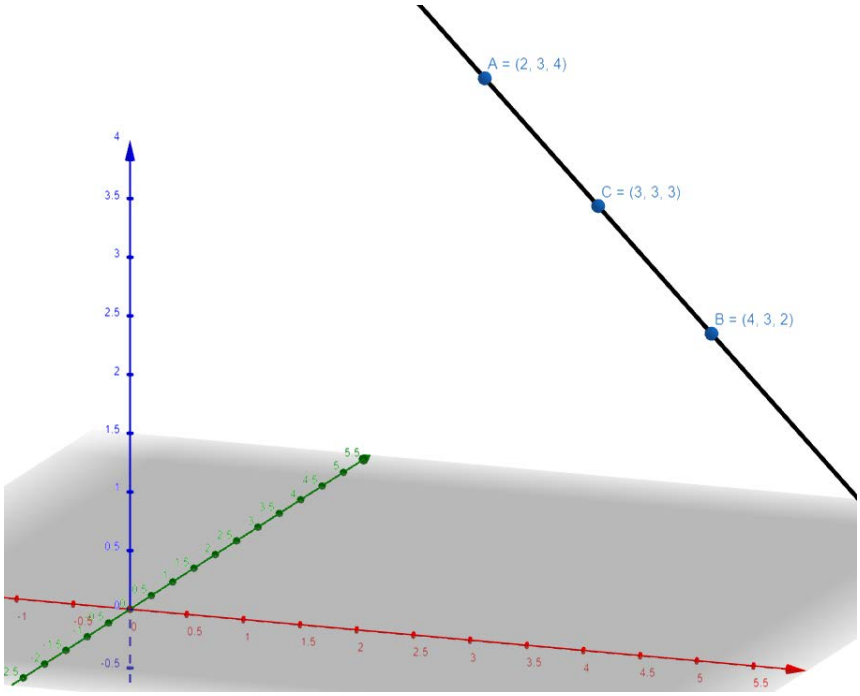
$$2 - 6 + 4 = 0 = k$$

線方程式為：

$$\begin{cases} x_1 - 2bx_2 + x_3 = 0 \\ -x_1 + 2bx_2 - x_3 = 0 \end{cases}$$

C 點(3,3,3)帶入(3-6+3=0)也符合，故 ABC 三點共線。

如下圖(從 GeoGebra 製作)所示，此三點共線：



ABC 三點共最多可以有$2^3 = 8$種 dichotomies。

由於三點共線，由圖可知，無法產出以下兩種情況：

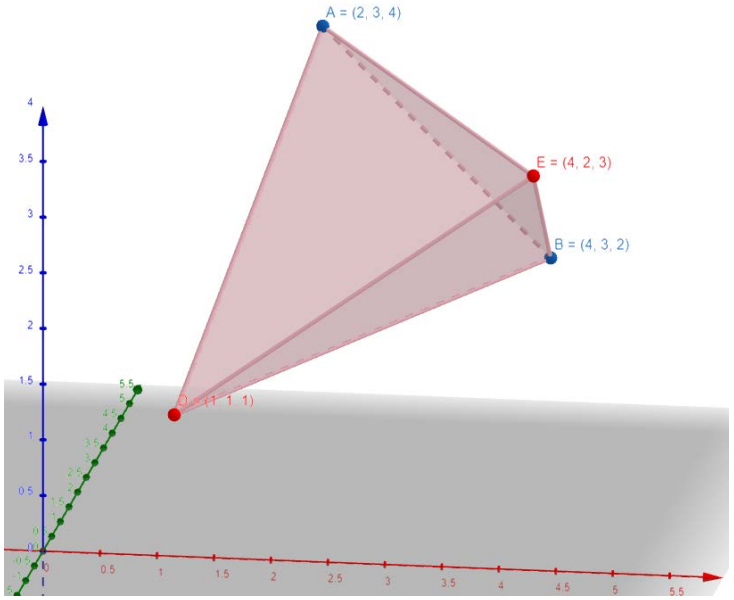
A(2,3,4)	B(4,3,2)	C(3,3,3)
+	-	+
-	+	-

所以此三點無法被 shattered。

[b] $\{(1,1,1),(2,3,4),(4,3,2),(4,2,3)\}$

由右圖所示，可見四點分布：

最多會有$2^4 = 16$種 dichotomies，由於其中八種情況對應另外八種，只是正負差異，故下面列舉出八種情況：



情況	D(1,1,1)	A(2,3,4)	B(4,3,2)	E(4,2,3)	會不會產生
1	+	+	+	+	會
2	+	-	-	-	會
3	+	+	-	-	會
4	+	+	+	-	會
5	+	-	+	-	會
6	+	-	-	+	會
7	+	-	+	+	會
8	+	+	-	+	會

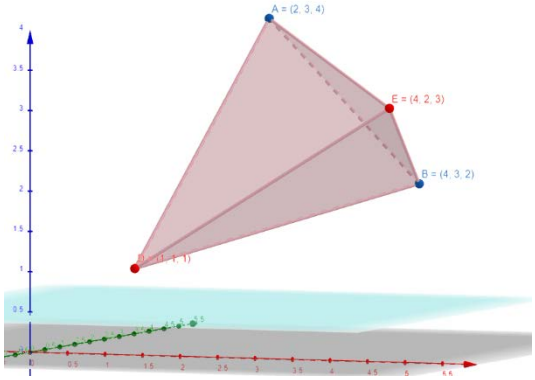
以下舉例出可產生該情況的可能情形：

情況 1

藍色平面$0.25x_3 = 0.13$，(w0,w1,w2,w3)=(-0.13,0,0,0.25)，

帶入 DABE 四點，可產出(D,A,B,E)=(+,+,+,+)。

反之用(w0,w1,w2,w3)=(0.13,0,0,-0.25)可產出(-,-,-,-)。

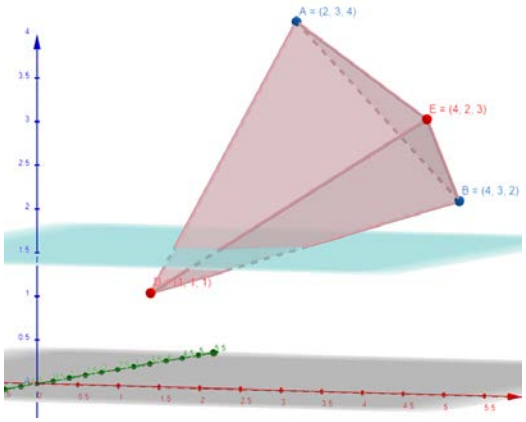


情況 2

藍色平面$-1.5x_3 = -2.25$，(w0,w1,w2,w3)=(2.25,0,0,-

1.5)，帶入 DABE 四點，可產出(D,A,B,E)= (+,-,-,-)。

反之用(w0,w1,w2,w3)=(-2.25,0,0,1.5)可產出(-,+,+,+)。

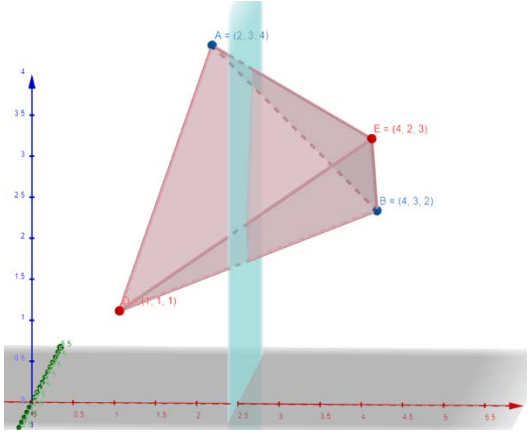


情況 3

藍色平面$-3x_1 = -7.5$，(w0,w1,w2,w3)=(7.5,0,0,-3)，帶入

DABE 四點，可產出(D,A,B,E)= (+,+,-,-)。

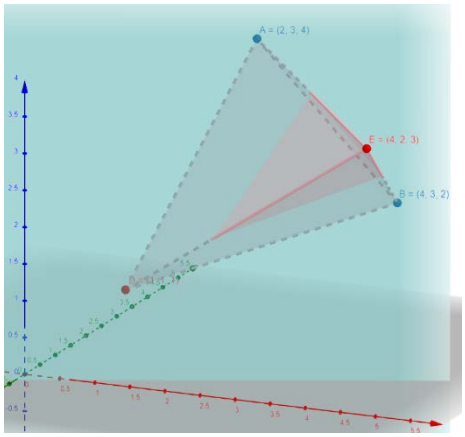
反之用(w0,w1,w2,w3)=(-7.5,0,0,3)可產出(-,-,+,+)。



情況 4

藍色平面$-1.5x_1 + 3.5x_2 - 1.5x_3 = -1$，(w0,w1,w2,w3)=(1,-1.5,3.5,-1.5)，帶入 DABE 四點，可產出(D,A,B,E)= (+,+,+,-)。

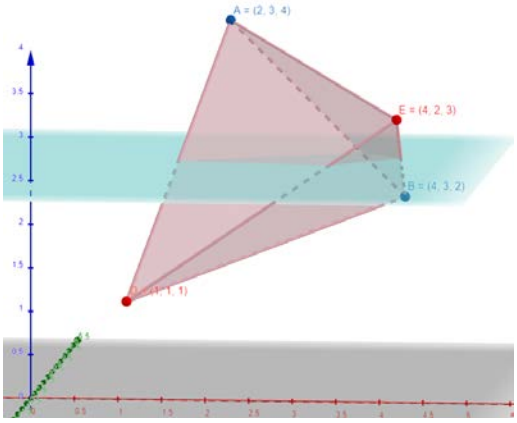
反之用(w0,w1,w2,w3)=(-1,1.5,-3.5,1.5)可產出(-,-,-,+)。



情況 5

藍色平面$-x_3 = -2.5$，(w0,w1,w2,w3)=(2.5,0,0,-1)，帶入 DABE 四點，可產出(D,A,B,E)= (+,-,+,-)。

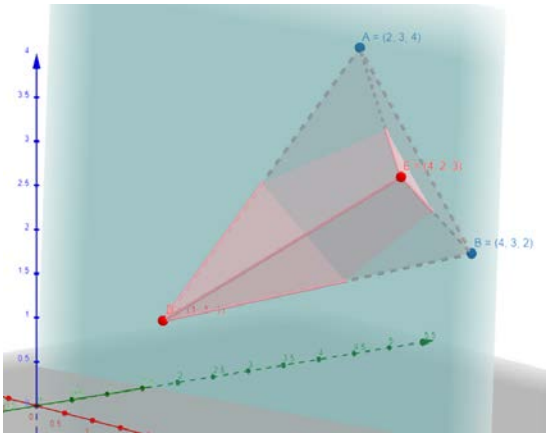
反之用(w0,w1,w2,w3)=(-2.5,0,0,1)可產出(-,+,-,+)。



情況 6

藍色平面$0.47x_1 - 2.71x_2 + 0.14x_3 = -4.36$，(w0,w1,w2,w3)=(4.36,0.47,-2.71,0.14)，帶入 DABE 四點，可產出(D,A,B,E)= (+,-,-,+)。

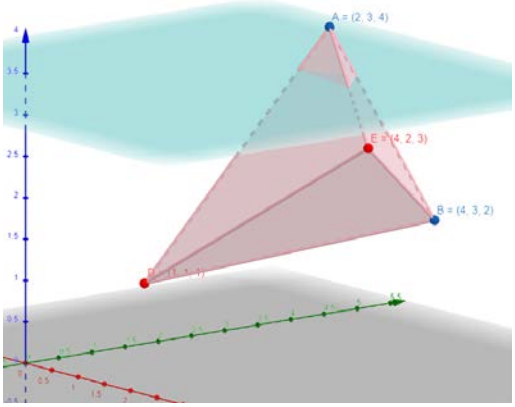
反之用(w0,w1,w2,w3)=(-4.36,-0.47,2.71,-0.14)可產出(-,+,+,-)。



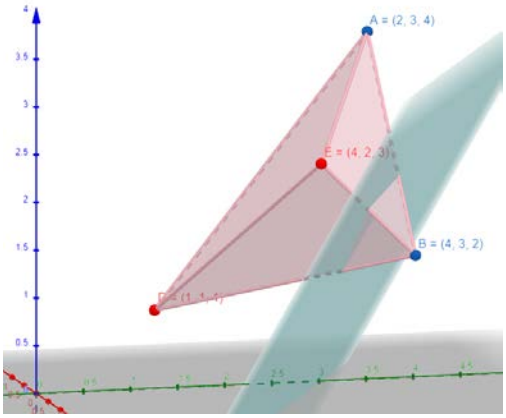
情況 7

藍色平面$-x_3 = -3.5$，(w0,w1,w2,w3)=(3.5,0,0,-1)，帶入 DABE 四點，可產出(D,A,B,E)= (+,-,+,+)。

反之用(w0,w1,w2,w3)=(-3.5,0,0,1)可產出(-,+,-,-)。



情況 8



藍色平面$-0.38x_1 - 0.72x_2 + 0.45x_3 = -2.17$，

(w0,w1,w2,w3)=(2.17,-0.38,-0.72,0.45)，帶入 DABE 四點，

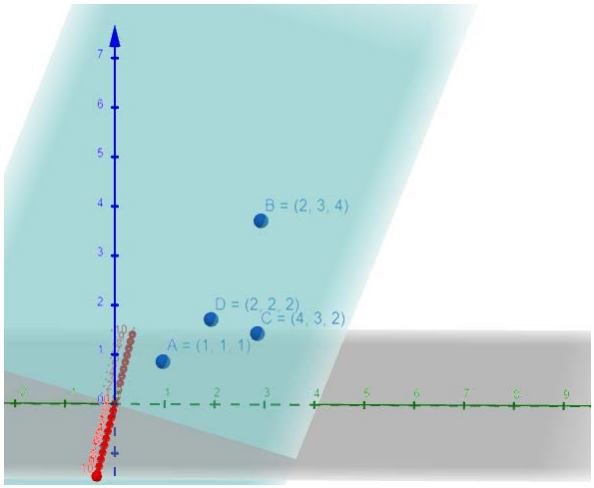
可產出(D,A,B,E)= (+,+,-,+)。

反之用(w0,w1,w2,w3)=(-2.17,0.38,0.72,-0.45)可產出

(-,-,+,-)。

得證，所有情形都能夠產生出來，此四點可以被 shattered。

$$[\mathbf{c}]\ \{(1,1,1),(2,3,4),(4,3,2),(2,2,2)\}$$



由圖可知，此四點共面，假設 A(1,1,1)、B(2,3,4)、C(4,3,2)、D(2,2,2)，利用 ABC 三點

求平面方程式，假設平面法向量為$\vec{n}$。

$$\overrightarrow{AB} = (1,2,3)\ \overrightarrow{AC} = (3,2,1)$$

由於$\overrightarrow{AB} \perp \vec{n}$ 、$\overrightarrow{AC} \perp \vec{n}$，所以$\overrightarrow{AB} \times \overrightarrow{AC}\ //\ \vec{n}$

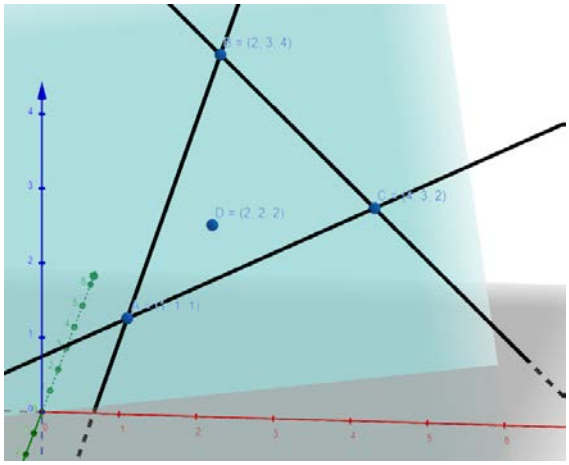
$$\overrightarrow{AB} \times \overrightarrow{AC} = (-1,2,-1)\ //\ \vec{n}$$

假設方程式為$-x_1 + 2x_2 - x_3 = k$，帶入 A(1,1,1)：

$$-x_1 + 2x_2 - x_3 = 0$$

帶入 ABCD 四點都符合，四點共平面。

ABDC 四點共最多可以有$2^4 = 16$種 dichotomies。

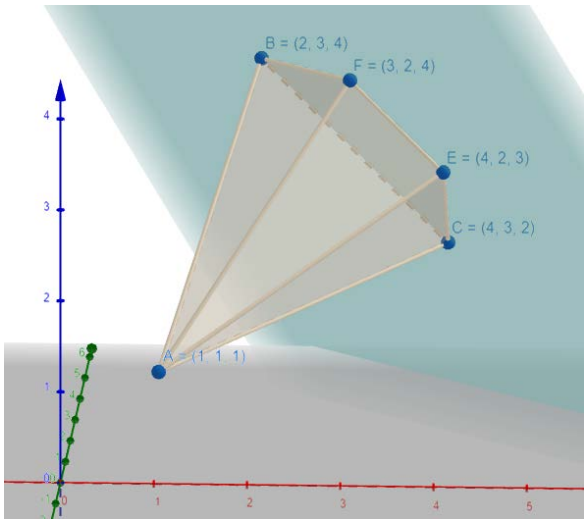


由圖可知，無法將 D 單獨抓出來，因此不能產出以下兩種情況：

A(1,1,1)	B(2,3,4)	C(4,3,2)	D(2,2,2)
+	+	+	-
-	-	-	+

所以此四點無法被 shattered。

[d] $\{(1,1,1),(2,3,4),(4,3,2),(4,2,3),(3,2,4)\}$



由圖可知，BCEF 四點共面，假設 A(1,1,1)、B(2,3,4)、C(4,3,2)、E(4,2,3)、F(3,2,4)，利用 BCE 三點求平面方程式，假設平面法向量為$\vec{n}$。

$$\overrightarrow{BC} = (2,0,-2)\ \overrightarrow{BE} = (2,-1,-1)$$

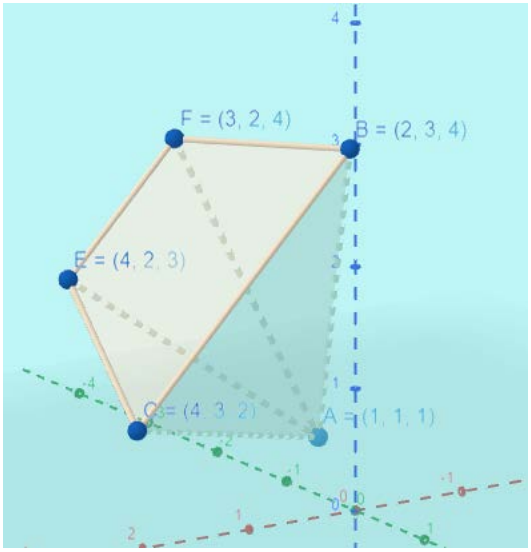
由於$\overrightarrow{BC} \perp \vec{n}$ 、$\overrightarrow{BE} \perp \vec{n}$，所以$\overrightarrow{BC} \times \overrightarrow{BE}\ //\ \vec{n}$

$$\overrightarrow{BC} \times \overrightarrow{BE} = (-1,-1,-1)\ //\ \vec{n}$$

假設方程式為$-x_1 - x_2 - x_3 = k$，帶入 B(2,3,4)：

$$-x_1 - x_2 - x_3 = -9$$

帶入 BCEF 四點都符合，四點共平面。



由圖可知，C、F 無法被單獨一起抓出來，B、E 也無法被單獨一起抓出來，所以不能產出以下四種情況：

A(1,1,1)	B(2,3,4)	C(4,3,2)	E(4,2,3)	F(3,2,4)
+	+	-	+	-
-	-	+	-	+
+	-	+	-	+
-	+	-	+	-

所以此五點無法被 shattered。

且由課本投影片可知$d_{vc} = d + 1$，3D perceptrons 時，$d_{vc} = d + 1 = 4$，所以五個點就一定無法被 shattered。

[e] none of the other choices (none of them can be shattered)

2. ANS [c]

What is the growth function of origin-passing perceptrons in 2D? Those perceptrons are all perceptrons with $w_0 = 0$. Choose the correct answer; explain your choice.

Hint: Put your input vectors on the unit circle, and perhaps think about a polar coordinate system.

[a] $2N + 2$
[b] $2N + 1$
[c] $2N$
[d] $2N - 1$
[e] $2N - 2$

2D origin-passing perceptrons 代表產出的線都會經過原點，w0=0。

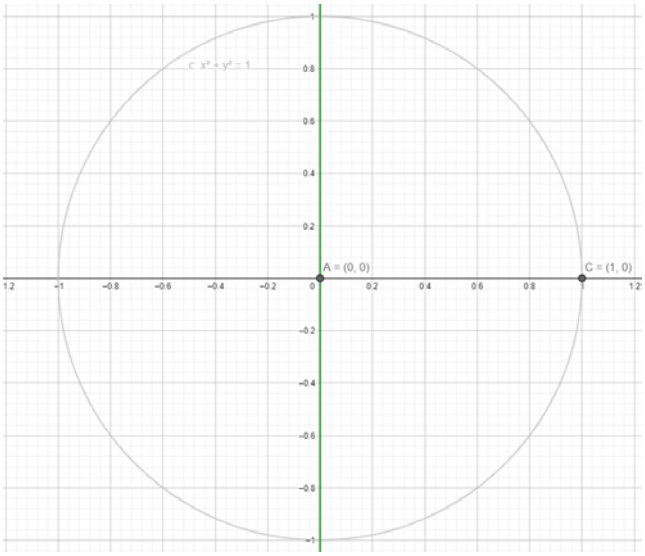
假設 input 的點都在單位圓上，線方程式為$w_1x_1 + w_2x_2 = 0$。

N=1：

假設有 C(0,1)，綠線$x_1 = 0$，(w_0, w_1, w_2)=(0,1,0)

可以讓 C 為正，反之(w_0, w_1, w_2)=(0,-1,0)可以讓

C 為負。故 N=1，$m_H(1) = 2$。



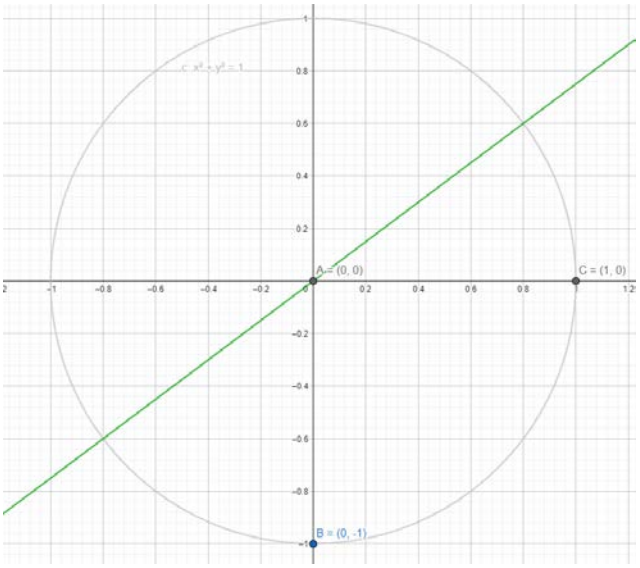
N=2：

假設有 C(0,1)、B(0,-1)兩點，綠線$-0.75x_1 +$

$x_2 = 0$，(w_0, w_1, w_2)=(0, -0.75,1)可以讓 BC 為

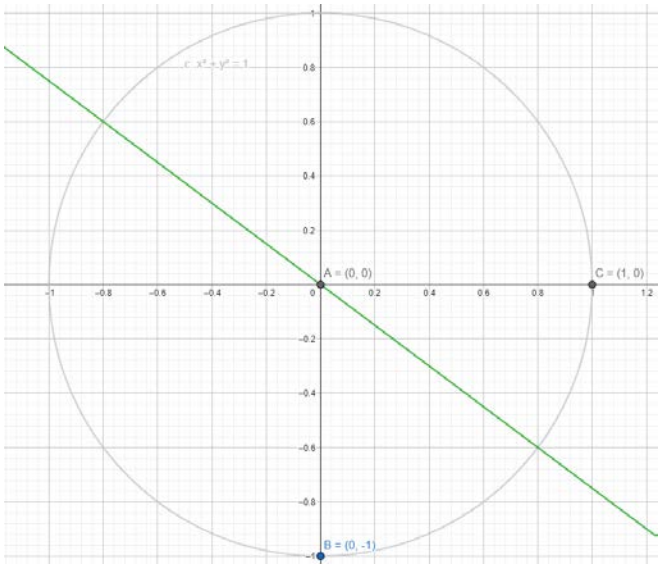
負，反之(w_0, w_1, w_2)= (0, 0.75,-1)可以讓 BC 為

正。



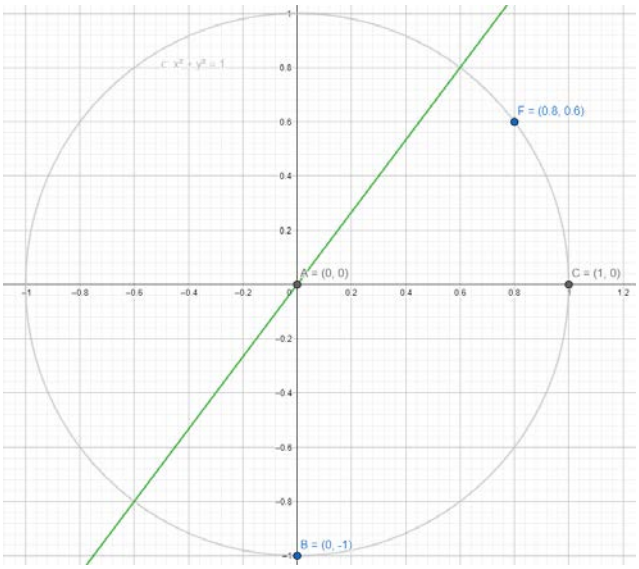
綠線$0.75x_1 + x_2 = 0$，(w_0, w_1, w_2)=(0, 0.75,1)可以讓 C 為正、B 為負，反之(w_0, w_1, w_2)= (0, -0.75,-1)可以讓 C 為負、B 為正。

故 N=2，$m_H(2) = 4$。

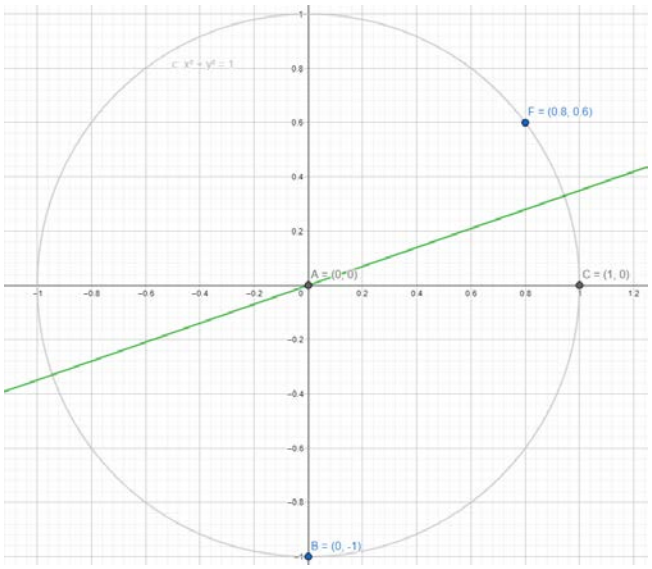


N=3：

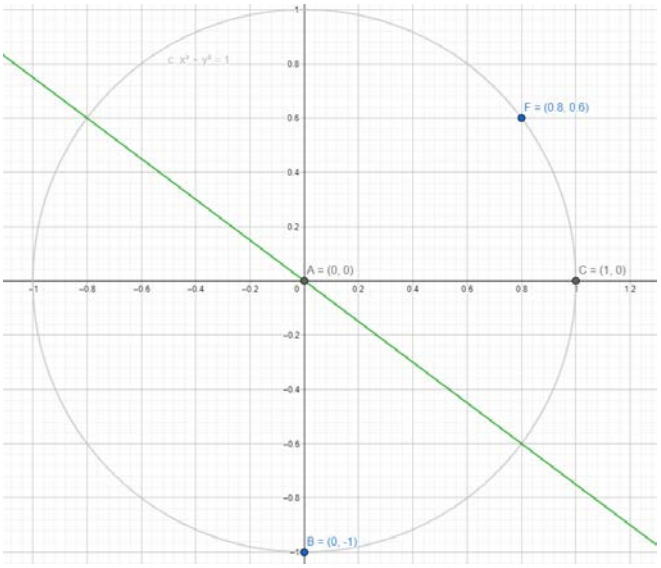
假設有 C(0,1)、B(0,-1)、F(0.8,0.4)三點，綠線$-\frac{4}{3}x_1 + x_2 = 0$，$(w_0, w_1, w_2)$=(0,$-\frac{4}{3}$,1)可以讓 BCF 為負，反之$(w_0, w_1, w_2)$= (0,$\frac{4}{3}$,-1)可以讓 BCF 為正。



綠線$-0.3490374001888x_1 + x_2 = 0$，$(w_0, w_1, w_2)$=(0, -0.3490374001888,1)可以讓 F 為正、BC 為負，反之(w_0, w_1, w_2)= (0, 0.3490374001888,-1)可以讓 F 為負、BC 為正。



綠線$0.75x_1 + x_2 = 0$，(w_0, w_1, w_2)=(0, 0.75,1)可以讓 CF 為正、B 為負，反之(w_0, w_1, w_2)= (0, -0.75,-1)可以讓 CF 為負、B 為正。



無法將 C 點單獨拉出來形成(C,B,F)=(+,-,-)或是(-,+,+)。所以故 N=3，$m_H(3) = 6$。

整理上述情形發現：

$$\text{N=1，} m_H(1) = 2 = 2 * 1\text{。}$$

$$\text{N=2，} m_H(2) = 4 = 2 * 2\text{。}$$

$$\text{N=3，} m_H(3) = 6 = 2 * 3\text{。}$$

得出：

$$m_H(N) = 2N$$

選 C。

Donut Hypothesis Set

3. ANS [a]

The "donut" hypothesis set in $\mathbb{R}^d$ contains hypothesis parameterized by two positive numbers a and b, where

$$h(\mathbf{x}) = \begin{cases} +1 & \text{if } a \leq \sum_{i=1}^{d} x_i^2 \leq b, \\ -1 & \text{otherwise.} \end{cases}$$

What is the growth function of the hypothesis set? Choose the correct answer; explain your choice.

[a] $\binom{N+1}{2} + 1$

[b] $\binom{N+1}{3} + 1$

[c] $\binom{N+1}{6} + 1$

[d] $\binom{N+1}{d} + 1$

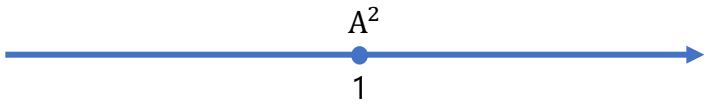
[e] none of the other choices

d=1：

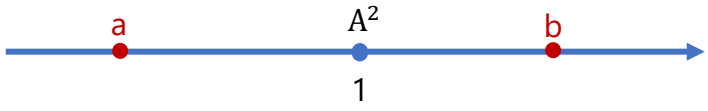
$$h(x) = \begin{cases} +1 & if\ a \leq x_1^2 \leq b, \\ -1 & otherwise. \end{cases}$$

N=1：

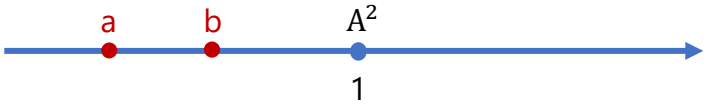
假設一點 A 為 1，



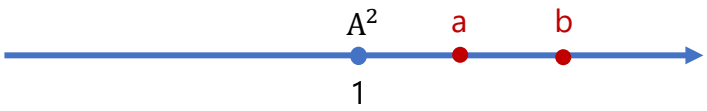
有兩種可能一個是讓 h(A)=+1 或是-1，h(A)=+1 時如下圖：



當 h(A)=-1 時如下圖：

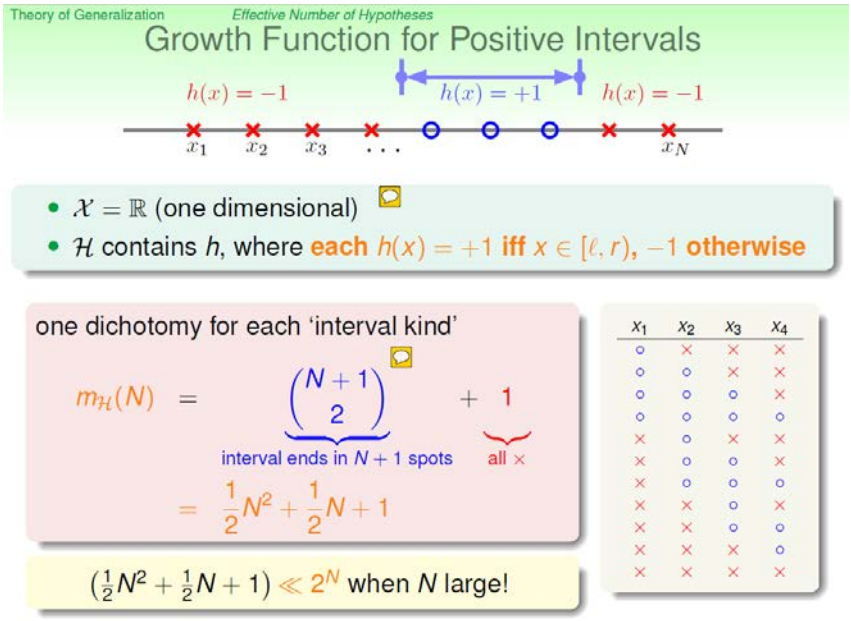


或是



共有兩種結果，$m_H(1) = 2$。

可以發現這其實是一個 positive interval 的情況，如課程投影片：



假設有 N 個點，就會產生 N+1 個區間，在 N+1 個區間中選出兩個來分別安置 a 與 b，故C_2^{N+1}。最後是讓所有點都為 -1 的情形，故要再加上 1。

可知 d=1 時，$m_H(N) = C_2^{N+1} + 1$。

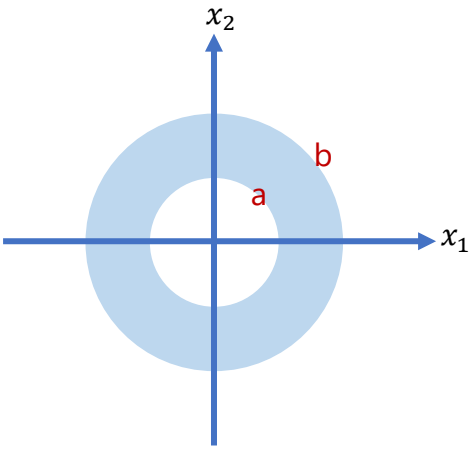
d=2：

$$h(x) = \begin{cases} +1 & if\ a \le x_1^2 + x_2^2 \le b, \\ -1 & otherwise. \end{cases}$$

二維的圖型如右圖，會是一個甜甜圈的形狀。

藍色區域為 a 與 b 為起來的區間，

若點落在藍色區域，會是+1，其他地方為-1。



假設 N=3，A(2,2)、B(3,3)、C(5,5)：

$$A_1^2 + A_2^2 = 8$$
$$B_1^2 + B_2^2 = 18$$
$$C_1^2 + C_2^2 = 50$$

故一樣是 positive interval 的情形，計算過後三點變成三個值，可以一樣拉成一維的狀況去思考。所以 d=2 時，$m_H(N) = C_2^{N+1} + 1$。

d=3：

$$h(x) = \begin{cases} +1 & if\ a \le x_1^2 + x_2^2 + x_3^2 \le b, \\ -1 & otherwise. \end{cases}$$

假設 N=3，A(2,2,2)、B(3,3,3)、C(5,5,5)：

$$A_1^2 + A_2^2 + A_3^2 = 12$$
$$B_1^2 + B_2^2 + B_3^2 = 27$$
$$C_1^2 + C_2^2 + C_3^2 = 75$$

一樣也是 positive interval 的情形，計算過後三點變成三個值，可以拉成一維的狀況去思考。所以 d=3 時，$m_H(N) = C_2^{N+1} + 1$。

整理上面可發現：

d=1，$m_H(N) = C_2^{N+1} + 1$

d=2，$m_H(N) = C_2^{N+1} + 1$

d=3，$m_H(N) = C_2^{N+1} + 1$

所以可知不管 d 是多少，經由$\sum_{i=1}^{d} x_i^2$的計算都會將點化成一個值，所以都是 positive interval 的情形，此 h(x)的成長函數$m_H(N) = C_2^{N+1} + 1$，答案為 a。

4. ANS [d]

Following the previous problem, what is the VC dimension of the donut hypothesis set? Choose the correct answer; explain your choice.

[a] d
[b] 6
[c] 3
[d] 2
[e] none of the other choices

VC Dimension

the formal name of **maximum non-**break point d_{VC}
= (minimum break point k - 1)

Definition

VC dimension of $\mathcal{H}$, denoted $d_{VC}(\mathcal{H})$ is

largest N for which $m_{\mathcal{H}}(N) = 2^N$
(the **most** inputs $\mathcal{H}$ that can shatter)

- d_{VC} = 'minimum k' - 1

$N \leq d_{VC} \implies \mathcal{H}$ can shatter some N inputs
$k > d_{VC} \implies k$ is a break point for $\mathcal{H}$

if $N \geq 2, d_{VC} \geq 2, m_{\mathcal{H}}(N) \leq N^{d_{VC}}$

根據課本投影片的定義，VC Dimension = max non-break point。

根據上一題得到的結果：

$$\text{h(x)的成長函數} m_H(N) = C_2^{N+1} + 1$$

$$m_H(1) = C_2^2 + 1 = 2 = 2^1$$
$$m_H(2) = C_2^3 + 1 = 4 = 2^2$$
$$m_H(3) = C_2^4 + 1 = 7 < 2^3$$

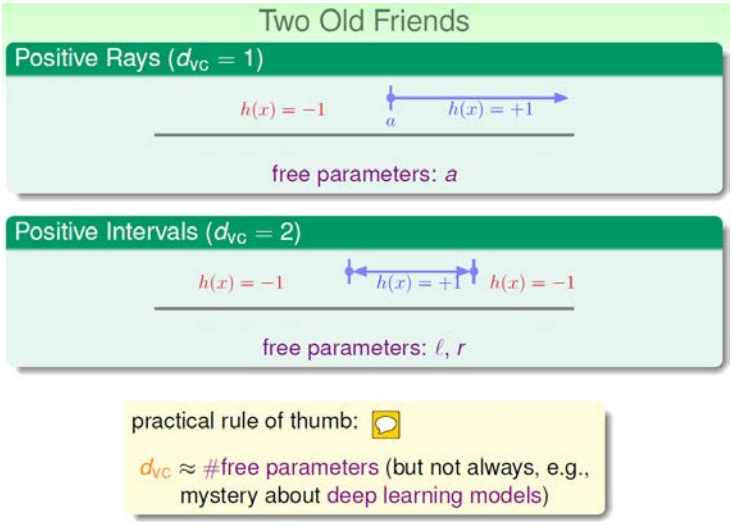
可以得知 N=3 是最小的 break point，N=2 就是最大的 non-break point，

所以$d_{VC} = 2$，答案選 d。

More on VC Dimension

5. ANS [e]

Which of the following hypothesis set is of a different VC dimension, compared to others? Choose the correct answer; explain your choice.

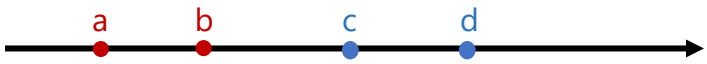


根據投影片說除了 deep learning models 以外，通常$d_{VC} = free\ parameters$的個數。

所以以下解題方式為找出該情形有幾個 free parameters。

[a] Unions of two positive intervals over $x \in \mathbb{R}$, which returns +1 if x is within at least one of the intervals.

有兩個 positive intervals，所以會有四個 free parameters，可能情況如下圖：



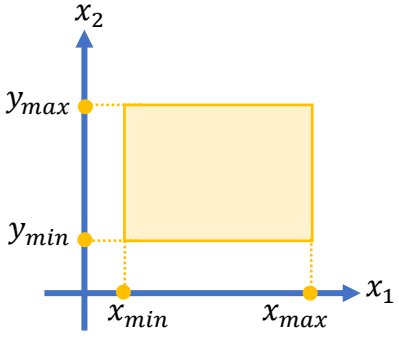
$$h(x) = \begin{cases} +1 & if\ a \le x_1^2 \le b\ or\ \ c \le x_1^2 \le d, \\ -1 & otherwise. \end{cases}$$

有四個 free parameters，所以$d_{VC} = 4$。

[b] Axis-aligned rectangle classifiers for $\mathbf{x} \in \mathbb{R}^2$, which returns +1 if $\mathbf{x}$ is inside a rectangle whose edges are parallel to the axes of $\mathbb{R}^2$.

若 **x** 在與軸平行的長方形內部就是+1，要產生一個長方形也需要四個 parameters。

如右圖：



有四個 free parameters，所以$d_{VC} = 4$。

[c] Positively-biased perceptrons over $\mathbf{x} \in \mathbb{R}^4$, which contains perceptrons with $w_0 > 0$.

4D Positively-biased perceptrons，有五個 parameters w_0 、w_1 、w_2 、w_3 、w_4，但是限定$w_0 > 0$。由於 perceptrons 最後是取 sign，所以每個 parameter 都有兩種可能就是+或-，而$w_0 > 0$相當於限制住一個 parameter 的結果唯一，所以w_0就不是 free parameter。所以只有四個 free parameters，$d_{VC} = 4$。

[d] Polynomial hypotheses of degree 3 for $x \in \mathbb{R}$, which are of the form

$$h(x) = \text{sign}(\sum_{i=0}^{3} w_i x^i).$$

將式子展開如下：

$$h(x) = sign(w_0 x^0 + w_1 x^1 + w_2 x^2 + w_3 x^3)$$

可見有四個 free parameters w_0 、w_1 、w_2 、w_3，$d_{VC} = 4$。

[e] none of the other choices

由於上述 ABCD 選項的d_{VC}都為 4，所以選 e。

6. ANS [d]

For a finite hypothesis set $\mathcal{H}$ with 1126 binary classifiers, what is the largest possible value of $d_{\text{vc}}(\mathcal{H})$? Choose the correct answer; explain your choice.

[a] 1126
[b] 112
[c] 11
[d] 10
[e] 1

H 有 1126 個假說(1126 binary classifiers)，代表一定可以 shatter 掉十個 free parameters，因為$2^{10} = 1024 < 1126$，所以最大可能的$d_{VC}(H) = 10$。

Deviation from Optimal Hypothesis

7. ANS [d]

Recall that the multiple-bin Hoeffding bound quantifies the BAD probability from *any* hypothesis h in the hypothesis set. That is,

$$\mathbb{P}\left[\exists h \in \mathcal{H} \text{ s.t. } |E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon\right] \leq 2M \exp\left(-2\epsilon^2 N\right).$$

Define the best-E_{in} hypothesis

$$g = \text{argmin}_{h\in\mathcal{H}} E_{\text{in}}(h)$$

and the best-E_{out} hypothesis (which is optimal but can only be obtained by a "cheating" algorithm)

$$g_* = \text{argmin}_{h\in\mathcal{H}} E_{\text{out}}(h).$$

Using the multiple-bin Hoeffding bound above, with probability more than $1-\delta$, which of the following is an upper bound of $E_{\text{out}}(g) - E_{\text{out}}(g_*)$? Choose the correct answer; explain your choice.

[a] $\sqrt{\frac{1}{2N}\ln\left(\frac{M}{\delta}\right)}$

[b] $\sqrt{\frac{1}{N}\ln\left(\frac{2M}{\delta}\right)}$

[c] $\sqrt{\frac{1}{2N}\ln\left(\frac{2M}{\delta}\right)}$

[d] $2\sqrt{\frac{1}{2N}\ln\left(\frac{2M}{\delta}\right)}$

[e] $\sqrt{\frac{1}{N}\ln\left(\frac{M}{\delta}\right)}$

with probability $\geq 1-\delta$, $|E_{in}(h) - E_{out}(h)| \leq \epsilon$

$$\text{set } \delta = 2Mexp(-2\epsilon^2 N)$$

$$\frac{\delta}{2M} = exp(-2\epsilon^2 N)$$

$$\frac{\delta}{2M} = \frac{1}{exp(2\epsilon^2 N)}$$

$$\frac{2M}{\delta} = exp(2\epsilon^2 N)$$

兩邊同取以e為底的log，也就是ln：

$$ln(\frac{2M}{\delta}) = 2\epsilon^2 N$$

$$\epsilon = \sqrt{\frac{1}{2N} ln(\frac{2M}{\delta})}$$

將$|E_{in}(h) - E_{out}(h)|$ 換成 $|E_{out}(g) - E_{out}(g^*)|$：

根據題目可知：

$$E_{in}(g) \leq E_{in}(g^*)$$
$$E_{out}(g) \geq E_{out}(g^*)$$

$$|E_{in}(h) - E_{out}(h)| \leq \epsilon$$

將絕對值去掉會產生兩種情況：

1. $E_{in}(h) - E_{out}(h) \leq \epsilon$
   $E_{in}(h) \leq \epsilon + E_{out}(h)$

   換成g^*

   $E_{in}(g^*) \leq \epsilon + E_{out}(g^*)$

2. $E_{out}(h) - E_{in}(h) \leq \epsilon$
   $E_{out}(h) \leq \epsilon + E_{in}(h)$

   換成g

   $E_{out}(g) \leq \epsilon + E_{in}(g)$

由情況 2 結果$E_{out}(g) \leq \epsilon + E_{in}(g)$，且已知 $E_{in}(g) \leq E_{in}(g^*)$：

$$E_{out}(g) \leq \epsilon + E_{in}(g) \leq \epsilon + E_{in}(g^*)$$

由情況 1 結果$E_{in}(g^*) \leq \epsilon + E_{out}(g^*)$可知：

$$E_{out}(g) \leq \epsilon + E_{in}(g) \leq \epsilon + E_{in}(g^*) \leq \epsilon + \left(\epsilon + E_{out}(g^*)\right) = 2\epsilon + E_{out}(g^*)$$

得出：

$$E_{out}(g) \leq 2\epsilon + E_{out}(g^*)$$
$$E_{out}(g) - E_{out}(g^*) \leq 2\epsilon$$

所以 $|E_{out}(g) - E_{out}(g^*)| \leq 2\epsilon = 2\sqrt{\frac{1}{2N} ln(\frac{2M}{\delta})}$，選 d。

VC Bound

8. ANS [b]

When using the positive ray model taught in class, given $\epsilon = 0.1$ and $\delta = 0.1$, among the five choices, what is the smallest N such that the BAD probability of the VC bound

$$\mathbb{P}\left[\exists h \in \mathcal{H} \text{ s.t. } |E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon\right] \leq 4 m_{\mathcal{H}}(2N) \exp\left(-\frac{1}{8}\epsilon^2 N\right)$$

is $\leq \delta$? Choose the correct answer; explain your choice.

[a] 10000
[b] 11000
[c] 12000
[d] 13000
[e] 14000

題目是要找到一個最小的 N，讓壞事情發生($|E_{in}(h) - E_{out}(h)| > \epsilon$)的機率≤ δ = 0.1，且題目給$\epsilon = 0.1$。

解題方向是一開始先假設<=delta=0.1 這件事成立，將 0.1 帶入。將式子化簡後，最後找到一個最小的 N 讓 bound 能 ≤ ϵ 並最接近 0.1，這樣就符合題目要求的條件。

$$4 m_{\mathcal{H}}(2N) \exp\left(-\frac{1}{8}\epsilon^2 N\right) \leq \delta = 0.1$$

由於 positive ray 的 $d_{vc} = 1$, $m_{\mathcal{H}}(2N) = 2N + 1$：

$$\Rightarrow 4(2N+1)^{d_{vc}} \exp\left(-\frac{1}{8}\epsilon^2 N\right) \leq 0.1$$

$$\Rightarrow 4(2N+1) \exp\left(-\frac{1}{8}\epsilon^2 N\right) \leq 0.1$$

$$\exp\left(-\frac{1}{8}\epsilon^2 N\right) \leq \frac{1}{40(2N+1)}$$

$$\frac{1}{\exp\left(\frac{1}{8}\epsilon^2 N\right)} \leq \frac{1}{40(2N+1)}$$

$$40(2N+1) \leq \exp\left(\frac{1}{8}\epsilon^2 N\right)$$

取 ln：

$$\Rightarrow \quad \ln(80N+40) \le \frac{1}{8}\epsilon^2 N$$

$$\frac{8}{N}\ln(80N+40) \le \epsilon^2 = (0.1)^2$$

$$\Rightarrow \quad \underline{\frac{8}{N}\ln(80N+40) \le 0.01} \quad (A)$$

(A) $N=10000$ 代入 (A)

$$\frac{8}{10000}\ln(800000+40) = 0.01087393$$

(B) $N=11000$ 代入 (A)

$$\frac{8}{11000}\ln(880000+40) = 0.00995547$$

(C) $N=12000$ 代入 (A)

$$\frac{8}{12000}\ln(80\times12000+40) = 0.00918315$$

(D) $N=13000$ 代入 (A)

$$\frac{8}{13000}\ln(80\times13000+40) = 0.00852601$$

(E) $N=14000$ 代入 (A)

$$\frac{8}{14000}\ln(80\times14000+40) = 0.00795935$$

綜合上述可得 B 選項出來的值符合 ≤ 0.01

並且最接近 0.01，選 B。

Hessian and Newton Method

9. ANS [b]

Let $E(\mathbf{w}): \mathbb{R}^d \to \mathbb{R}$ be a function. Denote the gradient $\mathbf{b}_E(\mathbf{w})$ and the Hessian $\mathbf{A}_E(\mathbf{w})$ by

$$\mathbf{b}_E(\mathbf{w}) = \nabla E(\mathbf{w}) = \begin{bmatrix} \frac{\partial E}{\partial w_1}(\mathbf{w}) \\ \frac{\partial E}{\partial w_2}(\mathbf{w}) \\ \vdots \\ \frac{\partial E}{\partial w_d}(\mathbf{w}) \end{bmatrix}_{d\times 1} \quad \text{and} \quad \mathbf{A}_E(\mathbf{w}) = \begin{bmatrix} \frac{\partial^2 E}{\partial w_1^2}(\mathbf{w}) & \frac{\partial^2 E}{\partial w_1 \partial w_2}(\mathbf{w}) & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_d}(\mathbf{w}) \\ \frac{\partial^2 E}{\partial w_2 \partial w_1}(\mathbf{w}) & \frac{\partial^2 E}{\partial w_2^2}(\mathbf{w}) & \cdots & \frac{\partial^2 E}{\partial w_2 \partial w_d}(\mathbf{w}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial w_d \partial w_1}(\mathbf{w}) & \frac{\partial^2 E}{\partial w_d \partial w_2}(\mathbf{w}) & \cdots & \frac{\partial^2 E}{\partial w_d^2}(\mathbf{w}) \end{bmatrix}_{d\times d}.$$

Then, the second-order Taylor's expansion of $E(\mathbf{w})$ around $\mathbf{u}$ is:

$$E(\mathbf{w}) \approx E(\mathbf{u}) + \mathbf{b}_E(\mathbf{u})^T(\mathbf{w}-\mathbf{u}) + \frac{1}{2}(\mathbf{w}-\mathbf{u})^T \mathbf{A}_E(\mathbf{u})(\mathbf{w}-\mathbf{u}).$$

Suppose $\mathbf{A}_E(\mathbf{u})$ is positive definite. What is the optimal direction $\mathbf{v}$ such that $\mathbf{w} \leftarrow \mathbf{u}+\mathbf{v}$ minimizes the right-hand-side of the Taylor's expansion above? Choose the correct answer; explain your choice.

Note that iterative optimization with $\mathbf{v}$ is generally called Newton's method.

[a] $+(\mathbf{A}_E(\mathbf{u}))^{-1}\mathbf{b}_E(\mathbf{u})$
[b] $-(\mathbf{A}_E(\mathbf{u}))^{-1}\mathbf{b}_E(\mathbf{u})$
[c] $+(\mathbf{A}_E(\mathbf{u}))^{+1}\mathbf{b}_E(\mathbf{u})$
[d] $-(\mathbf{A}_E(\mathbf{u}))^{+1}\mathbf{b}_E(\mathbf{u})$
[e] none of the other choices

$$E(w) = E(u) + b_E(u)^T(w-u) + \frac{1}{2}(w-u)^T A_E(u)(w-u)$$

$A_E(u)$ 為正定矩陣 ⇒ 則取 $E(w)$ 的極值為極小值

$E(w)$ 有極值的必要條件為在極值處一階導數為 0

根據 Taylor's expansion 二階展開，得 $\nabla E(w)$：

$$E(w) = E(u) + b_E(u)^T \cdot w - b_E(u)^T \cdot u + \frac{1}{2} w^T A_E(u) w - u^T A_E(u) w + \frac{1}{2} u^T A_E(u) u$$

$$= \frac{1}{2} w^T A_E(u) w + \underbrace{(b_E(u) - A_E(u)u)}_{b^T} w + \underbrace{\left[E(u) - b_E(u)^T \cdot u + \frac{1}{2} u^T A_E(u) u\right]}_{c}$$

$$= \frac{1}{2} w^T A_E(u) w + b^T w + c$$

對w微分

$$\Rightarrow \nabla E(w) = A_E(u)w + b$$

$$= A_E(u)w + b_E(u) - A_E(u)u = 0$$

$$A_E(u) \cdot w = A_E(u) \cdot u - b_E(u)$$

$$w = u - \frac{b_E(u)}{A_E(u)}$$

$$w = u - A_E(u)^{-1} \cdot b_E(u)$$

$$v = -A_E(u)^{-1} \cdot b_E(u)$$

$$\therefore \text{得證}$$

10. ANS [d]

Following the previous problem, considering minimizing $E_{\text{in}}(\mathrm{w})$ in logistic regression problem with Newton's method. Consider a data set $\{(\mathrm{x}_n, y_n)\}_{n=1}^{N}$ with the cross-entropy error function for E_{in}:

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

For any given w_t, let

$$h_t(\mathrm{x}) = \frac{1}{1+\exp(\mathrm{w}_t^T \mathrm{x})}.$$

What is the Hessian $\mathrm{A}_E(\mathrm{w}_t)$ with $E = E_{\text{in}}$? Choose the correct answer; explain your choice.

[a] $\frac{1}{N}\sum_{n=1}^{N} \mathrm{x}_n \mathrm{x}_n^T$

[b] $\frac{1}{N}\sum_{n=1}^{N} \|\mathrm{w}_t\|^2 \mathrm{x}_n \mathrm{x}_n^T$

[c] $\frac{1}{N}\sum_{n=1}^{N} h_t^2(y_n \mathrm{x}_n) \mathrm{x}_n \mathrm{x}_n^T$

[d] $\frac{1}{N}\sum_{n=1}^{N} h_t(y_n \mathrm{x}_n) h_t(-y_n \mathrm{x}_n) \mathrm{x}_n \mathrm{x}_n^T$

[e] none of the other choices

$$h(X) = \frac{1}{1+\exp(w_t^T X)}$$

$$E_{in}(w) = \frac{1}{N}\sum_{n=1}^{N} \ln\left(1+\exp(-y_n w^T x_n)\right)$$

由投影片中可知

$$\nabla E_{in}(w) = \frac{1}{N}\sum_{n=1}^{N} \left(\frac{\partial \ln(\square)}{\partial \square}\right)\left(\frac{\partial (1+\exp(\circ))}{\partial \circ}\right)\left(\frac{\partial -y_n w^T x_n}{\partial w_i}\right)$$

$$= \frac{1}{N}\sum_{n=1}^{N} \left(\frac{1}{\square}\right)(\exp(\circ))(-y_n x_{n,i})$$

$$= \frac{1}{N}\sum_{n=1}^{N} \left(\frac{\exp(\circ)}{1+\exp(\circ)}\right)(-y_n x_{n,i})$$

$$= \frac{1}{N}\sum_{n=1}^{N} \theta(-y_n w^T x_n)(-y_n x_n)$$

$$A_E(w_t) = \nabla(\nabla E_{in}(w)) \quad \because h(x) = \frac{1}{1+\exp(w_t^T x)}$$

$$= \nabla\left(\frac{1}{N}\sum_{n=1}^{N} h(-y_n x_n)(-y_n x_n)\right)$$

$$= \nabla\left(\frac{1}{N}\sum_{n=1}^{N} \left(\frac{1}{1+e^{-y_n w_t^T x_n}}\right)(-y_n x_n)\right)$$

$$= \frac{1}{N}\sum_{n=1}^{N} -\frac{-e^{-y_n w_t^T x_n}}{(1+e^{-y_n w_t^T x_n})^2} \cdot (-y_n x_n) \cdot (-y_n x_n)$$

$$= \frac{1}{N}\sum_{n=1}^{N} \frac{e^{-y_n w_t^T x_n}}{(1+e^{-y_n w_t^T x_n})^2} \cdot x_n x_n^T$$

$$= \frac{1}{N}\sum_{n=1}^{N} \frac{e^{-y_n w_t^T x_n} \times \frac{1}{e^{-y_n w_t^T x_n}}}{(1+e^{-y_n w_t^T x_n}) \times \frac{1}{e^{-y_n w_t^T x_n}}} \times \frac{1}{1+e^{-y_n w_t^T x_n}} \times x_n x_n^T$$

$$= \frac{1}{N}\sum_{n=1}^{N} \frac{1}{\frac{1}{e^{-y_n w_t^T x_n}+1}} \times \frac{1}{1+e^{-y_n w_t^T x_n}} \times x_n x_n^T$$

$$= \frac{1}{N}\sum_{n=1}^{N} \left(\frac{1}{e^{-y_n w_t^T x_n}+1}\right)^{-1} \times h(-y_n x_n) \times x_n x_n^T$$

$$= \frac{1}{N}\sum_{n=1}^{N} \frac{1}{e^{y_n w_t^T x_n}+1} \times h(-y_n x_n) \times x_n x_n^T$$

$$= \frac{1}{N}\sum_{n=1}^{N} h(y_n x_n) \times h(-y_n x_n) \times x_n x_n^T$$ ※得證

11. ANS [c]

In the lecture, we learned that the linear regression weights w must satisfy the normal equation $X^T X w = X^T y$. If $X^T X$ is not invertible, we cannot compute $w_{LIN} = (X^T X)^{-1} X^T y$. Then, one possible solution is

$$w_{LIN} = X^{\dagger} y,$$

where $X^{\dagger}$ is called the Moore-Penrose pseudo-inverse. Let $X = U \Sigma V^T$ be the singular value decomposition of the N by $d+1$ matrix X, with U and V being unitary matrices. The Moore-Penrose pseudo-inverse $X^{\dagger} = V \Sigma^{\dagger} U^T$ is a $d+1$ by N matrix, where $\Sigma^{\dagger}[i][n] = \frac{1}{\Sigma[n][i]}$ when $\Sigma[n][i]$ is nonzero, and 0 otherwise. Which of the following statements related to $X^{\dagger}$ is incorrect? Choose the **incorrect** statement; explain your choice.

[a] $(X^T X)^{-1} X^T = X^{\dagger}$ when $X^T X$ is invertible.

∵ unitary matric

∴ $UU^T = I = VV^T = VV^{-1} \Rightarrow V^T = V^{-1}$

[a] $(X^T X)^{-1} X^T = X^{\dagger}$

$$X^T X = (U \Sigma V^T)^T (U \Sigma V^T)$$
$$= V \Sigma^T U^T U \Sigma V^T$$
$$= V \Sigma^T \Sigma V^T$$

$$(V \Sigma^T \Sigma V^T)^{-1} X^T = (V^T)^{-1} (\Sigma^T \Sigma)^{-1} (V)^{-1} (V \Sigma^T U^T)$$
$$= (V^T)^{-1} (\Sigma^T \Sigma)^{-1} \Sigma^T U^T$$
$$= (V^T)^{-1} \cdot \Sigma^{\dagger} \cdot U^T \quad V^T = V^{-1}$$
$$= V \cdot \Sigma^{\dagger} \cdot U^T = X^{\dagger}$$

得證

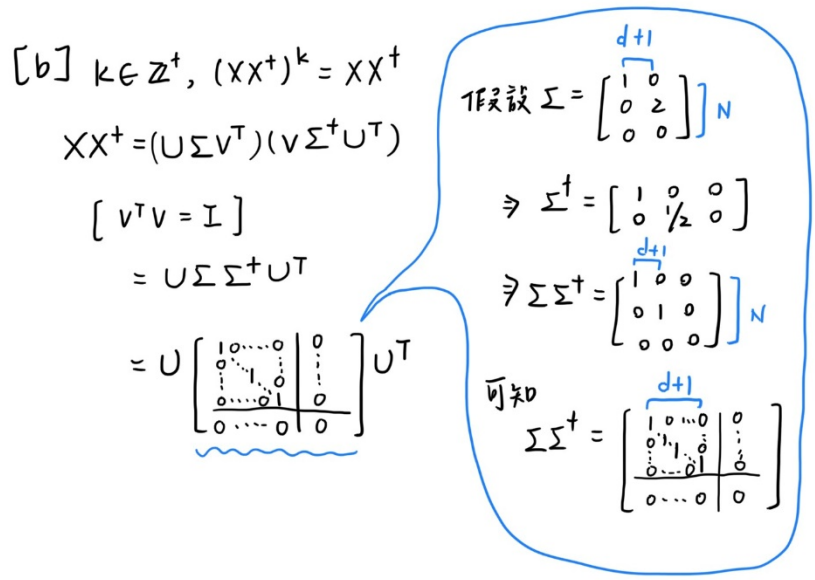
[b] For any $k \in \mathbb{Z}^+$, $(XX^{\dagger})^k = XX^{\dagger}$.

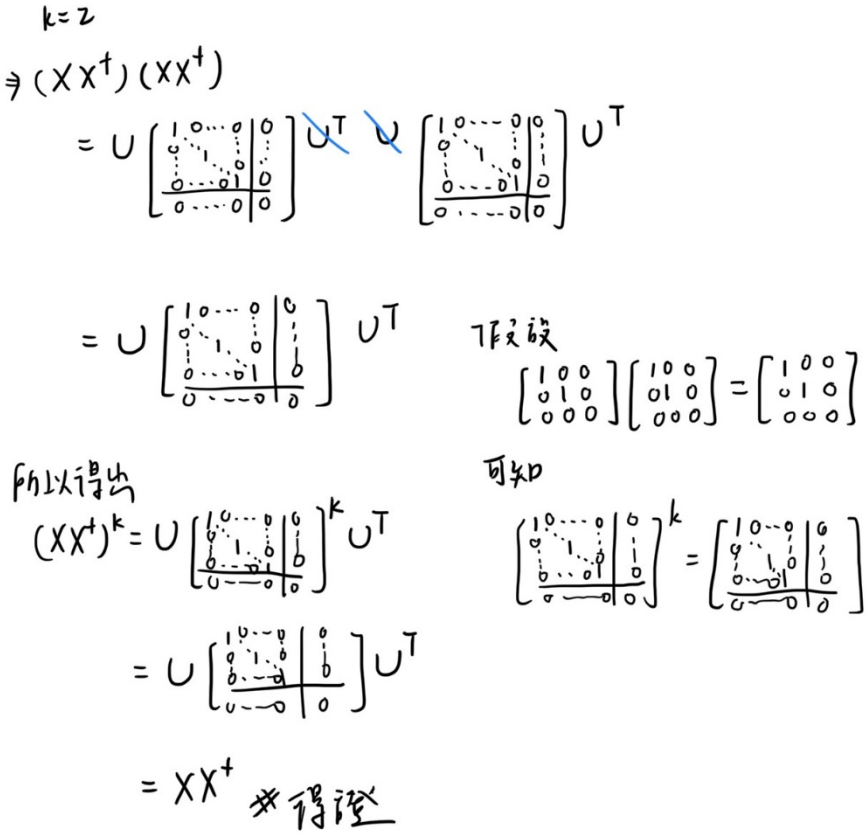
```python
# b
import numpy as np
import math
# N > d+1
X = [[2,4,6],[1,3,5],[1,4,7],[3,5,7]]
X = np.asarray(X)
X_cross = np.linalg.pinv(X)
XX_cross = np.dot(X,X_cross)
# k = 2,4
XX_cross_2 = np.dot(XX_cross, XX_cross)
XX_cross_4 = np.dot(XX_cross_2, XX_cross_2)
print(XX_cross)
print(XX_cross_2)
print(XX_cross_4)
```

```
[[ 0.26666667  0.13333333  0.13333333  0.4       ]
 [ 0.13333333  0.23333333  0.4         0.03333333]
 [ 0.13333333  0.4         0.73333333 -0.13333333]
 [ 0.4         0.03333333 -0.13333333  0.76666667]]
[[ 0.26666667  0.13333333  0.13333333  0.4       ]
 [ 0.13333333  0.23333333  0.4         0.03333333]
 [ 0.13333333  0.4         0.73333333 -0.13333333]
 [ 0.4         0.03333333 -0.13333333  0.76666667]]
[[ 0.26666667  0.13333333  0.13333333  0.4       ]
 [ 0.13333333  0.23333333  0.4         0.03333333]
 [ 0.13333333  0.4         0.73333333 -0.13333333]
 [ 0.4         0.03333333 -0.13333333  0.76666667]]
```

由程式可知， For any $k \in \mathbb{Z}^+$, $(XX^\dagger)^k = XX^\dagger$成立。

數學證明如下：





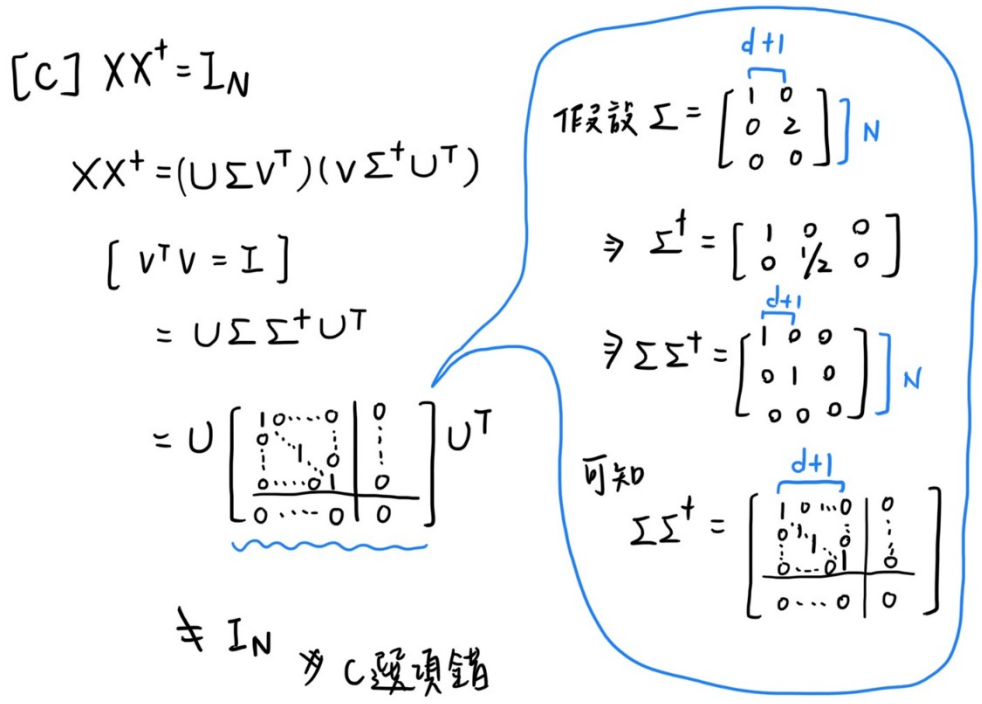
[c] $XX^{\dagger} = I_N$, the N by N identity matrix.

```
# c
import numpy as np
# N > d+1
X = [[2,4,6],[1,3,5],[1,4,7],[3,5,7]]
X = np.asarray(X)
X_cross = np.linalg.pinv(X)
np.dot(X,X_cross)
```

```
array([[ 0.26666667,  0.13333333,  0.13333333,  0.4       ],
       [ 0.13333333,  0.23333333,  0.4       ,  0.03333333],
       [ 0.13333333,  0.4       ,  0.73333333, -0.13333333],
       [ 0.4       ,  0.03333333, -0.13333333,  0.76666667]])
```

由程式可知$XX^{+} \neq I_N$。

數學證明如下：



[d] $\text{trace}(XX^{\dagger}) = \text{rank}(X)$.

```
# d
import numpy as np
# N > d+1
X = [[2,4,6],[1,3,5],[1,4,2],[3,5,2]]
X = np.asarray(X)
X_cross = np.linalg.pinv(X)
XX_cross = np.dot(X,X_cross)
trace = np.trace(XX_cross)
rank = np.linalg.matrix_rank(X)

print("trace: ",trace)
print("rank: ",rank)
```

```
trace:  3.0000000000000001
rank:  3
```

由程式可知$\text{trace}(XX^{+}) = rank(X)$。

數學證明如下：

[d]

trace matrix 為 matrix 對角線之合

rank 是矩陣中線性獨立的列 or 行向量的個數

$$XX^{+} = U\Sigma V^{T} V\Sigma^{+}U^{T}$$

$$= U\Sigma\Sigma^{+}U^{T}$$

假設

$$U = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \quad U^{-1} = U^{T} = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \Sigma^{+} = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad \Sigma\Sigma^{+} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

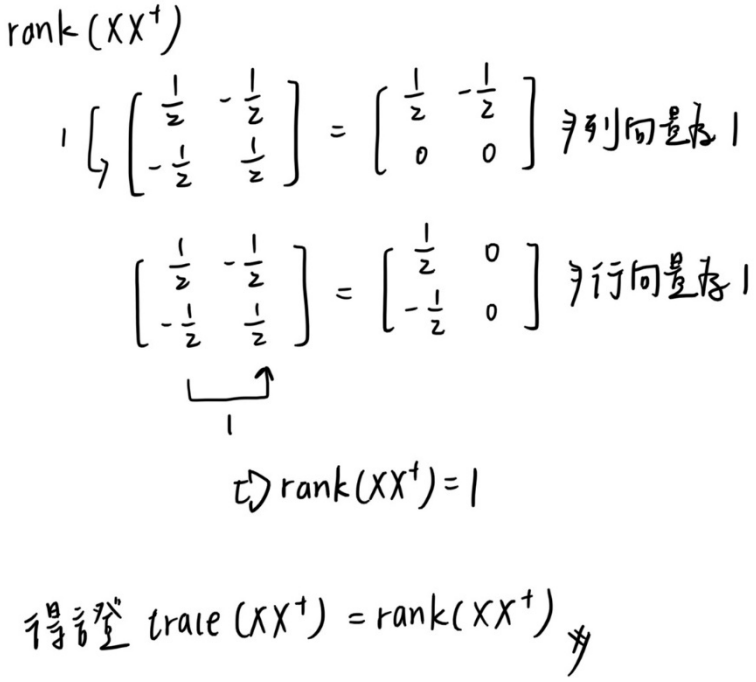
$$XX^{+} = U\Sigma\Sigma^{+}U^{T}$$

$$= \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

$$\text{trace}(XX^{+}) = \frac{1}{2} + \frac{1}{2} = 1$$



[e] none of the other choices

source code：https://colab.research.google.com/drive/12B__0kTFg-k-GVco1od0V8ORuj-Oj6AZ?usp=sharing

12. ANS [a]

In the lecture, we learned that the logistic regression can be derived by maximizing the likelihood function where each label y_n is generated from $P(+1|\mathrm{x}_n)$ "pretended by" $1/(1+\exp(-\mathrm{w}^T\mathrm{x}_n))$. Now, consider a case where each real-valued label y_n is generated from $p(y|\mathrm{x}_n)$, where p is used instead of P to denote a probability density function (instead of a probability mass function—you can ignore the subtle mathematical difference for now). We will consider $p(y|\mathrm{x}_n)$ to be pretended by a normal distribution with mean $\mathrm{w}^T\mathrm{x}_n$ and variance a^2. Assume that all inversions in the equations below exist, what is the optimal w^* that maximizes the likelihood function for this case? Choose the correct answer; explain your choice.

[a] $(\mathrm{X}^T\mathrm{X})^{-1}\mathrm{X}^T\mathrm{y}$

[b] $a\,(\mathrm{X}^T\mathrm{X})^{-1}\mathrm{X}^T\mathrm{y}$

[c] $a^2\,(\mathrm{X}^T\mathrm{X})^{-1}\mathrm{X}^T\mathrm{y}$

[d] $(\mathrm{X}^T\mathrm{X}+a^2\mathrm{I})^{\dagger}\mathrm{X}^T\mathrm{y}$

[e] none of the other choices

由 normal distribution 的 probability density function

$$\Rightarrow f(x;\mu,\sigma)=\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

可得 $$h(x)=f(y|x;\mu,\sigma)=\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right)$$

likelihood function，令 like(h)

$$\text{like}(h)=P(x_1)h(x_1)\times P(x_2)h(x_2)\times\cdots\times P(x_N)h(x_N)$$

$P(x_1)\cdot P(x_2)\cdots P(x_N)$ 對於每個 h 都相同，可忽略

$$\Rightarrow \text{like}(h)=\prod_{i=1}^{N}h(x_i)=\prod_{i=1}^{N}f(y_i|x_i;\mu,\sigma)$$

$$=\prod_{i=1}^{N}\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(y_i-\mu)^2}{2\sigma^2}\right)$$

[題目中說 $\mu=w^Tx_n$，$\sigma^2=a^2$]

$$=\prod_{i=1}^{N}(2\pi a^2)^{-1/2}\exp\left(-\frac{(y_i-w^Tx_i)^2}{2a^2}\right)$$

[由於 $e^a\cdot e^b=e^{a+b}$]

$$=(2\pi a^2)^{-N/2}\exp\left(-\frac{1}{2a^2}\sum_{i=1}^{N}(y_i-w^Tx_i)^2\right)$$

取 ln

$$\Rightarrow \ln likelih = \ln\left((2\pi a^2)^{-N/2}\right) \ln\left(exp\left(-\frac{1}{2a^2}\sum_{i=1}^{N}(y_i - w^T x_i)^2\right)\right)$$

$$= -\frac{N}{2}\ln(2\pi a^2) - \frac{1}{2a^2}\sum_{i=1}^{N}(y_i - w^T x_i)^2$$

$$= -\frac{N}{2}\ln(2\pi) - \frac{N}{2}\ln(a^2) - \frac{1}{2a^2}\sum_{i=1}^{N}(y_i - w^T x_i)^2$$

求 max of $\ln likelih$，對 w 微分，$\nabla \ln likelih = 0$

$$\nabla \ln likelih = \nabla\left(-\frac{N}{2}\ln(2\pi) - \frac{N}{2}\ln(a^2) - \frac{1}{2a^2}\sum_{i=1}^{N}(y_i - w^T x_i)^2\right)$$

$$= \nabla\left(-\frac{1}{2a^2}\sum_{i=1}^{N}(y_i - w^T x_i)^2\right)$$

$$= -\frac{1}{2a^2}\sum_{i=1}^{N}\left(2(y_i - w^T x_i)^1 \cdot (-x_i)\right)$$

$$= \frac{1}{a^2}\sum_{i=1}^{N}\left(x_i^T y_i - w^T x_i^T x_i\right)$$

$$= \frac{1}{a^2}\left(\sum_{i=1}^{N} x_i^T y_i - \sum_{i=1}^{N} w^T x_i^T x_i\right)$$

由於 $\nabla \ln likelih = 0$

$$\Rightarrow \sum_{i=1}^{N} x_i^T y_i - \sum_{i=1}^{N} w^T x_i^T x_i = 0$$

$$w^T = \frac{\sum_{i=1}^{N} x_i^T y_i}{\sum_{i=1}^{N} x_i^T x_i} = \left(\sum_{i=1}^{N} x_i^T x_i\right)^{-1} \cdot \sum_{i=1}^{N} x_i^T y_i$$

$$= \underbrace{(X^T X)^{-1} \cdot X^T y}_{\text{得證}}$$

Experiments with Linear Models

Next, we will play with linear regression, logistic regression, and their use for binary classification. We will also learn how the their different robustness to outliers. We will generate artificial $2D$ data for training and testing in the next problems. Each example $(\mathbf{x}, y)$ is assumed to be generated from the following process:

- Flip a fair coin to get either $y = +1$ or $y = -1$.
- If $y = +1$, generate $\mathbf{x} = (1, x_1, x_2)$ where (x_1, x_2) comes from a normal distribution of mean $[2, 3]$ and covariance $\begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix}$.
- If $y = -1$, generate $\mathbf{x} = (1, x_1, x_2)$ where (x_1, x_2) comes from a normal distribution of mean $[0, 4]$ and covariance $\begin{bmatrix} 0.4 & 0 \\ 0 & 0.4 \end{bmatrix}$.

Please generate $N = 200$ examples from the process as your training data set $\mathcal{D}$. Then, generate 5000 more examples from the process as your test data set (for evaluating E_{out}).

Hint: Be sure to check whether your normal distribution function needs you to provide the variance, which would be like 0.6 for the $y_n = +1$ cases, or the standard deviation, which would be like $\sqrt{0.6}$.

13. ANS [d]

(*) Implement the linear regression algorithm taught in the lecture. Run the algorithm for 100 times, each with a different random seed for generating the two data sets above. What is the average $E_{\text{in}}^{\text{sqr}}(\mathbf{w}_{\text{lin}})$, where $E_{\text{in}}^{\text{sqr}}$ denotes the *averaged* squared error over N examples? Choose the closest answer; provide your code.

[a] 0.04
[b] 0.16
[c] 0.24
[d] 0.28
[e] 0.40

題目要求：

- 重複跑 100 次
- 每次 iteration 要用不一樣的 random seed 產生 dataset
- training dataset(D) size N=200
- 求 average $E_{in}^{sqr}(\mathbf{w}_{lin})$

Source Code on Colab：
https://colab.research.google.com/drive/1ASj07njRTsRK0issYbNw4-5RkF5O6Qqw?usp=sharing

```
[ ]  import numpy as np
     import math
```

```python
def generateD(X, Y, size, iter):
    np.random.seed(iter)
    for i in range(size):
        #隨機取0或1，將0視為-1
        y = np.random.randint(0, 2)

        if y == 1:
            Y.append([y])
            mean = [2, 3]
            cov = [[0.6, 0], [0, 0.6]]    # diagonal covariance
        else:
            Y.append([y-1])
            mean = [0, 4]
            cov = [[0.4, 0], [0, 0.4]]    # diagonal covariance

        xi = np.random.multivariate_normal(mean, cov, 1)
        Xi = np.array([1])
        Xi = np.append(Xi, xi)
        X.append(Xi)
```

```python
if __name__ == '__main__':
    iteration = 100
    train_size = 200
    Ein_sum = 0
    for iter in range(iteration):
        # Generate Data
        trainX = []
        trainY = []
        generateD(trainX, trainY, train_size, iter)
        trainX = np.asarray(trainX)
        trainY = np.asarray(trainY)

        # Calculate pseudo-inverse X_cross
        trainX_cross = np.linalg.pinv(trainX)
        # print(trainX_cross, type(trainX_cross))

        # Calculate WLIN = X_cross · Y
        WLIN = np.matmul(trainX_cross, trainY)
        # print(WLIN)

        # Calculate Ein
        EinN = 0
        for i in range(train_size):
            # print("trainY[i]: ",trainY[i])
            h_xn = np.matmul(WLIN.T, trainX[i])
            # print("h_xn: ",h_xn)
            EinN += math.pow(h_xn - trainY[i], 2)
            # print("EinN: ", EinN)
        Ein = EinN/train_size
        # print(Ein)
        Ein_sum += Ein
    print(Ein_sum, iteration)
    # Calculate Ein_avg
    Ein_avg = Ein_sum/iteration
    print(Ein_avg)
```

自定義一個用來產生 Dataset 的 function 叫做 generateD，傳入的參數是 X、Y 的 list 跟 dataset 規定要產出的 size，還有目前是跑到第幾個 iteration 的 iter。

```
def generateD(X, Y, size, iter):
  # 利用 iter 變數來讓每一次 iteration 都有不一樣的 Random seed
  np.random.seed(iter)
  # 用 for 迴圈去跑要產出幾筆資料
    for i in range(size):
      #隨機取 0 或 1，將 0 視為-1
      y = np.random.randint(0, 2)
      # y=1 就按題目規定的方式指定 mean 跟 covariance
      if y == 1:
        Y.append([y]) #加上一筆 Y 的資料(新增一個 1)
        mean = [2, 3]
        cov = [[0.6, 0], [0, 0.6]]  # diagonal covariance

      # y=0(這邊用來代表實際 y=-1)按題目規定的方式指定 mean 跟 covariance

      else:
        Y.append([y-1]) #加上一筆 Y 的資料(新增一個-1)
        mean = [0, 4]
        cov = [[0.4, 0], [0, 0.4]]  # diagonal covariance
      # 利用 multivariate_normal 來產出指定好 mean 跟 covariance 的常態分佈資料
      xi = np.random.multivariate_normal(mean, cov, 1)
      Xi = np.array([1])  # 在每個 Xi 前補上 X0(X0=1)
      Xi = np.append(Xi, xi)
      X.append(Xi) #加上一筆 X 的資料
```

跑完 for 迴圈出來就能得到 size 大小，按題目規定為常態分佈的 X 與 Y 的 dataset。

以下為主程式

```
if __name__ == '__main__':
  iteration = 100  #設定好跑 100 次
  train_size = 200  #按題目規定 training set 大小為 200
  Ein_sum = 0 #初始化 Ein_sum=0
  # 按規定次數跑迴圈
  for iter in range(iteration):
    # Generate Data
    trainX = []
```

```
    trainY = []
    # 呼叫前面定義好的 generateD function 產生 training dataset
    generateD(trainX, trainY, train_size, iter)
    trainX = np.asarray(trainX)
    trainY = np.asarray(trainY)
    # 利用 np.linalg.pinv 算出 X 的 pseudo-inverse
    # Calculate pseudo-inverse X_cross
    trainX_cross = np.linalg.pinv(trainX)

    # 根據投影片定義算出 WLIN
    # Calculate WLIN = X_cross · Y
    WLIN = np.matmul(trainX_cross, trainY)
    # 計算E_in^sqr(w_lin)
    # Calculate Ein
    EinN = 0
    # 用 for 迴圈算出每筆 X 與 WLIN 的內積值也就是 h(Xn)
    # 然後透過 EinN 紀錄 h(Xn)與 label Y 的差異的平方總和
    for i in range(train_size):
      h_xn = np.matmul(WLIN.T, trainX[i])
      EinN += math.pow(h_xn - trainY[i], 2)
    # 求出 Ein，並用 Ein_sum 紀錄每次 iteration 的E_in^sqr(w_lin)總和
    Ein = EinN/train_size
    Ein_sum += Ein
  print(Ein_sum, iteration)
  # 最後將 Ein_sum/iteration 次數算出 average E_in^sqr(w_lin)
  # Calculate Ein_avg
  Ein_avg = Ein_sum/iteration
  print(Ein_avg)
```

$\mathbf{w}_{\text{LIN}} = X^{\dagger}\mathbf{y}$

- linear regression hypothesis: $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

in-sample

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(\underbrace{h(\mathbf{x}_n)}_{\mathbf{w}^T\mathbf{x}_n} - y_n)^2$$

得到結果如下：

```
0.28236212517755155
```

故答案選 d。

14. ANS [d]

(*) Following the previous problem, what is the average $\left|E_{\text{in}}^{0/1}(\mathbf{w}_{\text{lin}}) - E_{\text{out}}^{0/1}(\mathbf{w}_{\text{lin}})\right|$, where 0/1 denotes the 0/1 error (i.e. using $\mathbf{w}_{\text{lin}}$ for binary classification), $E_{\text{in}}^{0/1}$ denotes the *averaged* 0/1 error over N examples, $E_{\text{out}}^{(0/1)}$ is estimated using the averaged 0/1 error on the test data set above? Choose the closest answer; provide your code.

[a] 0.091
[b] 0.065
[c] 0.039
[d] 0.013
[e] 0.001

題目要求：

- 重複跑 100 次
- 每次 iteration 要用不一樣的 random seed 產生 dataset
- training dataset(D) size N=200 、test dataset size = 5000
- 求 average $|E_{in}^{0/1}(\mathbf{w}_{lin}) - E_{out}^{0/1}(\mathbf{w}_{lin})|$

Source Code on Colab：
https://colab.research.google.com/drive/1N9YSI30dzgRSw9R7IMNRFkYvIxTdGa39?usp=sharing

```
import numpy as np
import math
```

```
def generateD(X, Y, size, iter):
    np.random.seed(iter)
    for i in range(size):
        #隨機取0或1，將0視為-1
        y = np.random.randint(0, 2)

        if y == 1:
            Y.append([y])
            mean = [2, 3]
            cov = [[0.6, 0], [0, 0.6]]    # diagonal covariance
        else:
            Y.append([y-1])
            mean = [0, 4]
            cov = [[0.4, 0], [0, 0.4]]    # diagonal covariance

        xi = np.random.multivariate_normal(mean, cov, 1)
        Xi = np.array([1])
        Xi = np.append(Xi, xi)
        X.append(Xi)
```

```python
if __name__ == '__main__':
    iteration = 100
    train_size = 200
    test_size = 5000
    result = []

    for iter in range(iteration):
        # Generate Data
        trainX = []
        trainY = []
        testX = []
        testY = []
        generateD(trainX, trainY, train_size, iter)
        generateD(testX, testY, test_size, iter)
        trainX = np.asarray(trainX)
        trainY = np.asarray(trainY)
        testX = np.asarray(testX)
        testY = np.asarray(testY)

        # Calculate pseudo-inverse (X_cross=(XTX)^-1 XT)
        trainX_cross = np.linalg.pinv(trainX)

        # Calculate WLIN = X_cross · Y
        WLIN = np.matmul(trainX_cross, trainY)
        # print(WLIN)

        # Calculate Ein
        EinN = 0
        for i in range(train_size):
            h_xn = np.matmul(WLIN.T, trainX[i])
            if np.sign(h_xn) != trainY[i]:
                EinN += 1
        # print("EinN: ", EinN)
        Ein = EinN/train_size
        # print("Ein: ", Ein)

        # Calculate Eout
        EoutN = 0
        for i in range(test_size):
            h_xn = np.matmul(WLIN.T, testX[i])
            if np.sign(h_xn) != testY[i]:
                EoutN += 1
        # print("EoutN: ", EoutN)
        Eout = EoutN/test_size
        # print("Eout: ", Eout)

        result.append(abs(Ein - Eout))

    print(result)
    r = np.mean(result)
    print(r)
```

自定義一個用來產生 Dataset 的 function 叫做 generateD，傳入的參數是 X、Y 的 list 跟 dataset 規定要產出的 size，還有目前是跑到第幾個 iteration 的 iter。

```
def generateD(X, Y, size, iter):
  # 利用 iter 變數來讓每一次 iteration 都有不一樣的 Random seed
  np.random.seed(iter)
  # 用 for 迴圈去跑要產出幾筆資料
    for i in range(size):
      #隨機取 0 或 1，將 0 視為-1
      y = np.random.randint(0, 2)
      # y=1 就按題目規定的方式指定 mean 跟 covariance
      if y == 1:
        Y.append([y]) #加上一筆 Y 的資料(新增一個 1)
        mean = [2, 3]
        cov = [[0.6, 0], [0, 0.6]]  # diagonal covariance

      # y=0(這邊用來代表實際 y=-1)按題目規定的方式指定 mean 跟 covariance

      else:
        Y.append([y-1]) #加上一筆 Y 的資料(新增一個-1)
        mean = [0, 4]
        cov = [[0.4, 0], [0, 0.4]]  # diagonal covariance
      # 利用 multivariate_normal 來產出指定好 mean 跟 covariance 的常態分佈資料
      xi = np.random.multivariate_normal(mean, cov, 1)
      Xi = np.array([1])  # 在每個 Xi 前補上 X0(X0=1)
      Xi = np.append(Xi, xi)
      X.append(Xi) #加上一筆 X 的資料
```

跑完 for 迴圈出來就能得到 size 大小，按題目規定為常態分佈的 X 與 Y 的 dataset。

以下為主程式

```
if __name__ == '__main__':
  iteration = 100  #設定好跑 100 次
  train_size = 200  #按題目規定 training set 大小為 200
  test_size = 5000  #按題目規定 testing set 大小為 5000

  result = [] #初始化 result=0
  # 按規定次數跑迴圈
  for iter in range(iteration):
```

```
# Generate Data
trainX = []
trainY = []
testX = []
testY = []
```

呼叫前面定義好的 generateD function 產生 training 跟 testing dataset

```
generateD(trainX, trainY, train_size, iter)
generateD(testX, testY, test_size, iter)
trainX = np.asarray(trainX)
trainY = np.asarray(trainY)
testX = np.asarray(testX)
testY = np.asarray(testY)
```

利用 np.linalg.pinv 算出 X 的 pseudo-inverse

```
# Calculate pseudo-inverse
trainX_cross = np.linalg.pinv(trainX)
```

根據投影片定義算出 WLIN $\mathbf{w}_{\text{LIN}} = X^{\dagger}\mathbf{y}$

```
# Calculate WLIN = X_cross · Y
WLIN = np.matmul(trainX_cross, trainY)
```

計算$E_{in}^{0/1}(\mathbf{w}_{lin})$

```
# Calculate Ein
EinN = 0
```

用 for 迴圈算出每筆 X 與 WLIN 的內積值也就是 h(Xn)

- linear regression hypothesis: $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

然後透過 EinN 紀錄求出的 h(Xn)與 label Y 不一樣個數的總和

$$E_{\text{in}}(\mathbf{h}) = \frac{1}{N}\sum_{n=1}^{N}[\![h(\mathbf{x}_n) \neq y_n]\!]$$

```
for i in range(train_size):
  h_xn = np.matmul(WLIN.T, trainX[i])
  if np.sign(h_xn) != trainY[i]:
    EinN += 1
```

求出 Ein

```
Ein = EinN/train_size
```

利用 testing dataset 計算$E_{out}^{0/1}(\mathbf{w}_{lin})$

```
# Calculate Eout
EoutN = 0
```

```
  # 用 for 迴圈算出每筆 X 與 WLIN 的內積值也就是 h(Xn)
```

- linear regression hypothesis: $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

```
  # 然後透過 EoutN 紀錄求出的 h(Xn)與 label Y 不一樣個數的總和
```

$$E_{\text{out}}(\mathbf{h}) = \underset{\mathbf{x}\sim P}{\mathcal{E}}[\![h(\mathbf{x}) \neq f(\mathbf{x})]\!]$$

```
  for i in range(test_size):
    h_xn = np.matmul(WLIN.T, testX[i])
    if np.sign(h_xn) != testY[i]:
      EoutN += 1

  # 求出 Eout

  Eout = EoutN/test_size
  # 將這次 iteration 的|E_in^0/1(w_lin) − E_out^0/1(w_lin)|存在 result 中
  result.append(abs(Ein - Eout))

print(result)

 # 最後用 np.mean 算出 average |E_in^0/1(w_lin) − E_out^0/1(w_lin)|

r = np.mean(result)
print(r)
```

得到結果如下：

0.014493999999999998 故答案選 d。

15. ANS [b]

(*) Consider two algorithms. The first one, $\mathcal{A}$, is the linear regression algorithm above. The second one $\mathcal{B}$ is logistic regression, trained with gradient descent with $\eta = 0.1$ for $T = 500$ iterations, starting from $\mathbf{w}_0 = \mathbf{0}$. Run the algorithms on the same $\mathcal{D}$, and record $[E_{\text{out}}^{0/1}(\mathcal{A}(\mathcal{D})), E_{\text{out}}^{0/1}(\mathcal{B}(\mathcal{D}))]$. Repeat the process for 100 times, each with a different random seed for generating the two data sets above. What is the average $[E_{\text{out}}^{0/1}(\mathcal{A}(\mathcal{D})), E_{\text{out}}^{0/1}(\mathcal{B}(\mathcal{D}))]$? Choose the closest answer; provide your code.

[a] (0.018, 0.018)
[b] (0.058, 0.058)
[c] (0.058, 0.093)
[d] (0.138, 0.108)
[e] (0.268, 0.208)

題目要求：

- 重複跑 100 次
- 每次 iteration 要用不一樣的 random seed 產生 dataset
- training dataset(D) size N=200 、test dataset size = 5000
- 用η = 0.1，T = 500，$\mathbf{w}_0 = 0$，來跑 logistic regression(*B*)
- 求 average $[E_{out}^{0/1}(A(D)), E_{out}^{0/1}(B(D))]$

Source Code on Colab：
https://colab.research.google.com/drive/1VK2eKbVkeP2aMR6XdH9UdRCtgYe4ptnS?usp=sharing

```
import numpy as np
import math
```

```
def generateD(X, Y, size, iter):
    np.random.seed(iter)
    for i in range(size):
        #隨機取0或1，將0視為-1
        y = np.random.randint(0, 2)

        if y == 1:
            Y.append([y])
            mean = [2, 3]
            cov = [[0.6, 0], [0, 0.6]]    # diagonal covariance
        else:
            Y.append([y-1])
            mean = [0, 4]
            cov = [[0.4, 0], [0, 0.4]]    # diagonal covariance

        xi = np.random.multivariate_normal(mean, cov, 1)
        Xi = np.array([1])
        Xi = np.append(Xi, xi)
        X.append(Xi)
```

```python
if __name__ == '__main__':
    iteration = 100
    train_size = 200
    test_size = 5000
    Eout_lin = []
    Eout_log = []

    for iter in range(iteration):
        # Generate Data
        trainX = []
        trainY = []
        testX = []
        testY = []
        generateD(trainX, trainY, train_size, iter)
        generateD(testX, testY, test_size, iter)
        trainX = np.asarray(trainX)
        trainY = np.asarray(trainY)
        testX = np.asarray(testX)
        testY = np.asarray(testY)

    # Linear Reg
        # Calculate pseudo-inverse
        trainX_cross = np.linalg.pinv(trainX)
        # Calculate WLIN = X_cross · Y
        WLIN = np.matmul(trainX_cross, trainY)
        # print(WLIN)
        # Calculate Eout
        EoutN = 0
        for i in range(test_size):
            h_xn = np.matmul(WLIN.T, testX[i])
            if np.sign(h_xn) != testY[i]:
                EoutN += 1
        # print("EoutN: ", EoutN)
        Eout = EoutN/test_size
        # print("Eout: ", Eout)
        Eout_lin.append(Eout)
```

```
    # Logistic Reg
        T = 500
        eta = 0.1
        W = np.zeros(3)
        for i in range(T):
            GDN = 0
            # Gradient Descent : 1/N[1+...+N(theta(-yn WT Xn)(-yn Xn))]
            for j in range(train_size):
                # theta(-yn WT Xn) = exp(-yn WT Xn)/(1 + exp(-yn WT Xn))
                s = -trainY[j] * np.matmul(W.T, trainX[j])
                theta = np.exp(s)/(1 + np.exp(s))
                GDN += theta * -trainY[j] * trainX[j]
            # print("GDN: ", GDN)
            GD = GDN / train_size
            # print("GD: ", GD)
            # update : Wt+1 = Wt - eta*GD
            W = W - (eta * GD)
        # print("W: ", W)

        # Calculate Eout
        EoutN = 0
        l = []
        for i in range(test_size):
            h_xn = np.matmul(W.T, testX[i])
            if np.sign(h_xn) != testY[i]:
                EoutN += 1
        # print("EoutN: ", EoutN)
        Eout = EoutN/test_size
        # print("Eout: ", Eout)
        Eout_log.append(Eout)

    # print("Eout_lin: ",Eout_lin)
    # print("Eout_log: ",Eout_log)
    Eout_lin_avg = np.mean(Eout_lin)
    Eout_log_avg = np.mean(Eout_log)
    print("Eout_lin_avg: ",Eout_lin_avg)
    print("Eout_log_avg: ",Eout_log_avg)
```

自定義一個用來產生 Dataset 的 function 叫做 generateD，傳入的參數是 X、Y 的 list 跟 dataset 規定要產出的 size，還有目前是跑到第幾個 iteration 的 iter。

```
def generateD(X, Y, size, iter):
  # 利用 iter 變數來讓每一次 iteration 都有不一樣的 Random seed
  np.random.seed(iter)
  # 用 for 迴圈去跑要產出幾筆資料
    for i in range(size):
      #隨機取 0 或 1，將 0 視為-1
      y = np.random.randint(0, 2)
      # y=1 就按題目規定的方式指定 mean 跟 covariance
```

```
        if y == 1:
          Y.append([y]) #加上一筆 Y 的資料(新增一個 1)
          mean = [2, 3]
          cov = [[0.6, 0], [0, 0.6]]  # diagonal covariance
        # y=0(這邊用來代表實際 y=-1)按題目規定的方式指定 mean 跟 covariance
        else:
          Y.append([y-1]) #加上一筆 Y 的資料(新增一個-1)
          mean = [0, 4]
          cov = [[0.4, 0], [0, 0.4]]  # diagonal covariance
        # 利用 multivariate_normal 來產出指定好 mean 跟 covariance 的常態分佈資料
        xi = np.random.multivariate_normal(mean, cov, 1)
        Xi = np.array([1]) # 在每個 Xi 前補上 X0(X0=1)
        Xi = np.append(Xi, xi)
        X.append(Xi) #加上一筆 X 的資料
```

跑完 for 迴圈出來就能得到 size 大小，按題目規定為常態分佈的 X 與 Y 的 dataset。

以下為主程式

```
if __name__ == '__main__':
  iteration = 100
  train_size = 200
  test_size = 5000
  Eout_lin = []
  Eout_log = []
  for iter in range(iteration):
    # 按題目要求透過上述自定義 function 產生出 dataset
    # Generate Data
    trainX = []
    trainY = []
    testX = []
    testY = []
    generateD(trainX, trainY, train_size, iter)
    generateD(testX, testY, test_size, iter)
    trainX = np.asarray(trainX)
    trainY = np.asarray(trainY)
    testX = np.asarray(testX)
    testY = np.asarray(testY)
```

```
# Linear Reg
```

利用 np.linalg.pinv 算出 X 的 pseudo-inverse

```
# Calculate pseudo-inverse
trainX_cross = np.linalg.pinv(trainX)
```

根據投影片定義算出 WLIN $\mathbf{w}_{\text{LIN}} = \mathrm{X}^{\dagger}\mathbf{y}$

```
# Calculate WLIN = X_cross · Y
WLIN = np.matmul(trainX_cross, trainY)
```

利用 testing dataset 計算$E_{out}^{0/1}(A(D))$

```
# Calculate Eout
EoutN = 0
```

用 for 迴圈算出每筆 X 與 WLIN 的內積值也就是 h(Xn)

- linear regression hypothesis: $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

然後透過 EoutN 紀錄求出的 h(Xn)與 label Y 不一樣個數的總和

$$E_{\text{out}}(\mathbf{h}) = \underset{\mathbf{x}\sim P}{\mathcal{E}} [\![h(\mathbf{x}) \neq f(\mathbf{x})]\!]$$

```
for i in range(test_size):
  h_xn = np.matmul(WLIN.T, testX[i])
  if np.sign(h_xn) != testY[i]:
    EoutN += 1
# print("EoutN: ", EoutN)
Eout = EoutN/test_size
# print("Eout: ", Eout)
# 最後將結果存在 Eout_lin 中
Eout_lin.append(Eout)
```

接下來是算 Logistic Reg

```
# Logistic Reg
```

用η(eta) = 0.1 · T = 500 · $\mathbf{w}_0 = 0$ · 來跑 logistic regression(*B*)

```
T = 500
eta = 0.1
W = np.zeros(3)
for i in range(T):
  GDN = 0
```

$$\nabla E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \theta\left(-y_n \mathbf{w}^T \mathbf{x}_n\right)\left(-y_n \mathbf{x}_n\right)$$

根據投影片可知道 Gradient Descent 的算法

```
# Gradient Descent : 1/N[1+...+N(theta(-yn WT Xn)(-yn Xn))]
for j in range(train_size):
```

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

根據投影片可知道 theta 的算法

```
  # theta(-yn WT Xn) = exp(-yn WT Xn)/(1 + exp(-yn WT Xn))
  s = -trainY[j] * np.matmul(W.T, trainX[j])
  theta = np.exp(s)/(1 + np.exp(s))
  GDN += theta * -trainY[j] * trainX[j]
GD = GDN / train_size
```

Iterative Optimization

For $t = 0, 1, \ldots$

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \eta \mathbf{v}$$

when stop, return last $\mathbf{w}$ as g

- optimal $\mathbf{v}$: opposite direction of $\nabla E_{\text{in}}(\mathbf{w}_t)$

$$\mathbf{v} = -\frac{\nabla E_{\text{in}}(\mathbf{w}_t)}{\|\nabla E_{\text{in}}(\mathbf{w}_t)\|}$$

- gradient descent: for small η, $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta \frac{\nabla E_{\text{in}}(\mathbf{w}_t)}{\|\nabla E_{\text{in}}(\mathbf{w}_t)\|}$

根據投影片可知道更新的方法

```
# update : Wt+1 = Wt - eta*GD
W = W - (eta * GD)
```

利用 testing dataset 計算$E_{out}^{0/1}(B(D))$

```
# Calculate Eout
    EoutN = 0
    l = []
    # 用 for 迴圈算出每筆 X 與 W(前面算出來最後的 W)的內積值也就是 h(Xn)
    # 然後透過 EoutN 紀錄求出的 h(Xn)與 label Y 不一樣個數的總和
```

$$E_{\text{out}}(\mathbf{h}) = \underset{\mathbf{x}\sim P}{\mathcal{E}} [\![h(\mathbf{x}) \neq f(\mathbf{x})]\!]$$

```
    for i in range(test_size):
      h_xn = np.matmul(W.T, testX[i])
      if np.sign(h_xn) != testY[i]:
        EoutN += 1
    Eout = EoutN/test_size
    # 最後將結果存在 Eout_log 中
    Eout_log.append(Eout)
#求 average [E_out^0/1(A(D)), E_out^0/1(B(D))]
Eout_lin_avg = np.mean(Eout_lin)
Eout_log_avg = np.mean(Eout_log)
print("Eout_lin_avg: ",Eout_lin_avg)
print("Eout_log_avg: ",Eout_log_avg)
```

得到結果如下：

```
Eout_lin_avg:  0.057684000000000006
Eout_log_avg:  0.05918599999999999
```

故答案選 b。

16. ANS [c]

(*) Following the previous problem, in addition to the 200 examples in $\mathcal{D}$, add 20 outlier examples generated from the following process to your training data (but not to your test data). All outlier examples will be labeled $y = +1$ and $\mathbf{x} = [1, x_1, x_2]$ where (x_1, x_2) comes from a normal distribution of mean $[6, 0]$ and covariance $\begin{bmatrix} 0.3 & 0 \\ 0 & 0.1 \end{bmatrix}$. Name the new training data set $\mathcal{D}'$. Run the algorithms on the same $\mathcal{D}'$, and record $[E_{\text{out}}^{0/1}(\mathcal{A}(\mathcal{D}')), E_{\text{out}}^{0/1}(\mathcal{B}(\mathcal{D}'))]$. Repeat the process for 100 times, each with a different random seed for generating the two data sets above. What is the average $[E_{\text{out}}^{0/1}(\mathcal{A}(\mathcal{D}')), E_{\text{out}}^{0/1}(\mathcal{B}(\mathcal{D}'))]$? Choose the closest answer; provide your code.

[a] (0.070, 0.018)
[b] (0.240, 0.048)
[c] (0.090, 0.058)
[d] (0.090, 0.078)
[e] (0.270, 0.108)

題目要求：

- 重複跑 100 次
- 每次 iteration 要用不一樣的 random seed 產生 dataset
- training dataset(D) size N=200，多加上 20 outlier、test dataset size = 5000
- 用$\eta = 0.1$，$T = 500$，$\mathbf{w}_0 = 0$，來跑 logistic regression(*B*)
- 求 average $[E_{out}^{0/1}(A(D)), E_{out}^{0/1}(B(D))]$

Source Code on Colab：
https://colab.research.google.com/drive/1MWM654IhE0bhzUd1LH0L7e8oBgkhP0jH?usp=sharing

```
import numpy as np
import math
```

```
def generateD(X, Y, size, iter):
    np.random.seed(iter)
    for i in range(size):
        #隨機取0或1，將0視為-1
        y = np.random.randint(0, 2)

        if y == 1:
            Y.append([y])
            mean = [2, 3]
            cov = [[0.6, 0], [0, 0.6]]    # diagonal covariance
        else:
            Y.append([y-1])
            mean = [0, 4]
            cov = [[0.4, 0], [0, 0.4]]    # diagonal covariance

        xi = np.random.multivariate_normal(mean, cov, 1)
        Xi = np.array([1])
        Xi = np.append(Xi, xi)
        X.append(Xi)
```

```
def generateO(X, Y, size, iter):
    np.random.seed(iter)
    for i in range(size):

        Y.append([1])
        mean = [6, 0]
        cov = [[0.3, 0], [0, 0.1]]

        xi = np.random.multivariate_normal(mean, cov, 1)
        Xi = np.array([1])
        Xi = np.append(Xi, xi)
        X.append(Xi)
```

```
if __name__ == '__main__':
    iteration = 100
    train_size = 200
    test_size = 5000
    out_size = 20
    Eout_lin = []
    Eout_log = []

    for iter in range(iteration):
        # Generate Data
        trainX = []
        trainY = []
        testX = []
        testY = []
        generateD(trainX, trainY, train_size, iter)
        generateO(trainX, trainY, out_size, iter)
        generateD(testX, testY, test_size, iter)
        trainX = np.asarray(trainX)
        trainY = np.asarray(trainY)
        testX = np.asarray(testX)
        testY = np.asarray(testY)

    # Linear Reg
        # Calculate pseudo-inverse
        trainX_cross = np.linalg.pinv(trainX)
        # Calculate WLIN = X_cross · Y
        WLIN = np.matmul(trainX_cross, trainY)
        # print(WLIN)
        # Calculate Eout
        EoutN = 0
        for i in range(test_size):
            h_xn = np.matmul(WLIN.T, testX[i])
            if np.sign(h_xn) != testY[i]:
                EoutN += 1
        # print("EoutN: ", EoutN)
        Eout = EoutN/test_size
        # print("Eout: ", Eout)
        Eout_lin.append(Eout)
```

```
# Logistic Reg
    T = 500
    eta = 0.1
    W = np.zeros(3)
    for i in range(T):
        GDN = 0
        # Gradient Descent : 1/N[1+...+N(theta(-yn WT Xn)(-yn Xn))]
        for j in range(train_size + out_size):
            # theta(-yn WT Xn) = exp(-yn WT Xn)/(1 + exp(-yn WT Xn))
            s = -trainY[j] * np.matmul(W.T, trainX[j])
            theta = np.exp(s)/(1 + np.exp(s))
            GDN += theta * -trainY[j] * trainX[j]
        GD = GDN / (train_size + out_size)

        # update : Wt+1 = Wt - eta*GD
        W = W - (eta * GD)
    # print("W: ", W)

    # Calculate Eout
    EoutN = 0
    l = []
    for i in range(test_size):
        h_xn = np.matmul(W.T, testX[i])
        if np.sign(h_xn) != testY[i]:
            EoutN += 1
    # print("EoutN: ", EoutN)
    Eout = EoutN/test_size
    # print("Eout: ", Eout)
    Eout_log.append(Eout)

print("Eout_lin: ",Eout_lin)
print("Eout_log: ",Eout_log)
Eout_lin_avg = np.mean(Eout_lin)
Eout_log_avg = np.mean(Eout_log)
print("Eout_lin_avg: ",Eout_lin_avg)
print("Eout_log_avg: ",Eout_log_avg)
```

這題的程式碼大致跟 15 題一樣，只有多加了黃色螢光筆的地方。

自定義一個用來產生 Dataset 的 function 叫做 generateD，傳入的參數是 X、Y 的 list 跟 dataset 規定要產出的 size，還有目前是跑到第幾個 iteration 的 iter。

```
def generateD(X, Y, size, iter):
  # 利用 iter 變數來讓每一次 iteration 都有不一樣的 Random seed
  np.random.seed(iter)
  # 用 for 迴圈去跑要產出幾筆資料
    for i in range(size):
      #隨機取 0 或 1，將 0 視為-1
      y = np.random.randint(0, 2)
      # y=1 就按題目規定的方式指定 mean 跟 covariance
      if y == 1:
```

```
        Y.append([y]) #加上一筆 Y 的資料(新增一個 1)
        mean = [2, 3]
        cov = [[0.6, 0], [0, 0.6]]  # diagonal covariance
      # y=0(這邊用來代表實際 y=-1)按題目規定的方式指定 mean 跟 covariance
      else:
        Y.append([y-1]) #加上一筆 Y 的資料(新增一個-1)
        mean = [0, 4]
        cov = [[0.4, 0], [0, 0.4]]  # diagonal covariance
      # 利用 multivariate_normal 來產出指定好 mean 跟 covariance 的常態分佈資料
      xi = np.random.multivariate_normal(mean, cov, 1)
      Xi = np.array([1]) # 在每個 Xi 前補上 X0(X0=1)
      Xi = np.append(Xi, xi)
      X.append(Xi) #加上一筆 X 的資料
```

跑完 for 迴圈出來就能得到 size 大小，按題目規定為常態分佈的 X 與 Y 的 dataset。

自定義一個用來產生 Dataset 的 function 叫做 generateO，跟前面的方式一樣，只是常態分佈條件不一樣，用來產生 outlier。

```
def generateO(X, Y, size, iter):
  np.random.seed(iter)
  for i in range(size):

    Y.append([1])
    mean = [6, 0]
    cov = [[0.3, 0], [0, 0.1]]

    xi = np.random.multivariate_normal(mean, cov, 1)
    Xi = np.array([1])
    Xi = np.append(Xi, xi)
    X.append(Xi)
```

以下為主程式

```
if __name__ == '__main__':
  iteration = 100
  train_size = 200
  test_size = 5000
  out_size = 20
```

```
Eout_lin = []
Eout_log = []
for iter in range(iteration):
  # 按題目要求透過上述自定義 function 產生出 dataset
  # Generate Data
  trainX = []
  trainY = []
  testX = []
  testY = []
  generateD(trainX, trainY, train_size, iter)
  generateO(trainX, trainY, out_size, iter)
  generateD(testX, testY, test_size, iter)
  trainX = np.asarray(trainX)
  trainY = np.asarray(trainY)
  testX = np.asarray(testX)
  testY = np.asarray(testY)
# Linear Reg
```

$$\mathbf{w}_{\text{LIN}} = \mathrm{X}^{\dagger}\mathbf{y}$$

```
    # 利用 np.linalg.pinv 算出 X 的 pseudo-inverse
    # Calculate pseudo-inverse
    trainX_cross = np.linalg.pinv(trainX)
    # 根據投影片定義算出 WLIN
    # Calculate WLIN = X_cross · Y
    WLIN = np.matmul(trainX_cross, trainY)
    # 利用 testing dataset 計算E_out^{0/1}(A(D))
    # Calculate Eout
    EoutN = 0
    # 用 for 迴圈算出每筆 X 與 WLIN 的內積值也就是 h(Xn)
```

- linear regression hypothesis: $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

```
    # 然後透過 EoutN 紀錄求出的 h(Xn)與 label Y 不一樣個數的總和
```

$$E_{\text{out}}(\mathbf{h}) = \underset{\mathbf{x}\sim P}{\mathcal{E}} [\![h(\mathbf{x}) \neq f(\mathbf{x})]\!]$$

```
    for i in range(test_size):
      h_xn = np.matmul(WLIN.T, testX[i])
      if np.sign(h_xn) != testY[i]:
```

```
            EoutN += 1
        # print("EoutN: ", EoutN)
        Eout = EoutN/test_size
        # print("Eout: ", Eout)
        # 最後將結果存在 Eout_lin 中
        Eout_lin.append(Eout)
```

接下來是算 Logistic Reg

```
    # Logistic Reg

    # 用η(eta) = 0.1 · T = 500 · $\mathbf{w}_0 = 0$ · 來跑 logistic regression(B)

        T = 500
        eta = 0.1
        W = np.zeros(3)
        for i in range(T):
            GDN = 0
```

$$\nabla E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \theta\left(-y_n \mathbf{w}^T \mathbf{x}_n\right)\left(-y_n \mathbf{x}_n\right)$$

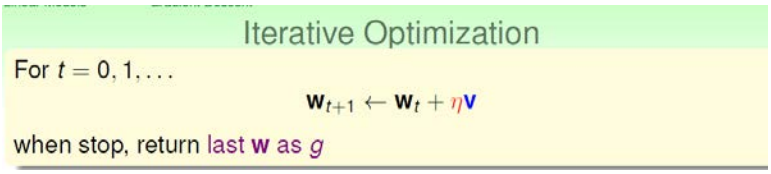
根據投影片可知道 Gradient Descent 的算法

```
            # Gradient Descent : 1/N[1+...+N(theta(-yn WT Xn)(-yn Xn))]
            for j in range(train_size + out_size):
```

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

根據投影片可知道 theta 的算法

```
                # theta(-yn WT Xn) = exp(-yn WT Xn)/(1 + exp(-yn WT Xn))
                s = -trainY[j] * np.matmul(W.T, trainX[j])
                theta = np.exp(s)/(1 + np.exp(s))
                GDN += theta * -trainY[j] * trainX[j]
        GD = GDN / (train_size + out_size)
```



- optimal $\mathbf{v}$: opposite direction of $\nabla E_{\text{in}}(\mathbf{w}_t)$

$$\mathbf{v} = -\frac{\nabla E_{\text{in}}(\mathbf{w}_t)}{\|\nabla E_{\text{in}}(\mathbf{w}_t)\|}$$

- gradient descent: for small η, $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta \frac{\nabla E_{\text{in}}(\mathbf{w}_t)}{\|\nabla E_{\text{in}}(\mathbf{w}_t)\|}$

根據投影片可知道更新的方法

```
# update : Wt+1 = Wt - eta*GD
W = W - (eta * GD)
```

利用 testing dataset 計算$E_{out}^{0/1}(B(D))$

```
# Calculate Eout
    EoutN = 0
    l = []
    # 用 for 迴圈算出每筆 X 與 W(前面算出來最後的 W)的內積值也就是 h(Xn)
    # 然後透過 EoutN 紀錄求出的 h(Xn)與 label Y 不一樣個數的總和
```

$$E_{\text{out}}(\mathbf{h}) = \mathop{\mathcal{E}}_{\mathbf{x}\sim P} [\![h(\mathbf{x}) \neq f(\mathbf{x})]\!]$$

```
    for i in range(test_size):
      h_xn = np.matmul(W.T, testX[i])
      if np.sign(h_xn) != testY[i]:
        EoutN += 1
    Eout = EoutN/test_size
    # 最後將結果存在 Eout_log 中
    Eout_log.append(Eout)
#求 average [E_out^0/1(A(D)),E_out^0/1(B(D))]
Eout_lin_avg = np.mean(Eout_lin)
Eout_log_avg = np.mean(Eout_log)
print("Eout_lin_avg: ",Eout_lin_avg)
print("Eout_log_avg: ",Eout_log_avg)
```

得到結果如下：

```
Eout_lin_avg:  0.09195600000000002
Eout_log_avg:  0.05915999999999998
```

故答案選 c。