$\boxed{\text{P1. B}}$ [B] $\underbrace{(1,1,1)}_{x_1}, \underbrace{(2,3,4)}_{x_2}, \underbrace{(4,3,2)}_{x_3}, \underbrace{(4,2,3)}_{x_4}$

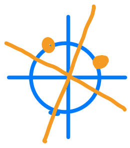
$$\left.\begin{array}{l} x_2 - x_1 = (1,2,3) \\ x_3 - x_1 = (3,2,1) \\ x_4 - x_1 = (3,1,2) \end{array}\right] \Rightarrow \operatorname{rank}\left(\begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 3 & 1 & 2 \end{bmatrix}\right) = 3$$

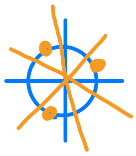
, 可知這3個vector不共平面

沒有發生退化,故必定可以被3D perceptron hypothesis shattered.

P2.C

當N=1時：{o, x}，2种 

當N=2時：{oo, xx, ox, xo}，4种 

當N=3時：{oxx, xoo, oxo, xox, oox, xxo}，6种 

只有2N符合規律

$\boxed{\text{P3. A}}$ Without loss of generality，令 $d=1$

此 donut hypothesis 可視為在 X 軸正向上的 positive interval hypothesis

By slide. L4. P15, positive interval hypothesis $m_H(\mathcal{H}) = \binom{N+1}{2} + 1$ #

故选 [A]

$\boxed{P4.D}$

$m_H(1) = \binom{2}{2} + 1 = 2$

$m_H(2) = \binom{3}{2} + 1 = 4$

$m_H(3) = \binom{4}{2} + 1 = 7 \neq 8 \leftarrow$ break point

故 $d_{vc}(H) = 2$ #

P5.C

[a] 1個 positive interval 有2個 degree of freedom，2個 positive interval 有4個 d.o.f.
故 $d_{vc}(H)=4$

[b] $\mathbb{R}^2$ 中的長方形有4個 d.o.f.，分別是中心 x.y 座標与長寬
故 $d_{vc}(H)=4$

[c] 根據 slide L4. P32. perception hypothesis set $d_{vc}=d+1$
當 $d=4$ 時，$d_{vc}=5$，且限制 $w_0>0$ 不影响 d_{vc}. 因為我們可以將原本能被 hypothesis set shatted 的 x input 平移至正半平面. d_{vc} 仍是 5 #

[d] 三次多項式 $(w_0+w_1x^1+w_2x^2+w_3x^3)$. 有4個 d.o.f 可以決定. 故 $d_{vc}(H)=4$ #

P6. A

1個 binary classifier 只有一个 degree of freedom，就是決定 Binary classifier 的 theshold

若有1126個 binary classifier 就有1126个 degree of freedom，故 d_{vc} 最多是1126.

P7.D

By Hoeffding inequlity

$P[|E_{in}(g)-E_{out}(g)|>\varepsilon] \leq 2M\exp(-2\varepsilon^2N)$

這是壞事發生的機率的upper bound,換言之,若要找好事發生的機率的lower bound,公式能寫成:

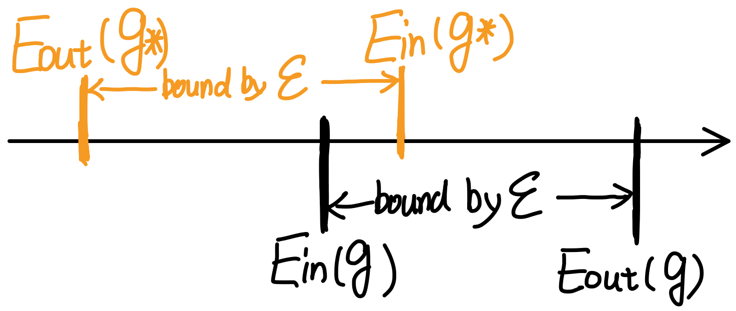
$\Rightarrow P[|E_{in}(g)-E_{out}(g)|<\varepsilon] \geq 1-2M\exp(-2\varepsilon^2N)$

g^*也能用Hoeffding inequdlity寫一樣的式子

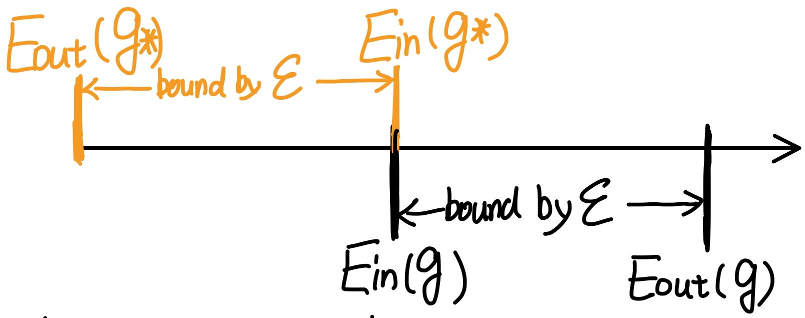
$\Rightarrow P[|E_{in}(g^*)-E_{out}(g^*)|<\varepsilon] \geq 1-2M\exp(-2\varepsilon^2N)$

由於 $g=\text{argmin}_{h\in H}E_{in}(h)$,故 $E_{in}(g) \leq E_{in}(g^*)$

我們可以利用inequlity的boundary(考慮最糟狀況)在數線上畫出下圖



且由於 $E_{in}(g) \leq E_{in}(g^*)$的限制,我們知道 $E_{out}(g)$與$E_{out}(g^*)$相距最遠的case發生在$E_{in}(g)=E_{in}(g^*)$時,如下圖所示,此時 $E_{out}(g)$與$E_{out}(g^*)$ bound by $\varepsilon+\varepsilon=2\varepsilon$



所以我們可以寫: $P[|E_{out}(g)-E_{out}(g^*)|<2\varepsilon] \geq 1-\delta$

其中 $1-\delta = 1-2M\exp(-2\varepsilon^2 N)$

$$\Rightarrow \delta = 2M\exp(-2\varepsilon^2 N)$$

$$\Rightarrow \ln\left(\frac{\delta}{2M}\right) = -2\varepsilon^2 N$$

$$\Rightarrow \frac{1}{2N}\ln\left(\frac{2M}{\delta}\right) = \varepsilon^2$$

$$\Rightarrow \varepsilon = \sqrt{\frac{1}{2N}\ln\left(\frac{2M}{\delta}\right)}$$

而我們要的 upper bound 是 2ε，故答案為 $\underline{2\varepsilon = 2\sqrt{\frac{1}{2N}\ln\left(\frac{2M}{\delta}\right)}}$ #

$\boxed{P8.B}$ By L4 投影片, P.25: positive rays $m_H(N) = N+1$

$4m_H(2N)\exp(-\frac{1}{8}\varepsilon^2 N)$, 代入 $\varepsilon = 0.1$

$\Rightarrow 4(2N+1)\exp(-\frac{1}{8}\cdot 0.01\cdot N)$ — ①

代 $N=10000$ 進入①式 $\Rightarrow 0.298$

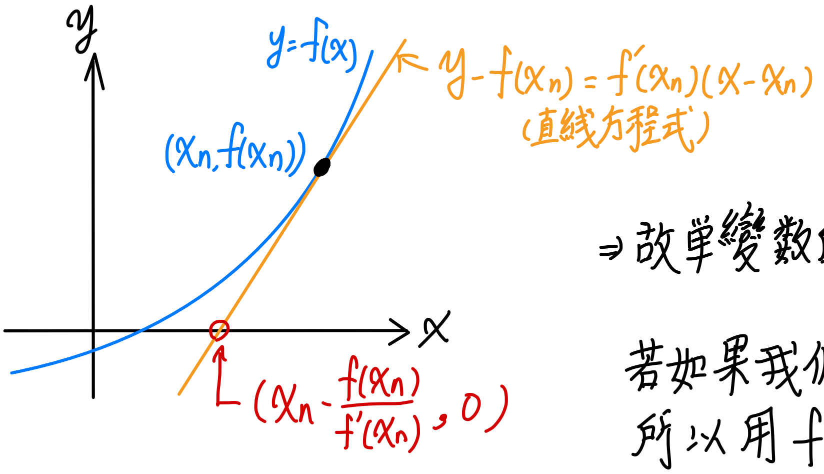
代 $N=11000$ 進入①式 $\Rightarrow 0.094$ # ≤ 0.1 故選B #

代 $N=12000$ 進入①式 $\Rightarrow 0.029$

代 $N=13000$ 進入①式 $\Rightarrow 0.009$

代 $N=14000$ 進入①式 $\Rightarrow 0.003$

P9.B



⇒故單變數的求根方式是 $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$

若如果我們要求極值，等同於在找 $f'(x)=0$ 的根
所以用 $f'(x_n)$ 替換 $f(x_n)$，我們可以得到

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

接下來將單變數的公式換成矩陣形式：$\underline{\vec{w}_{n+1} = \vec{w}_n - (A_E(u))^{-1} b_E(u)}$ #

$\boxed{\text{P10.D}}$ By L5 投影片 P.30

$$\nabla E_{in}(w) = \frac{1}{N}\sum_{i=1}^{N}\theta(-y_n w^T x_n)(-y_n x_n), \text{ where } \theta(s) = \frac{1}{1+e^{-s}}$$

求 $\frac{d}{dw}\nabla E_{in}(w) = \frac{1}{N}\sum_{i=1}^{N}\frac{d}{dw}\left[\frac{-y_n x_n}{1+e^{y_n w^T x_n}}\right]$

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

$$= \frac{1}{N}\sum_{i=1}^{N}\underbrace{(y_n^2)}_{1,\ \because\ y_n\in\{-1,1\}}\cdot\left(\frac{e^{y_n w^T x_n}}{1+2e^{y_n w^T x_n}+e^{2y_n w^T x_n}}\right)\cdot(x_n x_n^T)$$

$$= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{e^{-y_n w^T x_n}+2+e^{y_n w^T x_n}}\right)\cdot(x_n x_n^T)$$

$$= \frac{1}{N}\sum_{i=1}^{N}\underbrace{\left(\frac{1}{1+e^{-y_n w^T x_n}}\right)}\cdot\underbrace{\left(\frac{1}{1+e^{y_n w^T x_n}}\right)}\cdot(x_n x_n^T), \quad h_t(x) = \frac{1}{1+e^{w_t^T x}} \text{ 代換}$$

$$= \underline{\frac{1}{N}\sum_{i=1}^{N} h_t(-y_n x_n)\cdot h(y_n x_n)\cdot(x_n x_n^T)} \ \#$$

$\boxed{\text{P11.C}}$ 舉反例說明

設 $A=\begin{bmatrix}1&1\\1&1\end{bmatrix}$，則 $A^{\dagger}=\begin{bmatrix}1/4&1/4\\1/4&1/4\end{bmatrix}$，$\underline{AA^{\dagger}=\begin{bmatrix}1/2&1/2\\1/2&1/2\end{bmatrix}\neq\begin{bmatrix}1&0\\0&1\end{bmatrix}}$ #

P12 C

考慮 normal distribution $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

將 $\mu = w^T x_n$, $\sigma^2 = a^2$, $x = y_n$ 代入上式

$$f(y_n) = \frac{1}{a\sqrt{2\pi}} \cdot e^{-\frac{(y_n - w^T x_n)^2}{2a^2}}$$

$$\text{likelihood} = \prod_{n=1}^{N} f(y_n) = \prod_{n=1}^{N} \frac{1}{a\sqrt{2\pi}} \cdot e^{-\frac{(y_n - w^T x_n)^2}{2a^2}}$$

取 $\ln$: $\frac{N}{a\sqrt{2\pi}} + \sum_{n=1}^{N} \left[-\frac{(y_n - w^T x_n)^2}{2a^2} \right]$

$$\Rightarrow \frac{N}{a\sqrt{2\pi}} + \left(-\frac{1}{2a^2}\right) \cdot \sum_{n=1}^{N} (y_n - w^T x_n)^2$$

我們想求 min error, 故乘上 -1

$$\Rightarrow -\frac{N}{a\sqrt{2\pi}} + \frac{1}{2a^2} \cdot \left[y^T y - 2 y w^T x + (w^T x)^2 \right]$$ ← 此為 Error $\nabla E_{in}(w)$

求 $\nabla E_{in}(w) = 0$

$$\frac{\partial}{\partial w} \left[-\frac{N}{a\sqrt{2\pi}} + \frac{1}{2a^2} \cdot \left[y^T y - 2 w^T x y + (w^T x)^2 \right] \right] = 0$$

$$\Rightarrow \frac{1}{2a^2} \left[-2 x^T y + 2 x^T x w^T \right] = 0$$

$$\Rightarrow x^T x w^T = a^2 x^T y$$

$$\Rightarrow \underline{w^T = a^2 (x^T x)^{-1} \cdot x^T y}$$ #

P13. D

```python
import random
import numpy as np
import math

N_TRAIN = 200 # Number of training data
N_TEST = 5000 # Number of testing data
N_REPEAT = 100

def sign(x):
    if x >= 0:
        return 1
    else:
        return -1

Ein_acc = 0
for seed in range(N_REPEAT):
    random.seed(a=seed)
    # Generate training/testing data
    train_data = []
    test_data = []
    for _ in range(N_TRAIN + N_TEST):
        y = random.randint(0, 1)
        if y == 1:
            x1 = np.random.normal(2, math.sqrt(0.6), 1)[0]
            x2 = np.random.normal(3, math.sqrt(0.6), 1)[0]
        elif y == 0: # -1
            y = -1
            x1 = np.random.normal(0, math.sqrt(0.4), 1)[0]
            x2 = np.random.normal(4, math.sqrt(0.4), 1)[0]
        if len(train_data) < N_TRAIN:
            train_data.append((1, x1, x2, y))
        else:
            test_data.append((1, x1, x2, y))

    # Linear regression
    X = []
    Y = []
    for i in range(N_TRAIN):
        X.append(train_data[i][:3])
        Y.append([train_data[i][3]])
    X = np.array(X)
    Y = np.array(Y)
    X_pesudo = np.linalg.pinv(X, rcond=1e-15, hermitian=False)
    W_lin = np.matmul(X_pesudo, Y)

    # Calculate Ein (error on training dataset)
    for i in range(N_TRAIN):
        s = np.dot(W_lin.reshape(3), np.array(train_data[i][:3]))
        Ein_acc += (Y[i]*s-1)**2 * (1/N_TRAIN)

print("Ein = " + str(Ein_acc / N_REPEAT))
```

P14. D

```python
import random
import numpy as np
import math

N_TRAIN = 200 # Number of training data
N_TEST = 5000 # Number of testing data
N_REPEAT = 100

def sign(x):
    if x >= 0:
        return 1
    else:
        return -1

ans_acc = 0
for seed in range(N_REPEAT):
    random.seed(a=seed)
    # Generate training/testing data
    train_data = []
    test_data = []
    for _ in range(N_TRAIN + N_TEST):
        y = random.randint(0, 1)
        if y == 1:
            x1 = np.random.normal(2, math.sqrt(0.6), 1)[0]
            x2 = np.random.normal(3, math.sqrt(0.6), 1)[0]
        elif y == 0: # -1
            y = -1
            x1 = np.random.normal(0, math.sqrt(0.4), 1)[0]
            x2 = np.random.normal(4, math.sqrt(0.4), 1)[0]
        if len(train_data) < N_TRAIN:
            train_data.append((1, x1, x2, y))
        else:
            test_data.append((1, x1, x2, y))

    # Linear regression
    X = []
    Y = []
    for i in range(N_TRAIN):
        X.append(train_data[i][:3])
        Y.append([train_data[i][3]])
    X = np.array(X)
    Y = np.array(Y)
    X_pesudo = np.linalg.pinv(X, rcond=1e-15, hermitian=False)
    W_lin = np.matmul(X_pesudo, Y)

    # Calculate Ein (error on training dataset)
    Ein_acc = 0
    for i in range(N_TRAIN):
        s = np.dot(W_lin.reshape(3), np.array(train_data[i][:3]))
        if sign(train_data[i][3]*s) != 1:
            Ein_acc += 1 * (1/N_TRAIN)

    # Calculate Eout (error on training dataset)
    Eout_lin_acc = 0
    for i in range(N_TEST):
        s = np.dot(W_lin.reshape(3), np.array(test_data[i][:3]))
        if sign(test_data[i][3]*s) != 1:
            Eout_lin_acc += 1 * (1/N_TEST)

    ans_acc += abs(Ein_acc - Eout_lin_acc)

print("ans = " + str(ans_acc/N_REPEAT))
```

P15.B

```python
import random
import numpy as np
import math

N_TRAIN = 200 # Number of training data
N_TEST = 5000 # Number of testing data
N_REPEAT = 100
# Logistic Regression
N_MAX_ITERATION = 500
RHO = 0.1

def sign(x):
    if x >= 0:
        return 1
    else:
        return -1

def sigmoid(x):
    return math.exp(x) / (1 + math.exp(x))

Eout_lin_acc = 0
Eout_log_acc = 0
for seed in range(N_REPEAT):
    random.seed(a=seed)
    # Generate training/testing data
    train_data = []
    test_data = []
    for _ in range(N_TRAIN + N_TEST):
        y = random.randint(0, 1)
        if y == 1:
            x1 = np.random.normal(2, math.sqrt(0.6), 1)[0]
            x2 = np.random.normal(3, math.sqrt(0.6), 1)[0]
        elif y == 0: # -1
            y = -1
            x1 = np.random.normal(0, math.sqrt(0.4), 1)[0]
            x2 = np.random.normal(4, math.sqrt(0.4), 1)[0]
        if len(train_data) < N_TRAIN:
            train_data.append((1, x1, x2, y))
        else:
            test_data.append((1, x1, x2, y))

    # Linear regression
    X = []
    Y = []
    for i in range(N_TRAIN):
        X.append(train_data[i][:3])
        Y.append([train_data[i][3]])
    X = np.array(X)
    Y = np.array(Y)
    X_pesudo = np.linalg.pinv(X, rcond=1e-15, hermitian=False)
    W_lin = np.matmul(X_pesudo, Y)

    # Logistic regression
    W_log = np.array([0, 0, 0])
    for _ in range(N_MAX_ITERATION):
        acc = 0
        for i in range(N_TRAIN):
            xn = np.array(train_data[i][:3])
            yn = train_data[i][3]
            acc += (sigmoid(-1 * yn * np.dot(W_log, xn))) * (-1 * yn * xn)
        gradient = acc / (N_TRAIN)
        W_log = W_log - RHO * gradient

    # Calculate Linear Regression Eout (error on training dataset)
    for i in range(N_TEST):
        s = np.dot(W_lin.reshape(3), np.array(test_data[i][:3]))
        if sign(test_data[i][3]*s) != 1:
            Eout_lin_acc += 1 * (1/N_TEST)

    # Calculate Logistic Regression Eout (error on training dataset)
    for i in range(N_TEST):
        s = np.dot(W_log.reshape(3), np.array(test_data[i][:3]))
        if sign(test_data[i][3]*s) != 1:
            Eout_log_acc += 1 * (1/N_TEST)

print("(Eout_lin, Eout_log) = ({a}, {b})".format(a = str(Eout_lin_acc/N_REPEAT),
                                                  b = str(Eout_log_acc/N_REPEAT)))
```

P16.C

```
import random
import numpy as np
import math

N_TRAIN = 200 # Number of training data
N_TEST = 5000 # Number of testing data
N_REPEAT = 100
# Logistic Regression
N_MAX_ITERATION = 500
RHO = 0.1
# Outliner
N_OUTLINE = 20

def sign(x):
    if x >= 0:
        return 1
    else:
        return -1

def sigmoid(x):
    return math.exp(x) / (1 + math.exp(x))

Eout_lin_acc = 0
Eout_log_acc = 0
for seed in range(N_REPEAT):
    random.seed(a=seed)
    # Generate training/testing data
    train_data = []
    test_data = []
    for _ in range(N_TRAIN + N_TEST):
        y = random.randint(0, 1)
        if y == 1:
            x1 = np.random.normal(2, math.sqrt(0.6), 1)[0]
            x2 = np.random.normal(3, math.sqrt(0.6), 1)[0]
        elif y == 0: # -1
            y = -1
            x1 = np.random.normal(0, math.sqrt(0.4), 1)[0]
            x2 = np.random.normal(4, math.sqrt(0.4), 1)[0]
        if len(train_data) < N_TRAIN:
            train_data.append((1, x1, x2, y))
        else:
            test_data.append((1, x1, x2, y))

    # Outliner data
    for _ in range(N_OUTLINE):
        x1 = np.random.normal(6, math.sqrt(0.3), 1)[0]
        x2 = np.random.normal(0, math.sqrt(0.1), 1)[0]
        train_data.append((1, x1, x2, 1))

    # Linear regression
    X = []
    Y = []
    for i in range(N_TRAIN + N_OUTLINE):
        X.append(train_data[i][:3])
        Y.append([train_data[i][3]])
    X = np.array(X)
    Y = np.array(Y)
    X_pesudo = np.linalg.pinv(X, rcond=1e-15, hermitian=False)
    W_lin = np.matmul(X_pesudo, Y)

    # Logistic regression
    W_log = np.array([0, 0, 0])
    for _ in range(N_MAX_ITERATION):
        acc = 0
        for i in range(N_TRAIN + N_OUTLINE):
            xn = np.array(train_data[i][:3])
            yn = train_data[i][3]
            acc += (sigmoid(-1 * yn * np.dot(W_log, xn))) * (-1 * yn * xn)
        gradient = acc / (N_TRAIN + N_OUTLINE)
        W_log = W_log - RHO * gradient

    # Calculate Linear Regression Eout (error on training dataset)
    for i in range(N_TEST):
        s = np.dot(W_lin.reshape(3), np.array(test_data[i][:3]))
        if sign(test_data[i][3]*s) != 1:
            Eout_lin_acc += 1 * (1/N_TEST)

    # Calculate Logistic Regression Eout (error on training dataset)
    for i in range(N_TEST):
        s = np.dot(W_log.reshape(3), np.array(test_data[i][:3]))
        if sign(test_data[i][3]*s) != 1:
            Eout_log_acc += 1 * (1/N_TEST)

print("(Eout_lin, Eout_log) = ({a}, {b})".format(a = str(Eout_lin_acc/N_REPEAT),
                                                 b = str(Eout_log_acc/N_REPEAT)))
```