

Q1

$$\underbrace{x_0 = 1}_{\text{circled}} \quad \underbrace{XW = Y \Leftrightarrow W = X^{-1}Y}$$

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

$x_1 \qquad x_2 \qquad x_3$

$$x_2 - x_1 = (2, 0, -1)$$

$$x_3 - x_2 = (-1, 0, 1)$$

⇒ 三點共線

⇒ not shattered

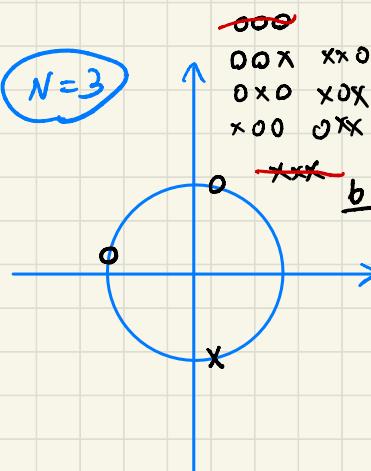
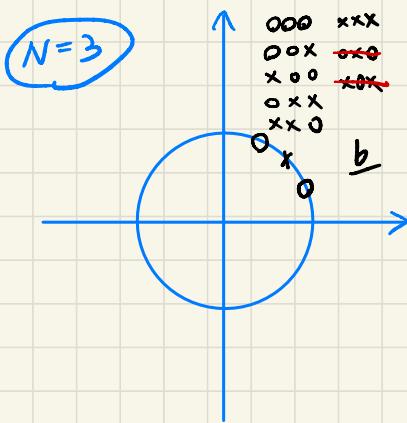
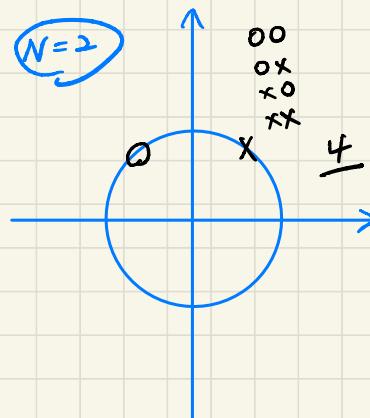
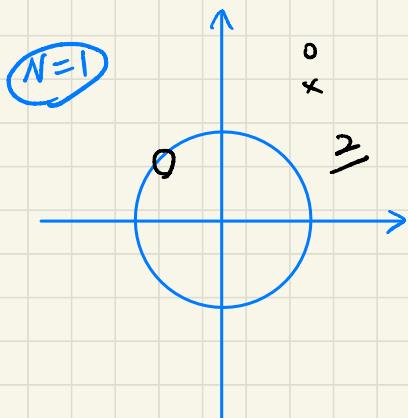
[b] inverse matrix exist. → shattered #

[c] singular matrix

[d] dvc = 4, N=5 → not shattered

Q2

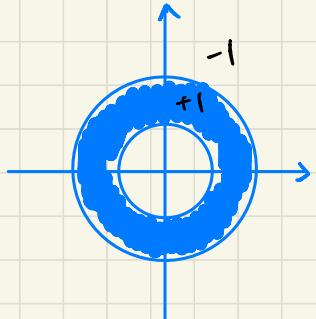
$$m_H(N) = \max_{x_1, \dots, x_N \in X} |H(x_1, x_2, \dots, x_N)|$$



Through the 4 examples above, $m_H(N)$ is $2N$. #

Q3

$$h(x) = \begin{cases} +1 & \text{if } a \leq \sum_{i=1}^d x_i^2 \leq b \\ -1 & \text{otherwise.} \end{cases} \quad (a, b > 0)$$



(like positive interval)

$$\underline{d=1}$$

$$N=3$$

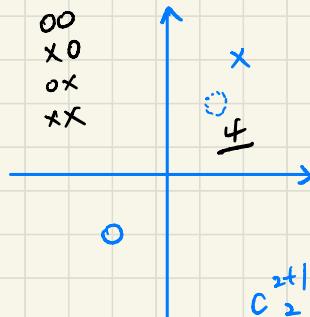


$$C_2^{2+1} + 1 = 7$$

$$\underline{d=2}$$

$$N=2$$

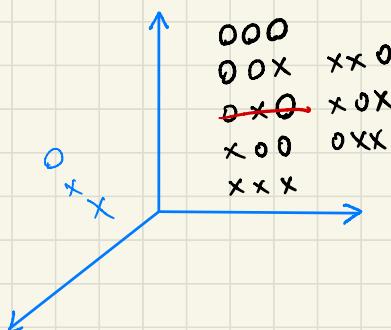
$$\begin{matrix} 00 \\ x0 \\ ox \\ xx \end{matrix}$$



$$C_2^{2+1} + 1 = 4$$

$$d=3$$

$$N=3$$



$$C_2^{3+1} + 1 = 7$$



$$\underline{C_2^{N+1} + 1}$$

$$* C_2^{N+1} + 1$$

$$= C_3^{3+1} + 1$$

$$= 4 + 1$$

$$= \cancel{X}$$

Q4

$$\text{growth function} \rightarrow C_2^{N+1} + 1$$

$$N=1, C_2^{1+1} + 1 = 2 = 2^1$$

$$N=2, C_2^{2+1} + 1 = 4 = 2^2$$

$$N=3, C_2^{3+1} + 1 = 7 \neq 2^3$$

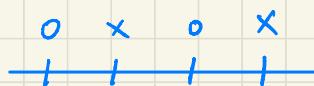
$$\Rightarrow \text{VC dimension} = 2$$

Q5

$$N=3$$



$$N=4$$



0000
000x
00Xx
0xxX

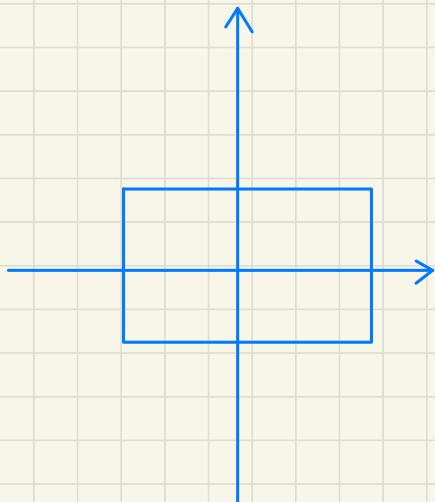
dvc = 4

$$N=5$$



~~0 x 0 x 0~~

[b]



$$N=1$$

$$N=2$$

0 x x x 0
0 0

$$N=3$$

0 0 0 x x
0 0

$$N=4$$

0 0 0 0
0 0

$N=5$ ：假設 rectangle 包含了 5 個點中具

(x) $x_{\max}, x_{\min}, y_{\max}, y_{\min}$ 特質

的點，則至少會存在1 個不具

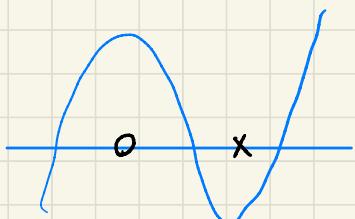
上述特質的點且其又被包含於 rectangle 內。因此若今天存在一組 dataset，且僅 不具上述特質的點

為 "-1"，則沒有任何 rectangle 能夠
區分這組 dataset。

$\Rightarrow \text{dvc should be } 4.$

$$[d] h(x) = \text{sign} \left(\sum_{i=0}^3 w_i x^i \right)$$

$$= \text{sign} (w_0 x^0 + w_1 x^1 + w_2 x^2 + w_3 x^3)$$



$N=1$

$N=2$

$N=3$

$N=4$

$N=5$

break point 發生
於 $N=5$
 $\Rightarrow \text{dvc} = 4$ #

000
00X
0X0
X00

0000
00X
0X0
X00

000
00X
0X0
X00

X00X0X

4 支點 (X)

[c]

There is a restriction in weight of perceptron : positively-biased. So I guess its VC dimension is less than $d+1 = 5$. And if we see it through degree of freedom, its w_0 is fixed to some extent. Hence, its real VC dimension could be $5-1 = \textcircled{4}$. #

Q6

$dvc \rightarrow \max N$ that let $m_H(N) = 2^N$

$$m_H(N) = \max_{x_1, x_2, \dots, x_N \in X} |H(x_1, x_2, \dots, x_N)|$$

H with 1126 binary classifiers

\Rightarrow can include up to 1024 dichotomies for 10 inputs

$$(\because 2^{10} < 1126 < 2^{11})$$

\Rightarrow largest possible value of $dvc(H) = 10$ #

Q7

Upper bound of $E_{\text{out}}(g) - E_{\text{out}}(g^*)$

$$P(|E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon) \leq 2M e^{-2\epsilon^2 N}$$
$$= \delta$$

$$\delta = 2M e^{-2\epsilon^2 N}$$

$$e^{-2\epsilon^2 N} = \frac{\delta}{2M}$$

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

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

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

with probability $\geq 1 - \delta$,

$$|E_{\text{in}}(h) - E_{\text{out}}(h)| \leq \sqrt{\frac{1}{2N} \ln \frac{2M}{\delta}}$$

$$-\sqrt{\frac{1}{2N} \ln \frac{2M}{\delta}} \leq E_{\text{in}}(h) - E_{\text{out}}(h) \leq \sqrt{\frac{1}{2N} \ln \frac{2M}{\delta}}$$

$$E_{\text{in}}(h) - \sqrt{\frac{1}{2N} \ln \frac{2M}{\delta}} \leq E_{\text{out}}(h) \leq E_{\text{in}}(h) + \sqrt{\frac{1}{2N} \ln \frac{2M}{\delta}}$$

In the inequality above, h can be any hypothesis in H .

Take example for dataset is linear separable.



For g

$$|E_{in}(g) - E_{out}(g)| < \epsilon$$

↓ equivalent (due to $E_{out}(g) \geq E_{in}(g)$)

$$E_{out}(g) - E_{in}(g) < \epsilon \quad \text{--- } ①$$

For g^*

$$|E_{in}(g^*) - E_{out}(g^*)| < \epsilon$$

↓ equivalent (due to $E_{in}(g^*) \geq E_{out}(g^*)$)

$$E_{in}(g^*) - E_{out}(g^*) < \epsilon \quad \text{--- } ②$$

① + ②

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

$$\therefore E_{in}(g^*) - E_{in}(g) \geq 0$$

$$\therefore E_{out}(g) - E_{out}(g^*) < 2\epsilon \quad \#$$

Q8

$$\underline{m_H(N) = N+1, \delta = 0.1}$$

$$4 m_H(2N) e^{-\frac{1}{8} \epsilon^2 N} = 4(2N+1) e^{-\frac{1}{8} \epsilon^2 N} \leq 0.1$$

$$\underline{\epsilon = 0.1}$$

$$\underline{\frac{4(2N+1) e^{-\frac{1}{8} \frac{1}{100} N}}{f(N)} \leq 0.1}$$

$$f(10000) \approx 0.298$$

$$f(11000) \approx 0.094 \checkmark \text{ (smallest } N)$$

$$f(12000) \approx 0.029$$

$$f(13000) \approx 0.009$$

$$f(14000) \approx 0.003$$

Q11

[c]

$XX^T = I_N$, the N by N identity matrix.

False.

Give an example :

$$X = \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix}$$

$$X^T = \begin{bmatrix} 0.1 & 0.2 \\ 0.1 & 0.2 \end{bmatrix}$$

$$XX^T = \begin{bmatrix} 0.2 & 0.4 \\ 0.4 & 0.8 \end{bmatrix} \neq \text{identity matrix.}$$

#

Q12

The maximum of the likelihood function will happen in the w which makes $w^T x_n$ equal to mean (M)
(Because for normal distribution, the position of its mean has the largest probability density)

Hence, we just need to choose a w which is just the same as the weight output of linear regression algorithm (W_{lin}). ($W_{lin}^T x_n$ will be equal to mean of $p(y|x_n) \rightarrow$ position of least square error)

That w is $[a] = (x^T x)^{-1} x^T y$. #

Q13

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Created on Sat Nov  6 10:09:13 2021
5
6 @author: md703
7 """
8
9 import numpy as np
10
11
12 def generateSamples(N):
13     samples = np.empty((N, 4)) # 4 --> x0, x1, x2, y
14     # y
15     samples[:, 3] = np.random.randint(0, 2, N)
16     samples[:, 3] = samples[:, -1]*2 - 1
17     # x0
18     samples[:, 0] = 1
19     # x1, x2
20     samples[samples[:, 3]==1, 1:3] = +\
21         np.random.multivariate_normal(mean=[2, 3],
22                                         cov=[[0.6, 0], [0, 0.6]],
23                                         size=sum(samples[:, 3]==1))
24     samples[samples[:, 3]==-1, 1:3] = +\
25         np.random.multivariate_normal(mean=[0, 4],
26                                         cov=[[0.4, 0], [0, 0.4]],
27                                         size=sum(samples[:, 3]==-1))
28
29
30
31 EinSet = []
32 for idx in range(100):
33     # generate data with a specific seed
34     np.random.seed(idx)
35     train = generateSamples(200)
36
37     # linear regression
38     dagger = np.linalg.pinv(train[:, :3])
39     wlin = np.matmul(dagger, train[:, 3])
40
41     # append squared-error for this wlin
42     EinSet.append(np.square(np.matmul(train[:, :3], wlin) - train[:, 3]).mean())
43
44 # final result
45 EinSetMean = np.array(EinSet).mean()
```

Q14

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 """
4 Created on Sat Nov  6 11:16:01 2021
5
6 @author: md703
7 """
8
9 import numpy as np
10
11
12 def generateSamples(N):
13     samples = np.empty((N, 4)) # 4 --> x0, x1, x2, y
14     # y
15     samples[:, 3] = np.random.randint(0, 2, N)
16     samples[:, 3] = samples[:, -1]*2 - 1
17     # x0
18     samples[:, 0] = 1
19     # x1, x2
20     samples[samples[:, 3]==1, 1:3] = +\
21         np.random.multivariate_normal(mean=[2, 3],
22                                         cov=[[0.6, 0], [0, 0.6]],
23                                         size=sum(samples[:, 3]==1))
24     samples[samples[:, 3]==-1, 1:3] = +\
25         np.random.multivariate_normal(mean=[0, 4],
26                                         cov=[[0.4, 0], [0, 0.4]],
27                                         size=sum(samples[:, 3]==-1))
28
29
30
31 differenceSet = []
32 for idx in range(100):
33     # generate data with a specific seed
34     np.random.seed(idx)
35     train = generateSamples(200)
36     test = generateSamples(5000)
37
38     # linear regression
39     dagger = np.linalg.pinv(train[:, :3])
40     wlin = np.matmul(dagger, train[:, 3])
41
42     # calculate Ein for this wlin
43     trainPred = np.matmul(train[:, :3], wlin)
44     trainPred[trainPred>=0] = 1
45     trainPred[trainPred<0] = -1
46     Ein = np.mean(trainPred != train[:, 3])
47
48     # calculate Eout for this wlin
49     testPred = np.matmul(test[:, :3], wlin)
50     testPred[testPred>=0] = 1
51     testPred[testPred<0] = -1
52     Eout = np.mean(testPred != test[:, 3])
53
54     # append difference
55     differenceSet.append(abs(Ein-Eout))
56
57 # final result
58 differenceSetMean = np.array(differenceSet).mean()
```

Q15

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created on Wed Nov 10 22:47:16 2021
5
6  @author: md703
7  """
8
9  import numpy as np
10
11 def generateSamples(N):
12     samples = np.empty((N, 4)) # 4 --> x0, x1, x2, y
13     # y
14     samples[:, 3] = np.random.randint(0, 2, N)
15     samples[:, 3] = samples[:, -1]*2 - 1
16     # x0
17     samples[:, 0] = 1
18     # x1, x2
19     samples[samples[:, 3]==1, 1:3] = +\
20         np.random.multivariate_normal(mean=[2, 3],
21                                         cov=[[0.6, 0], [0, 0.6]],
22                                         size=sum(samples[:, 3]==1))
23     samples[samples[:, 3]==-1, 1:3] = +\
24         np.random.multivariate_normal(mean=[0, 4],
25                                         cov=[[0.4, 0], [0, 0.4]],
26                                         size=sum(samples[:, 3]==-1))
27
28
29
30 def sigmoid(s):
31     return 1/(1+np.exp(-s))
32
33
34 if __name__ == '__main__':
35     # parameters
36     trainN = 200
37     testN = 5000
38     eta = 0.1
39     T = 500
40     repeatTimes = 100
41
42     # repeat process
43     resultSet = np.empty((repeatTimes, 2))
44     for idx in range(repeatTimes):
45         # generate data with a specific seed
46         np.random.seed(idx)
47         train = generateSamples(trainN)
48         test = generateSamples(testN)
49
50         # linear regression
51         dagger = np.linalg.pinv(train[:, :3])
52         wlin = np.matmul(dagger, train[:, 3])
53         testPred = np.matmul(test[:, :3], wlin)
54         testPred[testPred>=0] = 1
55         testPred[testPred<0] = -1
56         EoutLinregress = np.mean(testPred != test[:, 3])
57
58         # logistic regression
59         w = np.zeros(train.shape[1]-1)
60         for _ in range(T):
61             gradient = np.mean(sigmoid(-train[:, -1] * np.matmul(w, train[:, :3].T)) *
62                                 (-train[:, -1].reshape(-1, 1)*train[:, :3]).T, axis=1)
63             w = w - eta*gradient
64             testPred = sigmoid(np.matmul(w, test[:, :3].T))
65             testPred[testPred>=0.5] = 1
66             testPred[testPred<0.5] = -1
67             EoutLogregress = np.mean(testPred != test[:, 3])
68
69         # save the result of this process
70         resultSet[idx] = [EoutLinregress, EoutLogregress]
71
72     # final result
73     resultSetMean = resultSet.mean(axis=0)
```

Q1b

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created on Thu Nov 11 00:00:58 2021
5
6  @author: md703
7  """
8
9  import numpy as np
10
11 def generateSamples(N):
12     samples = np.empty((N, 4)) # 4 --> x0, x1, x2, y
13     # y
14     samples[:, 3] = np.random.randint(0, 2, N)
15     samples[:, 3] = samples[:, -1]*2 - 1
16     # x0
17     samples[:, 0] = 1
18     # x1, x2
19     samples[samples[:, 3]==1, 1:3] = +\
20         np.random.multivariate_normal(mean=[2, 3],
21                                         cov=[[0.6, 0], [0, 0.6]],
22                                         size=sum(samples[:, 3]==1))
23     samples[samples[:, 3]==-1, 1:3] = +\
24         np.random.multivariate_normal(mean=[0, 4],
25                                         cov=[[0.4, 0], [0, 0.4]],
26                                         size=sum(samples[:, 3]==-1))
27
28
29
30 def generateOutliers(N):
31     outLiers = np.empty((N, 4)) # 4 --> x0, x1, x2, y
32     # y
33     outLiers[:, 3] = 1
34     # x0
35     outLiers[:, 0] = 1
36     # x1, x2
37     outLiers[:, 1:3] = +\
38         np.random.multivariate_normal(mean=[6, 0],
39                                         cov=[[0.3, 0], [0, 0.1]],
40                                         size=sum(outLiers[:, 3]==1))
41
42
43
44 def sigmoid(s):
45     return 1/(1+np.exp(-s))
46
47
48 if __name__ == '__main__':
49     # parameters
50     trainN = 200
51     outliersN = 20
52     testN = 5000
53     eta = 0.1
54     T = 500
55     repeatTimes = 100
56
57     # repeat process
58     resultSet = np.empty((repeatTimes, 2))
59     for idx in range(repeatTimes):
60         # generate data with a specific seed
61         np.random.seed(idx)
62         train = generateSamples(trainN)
63         outliers = generateOutliers(outliersN) # generate outliers
64         train = np.concatenate((train, outliers), axis=0) # add outliers
65         test = generateSamples(testN)
66
67         # linear regression
68         dagger = np.linalg.pinv(train[:, :3])
69         wlin = np.matmul(dagger, train[:, 3])
70         testPred = np.matmul(test[:, :3], wlin)
71         testPred[testPred>=0] = 1
72         testPred[testPred<0] = -1
73         EoutLinregress = np.mean(testPred != test[:, 3])
74
75         # logistic regression
76         w = np.zeros(train.shape[1]-1)
77         for _ in range(T):
78             gradient = np.mean(sigmoid(-train[:, -1] * np.matmul(w, train[:, :3].T)) *
79                                 (-train[:, -1].reshape(-1, 1)*train[:, :3].T, axis=1))
80             w = w - eta*gradient
81             testPred = sigmoid(np.matmul(w, test[:, :3].T))
82             testPred[testPred>=0.5] = 1
83             testPred[testPred<0.5] = -1
84             EoutLogregress = np.mean(testPred != test[:, 3])
85
86         # save the result of this process
87         resultSet[idx] = [EoutLinregress, EoutLogregress]
88
89     # final result
90     resultSetMean = resultSet.mean(axis=0)
```