Q1

$x_0 = 1$ $\quad XW = Y \Leftrightarrow W = X^{-1}Y$

[a] $\{(2,3,4), (4,3,2), (3,3,3)\}$
$\quad\quad x_1 \quad\quad x_2 \quad\quad x_3$

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

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

$\Rightarrow$ 三點共線

$\Rightarrow$ not shattered

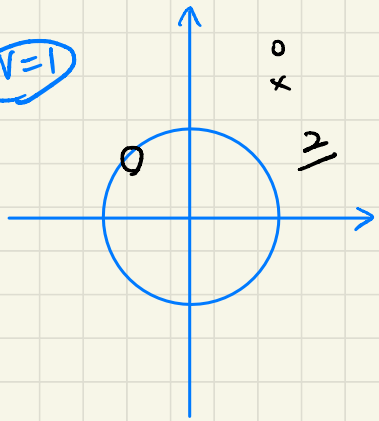
[b] inverse matrix exist. $\rightarrow$ shattered #

[c] singular matrix

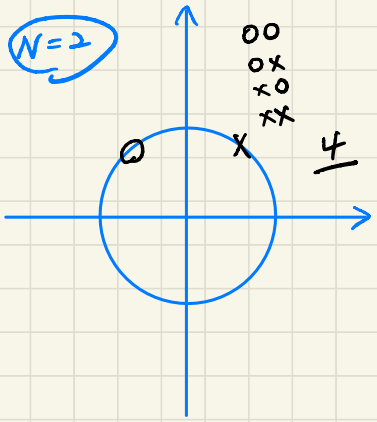
[d] dvc = 4, $N = 5$ $\rightarrow$ not shattered

Q2 $m_H(N) = \max_{x_1,\dots,x_N \in X} |H(x_1, x_2, \dots, x_N)|$

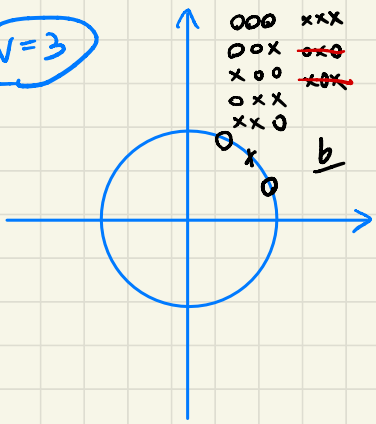
N=1



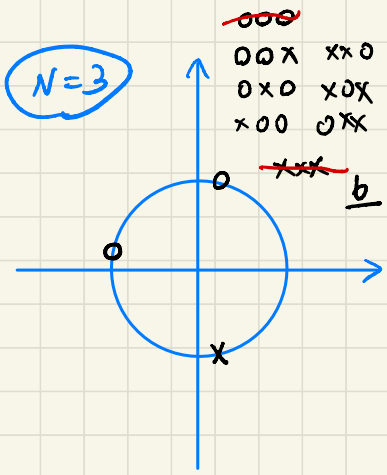
N=2



N=3



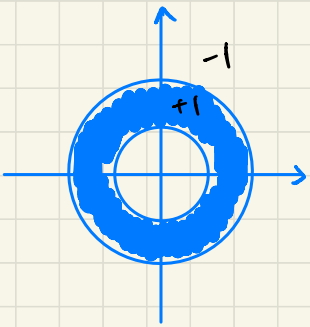
N=3



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

Q3

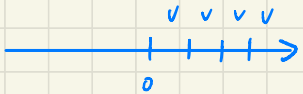
$$h(x) = \begin{cases} +1 & \text{if } a \le \sum_{i=1}^{d} x_i^2 \le b \quad (a, b > 0) \\ -1 & \text{otherwise.} \end{cases}$$



(like positive interval)

d=1

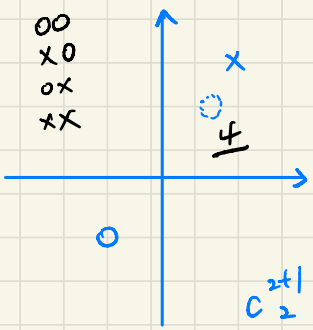
N=3



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

d=2

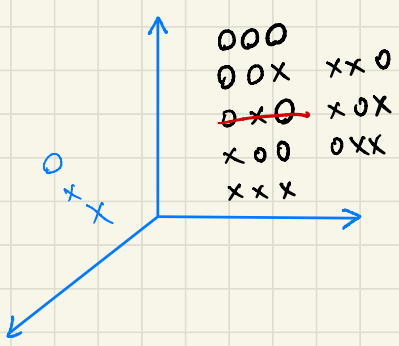
N=2



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

d=3

N=3



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

$C_2^{N+1} + 1$

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

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

$= 4 + 1$

$= 5$

Q4

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

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

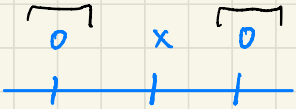
$N=2,\ C_2^{2+1} + 1 = 4 = 2^2$

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

$\Rightarrow$ VC dimension $= 2$

Q5

[a]



$N = 3$

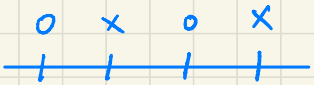
ooo xxx

oxx xoo

xox oxo

xxo oox

$N = 4$



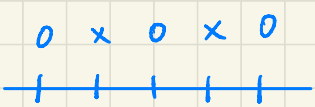
oooo

ooox

ooxx

oxxx

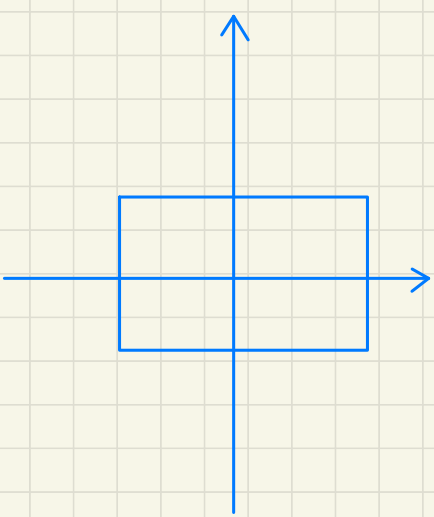
$N = 5$



~~oxoxo~~

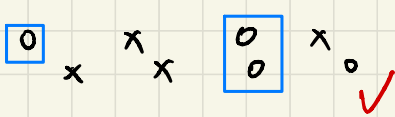
$d_{VC} = 4$

[b]



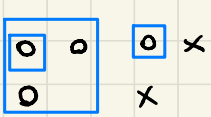
$N = 1$

$N = 2$



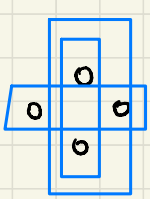
✓

$N = 3$



✓

$N = 4$



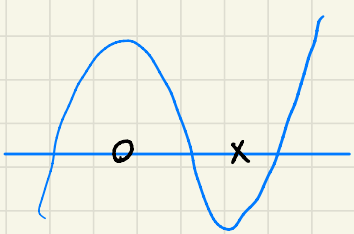
✓

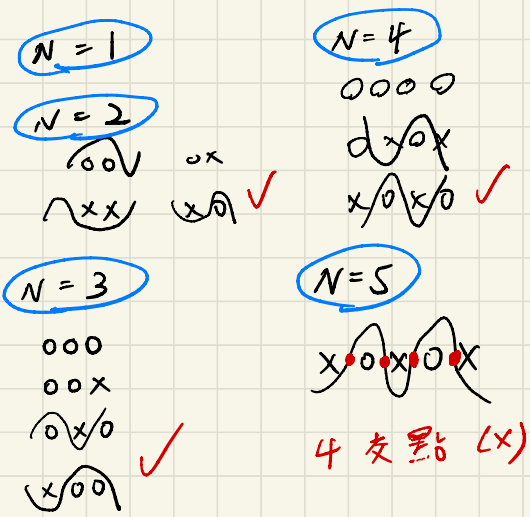
$N=5$：假設 rectangle 包含 5 個點中具 x_{max}、x_{min}、y_{max}、y_{min} 特質的點，則至少會存在1個不具上述特質的點且其又被包含於 rectangle 內，因此若今天存在一組 dataset，且僅不具上述特質的點為"-1"，則沒有任何 rectangle 能夠區分這組 dataset。 (X)

$\Rightarrow$ dvc should be 4.

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

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





break point 發生於 $N=5$

$\Rightarrow$ dvc = 4 #

[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 $5-1=4$. #

2b

$d_{VC} \rightarrow$ max N that let $m_H(N) = 2^N$

$$m_H(N) = \max_{x_1, x_2, \cdots, x_N \in \mathcal{X}} \left| H(x_1, x_2, \cdots, x_N) \right|$$

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 $d_{VC}(H) = 10$ #

27

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

$$P(|E_{in}(h) - E_{out}(h)| > \epsilon) \leq 2Me^{-2\epsilon^2 N}$$

$$= \delta$$

$$\delta = 2Me^{-2\epsilon^2 N}$$

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

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

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

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

with probability $\geq 1 - \delta$,

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

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

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

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

Take example for dataset is linear separable.

For g

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

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

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

For g^*

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

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

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

① + ②

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

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

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

Q8

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

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

$\epsilon = 0.1$

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

$f(10000) \approx 0.298$

$f(11000) \approx 0.094$ ✓ (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 (μ)

(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 $p(y|x_n)$ → position of least square error)

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

Q13

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Nov  6 10:09:13 2021

@author: md703
"""

import numpy as np


def generateSamples(N):
    samples = np.empty((N, 4))  # 4 --> x0, x1, x2, y
    # y
    samples[:, 3] = np.random.randint(0, 2, N)
    samples[:, 3] = samples[:, -1]*2 - 1
    # x0
    samples[:, 0] = 1
    # x1, x2
    samples[samples[:, 3]==1, 1:3] = +\
        np.random.multivariate_normal(mean=[2, 3],
                                      cov=[[0.6, 0], [0, 0.6]],
                                      size=sum(samples[:, 3]==1))
    samples[samples[:, 3]==-1, 1:3] = +\
        np.random.multivariate_normal(mean=[0, 4],
                                      cov=[[0.4, 0], [0, 0.4]],
                                      size=sum(samples[:, 3]==-1))
    return samples


EinSet = []
for idx in range(100):
    # generate data with a specific seed
    np.random.seed(idx)
    train = generateSamples(200)

    # linear regression
    dagger = np.linalg.pinv(train[:, :3])
    wlin = np.matmul(dagger, train[:, 3])

    # append squared-error for this wlin
    EinSet.append(np.square(np.matmul(train[:, :3], wlin) - train[:, 3]).mean())

# final result
EinSetMean = np.array(EinSet).mean()
```

Q14

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Nov  6 11:16:01 2021

@author: md703
"""

import numpy as np


def generateSamples(N):
    samples = np.empty((N, 4))  # 4 --> x0, x1, x2, y
    # y
    samples[:, 3] = np.random.randint(0, 2, N)
    samples[:, 3] = samples[:, -1]*2 - 1
    # x0
    samples[:, 0] = 1
    # x1, x2
    samples[samples[:, 3]==1, 1:3] = +\
        np.random.multivariate_normal(mean=[2, 3],
                                      cov=[[0.6, 0], [0, 0.6]],
                                      size=sum(samples[:, 3]==1))
    samples[samples[:, 3]==-1, 1:3] = +\
        np.random.multivariate_normal(mean=[0, 4],
                                      cov=[[0.4, 0], [0, 0.4]],
                                      size=sum(samples[:, 3]==-1))
    return samples


differenceSet = []
for idx in range(100):
    # generate data with a specific seed
    np.random.seed(idx)
    train = generateSamples(200)
    test = generateSamples(5000)

    # linear regression
    dagger = np.linalg.pinv(train[:, :3])
    wlin = np.matmul(dagger, train[:, 3])

    # calculate Ein for this wlin
    trainPred = np.matmul(train[:, :3], wlin)
    trainPred[trainPred>=0] = 1
    trainPred[trainPred<0] = -1
    Ein = np.mean(trainPred != train[:, 3])

    # calculate Eout for this wlin
    testPred = np.matmul(test[:, :3], wlin)
    testPred[testPred>=0] = 1
    testPred[testPred<0] = -1
    Eout = np.mean(testPred != test[:, 3])

    # append difference
    differenceSet.append(abs(Ein-Eout))

# final result
differenceSetMean = np.array(differenceSet).mean()
```

Q15

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Nov 10 22:47:16 2021

@author: md703
"""

import numpy as np

def generateSamples(N):
    samples = np.empty((N, 4))  # 4 --> x0, x1, x2, y
    # y
    samples[:, 3] = np.random.randint(0, 2, N)
    samples[:, 3] = samples[:, -1]*2 - 1
    # x0
    samples[:, 0] = 1
    # x1, x2
    samples[samples[:, 3]==1, 1:3] = +\
        np.random.multivariate_normal(mean=[2, 3],
                                      cov=[[0.6, 0], [0, 0.6]],
                                      size=sum(samples[:, 3]==1))
    samples[samples[:, 3]==-1, 1:3] = +\
        np.random.multivariate_normal(mean=[0, 4],
                                      cov=[[0.4, 0], [0, 0.4]],
                                      size=sum(samples[:, 3]==-1))
    return samples


def sigmoid(s):
    return 1/(1+np.exp(-s))


if __name__ == '__main__':
    # parameters
    trainN = 200
    testN = 5000
    eta = 0.1
    T = 500
    repeatTimes = 100

    # repeat process
    resultSet = np.empty((repeatTimes, 2))
    for idx in range(repeatTimes):
        # generate data with a specific seed
        np.random.seed(idx)
        train = generateSamples(trainN)
        test = generateSamples(testN)

        # linear regression
        dagger = np.linalg.pinv(train[:, :3])
        wlin = np.matmul(dagger, train[:, 3])
        testPred = np.matmul(test[:, :3], wlin)
        testPred[testPred>=0] = 1
        testPred[testPred<0] = -1
        EoutLinregress = np.mean(testPred != test[:, 3])

        # logistic regression
        w = np.zeros(train.shape[1]-1)
        for _ in range(T):
            gradient = np.mean(sigmoid(-train[:, -1] *  np.matmul(w, train[:, :3].T)) *
                               (-train[:, -1].reshape(-1, 1)*train[:, :3]).T, axis=1)
            w = w - eta*gradient
        testPred = sigmoid(np.matmul(w, test[:, :3].T))
        testPred[testPred>=0.5] = 1
        testPred[testPred<0.5] = -1
        EoutLogregress = np.mean(testPred != test[:, 3])

        # save the result of this process
        resultSet[idx] = [EoutLinregress, EoutLogregress]

# final result
resultSetMean = resultSet.mean(axis=0)
```

Q16

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Nov 11 00:00:58 2021

@author: md703
"""

import numpy as np

def generateSamples(N):
    samples = np.empty((N, 4))  # 4 --> x0, x1, x2, y
    # y
    samples[:, 3] = np.random.randint(0, 2, N)
    samples[:, 3] = samples[:, -1]*2 - 1
    # x0
    samples[:, 0] = 1
    # x1, x2
    samples[samples[:, 3]==1, 1:3] = +\
        np.random.multivariate_normal(mean=[2, 3],
                                      cov=[[0.6, 0], [0, 0.6]],
                                      size=sum(samples[:, 3]==1))
    samples[samples[:, 3]==-1, 1:3] = +\
        np.random.multivariate_normal(mean=[0, 4],
                                      cov=[[0.4, 0], [0, 0.4]],
                                      size=sum(samples[:, 3]==-1))
    return samples


def generateOutliers(N):
    outLiers = np.empty((N, 4))  # 4 --> x0, x1, x2, y
    # y
    outLiers[:, 3] = 1
    # x0
    outLiers[:, 0] = 1
    # x1, x2
    outLiers[:, 1:3] = +\
        np.random.multivariate_normal(mean=[6, 0],
                                      cov=[[0.3, 0], [0, 0.1]],
                                      size=sum(outLiers[:, 3]==1))
    return outLiers


def sigmoid(s):
    return 1/(1+np.exp(-s))


if __name__ == '__main__':
    # parameters
    trainN = 200
    outliersN = 20
    testN = 5000
    eta = 0.1
    T = 500
    repeatTimes = 100

    # repeat process
    resultSet = np.empty((repeatTimes, 2))
    for idx in range(repeatTimes):
        # generate data with a specific seed
        np.random.seed(idx)
        train = generateSamples(trainN)
        outliers = generateOutliers(outliersN)  # generate outliers
        train = np.concatenate((train, outliers), axis=0)  # add outliers
        test = generateSamples(testN)

        # linear regression
        dagger = np.linalg.pinv(train[:, :3])
        wlin = np.matmul(dagger, train[:, 3])
        testPred = np.matmul(test[:, :3], wlin)
        testPred[testPred>=0] = 1
        testPred[testPred<0] = -1
        EoutLinregress = np.mean(testPred != test[:, 3])

        # logistic regression
        w = np.zeros(train.shape[1]-1)
        for _ in range(T):
            gradient = np.mean(sigmoid(-train[:, -1] * np.matmul(w, train[:, :3].T)) *
                               (-train[:, -1].reshape(-1, 1)*train[:, :3]).T, axis=1)
            w = w - eta*gradient
        testPred = sigmoid(np.matmul(w, test[:, :3].T))
        testPred[testPred>=0.5] = 1
        testPred[testPred<0.5] = -1
        EoutLogregress = np.mean(testPred != test[:, 3])

        # save the result of this process
        resultSet[idx] = [EoutLinregress, EoutLogregress]

# final result
resultSetMean = resultSet.mean(axis=0)
```