# **HTML 2023 HW3**

tags: Personal

Discuss with anonymous and TAs.

## **Linear Models and More**

#### **P1**

- ullet CPU time: aN for training a binary classifier on a size N binary classification data set
- ullet We have size-N K-class classification data set, which means each class is of size  $rac{N}{K}$
- By performing one-versus-one(OVO), we need to compute  $\binom{K}{2}$  binary classification on the dataset.
- For each binary calssification, we only select 2 classes for computing the result, which means for each binary classification, only  $2 \times \frac{N}{K}$  dataset will be used.
- Therefore, the total CPU time is

$$ainom{K}{2} imes 2 imes rac{N}{K}=arac{K!}{2!(K-2)!}rac{2N}{K}=a(K-1)N$$

As a result, we should choose  $\left[b\right]$  as our solution.

## **P2**

Since quadratic hypothesis can implement all possible quadratic curve boundaries, such as circle, hyperbola, and parabola, and includes line, and constants. Therefore, we can assumed that  $x_i=(x_{i1},x_{i2})$ , and transformation function  $\Phi_2(x)=(1,x_1,x_2,x_1^2,x_1x_2,x_2^2)=z$ . From the transformation, we know that the  $d_{vc}(\mathcal{H}_{\Phi_2(x)})\leq 6$ , it is trivial that six inputs can be shattered after the transformation. For a more detailed calculation, we have

$$egin{array}{lll} x_1 &= (2,0), & x_2 &= (0,2), \ x_3 &= (-2,0) \ x_4 &= (0,-2), \ x_5 &= (0,0), \ x_6 &= (1,1) \end{array}$$

$$egin{array}{ll} \Phi_2(x_1) = (1,2,0,4,0,0) &= z_1 \ \Phi_2(x_2) = (1,0,2,0,0,4) &= z_2 \ \Phi_2(x_3) = (1,-2,0,4,0,0) &= z_3 \ \Phi_2(x_4) = (1,0,-2,0,0,4) &= z_4 \ \Phi_2(x_5) = (1,0,0,0,0,0) &= z_5 \ \Phi_2(x_6) = (1,1,1,1,1,1) &= z_6 \end{array}$$

$$z = egin{bmatrix} z_1^T \ z_2^T \ z_3^T \ z_4^T \ z_5^T \ z_6^T \end{bmatrix} = egin{bmatrix} 1 & 2 & 0 & 4 & 0 & 0 \ 1 & 0 & 2 & 0 & 0 & 4 \ 1 & -2 & 0 & 4 & 0 & 0 \ 1 & 0 & -2 & 0 & 0 & 4 \ 1 & 0 & 0 & 0 & 0 & 0 \ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}, w \ = egin{bmatrix} w_1^T \ w_2^T \ w_3^T \ w_4^T \ w_5^T \ w_6^T \end{bmatrix}$$

Since z is invertible, let y=zw, then for each  $y_i, 0 \le i \le 6$ , there exists a  $w=z^{-1}y$ , which means  $z_1, z_2, z_3, z_4, z_5, z_6$  can be shattered by quadratic hypothesis set, therefore,  $x_1, x_2, x_3, x_4, x_5, x_6$  can also be shattered by the union of quadratic, linear, or constant hypothesis sets.

As a result, we should choose  $\left[e\right]$  as our solution.

**P3** 

By hand

$$egin{array}{ll} x_1=(0,0), & y_1=-1 \ x_2=(4,0), & y_2=+1 \ x_3=(-4,0), & y_3=+1 \ x_4=(0,2), & y_4=-1 \ x_5=(0,-2), & y_5=-1 \ \end{array} \ egin{array}{ll} w_1=(-1,0,0,0.5,0,-0.5) \ w_2=(-1,0,0,-0.5,0,0.5) \ w_3=(-2,0,0,1,0,1) \ w_4=(-1,0,0,0.2,0,0.1) \end{array}$$

$$egin{array}{ll} \Phi(x) &= (1,x_1,x_2,x_1^2,x_1x_2,x_2^2) \ \Phi(x_1) &= (1,0,0,0,0,0) \ \Phi(x_2) &= (1,4,0,16,0,0) \ \Phi(x_3) &= (1,-4,0,16,0,0) \ \Phi(x_4) &= (1,0,2,0,0,4) \ \Phi(x_5) &= (1,0,-2,0,0,4) \end{array}$$

	$\mathrm{sign}(w_i^T\phi(x_i))$	$y_i$	$\mathrm{sign}(w_i^T\phi(x_i)))=y_i$
$w_1$	[-1,1,1,-1,-1]	[-1,1,1,-1,-1]	[0,0,0,0,0]
$w_2$	[-1,-1,-1,1,1]	[-1,1,1,-1,-1]	[O,X,X,X,X]
$w_3$	[-1,1,1,1,1]	[-1,1,1,-1,-1]	[O,O,O,X,X]
$w_4$	[-1,1,1,-1,-1]	[-1,1,1,-1,-1]	[0,0,0,0,0]

#### By python

```
1
     import numpy as np
 2
 3
     def phi(x):
 4
         return [1, x[0], x[1], x[0]*x[0], x[0]*x[1], x[1]*x[1]
 5
     def main(dataset, weight_vector):
 6
 7
         result = 0
 8
         for i in range(len(weight_vector)):
             print("weight_vector: ", weight_vector[i])
 9
             counter = 0
10
11
             for j in range(len(dataset)):
                 hypothesis = int(np.sign(np.dot(weight_vector[i], phi(dataset[j]))))
12
                 if hypothesis == dataset[j][2]:
13
                      print("0:", "(", hypothesis, ",", dataset[j][2], ")")
14
15
                      counter += 1
16
                 else:
                      print("X:", "(", hypothesis, ",", dataset[j][2], ")")
17
             if counter == len(dataset):
18
                 print("===separates the dataset correct===")
19
20
                 result += 1
21
             else:
22
                 print("===separates the dataset incorrect===")
23
         print("There are", result, "weight vectors that can seperate dataset.")
```

```
if __name__ == '__main__':
1
2
         dataset = np.array([
 3
             # x_1,x_2, y
             [0, 0, -1],
 4
             [4, 0, 1],
 5
             [-4, 0, 1],
 6
7
             [0, 2, -1],
             [0, -2, -1],
8
9
         ])
10
         weight_vector = np.array([
11
             [-1, 0, 0, 0.5, 0, -0.5],
12
             [-1, 0, 0, -0.5, 0, 0.5],
13
14
             [-2, 0, 0, 1, 0, 1],
             [-1, 0, 0, 0.2, 0, 0.1],
15
16
         ])
17
         main(dataset, weight_vector)
18
19
```

```
1
    weight_vector: [-1. 0. 0. 0.5 0. -0.5]
2
    0: (-1,-1)
3
    0: (1,1)
4
    0: (1,1)
5
    0: (-1,-1)
    0: (-1,-1)
6
7
    ===separates the dataset correct===
    weight_vector: [-1. 0. 0. -0.5 0. 0.5]
8
9
    0: (-1,-1)
    X: (-1, 1)
10
11
    X: (-1,1)
    X: (1,-1)
12
13
    X: (1,-1)
    ===separates the dataset incorrect===
14
15
    weight vector: [-2. 0. 0. 1. 0. 1.]
16
    0: (-1,-1)
    0: (1,1)
17
    0: (1,1)
18
19
    X: (1,-1)
20
    X: (1,-1)
21
    ===separates the dataset incorrect===
22
    weight vector: [-1. 0. 0. 0.2 0.
23
    0: ( -1 , -1 )
24
    0: (1,1)
25
    0: (1,1)
26
    0: (-1,-1)
    0: (-1,-1)
27
    ===separates the dataset correct===
28
29
    There are 2 weight vectors that can seperate dataset.
```

As a result, we should choose  $\left[c\right]$  as our solution.

$$egin{aligned} \phi(x) &= [(\Gamma x_n)^T] = x \Gamma^T \ & ilde{w} = ((x \Gamma^T)^T (x \Gamma^T))^{-1} (x \Gamma^T)^T y \ &= ((x \Gamma^T)^{-1} ((x \Gamma^T)^T)^{-1} (x \Gamma^T)^T y \ &= (x \Gamma^T)^{-1} y \ &= (\Gamma^T)^{-1} x^{-1} y \ &= (\Gamma^T)^{-1} x^{-1} y \ &= x^{-1} (x^T)^{-1} x^T y \ &= x^{-1} (x^T)^{-1} x^T y \ &= x^{-1} y \ & ilde{w} = (\Gamma^T)^{-1} w_{lin} \ &w_{lin} = \Gamma^T ilde{w} \ &E_{in}( ilde{w}) = rac{1}{N} \sum_{n=1}^N ((\Gamma x_n)^T ilde{w} - y_n)^2 \ &= rac{1}{N} \sum_{n=1}^N (x_n^T \Gamma^T (\Gamma^T)^{-1} w_{lin} - y_n)^2 \ &= rac{1}{N} \sum_{n=1}^N (x_n^T w_{lin} - y_n)^2 \ &= E_{in}(w_{lin}) \end{aligned}$$

As a result, we should choose  $\left[b\right]$  as our solution.

### **P5**

Define  $\Phi_{(k)}(x)=(1,x_k)=(w_0,w_1)$ , then we have  $w_0+w_1x_k=0\Leftrightarrow x_k=-\frac{w_0}{w_1}$  (threshold), which means if  $x_k$  is larger than threshold, label +1 otherwize, label -1. Hence, we can consider this feature transform as a decision stumps where  $m_H(N)=2N$  for any  $\Phi_{(k)}(x)=(1,x_k)$ . Except for all positive or all negative conditions, there exists N inputs and N-1 intervals between input values can be chosen by  $\mathcal{H}_k$ , this tells us that there are 2(N-1) possible conditions, and there are 2 conditions should be included, which is all positive or all negatives. Moreover, there have d transformation functions, which means  $\bigcup_{k=1}^d$  have  $m_H(N)=2(N-1)d+2=2Nd$  From the definition of  $d_{vc}$ , we have

$$egin{aligned} &2^N \leq 2(N-1)d+2 = 2Nd \ &\Rightarrow rac{2N}{2} \leq Nd \Rightarrow 2^{N-1} \leq Nd \ &\Rightarrow N-1 \leq \log_2 N + \log_2 d \leq rac{N}{2} + \log_2 d \because \log_2 d \leq rac{d}{2} ext{(by hint)} \ &\Rightarrow 2N-2 \leq N+2\log_2 d \ &\Rightarrow N \leq 2\log_2 d + 2 = 2(\log_2 d + 1) \ &\therefore d_{vc}(igcup_{k=1}^d \mathcal{H}_k) \leq 2(\log_2 d + 1) \end{aligned}$$

As a result, we should choose [b] as our solution.

#### **P6**

The feature transform means for each x in dataset, we have the transformation

$$\Phi(x) = z = egin{bmatrix} [x = x_1] \ [x = x_2] \ dots \ [x = x_N] \end{bmatrix}$$

For each  $[x=x_i], i=1,2,\ldots,N$  means if  $x=x_i$ , then the value is 1 otherwise 0. From the assumption, it shows that all  $x_n$  are different, which means

$$\Phi(x_1)=z_1=egin{bmatrix}1\0\dots\0\end{bmatrix}, \Phi(x_2)=z_2=egin{bmatrix}0\1\dots\0\end{bmatrix}, \cdots, \Phi(x_N)=z_N=egin{bmatrix}0\0\dots\1\end{bmatrix}$$

Therefore,  $(\Phi(x))_n$  is a one-hot table. Then let

$$z = egin{bmatrix} 1 & z_1^T \ 1 & z_2^T \ dots & dots \ 1 & z_N^T \end{bmatrix} = I_{n imes n} ext{(Identity matrix)}$$

[a]

We know that the rank of z is N, which means z has full column rank. Let  $y=z\tilde{w}$ , then for each  $y_i, 0 \leq i \leq N$ , there exists a  $\tilde{w}=z^{-1}y$ , therefore, we know that

$$egin{aligned} dots & ilde{w} = (x^Tx)^{-1}x^Ty \ & = egin{bmatrix} 1 & z_1^T \ 1 & z_2^T \ dots & dots \ 1 & z_N^T \end{bmatrix}^{-1} egin{bmatrix} y_1 \ y_2 \ dots \ y_N \end{bmatrix} \ dots & ilde{w}_n = y_n \end{aligned}$$

[b]

Since 
$$E_{in}=(g(x)-y)^2=( ilde w^T\Phi(x)-y)^2=rac{1}{N}\sum_{n=1}^N(y_n-y_n)^2=0$$
, therefore, we have  $E_{in}=0$ 

[c]

Since 
$$g(x_n) = ilde{w}^T \Phi(x_n) = ilde{w}^T z_n = y_n$$
, therefore we know that  $g(2x_n) = y_n$ 

[d]

From the definition of z, we know that for each  $[x=x_i], i=1,2,\ldots,N$  if  $x=x_i$ , then the value is 1, and for any n, if  $x\neq x_i$  the value is 0. Hence,  $g(x)=\tilde{w}^T\Phi(x\neq x_n)=0$ 

As a result, we should choose [c] as our solution.

## Playing with Regularization

**P7** 

$$\begin{split} E_{aug}(w) &= E_{in}(w) + \frac{\lambda}{3} \|w\|_{1} \\ &= \frac{1}{N} \sum_{n=1}^{N} (w_{0} + w_{1}x_{n} - y_{n})^{2} + \frac{\lambda}{3} (|w_{0}| + |w_{1}|) \\ \frac{\partial E_{aug}(w)}{\partial w_{0}} &= \frac{1}{N} \sum_{n=1}^{N} 2(w_{0} + w_{1}x_{n} - y_{n}) + \frac{\lambda}{3} (\frac{w_{0}}{|w_{0}|}) \\ &\Rightarrow \frac{2}{3} (3w_{0} + 3w_{1} - 3) + \frac{3}{3} (\frac{w_{0}}{|w_{0}|}) \\ &\Rightarrow 2w_{0} + 2w_{1} - 2 + (\frac{w_{0}}{|w_{0}|}) = 0 \Leftarrow (1) \\ \frac{\partial E_{aug}(w)}{\partial w_{1}} &= \frac{1}{N} \sum_{n=1}^{N} 2x_{n}(w_{0} + w_{1}x_{n} - y_{n}) + \frac{\lambda}{3} (\frac{w_{1}}{|w_{1}|}) \\ &\Rightarrow \frac{2}{3} (3w_{0} + 17w_{1} + 2) + \frac{3}{3} (\frac{w_{1}}{|w_{1}|}) \\ &\Rightarrow 2w_{0} + \frac{34}{3}w_{1} + \frac{4}{3} + (\frac{w_{1}}{|w_{1}|}) = 0 \Leftarrow (2) \\ (2) - (1) &\Rightarrow \frac{28}{3} w_{1} + \frac{10}{3} + (\frac{w_{1}}{|w_{1}|} - \frac{w_{0}}{|w_{0}|}) = 0 \end{split}$$

if 
$$\frac{w_1}{|w_1|} = 1$$
,  $\frac{w_0}{|w_0|} = 1 \Rightarrow w_1 = -\frac{5}{14}$ ,  $w_0 = \frac{6}{7}$   
if  $\frac{w_1}{|w_1|} = 1$ ,  $\frac{w_0}{|w_0|} = -1 \Rightarrow w_1 = -\frac{4}{7}$ ,  $w_0 = \frac{29}{14}$   
if  $\frac{w_1}{|w_1|} = -1$ ,  $\frac{w_0}{|w_0|} = 1 \Rightarrow w_1 = -\frac{1}{7}$ ,  $w_0 = \frac{9}{14}$   
if  $\frac{w_1}{|w_1|} = -1$ ,  $\frac{w_0}{|w_0|} = -1 \Rightarrow w_1 = -\frac{5}{14}$ ,  $w_0 = \frac{13}{7}$   
 $\therefore w_1 = -\frac{1}{7}$ ,  $w_0 = \frac{9}{14}$   

$$E_{aug}(w) = \frac{1}{N} \sum_{n=1}^{N} (\frac{9}{14} - \frac{1}{7}x_n - y_n)^2 + \frac{\lambda}{3} (\frac{11}{14})$$

$$= \frac{1}{3} ((-\frac{9}{14})^2 + (\frac{3}{14})^2 + (-\frac{15}{14})^2) + \frac{11}{14}$$

$$= \frac{1}{3} (\frac{45}{28}) + \frac{11}{14} = \frac{15}{28} + \frac{22}{28}$$

$$= \frac{37}{28} \approx 1.321$$

As a result, we should choose  $\left[e\right]$  as our solution.

**P8** 

$$egin{align} E_{aug}(w) &= E_{in}(w) + rac{\lambda}{N} \|w\|_2^2 \ &= rac{1}{N} \sum_{n=1}^N (w_0 + w_1 x_n - y_n)^2 + rac{\lambda}{N} (w_0^2 + w_1^2) \ rac{\partial E_{aug}(w)}{\partial w_0} &= rac{1}{N} \sum_{n=1}^N 2(w_0 + w_1 x_n - y_n) + rac{\lambda}{N} (2w_0) \ &\Rightarrow 2w_0 - 8 + \lambda w_0 = 0 \ w_0 &= rac{8}{2 + \lambda} \ rac{\partial E_{aug}(w)}{\partial w_0} &= rac{1}{N} \sum_{n=1}^N 2x_1(w_0 + w_1 x_n - y_n) + rac{\lambda}{N} (2w_1) \ &\Rightarrow 8w_1 - 20 + \lambda w_1 = 0 \ w_1 &= rac{20}{8 + \lambda} \ \end{cases}$$

Therefore, we can take test example back to the equation, then we have

$$egin{aligned} y &= w_{reg}^T \left[egin{array}{c} 1 \ x \end{array}
ight] = w_0 + w_1 x \ 4 &= rac{8}{2+\lambda} + rac{20}{8+\lambda} \ \lambda &= 2 \end{aligned}$$

As a result, we should choose [b] as our solution.

**P9** 

$$egin{aligned} E_{aug}(w) &= E_{in}(w)rac{\lambda}{N}w^Tw \ &= E_{in}(w)rac{\lambda}{N}(w_0^2+w_1^2+\cdots+w_d^2) \ rac{\partial E_{aug}(w)}{\partial w_i} &= rac{\partial 
abla E_{in}(w)}{\partial w_i} + rac{2\lambda}{N}w_i \ 
onumber &= 
abla E_{in}(w) + rac{2\lambda}{N} egin{bmatrix} w_0 \ w_1 \ dots \ w_d \ 
onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \ 

onumber &= 
abla E_{in}(w) + rac{2\lambda}{N}w \$$

As a result, we should choose [b] as our solution.

## Virtual Examples and Regularization

## P10

In order to find the minimum w, we can set the derivative of the optimal solution to be zero, hence, we have

$$egin{aligned} 0 &= rac{\partial}{\partial w} (rac{1}{N+K} (\sum_{n=1}^N (y_n - w^T x_n)^2 + \sum_{k=1}^K ( ilde{y}_k - w^T ilde{x}_k)^2)) \ &= rac{1}{N+K} (\sum_{n=1}^N -2 x_n (y_n - w^T x_n) + \sum_{k=1}^K -2 ilde{x}_k ( ilde{y}_k - w^T ilde{x}_k)) \ &= rac{2}{N+K} (\sum_{n=1}^N w^T x_n^2 - \sum_{n=1}^N x_n y_n + \sum_{k=1}^K w^T ilde{x}_k^2 - \sum_{k=1}^K ilde{x}_k ilde{y}_k) \ &\sum_{n=1}^N w^T x_n^2 + \sum_{k=1}^K w^T ilde{x}_k^2 = \sum_{n=1}^N x_n y_n + \sum_{k=1}^K ilde{x}_k ilde{y}_k \ &w = (x^T x + ilde{x}^T ilde{x})^{-1} (x^T y + ilde{x}^T ilde{y}) \end{aligned}$$

Then, we can find the minimum  $w_{reg}$  with the same method, which is setting the derivative of the optimal solution to be zero, and we get

$$0 = \frac{\partial}{\partial w} \left( \frac{1}{N} \|Xw - y\|^2 + \frac{\lambda}{N} \|w\|^2 \right)$$

$$= \frac{\partial}{\partial w} \left( \frac{1}{N} \sum_{n=1}^{N} (X_n w - y_n)^2 + \frac{\lambda}{N} w^2 \right)$$

$$= \frac{1}{N} \sum_{n=1}^{N} 2X_n (X_n w - y_n) + \frac{2\lambda I}{N} w$$

$$= \frac{2}{N} \sum_{n=1}^{N} w^T X_n^2 - \sum_{n=1}^{N} X_n y_n + \frac{2\lambda I}{N} w$$

$$= \frac{2}{N} \left( \sum_{n=1}^{N} w^T X_n^2 - \sum_{n=1}^{N} X_n y_n + \lambda I w \right)$$

$$\sum_{n=1}^{N} w^T X_n^2 + \lambda I w = \sum_{n=1}^{N} X_n y_n$$

$$w = (X^T X + \lambda I)^{-1} X^T y$$

Compare with the coefficients, we know that  $w=w_{reg}$ , therefore, we have

$$egin{aligned} w &= (x^Tx + ilde{x}^T ilde{x})^{-1}(x^Ty + ilde{x}^T ilde{y}) \ w_{reg} &= (X^TX + \lambda I)^{-1}X^Ty \ w &= w_{reg} \ (x^Tx + ilde{x}^T ilde{x})^{-1}(x^Ty + ilde{x}^T ilde{y}) &= (X^TX + \lambda I)^{-1}X^Ty \ ilde{X} &= \sqrt{\lambda}I_{d+1} \ ilde{y} &= 0 \end{aligned}$$

As a result, we should choose  $\left[b\right]$  as our solution.

$$\mathbb{E}(x_h^T x_h) = \mathbb{E}\left(\begin{bmatrix} \mid & \dots & \mid & \mid & \dots & \mid \\ x_1 & \dots & x_N & \bar{x}_1 & \dots & \bar{x}_N \\ \mid & \dots & \mid & \mid & \dots & \mid \end{bmatrix}, \begin{bmatrix} - & x_1 & - \\ \vdots & \vdots & \vdots & \vdots \\ - & x_N & - \\ - & \bar{x}_1 & - \\ \vdots & \vdots & \vdots \\ - & \bar{x}_N & - \end{bmatrix}\right)$$

$$= \mathbb{E}\left(\begin{bmatrix} \mid & \dots & \mid & \mid & \dots & \mid \\ x_1 & \dots & x_N & x_1 + \epsilon & \dots & x_N + \epsilon \\ \mid & \dots & \mid & \mid & \dots & \mid \end{bmatrix}, \begin{bmatrix} - & x_1 & - \\ \vdots & \vdots & \vdots \\ - & x_N & - \\ - & x_1 + \epsilon & - \\ \vdots & \vdots & \vdots \\ - & x_N + \epsilon & - \end{bmatrix}\right)$$

$$= \mathbb{E}\left(\begin{bmatrix} \mid & \dots & \mid & \mid & \dots & \mid \\ x_1 & \dots & x_N & x_1 + \epsilon & \dots & x_N + \epsilon \\ \mid & \dots & \mid & \mid & \dots & \mid \end{bmatrix}, \begin{bmatrix} x_{11} \\ \vdots \\ x_{N1} \\ \vdots \\ x_{Nd} + \epsilon \end{bmatrix}\right)$$

$$+ \begin{bmatrix} \mid & \dots & \mid & \mid & \dots & \mid \\ x_1 & \dots & x_N & x_1 + \epsilon & \dots & x_N + \epsilon \\ \mid & \dots & \mid & \mid & \dots & \mid \end{bmatrix}, \begin{bmatrix} x_{11} \\ \vdots \\ x_{Nd} \\ x_{1d} + \epsilon \\ \vdots \\ x_{Nd} + \epsilon \end{bmatrix}\right)$$

$$= \mathbb{E}(x^T x + x^T x) + \mathbb{P}\left(\begin{bmatrix} \epsilon^2 & 0 & \dots & 0 & 0 \\ 0 & \epsilon^2 & \dots & 0 & 0 \\ \vdots & 0 & \ddots & 0 & \vdots \\ 0 & 0 & \dots & \epsilon^2 & 0 \\ 0 & 0 & \dots & 0 & \epsilon^2 \end{bmatrix}\right)$$

$$= \mathbb{E}(x^T x + x^T x) + N \cdot \mathbb{P}(\epsilon^2) I_{d-1}$$

$$= \mathbb{E}(x^T x + x^T x) + N \cdot \frac{1}{2r} \int_{-r}^{r} \epsilon^2 I_{d+1}$$

$$= x^T x + x^T x + N \cdot \frac{1}{2r} \frac{2r^3}{3} I_{d+1}$$

$$= 2x^T x + \frac{N}{3} r^2 I_{d+1}$$

As a result, we should choose [c] as our solution.

### P12

In order to find the minimum  $y \in \mathbb{R}$ , we can set the derivative of the optimal solution to be zero, which means

$$egin{aligned} 0 &= rac{\partial}{\partial y} (rac{1}{N} \sum_{n=1}^N (y-y_n)^2 + rac{\lambda}{N} \Omega(y)) \ &= rac{1}{N} \sum_{n=1}^N 2 (y-y_n) + rac{\lambda}{N} \Omega'(y) \ &= rac{2}{N} (\sum_{n=1}^N y - \sum_{n=1}^N y_n) + rac{\lambda}{N} \Omega'(y) \ &= rac{2}{N} (Ny - \sum_{n=1}^N y_n) + rac{\lambda}{N} \Omega'(y) \ &= 2y - rac{2}{N} \sum_{n=1}^N y_n + rac{\lambda}{N} \Omega'(y) \ &rac{\lambda}{N} \Omega'(y) = rac{2}{N} \sum_{n=1}^N y_n - 2y \end{aligned}$$

$$\begin{split} & \because \text{optimal } y^* = \frac{(\sum_{n=1}^N y_n) + K}{N + 2K} \\ & \therefore \frac{\lambda}{N} \Omega'(y) = \frac{2}{N} \sum_{n=1}^N y_n - 2(\frac{(\sum_{n=1}^N y_n) + K}{N + 2K}) \\ & = \frac{1}{N + 2K} (\frac{2 \cdot (N + 2K) \cdot (\sum_{n=1}^N y_n)}{N} - 2 \cdot (\sum_{n=1}^N y_n) - 2K) \\ & = \frac{1}{N + 2K} (2 \cdot (\sum_{n=1}^N y_n) + 4 \cdot \frac{K}{N} (\sum_{n=1}^N y_n) - 2 \cdot (\sum_{n=1}^N y_n) - 2K)) \\ & = \frac{1}{N + 2K} (\frac{4K}{N} (\sum_{n=1}^N y_n) - 2K) \\ & = \frac{2K}{N + 2K} (\frac{2}{N} (\sum_{n=1}^N y_n) - 1) \\ & \Omega'(y) = \frac{N}{\lambda} \cdot \frac{2K}{N + 2K} \cdot (\frac{2 \cdot (\sum_{n=1}^N y_n) - N}{N}) \\ & = \frac{2K}{\lambda} \cdot \frac{1}{N + 2K} \cdot (2 \cdot (\sum_{n=1}^N y_n) - N) \\ & \because \text{optimal } y^* = \frac{(\sum_{n=1}^N y_n) + K}{N + 2K} \\ & \therefore \Omega'(y) = \frac{2K}{\lambda} \cdot (2y - 1) \\ & = \frac{2K}{\lambda} \cdot 2 \cdot (y - \frac{1}{2}) \\ & = \frac{2K}{\lambda} \cdot 2 \cdot (y - 0.5) \\ & \Omega(y) = \frac{2K}{\lambda} (y - 0.5)^2 \end{split}$$

As a result, we should choose  $\left[b\right]$  as our solution.

## **Experiments with Linear and Nonlinear Models**

P13

```
1
     import numpy as np
 2
 3
     def calculate_E_in_sqr(x, y, w):
 4
         return np.mean([(np.dot(w, x[i]) - y[i])**2 for i in range(len(x))])
 5
     def calculate w lin(x, y):
 6
 7
         # pseudo inverse: np.linalg.pinv(np.array([values]))
 8
         return np.dot(np.linalg.pinv(x), y)
9
10
     def read_file(filename):
         data = np.loadtxt(filename, dtype=float)
11
12
         x = data[:,:-1]
         y = data[:,-1]
13
14
         return x, y
15
     def main(args):
16
17
         x_train, y_train = read_file(args['filename_train'])
         x_train = np.c_[np.ones(len(x_train)), x_train] # x_{n, 0}=1
18
19
         weight = calculate_w_lin(x_train, y_train)
20
21
         E_in = calculate_E_in_sqr(x_train, y_train, weight)
         print("E_in: ", E_in)
22
23
24
     if __name__ == '__main__':
25
         args = {
26
             'filename_test': "hw3_test.dat",
              'filename_train': "hw3_train.dat"
27
28
         }
29
30
         main(args)
```

1 E\_in: 0.792234776110557

As a result, we should choose  $\left[c\right]$  as our solution.

```
1
     import numpy as np
 2
     import random
 3
     from tqdm import trange
 4
 5
     def calculate_E_in_sqr(x, y, w):
 6
         return np.mean([(np.dot(w, x[i]) - y[i])**2 for i in range(len(x))])
 7
 8
     def read_file(args, filename):
9
         data = np.loadtxt(filename, dtype=float)
10
         x = data[:,:-1]
         y = data[:,-1]
11
12
         args['features'] = len(x[0])
13
         args['size'] = len(x)
14
         return x, y
15
     def SGD(args, x, y):
16
17
         error_list = list()
18
         w_list = list()
19
         for i in trange(args['repeat_time']):
20
             random.seed()
             w = np.zeros(args['feature'])
21
22
             for j in range(args['iteration']):
23
                 pick = random.randint(0, args['size']-1)
                 w = w + args['learning_rate'] * 2 * (y[pick] - np.dot(w, x[pick])) * x[
24
             error_list.append(calculate_E_in_sqr(x, y, w))
25
26
             w_list.append(w)
         return w_list, error_list
27
```

4

```
1
     def main(args):
 2
         x_train, y_train = read_file(args, args['filename_train'])
 3
         x_train = np.c_[np.ones(len(x_train)), x_train] # x_{n, 0}=1
         args['feature'] = len(x_train[0])
 4
 5
         weight_list, E_in_list = SGD(args, x_train, y_train)
 6
 7
         E_in_average = np.mean(E_in_list)
         print("E_in_average: ", E_in_average)
 8
9
     if __name__ == '__main__':
10
11
         args = {
             'filename_test': "hw3_test.dat",
12
13
             'filename_train': "hw3_train.dat",
             'feature': 11, # without include y
14
15
             'iteration': 800,
16
             'learning rate': 0.001,
17
             'repeat_time': 1000,
             'size': 0
18
19
         }
20
         main(args)
21
```

As a result, we should choose [d] as our solution.

E\_in\_average: 0.8225428632462761

### P15

We use the same code above, then update the SGD steps and the error function. The updated functions show below.

```
1
     def calculate_E_in_ce(x, y, w):
 2
         return np.mean(np.log(1 + np.exp(-y * np.dot(x, w))))
 3
 4
     def SGD(args, x, y):
         error_list = list()
 5
         w list = list()
 6
 7
         for i in trange(args['repeat_time']):
 8
             random.seed()
 9
             w = np.zeros(args['feature'])
             for j in range(args['iteration']):
10
                 pick = random.randint(0, args['size']-1)
11
                 \# w = w + args['learning_rate'] * 2 * (y[pick] - np.dot(w, x[pick])) *
12
13
                 w = w + args['learning_rate'] * sigmoid(-y[pick] * np.dot(w, x[pick]))
             error_list.append(calculate_E_in_ce(x, y, w))
14
15
             w list.append(w)
16
         return w_list, error_list
```

1 E\_in\_average: 0.6571562557258926

As a result, we should choose [c] as our solution.

### P16

We use the same code above, then update the initialization of w from w=0 to  $w=w_{lin}$ . The updated functions show below.

```
1
     def calculate_w_lin(x, y):
 2
         # pseudo inverse: np.linalg.pinv(np.array([values]))
 3
         return np.dot(np.linalg.pinv(x), y)
 4
 5
     def SGD(args, x, y):
         error list = list()
 6
 7
         w_list = list()
 8
         for i in trange(args['repeat_time']):
 9
             random.seed()
             w = args['w_lin']
10
             for j in range(args['iteration']):
11
                  pick = random.randint(0, args['size']-1)
12
13
                  \# w = w + args['learning_rate'] * 2 * (y[pick] - np.dot(w, x[pick])) *
                  w = w + args['learning_rate'] * sigmoid(-y[pick] * np.dot(w, x[pick]))
14
             error list.append(calculate E in ce(x, y, w))
15
             w list.append(w)
16
17
         return w_list, error_list
18
19
     def main(args):
20
         x_train, y_train = read_file(args, args['filename_train'])
         x_train = np.c_[np.ones(len(x_train)), x_train] # x_{n, 0}=1
21
22
         args['feature'] = len(x_train[0])
23
         args['w_lin'] = calculate_w_lin(x_train, y_train)
24
25
         weight_list, E_in_list = SGD(args, x_train, y_train)
26
         E_in_average = np.mean(E_in_list)
         print("E_in_average: ", E_in_average)
27
28
29
     if __name__ == '__main__':
30
         args = {
              'filename_test': "hw3_test.dat",
31
              'filename_train': "hw3_train.dat",
32
33
              'feature': 11, # without include y
34
              'iteration': 800,
              'learning rate': 0.001,
35
36
              'repeat_time': 1000,
              'size': 0,
37
38
              'w lin': 0, # weight vector is a list
39
         }
40
         main(args)
41
```

1 E\_in\_average: 0.6051732441261142

As a result, we should choose [a] as our solution.

```
1
     import numpy as np
 2
     import random
 3
     from tqdm import trange
 4
 5
     def calculate_E_in_01(x, y, w):
         return np.mean(np.dot(x, w) * y <= 0)
 6
 7
     def calculate w lin(x, y):
 8
 9
         # pseudo inverse: np.linalg.pinv(np.array([values]))
10
         return np.dot(np.linalg.pinv(x), y)
11
12
     def read_file(args, filename):
13
         data = np.loadtxt(filename, dtype=float)
14
         x = data[:,:-1]
         y = data[:,-1]
15
         args['features'] = len(x[0])
16
17
         args['size'] = len(x)
18
         return x, y
19
20
     def SGD(args, x, y):
         error_list = list()
21
22
         w_list = list()
23
         for i in trange(args['repeat_time']):
             random.seed()
24
25
             w = args['w_lin']
             for j in range(args['iteration']):
26
27
                 pick = random.randint(0, args['size']-1)
                 \# w = w + args['learning_rate'] * 2 * (y[pick] - np.dot(w, x[pick])) *
28
                 w = w + args['learning_rate'] * sigmoid(-y[pick] * np.dot(w, x[pick]))
29
30
             error_list.append(calculate_E_in_01(x, y, w))
             w_list.append(w)
31
32
         return w_list, error_list
33
34
     def sigmoid(s):
         return (1 / (1 + np.exp(-s)))
35
```

**←** 

```
1
     def main(args):
 2
         x_train, y_train = read_file(args, args['filename_train'])
         x_train = np.c_[np.ones(len(x_train)), x_train] # x_{n, 0}=1
 3
 4
         args['feature'] = len(x_train[0])
         args['w_lin'] = calculate_w_lin(x_train, y_train)
 5
         weight_list, E_in_list = SGD(args, x_train, y_train)
 6
 7
         x_test, y_test = read_file(args, args['filename_test'])
 8
 9
         x_{test} = np.c_{np.ones}(len(x_{test})), x_{test} # x_{n}, 0}=1
         args['feature'] = len(x_train[0])
10
         E_out_list = np.array([calculate_E_in_01(x_test, y_test, weight_list[i]) for i
11
         print("|E_in-E_out|: ", np.mean(np.abs(E_in_list - E_out_list)))
12
13
     if __name__ == '__main__':
14
15
         args = {
16
              'filename_test': "hw3_test.dat",
17
              'filename_train': "hw3_train.dat",
18
              'feature': 11, # without include y
              'iteration': 800,
19
20
              'learning_rate': 0.001,
21
              'repeat_time': 1000,
              'size': 0,
22
23
              'w_lin': 0, # weight_vector is a list
24
         }
25
         main(args)
26
```

1 | |E\_in-E\_out|: 0.030122500000000017

As a result, we should choose [a] as our solution.

```
import numpy as np
 1
 2
     from tqdm import trange
 3
 4
     def calculate E in 01(x, y, w):
 5
         return np.mean(np.dot(x, w) * y <= 0)
 6
 7
     def calculate_w_lin(x, y):
 8
         # pseudo inverse: np.linalg.pinv(np.array([values]))
9
         return np.dot(np.linalg.pinv(x), y)
10
     def read file(args, filename):
11
         data = np.loadtxt(filename, dtype=float)
12
13
         x = data[:,:-1]
         y = data[:,-1]
14
15
         return x, y
16
17
     def main(args):
18
         x_train, y_train = read_file(args, args['filename_train'])
19
         x_train = np.c_[np.ones(len(x_train)), x_train] # x_{n, 0}=1
20
         x_test, y_test = read_file(args, args['filename_test'])
21
         x_{test} = np.c_{np.ones(len(x_{test})), x_{test}} # x_{n, 0}=1
22
23
         weight = calculate_w_lin(x_train, y_train)
24
         E in = calculate E in 01(x train, y train, weight)
25
         E_out = calculate_E_in_01(x_test, y_test, weight)
26
27
         print("E_in: ", E_in)
28
         print("E_out: ", E_out)
29
         print("|E_in-E_out|: ", np.abs(E_in - E_out))
30
     if __name__ == '__main__':
31
32
         args = {
33
              'filename_test': "hw3_test.dat",
34
             'filename_train': "hw3_train.dat",
35
         }
36
37
         main(args)
 1
     E in: 0.3
 2
     E_out: 0.34
     |E in-E out|: 0.04000000000000036
```

As a result, we should choose  $\left[b\right]$  as our solution.

We use the same code above, then update the transformation function and apply it to the training and testing dataset. The updated functions show below.

```
1
     def transformation(x, Q):
 2
         # x 0 is included
 3
         x_{origin} = x[:, 1:]
         transformed x = x.copy()
 4
 5
         for q in range(2, Q+1):
 6
             transformed_x = np.hstack((transformed_x, x_origin**q))
 7
         return transformed x
 8
 9
     def main(args):
         x_train, y_train = read_file(args, args['filename_train'])
10
         x_{train} = np.c_{np.ones(len(x_{train})), x_{train}} # x_{n, 0}=1
11
         x_test, y_test = read_file(args, args['filename_test'])
12
13
         x_{test} = np.c_{np.ones(len(x_{test})), x_{test}} # x_{n, 0}=1
14
         x_train = transformation(x_train, args['Q'])
15
         x_test = transformation(x_test, args['Q'])
16
17
         weight = calculate_w_lin(x_train, y_train)
18
         E_in = calculate_E_in_01(x_train, y_train, weight)
19
         E_out = calculate_E_in_01(x_test, y_test, weight)
20
         print("E_in: ", E_in)
21
         print("E out: ", E out)
22
         print("|E_in-E_out|: ", np.abs(E_in - E_out))
23
24
25
     if __name__ == '__main__':
26
         args = {
27
             'filename_test': "hw3_test.dat",
28
             'filename_train': "hw3_train.dat",
29
             'Q': 2,
30
         }
31
32
         main(args)
 1
     E in: 0.23
 2
     E out: 0.3125
```

As a result, we should choose [c] as our solution.

## P20

We use the same code above, then update the parameter of args. The updated functions show below.

```
1
    if __name__ == '__main__':
2
        args = {
            'filename_test': "hw3_test.dat",
3
            'filename_train': "hw3_train.dat",
4
5
            'Q': 8,
6
        }
7
8
        main(args)
1
    E_in: 0.0
2
    E_out: 0.415
3 |E_in-E_out|: 0.415
```

As a result, we should choose  $\left[d\right]$  as our solution.