HTML_2023_HW3

tags: `Personal`

```
Discuss with anonymous and TAs.
```

Linear Models and More

P1

- CPU time: aN for training a binary classifier on a size N binary classification data set
- We have size-N K-class classification data set, which means each class is of size $\frac{N}{K}$
- 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

$$a\binom{K}{2} \times 2 \times \frac{N}{K} = a\frac{K!}{2!(K-2)!}\frac{2N}{K} = a(K-1)N$$

As a result, we should choose $[b]$ 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

$$\begin{aligned} x_1 &= (2,0), \quad & x_2 &= (0,2), & x_3 &= (-2,0) \\ x_4 &= (0,-2), & x_5 &= (0,0), & x_6 &= (1,1) \end{aligned}$$

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

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

Since z is invertible, let $y = zw$, then for each $y_i, 0 \leq i \leq 6$, 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 $[e]$ as our solution.

P3

By hand

$$
\begin{aligned}
x_1 &= (0,0), & y_1 &= -1\\
x_2 &= (4,0), & y_2 &= +1\\
x_3 &= (-4,0), & y_3 &= +1\\
x_4 &= (0,2), & y_4 &= -1\\
x_5 &= (0,-2), & y_5 &= -1
\end{aligned}
$$

$$
\begin{aligned}
w_1 &= (-1,0,0,0.5,0,-0.5)\\
w_2 &= (-1,0,0,-0.5,0,0.5)\\
w_3 &= (-2,0,0,1,0,1)\\
w_4 &= (-1,0,0,0.2,0,0.1)
\end{aligned}
$$

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

	$\text{sign}(w_i^T\phi(x_i))$	y_i	$\text{sign}(w_i^T\phi(x_i))) = y_i$
w_1	[-1,1,1,-1,-1]	[-1,1,1,-1,-1]	[O,O,O,O,O]
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]	[O,O,O,O,O]

By python

```python
import numpy as np

def phi(x):
    return [1, x[0], x[1], x[0]*x[0], x[0]*x[1], x[1]*x[1]]

def main(dataset, weight_vector):
    result = 0
    for i in range(len(weight_vector)):
        print("weight_vector: ", weight_vector[i])
        counter = 0
        for j in range(len(dataset)):
            hypothesis = int(np.sign(np.dot(weight_vector[i], phi(dataset[j]))))
            if hypothesis == dataset[j][2]:
                print("O:", "(", hypothesis, ",", dataset[j][2], ")")
                counter += 1
            else:
                print("X:", "(", hypothesis, ",", dataset[j][2], ")")
        if counter == len(dataset):
            print("===separates the dataset correct===")
            result += 1
        else:
            print("===separates the dataset incorrect===")
    print("There are", result, "weight vectors that can seperate dataset.")
```

```python
if __name__ == '__main__':
    dataset = np.array([
        # x_1,x_2, y
        [0, 0, -1],
        [4, 0, 1],
        [-4, 0, 1],
        [0, 2, -1],
        [0, -2, -1],
    ])

    weight_vector = np.array([
        [-1, 0, 0, 0.5, 0, -0.5],
        [-1, 0, 0, -0.5, 0, 0.5],
        [-2, 0, 0, 1, 0, 1],
        [-1, 0, 0, 0.2, 0, 0.1],
    ])

    main(dataset, weight_vector)

```

```
weight_vector:  [-1.   0.   0.   0.5  0.  -0.5]
O: ( -1 , -1 )
O: ( 1 , 1 )
O: ( 1 , 1 )
O: ( -1 , -1 )
O: ( -1 , -1 )
===separates the dataset correct===
weight_vector:  [-1.   0.   0.  -0.5  0.   0.5]
O: ( -1 , -1 )
X: ( -1 , 1 )
X: ( -1 , 1 )
X: ( 1 , -1 )
X: ( 1 , -1 )
===separates the dataset incorrect===
weight_vector:  [-2.  0.  0.  1.  0.  1.]
O: ( -1 , -1 )
O: ( 1 , 1 )
O: ( 1 , 1 )
X: ( 1 , -1 )
X: ( 1 , -1 )
===separates the dataset incorrect===
weight_vector:  [-1.   0.   0.   0.2  0.   0.1]
O: ( -1 , -1 )
O: ( 1 , 1 )
O: ( 1 , 1 )
O: ( -1 , -1 )
O: ( -1 , -1 )
===separates the dataset correct===
There are 2 weight vectors that can seperate dataset.
```

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

P4

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

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

P5

Define $\Phi_{(k)}(x) = (1, x_k) = (w_0, w_1)$, then we have $w_0 + w_1 x_k = 0 \Leftrightarrow x_k = -\frac{w0}{w1}$(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

$$
\begin{aligned}
&2^N \leq 2(N-1)d + 2 = 2Nd \\
&\Rightarrow \frac{2N}{2} \leq Nd \Rightarrow 2^{N-1} \leq Nd \\
&\Rightarrow N-1 \leq \log_2 N + \log_2 d \leq \frac{N}{2} + \log_2 d \because \log_2 d \leq \frac{d}{2} (\text{by hint}) \\
&\Rightarrow 2N - 2 \leq N + 2\log_2 d \\
&\Rightarrow N \leq 2\log_2 d + 2 = 2(\log_2 d + 1) \\
&\therefore\ d_{vc}(\bigcup_{k=1}^{d} \mathcal{H}_k) \leq 2(\log_2 d + 1)
\end{aligned}
$$

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

P6

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

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

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

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

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

$$
z = \begin{bmatrix} 1 & z_1^T \\ 1 & z_2^T \\ \vdots & \vdots \\ 1 & z_N^T \end{bmatrix} = I_{n\times n} (\text{Identity matrix})
$$

[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

$$\because \tilde{w} = (x^T x)^{-1} x^T y$$

$$= \begin{bmatrix} 1 & z_1^T \\ 1 & z_2^T \\ \vdots & \vdots \\ 1 & z_N^T \end{bmatrix}^{-1} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

$$\therefore \tilde{w}_n = y_n$$

[b]

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

[c]

Since $g(x_n) = \tilde{w}^T \Phi(x_n) = \tilde{w}^T z_n = y_n$, 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

$$
\begin{aligned}
E_{aug}(w) &= E_{in}(w) + \frac{\lambda}{3}\|w\|_1 \\
&= \frac{1}{N}\sum_{n=1}^{N}(w_0 + w_1 x_n - y_n)^2 + \frac{\lambda}{3}(|w_0| + |w_1|) \\
\frac{\partial E_{aug}(w)}{\partial w_0} &= \frac{1}{N}\sum_{n=1}^{N} 2(w_0 + w_1 x_n - y_n) + \frac{\lambda}{3}(\frac{w_0}{|w_0|}) \\
&\Rightarrow \frac{2}{3}(3w_0 + 3w_1 - 3) + \frac{3}{3}(\frac{w_0}{|w_0|}) \\
&\Rightarrow 2w_0 + 2w_1 - 2 + (\frac{w_0}{|w_0|}) = 0 \Leftarrow (1) \\
\frac{\partial E_{aug}(w)}{\partial w_1} &= \frac{1}{N}\sum_{n=1}^{N} 2x_n(w_0 + w_1 x_n - y_n) + \frac{\lambda}{3}(\frac{w_1}{|w_1|}) \\
&\Rightarrow \frac{2}{3}(3w_0 + 17w_1 + 2) + \frac{3}{3}(\frac{w_1}{|w_1|}) \\
&\Rightarrow 2w_0 + \frac{34}{3}w_1 + \frac{4}{3} + (\frac{w_1}{|w_1|}) = 0 \Leftarrow (2) \\
\text{(2) - (1)} &\Rightarrow \frac{28}{3}w_1 + \frac{10}{3} + (\frac{w_1}{|w_1|} - \frac{w_0}{|w_0|}) = 0
\end{aligned}
$$

$$
\begin{aligned}
\text{if } \frac{w_1}{|w_1|} &= \ \ 1, \frac{w_0}{|w_0|} = \ \ 1 \Rightarrow w_1 = -\frac{5}{14}, w_0 = \frac{6}{7} \\
\text{if } \frac{w_1}{|w_1|} &= \ \ 1, \frac{w_0}{|w_0|} = -1 \Rightarrow w_1 = -\frac{4}{7}, w_0 = \frac{29}{14} \\
\text{if } \frac{w_1}{|w_1|} &= -1, \frac{w_0}{|w_0|} = \ \ 1 \Rightarrow w_1 = -\frac{1}{7}, w_0 = \frac{9}{14} \\
\text{if } \frac{w_1}{|w_1|} &= -1, \frac{w_0}{|w_0|} = -1 \Rightarrow w_1 = -\frac{5}{14}, w_0 = \frac{13}{7}
\end{aligned}
$$

$$
\begin{aligned}
\therefore w_1 &= -\frac{1}{7}, w_0 = \frac{9}{14} \\
E_{aug}(w) &= \frac{1}{N}\sum_{n=1}^{N}(\frac{9}{14} - \frac{1}{7}x_n - y_n)^2 + \frac{\lambda}{3}(\frac{11}{14}) \\
&= \frac{1}{3}((-\frac{9}{14})^2 + (\frac{3}{14})^2 + (-\frac{15}{14})^2) + \frac{11}{14} \\
&= \frac{1}{3}(\frac{45}{28}) + \frac{11}{14} = \frac{15}{28} + \frac{22}{28} \\
&= \frac{37}{28} \approx 1.321
\end{aligned}
$$

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

P8

$$
\begin{aligned}
E_{aug}(w) &= E_{in}(w) + \frac{\lambda}{N}\|w\|_2^2 \\
&= \frac{1}{N}\sum_{n=1}^{N}(w_0 + w_1x_n - y_n)^2 + \frac{\lambda}{N}(w_0^2 + w_1^2) \\
\frac{\partial E_{aug}(w)}{\partial w_0} &= \frac{1}{N}\sum_{n=1}^{N}2(w_0 + w_1x_n - y_n) + \frac{\lambda}{N}(2w_0) \\
&\Rightarrow 2w_0 - 8 + \lambda w_0 = 0 \\
w_0 &= \frac{8}{2+\lambda} \\
\frac{\partial E_{aug}(w)}{\partial w_0} &= \frac{1}{N}\sum_{n=1}^{N}2x_1(w_0 + w_1x_n - y_n) + \frac{\lambda}{N}(2w_1) \\
&\Rightarrow 8w_1 - 20 + \lambda w_1 = 0 \\
w_1 &= \frac{20}{8+\lambda}
\end{aligned}
$$

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

$$
\begin{aligned}
&y = w_{reg}^T \begin{bmatrix} 1 \\ x \end{bmatrix} = w_0 + w_1 x \\
&4 = \frac{8}{2+\lambda} + \frac{20}{8+\lambda} \\
&\lambda = 2
\end{aligned}
$$

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

P9

$$
\begin{aligned}
E_{aug}(w) &= E_{in}(w)\frac{\lambda}{N}w^T w \\
&= E_{in}(w)\frac{\lambda}{N}(w_0^2 + w_1^2 + \cdots + w_d^2) \\
\frac{\partial E_{aug}(w)}{\partial w_i} &= \frac{\partial \nabla E_{in}(w)}{\partial w_i} + \frac{2\lambda}{N} w_i \\
\nabla E_{aug}(w) &= \nabla E_{in}(w) + \frac{2\lambda}{N} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} \\
&= \nabla E_{in}(w) + \frac{2\lambda}{N} w \\
\therefore w_{t+1} &\leftarrow w_t - \eta \nabla E_{aug}(w_t) \\
w_{t+1} &\leftarrow w_t - \eta(\nabla E_{in}(w_t) + \frac{2\lambda}{N} w_t) \\
w_{t+1} &\leftarrow (1 - \frac{2\eta\lambda}{N}) w_t - \eta \nabla E_{in}(w_t) \\
a &= 1 - \frac{2\eta\lambda}{N}
\end{aligned}
$$

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

$$
\begin{aligned}
0 &= \frac{\partial}{\partial w}\left(\frac{1}{N+K}\left(\sum_{n=1}^{N}(y_n - w^T x_n)^2 + \sum_{k=1}^{K}(\tilde{y}_k - w^T\tilde{x}_k)^2\right)\right) \\
&= \frac{1}{N+K}\left(\sum_{n=1}^{N} -2x_n(y_n - w^T x_n) + \sum_{k=1}^{K} -2\tilde{x}_k(\tilde{y}_k - w^T\tilde{x}_k)\right) \\
&= \frac{2}{N+K}\left(\sum_{n=1}^{N} w^T x_n^2 - \sum_{n=1}^{N} x_n y_n + \sum_{k=1}^{K} w^T\tilde{x}_k^2 - \sum_{k=1}^{K}\tilde{x}_k\tilde{y}_k\right) \\
\sum_{n=1}^{N} w^T x_n^2 + \sum_{k=1}^{K} w^T\tilde{x}_k^2 &= \sum_{n=1}^{N} x_n y_n + \sum_{k=1}^{K}\tilde{x}_k\tilde{y}_k \\
w &= (x^T x + \tilde{x}^T\tilde{x})^{-1}(x^T y + \tilde{x}^T\tilde{y})
\end{aligned}
$$

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

$$
\begin{aligned}
0 &= \frac{\partial}{\partial w}\left(\frac{1}{N}\|Xw - y\|^2 + \frac{\lambda}{N}\|w\|^2\right) \\
&= \frac{\partial}{\partial w}\left(\frac{1}{N}\sum_{n=1}^{N}(X_n w - y_n)^2 + \frac{\lambda}{N}w^2\right) \\
&= \frac{1}{N}\sum_{n=1}^{N} 2X_n(X_n w - y_n) + \frac{2\lambda I}{N}w \\
&= \frac{2}{N}\sum_{n=1}^{N} w^T X_n^2 - \sum_{n=1}^{N} X_n y_n + \frac{2\lambda I}{N}w \\
&= \frac{2}{N}\left(\sum_{n=1}^{N} w^T X_n^2 - \sum_{n=1}^{N} X_n y_n + \lambda I w\right) \\
\sum_{n=1}^{N} w^T X_n^2 + \lambda I w &= \sum_{n=1}^{N} X_n y_n \\
w &= (X^T X + \lambda I)^{-1} X^T y
\end{aligned}
$$

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

$$
\begin{aligned}
w &= (x^T x + \tilde{x}^T\tilde{x})^{-1}(x^T y + \tilde{x}^T\tilde{y}) \\
w_{reg} &= (X^T X + \lambda I)^{-1} X^T y \\
w &= w_{reg} \\
(x^T x + \tilde{x}^T\tilde{x})^{-1}(x^T y + \tilde{x}^T\tilde{y}) &= (X^T X + \lambda I)^{-1} X^T y \\
\tilde{X} &= \sqrt{\lambda} I_{d+1} \\
\tilde{y} &= 0
\end{aligned}
$$

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

P11

$$
\begin{aligned}
\mathbb{E}(x_h^T x_h) &= \mathbb{E}\left(\begin{bmatrix} | & \dots & | & | & \dots & | \\ x_1 & \dots & x_N & \tilde{x}_1 & \dots & \tilde{x}_N \\ | & \dots & | & | & \dots & | \end{bmatrix} \cdot \begin{bmatrix} - & x_1 & - \\ \vdots & \vdots & \vdots \\ - & x_N & - \\ - & \tilde{x}_1 & - \\ \vdots & \vdots & \vdots \\ - & \tilde{x}_N & - \end{bmatrix} \right) \\
&= \mathbb{E}\left(\begin{bmatrix} | & \dots & | & | & \dots & | \\ x_1 & \dots & x_N & x_1+\epsilon & \dots & x_N+\epsilon \\ | & \dots & | & | & \dots & | \end{bmatrix} \cdot \begin{bmatrix} - & x_1 & - \\ \vdots & \vdots & \vdots \\ - & x_N & - \\ - & x_1+\epsilon & - \\ \vdots & \vdots & \vdots \\ - & x_N+\epsilon & - \end{bmatrix} \right) \\
&= \mathbb{E}\left(\begin{bmatrix} | & \dots & | & | & \dots & | \\ x_1 & \dots & x_N & x_1+\epsilon & \dots & x_N+\epsilon \\ | & \dots & | & | & \dots & | \end{bmatrix} \cdot \begin{bmatrix} x_{11} \\ \vdots \\ x_{N1} \\ x_{11}+\epsilon \\ \vdots \\ x_{N1}+\epsilon \end{bmatrix} + \dots \right. \\
&\qquad \left. + \begin{bmatrix} | & \dots & | & | & \dots & | \\ x_1 & \dots & x_N & x_1+\epsilon & \dots & x_N+\epsilon \\ | & \dots & | & | & \dots & | \end{bmatrix} \cdot \begin{bmatrix} x_{1d} \\ \vdots \\ x_{Nd} \\ x_{1d}+\epsilon \\ \vdots \\ x_{Nd}+\epsilon \end{bmatrix} \right) \\
&= \mathbb{E}(x^T x + x^T x) + \mathbb{P}\left(\begin{bmatrix} \epsilon^2 & 0 & \dots & 0 & 0 \\ 0 & \epsilon^2 & \dots & 0 & 0 \\ \vdots & 0 & \ddots & 0 & \vdots \\ 0 & 0 & \dots & \epsilon^2 & 0 \\ 0 & 0 & \dots & 0 & \epsilon^2 \end{bmatrix} \right) \\
&= \mathbb{E}(x^T x + x^T x) + N \cdot \mathbb{P}(\epsilon^2) I_{d+1} \\
&= \mathbb{E}(x^T x + x^T x) + N \cdot \frac{1}{2r} \int_{-r}^{r} \epsilon^2 I_{d+1} \\
&= x^T x + x^T x + N \cdot \frac{1}{2r} \frac{2r^3}{3} I_{d+1} \\
&= 2x^T x + \frac{N}{3} r^2 I_{d+1}
\end{aligned}
$$

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

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

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

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

Experiments with Linear and Nonlinear Models

P13

```
import numpy as np

def calculate_E_in_sqr(x, y, w):
    return np.mean([(np.dot(w, x[i]) - y[i])**2 for i in range(len(x))])

def calculate_w_lin(x, y):
    # pseudo inverse: np.linalg.pinv(np.array([values]))
    return np.dot(np.linalg.pinv(x), y)

def read_file(filename):
    data = np.loadtxt(filename, dtype=float)
    x = data[:,:-1]
    y = data[:,-1]
    return x, y

def main(args):
    x_train, y_train = read_file(args['filename_train'])
    x_train = np.c_[np.ones(len(x_train)), x_train] # x_{n, 0}=1

    weight = calculate_w_lin(x_train, y_train)
    E_in = calculate_E_in_sqr(x_train, y_train, weight)
    print("E_in: ", E_in)

if __name__ == '__main__':
    args = {
        'filename_test': "hw3_test.dat",
        'filename_train': "hw3_train.dat"
    }

    main(args)
```

```
E_in:  0.792234776110557
```

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

P14

```
import numpy as np
import random
from tqdm import trange

def calculate_E_in_sqr(x, y, w):
    return np.mean([(np.dot(w, x[i]) - y[i])**2 for i in range(len(x))])

def read_file(args, filename):
    data = np.loadtxt(filename, dtype=float)
    x = data[:,:-1]
    y = data[:,-1]
    args['features'] = len(x[0])
    args['size'] = len(x)
    return x, y

def SGD(args, x, y):
    error_list = list()
    w_list = list()
    for i in trange(args['repeat_time']):
        random.seed()
        w = np.zeros(args['feature'])
        for j in range(args['iteration']):
            pick = random.randint(0, args['size']-1)
            w = w + args['learning_rate'] * 2 * (y[pick] - np.dot(w, x[pick])) * x[
        error_list.append(calculate_E_in_sqr(x, y, w))
        w_list.append(w)
    return w_list, error_list
```

```
def main(args):
    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
    args['feature'] = len(x_train[0])

    weight_list, E_in_list = SGD(args, x_train, y_train)
    E_in_average = np.mean(E_in_list)
    print("E_in_average: ", E_in_average)

if __name__ == '__main__':
    args = {
        'filename_test': "hw3_test.dat",
        'filename_train': "hw3_train.dat",
        'feature': 11,  # without include y
        'iteration': 800,
        'learning_rate': 0.001,
        'repeat_time': 1000,
        'size': 0
    }

    main(args)
```

```
E_in_average:  0.8225428632462761
```

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

P15

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

```
def calculate_E_in_ce(x, y, w):
    return np.mean(np.log(1 + np.exp(-y * np.dot(x, w))))

def SGD(args, x, y):
    error_list = list()
    w_list = list()
    for i in trange(args['repeat_time']):
        random.seed()
        w = np.zeros(args['feature'])
        for j in range(args['iteration']):
            pick = random.randint(0, args['size']-1)
            # 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]))
        error_list.append(calculate_E_in_ce(x, y, w))
        w_list.append(w)
    return w_list, error_list
```

```
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.

```python
def calculate_w_lin(x, y):
    # pseudo inverse: np.linalg.pinv(np.array([values]))
    return np.dot(np.linalg.pinv(x), y)

def SGD(args, x, y):
    error_list = list()
    w_list = list()
    for i in trange(args['repeat_time']):
        random.seed()
        w = args['w_lin']
        for j in range(args['iteration']):
            pick = random.randint(0, args['size']-1)
            # 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]))
        error_list.append(calculate_E_in_ce(x, y, w))
        w_list.append(w)
    return w_list, error_list

def main(args):
    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
    args['feature'] = len(x_train[0])
    args['w_lin'] = calculate_w_lin(x_train, y_train)

    weight_list, E_in_list = SGD(args, x_train, y_train)
    E_in_average = np.mean(E_in_list)
    print("E_in_average: ", E_in_average)

if __name__ == '__main__':
    args = {
        'filename_test': "hw3_test.dat",
        'filename_train': "hw3_train.dat",
        'feature': 11,  # without include y
        'iteration': 800,
        'learning_rate': 0.001,
        'repeat_time': 1000,
        'size': 0,
        'w_lin': 0, # weight_vector is a list
    }

    main(args)
```

```
E_in_average:  0.6051732441261142
```

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

P17

```
import numpy as np
import random
from tqdm import trange

def calculate_E_in_01(x, y, w):
    return np.mean(np.dot(x, w) * y <= 0)

def calculate_w_lin(x, y):
    # pseudo inverse: np.linalg.pinv(np.array([values]))
    return np.dot(np.linalg.pinv(x), y)

def read_file(args, filename):
    data = np.loadtxt(filename, dtype=float)
    x = data[:,:-1]
    y = data[:,-1]
    args['features'] = len(x[0])
    args['size'] = len(x)
    return x, y

def SGD(args, x, y):
    error_list = list()
    w_list = list()
    for i in trange(args['repeat_time']):
        random.seed()
        w = args['w_lin']
        for j in range(args['iteration']):
            pick = random.randint(0, args['size']-1)
            # 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]))
        error_list.append(calculate_E_in_01(x, y, w))
        w_list.append(w)
    return w_list, error_list

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

```
def main(args):
    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
    args['feature'] = len(x_train[0])
    args['w_lin'] = calculate_w_lin(x_train, y_train)
    weight_list, E_in_list = SGD(args, x_train, y_train)

    x_test, y_test = read_file(args, args['filename_test'])
    x_test = np.c_[np.ones(len(x_test)), x_test] # x_{n, 0}=1
    args['feature'] = len(x_train[0])
    E_out_list = np.array([calculate_E_in_01(x_test, y_test, weight_list[i]) for i
    print("|E_in-E_out|: ", np.mean(np.abs(E_in_list - E_out_list)))

if __name__ == '__main__':
    args = {
        'filename_test': "hw3_test.dat",
        'filename_train': "hw3_train.dat",
        'feature': 11,  # without include y
        'iteration': 800,
        'learning_rate': 0.001,
        'repeat_time': 1000,
        'size': 0,
        'w_lin': 0, # weight_vector is a list
    }

    main(args)
```

```
|E_in-E_out|:  0.030122500000000017
```

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

P18

```
import numpy as np
from tqdm import trange

def calculate_E_in_01(x, y, w):
    return np.mean(np.dot(x, w) * y <= 0)

def calculate_w_lin(x, y):
    # pseudo inverse: np.linalg.pinv(np.array([values]))
    return np.dot(np.linalg.pinv(x), y)

def read_file(args, filename):
    data = np.loadtxt(filename, dtype=float)
    x = data[:,:-1]
    y = data[:,-1]
    return x, y

def main(args):
    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
    x_test, y_test = read_file(args, args['filename_test'])
    x_test = np.c_[np.ones(len(x_test)), x_test] # x_{n, 0}=1

    weight = calculate_w_lin(x_train, y_train)
    E_in = calculate_E_in_01(x_train, y_train, weight)
    E_out = calculate_E_in_01(x_test, y_test, weight)

    print("E_in: ", E_in)
    print("E_out: ", E_out)
    print("|E_in-E_out|: ", np.abs(E_in - E_out))

if __name__ == '__main__':
    args = {
        'filename_test': "hw3_test.dat",
        'filename_train': "hw3_train.dat",
    }

    main(args)
```

```
E_in:  0.3
E_out:  0.34
|E_in-E_out|:  0.040000000000000036
```

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

P19

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.

```
def transformation(x, Q):
    # x_0 is included
    x_origin = x[:, 1:]
    transformed_x = x.copy()
    for q in range(2, Q+1):
        transformed_x = np.hstack((transformed_x, x_origin**q))
    return transformed_x

def main(args):
    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
    x_test, y_test = read_file(args, args['filename_test'])
    x_test = np.c_[np.ones(len(x_test)), x_test] # x_{n, 0}=1
    x_train = transformation(x_train, args['Q'])
    x_test = transformation(x_test, args['Q'])

    weight = calculate_w_lin(x_train, y_train)
    E_in = calculate_E_in_01(x_train, y_train, weight)
    E_out = calculate_E_in_01(x_test, y_test, weight)

    print("E_in: ", E_in)
    print("E_out: ", E_out)
    print("|E_in-E_out|: ", np.abs(E_in - E_out))

if __name__ == '__main__':
    args = {
        'filename_test': "hw3_test.dat",
        'filename_train': "hw3_train.dat",
        'Q': 2,
    }

    main(args)
```

```
E_in:  0.23
E_out:  0.3125
|E_in-E_out|:  0.08249999999999999
```

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.

```
if __name__ == '__main__':
    args = {
        'filename_test': "hw3_test.dat",
        'filename_train': "hw3_train.dat",
        'Q': 8,
    }

    main(args)
```

```
E_in:  0.0
E_out:  0.415
|E_in-E_out|:  0.415
```

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