5. For the diagonal of $A_E(\vec{w})$, we have $\frac{\partial^2}{\partial w_j \partial w_i} E(\vec{w})$ $i,j = 1, \dots, d$

$$\frac{\partial^2}{\partial w_j \partial w_i} E(\vec{w}) = \frac{\partial^2}{\partial w_j \partial w_i} E_{in}(\vec{w})$$

$$= \frac{\partial^2}{\partial w_j \partial w_i} \frac{1}{N} \sum_{n=1}^{N} \ln(1+e^{-y_n \vec{w}^T \vec{x}_n})$$

$$= \frac{1}{N} \frac{\partial}{\partial w_j} \left( \frac{\partial}{\partial w_i} \sum_{n=1}^{N} \ln(1+e^{-y_n \vec{w}^T \vec{x}_n}) \right) - *$$

first calculate this part.

Let $f_n = \ln(1+e^{-y_n \vec{w}^T \vec{x}_n})$

$$\frac{\partial}{\partial w_i} \ln(1+e^{-y_n \vec{w}^T \vec{x}_n}) = \frac{\partial f_n}{\partial w_i}$$

Let $g_n = -y_n \vec{w}^T \vec{x}_n$

$\frac{d f_n}{d g_n} \cdot \frac{\partial g_n}{\partial w_i}$

$$= \frac{1}{1+e^{g_n}} (-y_n x_{n,i})$$

where $x_{n,i}$ is the $i$-th term of $\vec{x}_n$

$$= \frac{1}{1+e^{-y_n \vec{w}^T \vec{x}_n}} (-y_n x_{n,i})$$

Plug in to $*$:

$$= -\frac{1}{N} \frac{\partial}{\partial w_j} \left( \sum_{n=1}^{N} \frac{y_n x_{n,i}}{1+e^{y_n \vec{w}^T \vec{x}_n}} \right)$$

$$= -\frac{1}{N} \sum_{n=1}^{N} \frac{\partial}{\partial w_j} \left( \frac{y_n x_{n,i}}{1+e^{y_n \vec{w}^T \vec{x}_n}} \right)$$

$$= -\frac{1}{N} \sum_{n=1}^{N} y_n x_{n,i} \cdot \frac{\partial}{\partial w_j} \left( \frac{1}{1+e^{y_n \vec{w}^T \vec{x}_n}} \right)$$

$$= -\frac{1}{N} \sum_{n=1}^{N} y_n x_{n,i} \cdot \frac{-y_n x_{n,j} \cdot e^{-y_n \vec{w}^T \vec{x}_n}}{(1+e^{-y_n \vec{w}^T \vec{x}_n})^2}$$

$$= \frac{1}{N} \sum_{n=1}^{N} \frac{y_n^2 x_{n,i} x_{n,j} e^{-y_n \vec{w}^T \vec{x}_n}}{(1+e^{-y_n \vec{w}^T \vec{x}_n})^2}$$

is the $(i,j)$-th $(j,i$-th$)$ term of the Hessian $A_E(\vec{w})$

$A_E(\vec{w}) \in \mathbb{R}^{d \times d}$

$\parallel$

$\underset{N \times N}{X^T D} \underset{N \times d}{X}$

If we further decompose the result, we can obtain:

$X = \begin{bmatrix} \vec{x}_1^T \\ \vec{x}_2^T \\ \vdots \\ \vec{x}_N^T \end{bmatrix}$

$$\frac{1}{N}\sum_{n=1}^{N} \frac{y_n^2 x_{n,i} x_{n,j} e^{-y_n \vec{w}^T \vec{x}_n}}{(1+e^{-y_n \vec{w}^T \vec{x}_n})^2}$$

$$= \frac{1}{N}\sum_{n=1}^{N} x_{n,i}\, x_{n,j} \frac{y_n^2 e^{-y_n \vec{w}^T \vec{x}_n}}{(1+e^{-y_n \vec{w}^T \vec{x}_n})^2}$$

$\because$ Let $\vec{x}_n = [x_{n1}\ x_{n2} \cdots x_{nd}]^T$

$$\vec{x}_n \vec{x}_n^T = \begin{bmatrix} x_{n1} \\ \vdots \\ x_{nd} \end{bmatrix} [x_{n1} \cdots x_{nd}] = \begin{bmatrix} x_{n1}^2 & x_{n1}x_{n2} & \cdots & x_{n1}x_{nd} \\ x_{n2}x_{n1} & x_{n2}^2 & & \vdots \\ \vdots & & \ddots & \\ x_{nd}x_{n1} & \cdots & & x_{nd}^2 \end{bmatrix}$$

so that:

$$A_E(\vec{w}_t) = \frac{1}{N}\sum_{n=1}^{N} \underbrace{\boxed{\frac{y_n^2 e^{-y_n \vec{w}_t^T \vec{x}_n}}{(1+e^{-y_n \vec{w}_t^T \vec{x}_n})^2}}}_{= l_n} \vec{x}_n \vec{x}_n^T$$

$$= \frac{1}{N}(l_1 \vec{x}_1 \vec{x}_1^T + l_2 \vec{x}_2 \vec{x}_2^T + \cdots + l_N \vec{x}_N \vec{x}_N^T)$$

$$= \vec{x}_1 (\tfrac{1}{N} l_1) \vec{x}_1^T + \vec{x}_2 (\tfrac{1}{N} l_2) \vec{x}_2^T + \cdots + \vec{x}_N (\tfrac{1}{N} l_N) \vec{x}_N^T$$

$$= [\vec{x}_1 \cdots \vec{x}_N] \begin{bmatrix} \frac{1}{N} l_1 & & 0 \\ & \frac{1}{N} l_2 & \\ 0 & \ddots & \frac{1}{N} l_N \end{bmatrix}_{N \times N} \begin{bmatrix} \vec{x}_1^T \\ \vdots \\ \vec{x}_N^T \end{bmatrix}$$

$\because X = \begin{bmatrix} \vec{x}_1^T \\ \vdots \\ \vec{x}_N^T \end{bmatrix}$

$$= X^T D X$$

$\therefore X^T = [\vec{x}_1 \cdots \vec{x}_N]$

where $D = \begin{bmatrix} \frac{1}{N} l_1 & & 0 \\ & \frac{1}{N} l_2 & \\ 0 & \ddots & \frac{1}{N} l_N \end{bmatrix}_{N \times N}$ is a diagonal matrix

$$l_n = \frac{y_n^2 e^{-y_n \vec{w}_t^T \vec{x}_n}}{(1+e^{-y_n \vec{w}_t^T \vec{x}_n})^2} = \frac{y_n^2 e^{-y_n \vec{w}_t^T \vec{x}_n}}{h_t(y_n \vec{x}_n)^2} \quad \square$$

$\because \theta(s) = \dfrac{1}{1+e^{-s}}$ $\quad h_t(\vec{x}) = \theta(\vec{w}_t^T \vec{x}) = (1+e^{-\vec{w}_t^T \vec{x}})^{-1}$

$\rightarrow h_t(y_n \vec{x}_n) = \theta(y_n \vec{w}_t^T \vec{x}_n) = (1+e^{-y_n \vec{w}_t^T \vec{x}_n})$

6. K-class classification → output space $\mathcal{Y} = \{1, 2, \dots, K\}$

Matrix $W$ → represents a hypothesis $h_y()$

$$W = [\vec{w}_1\ \vec{w}_2 \cdots \vec{w}_k \cdots \vec{w}_K]_{(d+1)\times K} \qquad \vec{w}_k \in \mathbb{R}^{d+1}\ \forall k \in \{1, \dots, K\}$$

$$h_y(\vec{x}) = \frac{e^{\vec{w}_y^T \vec{x}}}{\sum_{k=1}^{K} e^{\vec{w}_k^T \vec{x}}} \xrightarrow{\text{approx.}} P(y|\vec{x})$$

$D = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \dots, (\vec{x}_N, y_N)\} \overset{iid}{\sim} p(\vec{x})$, target distribution $p(y|\vec{x})$

$$l(h_y|D) = p(D|h_y) \propto \prod_{n=1}^{N} h_{y_n}(\vec{x}_n)$$

$$\min -\ln \text{likelihood} \propto \min -\ln \prod_{n=1}^{N} h_{y_n}(\vec{x}_n) = \min \sum_{n=1}^{N} -\ln h_{y_n}(\vec{x}_n) = \min \sum_{n=1}^{N} err(W, \vec{x}_n, y_n)$$

$$= \min \sum_{n=1}^{N} -\ln \frac{e^{\vec{w}_{y_n}^T \vec{x}_n}}{\sum_{k=1}^{K} e^{\vec{w}_k^T \vec{x}_n}}$$

$$= \min \sum_{n=1}^{N} \left[ -\vec{w}_{y_n}^T \vec{x}_n + \ln\left(\sum_{k=1}^{K} e^{\vec{w}_k^T \vec{x}_n}\right) \right]$$

$$\frac{\partial\, err(W, \vec{x}, y)}{\partial \vec{w}_i} = \frac{\partial}{\partial \vec{w}_i}\left[-\vec{w}_y^T \vec{x} + \ln\left(\sum_{k=1}^{K} e^{\vec{w}_k^T \vec{x}}\right)\right]$$

$$= \frac{\partial}{\partial \vec{w}_i}(-\vec{w}_y^T \vec{x}) + \frac{\partial}{\partial \vec{w}_i}\left(\sum_{k=1}^{K} e^{\vec{w}_k^T \vec{x}}\right)$$

$$= \begin{cases} -\vec{x} + h_i(\vec{x}) \cdot \vec{x} & y = i \\ h_i(\vec{x}) \cdot \vec{x} & y \neq i \end{cases}$$

$\frac{\partial}{\partial \vec{w}_i}(-\vec{w}_y^T \vec{x})$ → if $y = i$, then $\frac{\partial}{\partial \vec{w}_i}(-\vec{w}_y^T \vec{x}) = \frac{\partial}{\partial \vec{w}_i}(-\vec{w}_i^T \vec{x}) = -\vec{x}$

if $y \neq i$, then $\frac{\partial}{\partial \vec{w}_i}(-\vec{w}_y^T \vec{x}) = 0$

$$\left( \frac{\partial\, err(W, \vec{x}_n, y_n)}{\partial \vec{w}_i} = \begin{cases} -\vec{x}_n + h_i(\vec{x}_n) \cdot \vec{x}_n & y_n = i \\ h_i(\vec{x}_n) \cdot \vec{x}_n & y_n \neq i \end{cases} \right)$$

$$\frac{\partial}{\partial \vec{w}_i}\left(\sum_{k=1}^{K} e^{\vec{w}_k^T \vec{x}}\right) \underset{\text{chain rule}}{=} \underbrace{\boxed{\frac{1}{\sum_{k=1}^{K} e^{\vec{w}_k^T \vec{x}}} \cdot e^{\vec{w}_i^T \vec{x}}}}_{= h_i(\vec{x})} \cdot \vec{x}$$

$$= h_i(\vec{x}) \cdot \vec{x}$$

Let $V = [\vec{v}_1\ \vec{v}_2 \cdots \vec{v}_K]_{(d+1)\times K}$ where $\vec{v}_i \in \mathbb{R}^{d+1}\ \forall i \in \{1, \dots, K\}$

$\vec{v}_1$: update direction for $\vec{w}_1$

$\forall i \in \{1, \dots, K\}\ i \neq y_n$, $\vec{v}_i = -h_i(\vec{x}_n) \cdot \vec{x}_n$

For $i = y_n$, $\vec{v}_i = \vec{v}_{y_n} = -\left[-\vec{x}_n + h_i(\vec{x}_n)\vec{x}_n\right] = \vec{x}_n - h_i(\vec{x}_n)\vec{x}_n = \vec{x}_n(1 - h_i(\vec{x}_n))$

i.e. $$V = \left[\vec{x}_n \cdot -h_i(\vec{x}_n) \cdots \boxed{\vec{x}_n(1 - h_i(\vec{x}_n))} \cdots \vec{x}_n \cdot -h_i(\vec{x}_n)\right]$$

$$= \vec{x}_n \left[-h_i(\vec{x}_n) \cdots -h_i(\vec{x}_n)\ \underbrace{\boxed{(1 - h_i(\vec{x}_n))}}_{i = y_n}\ -h_i(\vec{x}_n) \cdots -h_i(\vec{x}_n)\right]$$

$$= \vec{x}_n \cdot \vec{u}^T$$

$$\therefore \vec{u}^T = [-h_i(\vec{x}_n) \cdots (1 - h_i(\vec{x}_n)) \cdots -h_i(\vec{x}_n)]$$

$$\Rightarrow \vec{u} = [-h_i(\vec{x}_n) \cdots (1 - h_i(\vec{x}_n)) \cdots -h_i(\vec{x}_n)]^T \in \mathbb{R}^{K\times 1} \quad \square$$

7. $k=2$, $y_n \to y_n' = 2y_n - 3$ logistic regression $\to \vec{w}_{lr} \Rightarrow$ function of $(\vec{w}_1^*, \vec{w}_2^*)$

$\uparrow$ opt. sol from MLR

$\because k=2 \therefore y_n \in \{1,2\}$

$$\begin{cases} \text{if } y_n = 1, \text{ then } y_n' = 2-3 = -1 \\ \text{if } y_n = 2, \text{ then } y_n' = 4-3 = 1 \end{cases}$$

Thus, we form the binary classification as:

$$\begin{cases} y_n = 1 \longrightarrow y_n' = -1 \\ y_n = 2 \longrightarrow y_n' = 1 \end{cases}$$

Since $P(y=k \mid \vec{x}) = \dfrac{e^{\vec{w}_k^{*T}\vec{x}}}{e^{\vec{w}_1^{*T}\vec{x}} + e^{\vec{w}_2^{*T}\vec{x}}}$ (MLR)

and

$P(y'=1 \mid \vec{x}) = \dfrac{1}{1+e^{-\vec{w}_{lr}^T\vec{x}}}$, $P(y'=-1 \mid \vec{x}) = \dfrac{1}{1+e^{\vec{w}_{lr}^T\vec{x}}}$ (logistic regression)

$\Rightarrow$ from $\begin{cases} P(y=1 \mid \vec{x}) = P(y'=-1 \mid \vec{x}) \\ P(y=2 \mid \vec{x}) = P(y'=1 \mid \vec{x}) \end{cases}$

we can get:

$$\begin{cases} \dfrac{e^{\vec{w}_1^{*T}\vec{x}}}{e^{\vec{w}_1^{*T}\vec{x}} + e^{\vec{w}_2^{*T}\vec{x}}} = \dfrac{1}{1+e^{\vec{w}_{lr}^T\vec{x}}} & \text{— ①} \\[2em] \dfrac{e^{\vec{w}_2^{*T}\vec{x}}}{e^{\vec{w}_1^{*T}\vec{x}} + e^{\vec{w}_2^{*T}\vec{x}}} = \dfrac{1}{1+e^{-\vec{w}_{lr}^T\vec{x}}} & \text{— ②} \end{cases}$$

①: $e^{\vec{w}_1^{*T}\vec{x}}(1+e^{\vec{w}_{lr}^T\vec{x}}) = e^{\vec{w}_1^{*T}\vec{x}} + e^{\vec{w}_2^{*T}\vec{x}}$

$\Rightarrow \cancel{e^{\vec{w}_1^{*T}\vec{x}}} + e^{\vec{w}_1^{*T}\vec{x}} e^{\vec{w}_{lr}^T\vec{x}} = \cancel{e^{\vec{w}_1^{*T}\vec{x}}} + e^{\vec{w}_2^{*T}\vec{x}}$

$\Rightarrow e^{\vec{w}_1^{*T}\vec{x} + \vec{w}_{lr}^T\vec{x}} = e^{\vec{w}_2^{*T}\vec{x}}$

$\Rightarrow \vec{w}_1^{*T}\vec{x} + \vec{w}_{lr}^T\vec{x} = \vec{w}_2^{*T}\vec{x}$

$\Rightarrow \underline{\vec{w}_{lr}^T}\vec{x} = -\vec{w}_1^{*T}\vec{x} + \vec{w}_2^{*T}\vec{x} = \underline{(-\vec{w}_1^* + \vec{w}_2^*)}^T\vec{x}$

Thus, $\vec{w}_{lr} = -\vec{w}_1^* + \vec{w}_2^*$ $\square$

8. target function: $f(x) = 1-2x^2$

sample $x$ uniformly from $[0,1]$ $\to$ $h(x) = w_0 + w_1 x$, squared error

training set $D = \{ (x_1, f(x_1)), (x_2, f(x_2)) \}$ $x_1, x_2$ uniformly sampled from $[0,1]$

$$\Rightarrow E_{in}(g) = \sum_{i=1}^{2} (h(x_i) - f(x_i))^2$$
$$= \sum_{i=1}^{2} \left[ w_0 + w_1 x_i - (1-2x_i^2) \right]^2$$
$$= \sum_{i=1}^{2} \left[ w_0 + w_1 x_i - 1 + 2x_i^2 \right]^2$$

$\because$ we're fitting 2 points to a line

$\therefore$ can find such line that can fit both 2 points $\forall$ possible $D$.

$\to$ Therefore, $E_{in}(g) = 0$

$$\begin{cases} \frac{\partial}{\partial w_0} E_{in}(g) = \sum_{i=1}^{2} 2(w_0 + w_1 x_i - 1 + 2x_i^2) \cdot 1 = 0 & \text{— ①} \\ \frac{\partial}{\partial w_1} E_{in}(g) = \sum_{i=1}^{2} 2(w_0 + w_1 x_i - 1 + 2x_i^2) x_i = 0 & \text{— ②} \end{cases}$$

$$\Rightarrow \begin{cases} ① \ (w_0 + w_1 x_1 - 1 + 2x_1^2) + (w_0 + w_1 x_2 - 1 + 2x_2^2) = 0 \\ ② \ (w_0 + w_1 x_1 - 1 + 2x_1^2) x_1 + (w_0 + w_1 x_2 - 1 + 2x_2^2) x_2 = 0 \end{cases}$$

$① \times x_1$: $(w_0 + w_1 x_1 - 1 + 2x_1^2) x_1 + (w_0 + w_1 x_2 - 1 + 2x_2^2) x_1 = 0$

$\to ① \times x_1 - ②$: $(w_0 + w_1 x_2 - 1 + 2x_2^2)(x_1 - x_2) = 0$

$① \times x_2$: $(w_0 + w_1 x_1 - 1 + 2x_1^2) x_2 + (w_0 + w_1 x_2 - 1 + 2x_2^2) x_2 = 0$

$\to ② - ① \times x_2$: $(w_0 + w_1 x_1 - 1 + 2x_1^2)(x_1 - x_2) = 0$

If we ignore degenerate cases that $x_1 = x_2$, then

$$\begin{cases} w_0 + w_1 x_2 - 1 + 2x_2^2 = 0 \\ w_0 + w_1 x_1 - 1 + 2x_1^2 = 0 \end{cases}$$

$$\Rightarrow \begin{cases} w_0 = -w_1 x_2 + 1 - 2x_2^2 \\ -w_1 x_2 + \cancel{1} - 2x_2^2 + w_1 x_1 - \cancel{1} + 2x_1^2 = 0 \end{cases}$$

$$\Rightarrow \begin{cases} w_0 = -w_1 x_2 + 1 - 2x_2^2 \\ w_1 (x_1 - x_2) + 2(x_1^2 - x_2^2) = 0 \end{cases} \to$$

$w_1(x_1 - x_2) + 2(x_1 - x_2)(x_1 + x_2) = 0$

$[w_1 + 2(x_1 + x_2)](x_1 - x_2) = 0$

$\Downarrow$

$w_1 + 2(x_1 + x_2) = 0 \Rightarrow$

$w_1 = -2(x_1 + x_2)$

$w_0 = 2(x_1 + x_2)x_2 + 1 - 2x_2^2$

$= 2x_1 x_2 + \cancel{2x_2^2} + 1 - \cancel{2x_2^2}$

$$\Rightarrow \begin{cases} w_0 = 2x_1 x_2 + 1 \\ w_1 = -2(x_1 + x_2) \end{cases}$$

Thus, $g(x) = 2x_1 x_2 + 1 - 2(x_1 + x_2) x$

$$E_D[E_{out}(g)]$$

$$= E_D\left[E_x\left[(f(x)-g(x))^2\right]\right]$$

$$= E_D\left[E_x\left[\left(\cancel{1}-2x^2-2x_1x_2-\cancel{1}+2(x_1+x_2)x\right)^2\right]\right]$$

$$(-2x^2-2x_1x_2+2x_1x+2x_2x)^2$$

$$= 4x^4+4x_1x_2x^2-4x_1x^2-4x_2x^2$$

$$+4x_1x_2x^2+4x_1^2x_2^2-4x_1^2x_2x-4x_1x_2^2x$$

$$-4x_1x^3-4x_1^2x_2x+4x_1^2x^2+4x_1x_2x^2$$

$$-4x_2x^3-4x_1x_2^2x+4x_1x_2x^2+4x_2^2x^2$$

$$= E_D\left[\int_0^1 (2x_1x_2-2(x_1+x_2)x+2x^2)^2\,dx\right]$$

$$= E_D\Big[\tfrac{4}{5}+\tfrac{4}{3}x_1x_2-\tfrac{4}{3}x_1-\tfrac{4}{3}x_2$$
$$+\tfrac{4}{3}x_1x_2+4x_1^2x_2^2-2x_1^2x_2-2x_1x_2^2$$
$$-x_1-2x_1^2x_2+\tfrac{4}{3}x_1^2+\tfrac{4}{3}x_1x_2$$
$$-x_2-2x_1x_2^2+\tfrac{4}{3}x_1x_2+\tfrac{4}{3}x_2^2\Big]$$

$$= \int_0^1\int_0^1\Big[\tfrac{4}{5}+\tfrac{4}{3}x_1x_2-\tfrac{4}{3}x_1-\tfrac{4}{3}x_2$$
$$+\tfrac{4}{3}x_1x_2+4x_1^2x_2^2-2x_1^2x_2-2x_1x_2^2$$
$$-x_1-2x_1^2x_2+\tfrac{4}{3}x_1^2+\tfrac{4}{3}x_1x_2$$
$$-x_2-2x_1x_2^2+\tfrac{4}{3}x_1x_2+\tfrac{4}{3}x_2^2\Big]dx_1\,dx_2$$

$$= \int_0^1\Big[\tfrac{4}{5}+\tfrac{4}{3}x_2-\tfrac{2}{3}-\tfrac{4}{3}x_2+\tfrac{2}{3}x_2+\tfrac{4}{3}x_2^2-\tfrac{2}{3}x_2-x_2^2-\tfrac{1}{2}-\tfrac{2}{3}x_2+\tfrac{4}{9}+\tfrac{2}{3}x_2$$
$$-x_2-x_2^2+\tfrac{2}{3}x_2+\tfrac{4}{3}x_2^2\Big]dx_2$$

$$= \tfrac{4}{5}+\cancel{\tfrac{2}{3}}-\cancel{\tfrac{2}{3}}-\tfrac{2}{3}+\cancel{\tfrac{1}{3}}+\tfrac{4}{9}-\cancel{\tfrac{1}{3}}-\cancel{\tfrac{1}{3}}-\tfrac{1}{2}-\cancel{\tfrac{1}{3}}+\tfrac{4}{9}+\cancel{\tfrac{1}{3}}-\tfrac{1}{2}-\tfrac{1}{3}+\cancel{\tfrac{1}{3}}+\tfrac{4}{9}$$

$$= \tfrac{4}{5}-\underline{\tfrac{2}{3}}+\tfrac{12}{9}\underline{-1}-\underline{\tfrac{1}{3}} \qquad -2 \qquad \tfrac{4}{3}+\tfrac{4}{5}=\tfrac{20+12}{15}=\tfrac{32}{15}$$

$$= \tfrac{32}{15}-2$$

$$= \tfrac{2}{15} \qquad \square$$

9. training data set $(\vec{x}_1, y_1)\ (\vec{x}_2, y_2), \cdots, (\vec{x}_N, y_N)$

virtual examples $(\tilde{x}_1, y_1)\ (\tilde{x}_2, y_2), \cdots, (\tilde{x}_N, y_N)$

each $\tilde{x}_n = \vec{x}_n + \vec{\varepsilon}$ where $\vec{\varepsilon} \sim N(\vec{0}_{d+1}, \sigma^2 I_{d+1})$ $\vec{\varepsilon} \in \mathbb{R}^{d+1}$ let $\vec{\varepsilon} = [\varepsilon_0\ \varepsilon_1 \cdots \varepsilon_d]^T$

i.e. $E[\varepsilon_i] = 0 \quad \forall i = 0 \sim d$ $\quad (\varepsilon_i \sim N(0,1) \quad \forall i = 0 \sim d)$

$Var(\varepsilon_i) = 1 \quad \forall i = 0 \sim d$

$X_h = [\vec{x}_1 \cdots \vec{x}_N\ \tilde{x}_1 \cdots \tilde{x}_N]^T \rightarrow X_h^T = [\vec{x}_1 \cdots \vec{x}_N\ \tilde{x}_1 \cdots \tilde{x}_N]$

$$= \begin{bmatrix} \vec{x}_1^T \\ \vdots \\ \vec{x}_N^T \\ \tilde{x}_1^T \\ \vdots \\ \tilde{x}_N^T \end{bmatrix}$$

$$X^TX = [\vec{x}_1^T \cdots \vec{x}_N^T] \begin{bmatrix} \vec{x}_1 \\ \vdots \\ \vec{x}_N \end{bmatrix} = \sum_{n=1}^{N} \vec{x}_n \vec{x}_n^T$$

$$\boxed{E[X_h^T X_h] = \alpha X^T X + \beta \sigma^2 I_{d+1} \quad \vec{\varepsilon} \sim N(\vec{0}_{d+1}, \sigma^2 I_{d+1})}$$

$$E[X_h^T X_h] = E\left[ [\vec{x}_1 \cdots \vec{x}_N\ \tilde{x}_1 \cdots \tilde{x}_N] \begin{bmatrix} \vec{x}_1^T \\ \vdots \\ \vec{x}_N^T \\ \tilde{x}_1^T \\ \vdots \\ \tilde{x}_N^T \end{bmatrix} \right]$$

$$= E[\vec{x}_1\vec{x}_1^T + \cdots + \vec{x}_N\vec{x}_N^T + \tilde{x}_1\tilde{x}_1^T + \cdots + \tilde{x}_N\tilde{x}_N^T]$$

$$= E\left( \sum_{n=1}^{N} [\vec{x}_n\vec{x}_n^T + (\vec{x}_n + \vec{\varepsilon})(\vec{x}_n + \vec{\varepsilon})^T] \right)$$

$(\vec{x}_n + \vec{\varepsilon})(\vec{x}_n + \vec{\varepsilon})^T \qquad \vec{x}_n \in \mathbb{R}^{d+1}\ \vec{\varepsilon} \in \mathbb{R}^{d+1}$

$= (\vec{x}_n + \vec{\varepsilon})(\vec{x}_n^T + \vec{\varepsilon}^T)$

$= \vec{x}_n\vec{x}_n^T + \vec{x}_n\vec{\varepsilon}^T + \vec{\varepsilon}\vec{x}_n^T + \vec{\varepsilon}\vec{\varepsilon}^T$

$$= \sum_{n=1}^{N} E[2\vec{x}_n\vec{x}_n^T + \vec{x}_n\vec{\varepsilon}^T + \vec{\varepsilon}\vec{x}_n^T + \vec{\varepsilon}\vec{\varepsilon}^T]$$

$$= \sum_{n=1}^{N} \left[ E[2\vec{x}_n\vec{x}_n^T] + E[\vec{x}_n\vec{\varepsilon}^T] + E[\vec{\varepsilon}\vec{x}_n^T] + E[\vec{\varepsilon}\vec{\varepsilon}^T] \right]$$

$$= \sum_{n=1}^{N} \left[ 2\vec{x}_n\vec{x}_n^T + \vec{x}_n E[\vec{\varepsilon}^T] + E[\vec{\varepsilon}]\vec{x}_n^T + E[\vec{\varepsilon}\vec{\varepsilon}^T] \right]$$

$$= 2\boxed{\sum_{n=1}^{N} \vec{x}_n\vec{x}_n^T} + \sum_{n=1}^{N} \vec{x}_n [0 \cdots 0]_{1,d+1} + \sum_{n=1}^{N} \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}_{d+1,1} \vec{x}_n^T + \sum_{n=1}^{N} \sigma^2 I_{d+1}$$

$$= 2X^TX + N\sigma^2 I_{d+1}$$

$$\vec{\varepsilon}\vec{\varepsilon}^T = \begin{bmatrix} \varepsilon_0 \\ \varepsilon_1 \\ \vdots \\ \varepsilon_d \end{bmatrix} [\varepsilon_0\ \varepsilon_1 \cdots \varepsilon_d] = \begin{bmatrix} \varepsilon_0^2 & \varepsilon_0\varepsilon_1 & \cdots & \varepsilon_0\varepsilon_d \\ \varepsilon_1\varepsilon_0 & \varepsilon_1^2 & & \vdots \\ \vdots & & \ddots & \\ \varepsilon_d\varepsilon_0 & \cdots & & \varepsilon_d^2 \end{bmatrix}$$

$$E[\vec{\varepsilon}\vec{\varepsilon}^T] = E[(\vec{\varepsilon} - E[\vec{\varepsilon}])(\vec{\varepsilon} - E[\vec{\varepsilon}])^T]$$

$= Var(\vec{\varepsilon})$

$=$ covariance matrix

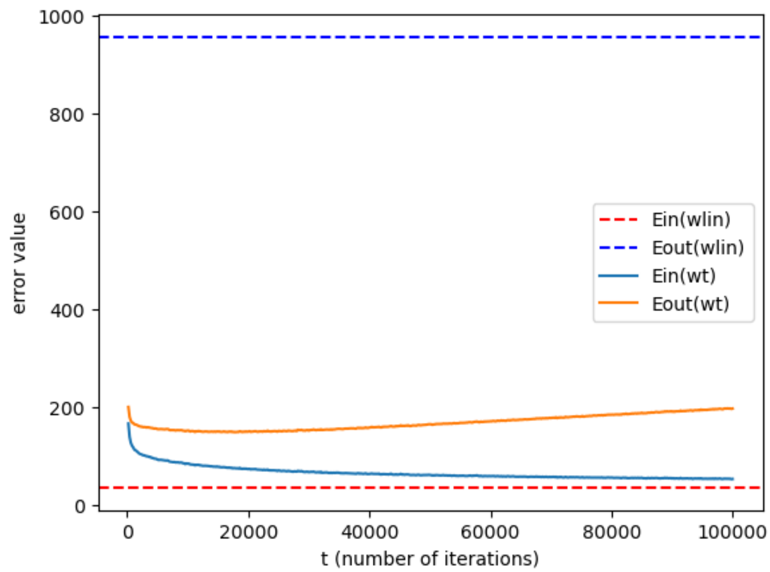
$$\begin{bmatrix} \sigma^2 & & 0 \\ & \sigma^2 & \\ 0 & & \ddots\ \sigma^2 \end{bmatrix}_{(d+1)\times(d+1)}$$

$= \sigma^2 I_{d+1}$

Thus, $\alpha = 2$, $\beta = N$ □

# ML homework 4: question 10

The resulting figure is as below:




Where the values of the in sample error and out of sample error which is not mentioned in the above plot is as below:

## Value of $E_{in}(\vec{w}_{lin})$

This value $E_{in}(\vec{w}_{lin})$ is the `avg_in_sample` in the code, and is the red dotted line in the plot.

```
avg_in_sample
```

[521] ✓ 0.0s Python

```
35.689598963498284
```

## Value of $E_{out}(\vec{w}_{lin})$

This value $E_{out}(\vec{w}_{lin})$ is the `avg_out_sample` in the code, and is the blue dotted line in the plot.

```
avg_out_sample
```

[522] ✓ 0.0s Python

```
956.5060504597701
```

## Meaning:

We can see that at the initial values of t, both the average in-sample error and out of sample error using SGD decreases rapidly, this dues to the fact that we initialize $\mathrm{w_t}$ as a zero vector, which is far from the real weight vector. Therefore, at the first iterations, we make great change on the weight vector, causing great error reduction (because the gradient is large.) However, as the weight vector gets near to the optimal weight vector, the decrease of the error became smaller.

The reason why our resulting SGD has a higher in-sample error on average may due to the reason that we only use a small subset of the dataset (choose from $\mathrm{N} = 64$ examples) to update the $100000$ iterations. This would introduce variability, and would make it hard to reach the weight vector that can have same low error as normal linear regression.

Also, another reason is that we have a closed form equation to calculate $\mathrm{w_{LIN}}$, but we update $\mathrm{w_t}$ randomly at each iteration, so we may not reach the exact solution as the normal regression.

However, we can see that the gap between the in-sample error and out of sample error for SGD is smaller than the normal regression, meaning that SGD provides smaller generalization error. I think it might due to the randomness we introduced in SGD, which is the reason of higher in-sample error as we mentioned. This also meet our expectation that smaller in-sample error is not better, we should not choose weight vectors that simply generates small in-sample error.

## Code:

# linear regression

In the linear regression function, we do the following steps:

1. calculate the weight vector $w_{LIN}$ by using the normal equation $w_{LIN} = (X^T X)^{-1} X^T y$

> the inverse of $X^T X$ is calculated by using `np.linalg.inv()`, and the matrix multiplication is done by using `@`

2. calculate the in-sample error

> $E_{in}(\vec{w}_{LIN}) = \frac{1}{N}(X^T \vec{w}_{LIN} - \vec{y})^2$

3. estimate the out of sample error

> calculated similarly to the in-sample error, with the data matrix and real value array changed.

```python
def linear_regression(X_in_mat, y_in_arr, X_out_mat, y_out_arr):

    w_lin = np.linalg.inv(X_in_mat.T @ X_in_mat) @ X_in_mat.T @ y_in_arr

    in_sample_error = np.mean((X_in_mat @ w_lin - y_in_arr) ** 2)
    out_of_sample_error = np.mean((X_out_mat @ w_lin - y_out_arr) ** 2)

    in_out_sample_error.append((in_sample_error, out_of_sample_error))

```

✓ 0.0s

# stochastic gradient descent (SGD)

In each iteration, we do the followings:

1. choose a random index from the list of sample indices, and save the corresponding data in `x_i` (a dict), `y_i` (a float)
2. convert the dictionary to a proper form and save in `input_vector`
3. calculate $-\nabla err(\vec{w}_t, \vec{x}_n, y_n)$ and save in variable `negative_stochastic_gradient`
4. update $w_t$

Each time after the above things are done, we check if t is a multiple of 200, if so, calculate the in sample error and out of sample error

```python
def sgd(sample_ind, X_in_mat, y_in_arr, X_out_mat, y_out_arr):

    w_t =np.zeros(12 + 1)

    for iteration in tqdm(range(1, 100001)):
        random_index = np.random.choice(sample_ind)
        x_i = X[random_index]
        y_i = y[random_index]

        input_vector = np.concatenate((np.array([1]), np.zeros(12)))
        for index, value in x_i.items():
            input_vector[index] = value

        update_direction = 2 * (y_i - w_t @ input_vector) * input_vector

        w_t += eta * update_direction

        if iteration % 200 == 0 and iteration != 0:
            error_record_index = (iteration // 200) - 1
            mult_200_E_in[error_record_index] += (np.mean((X_in_mat @ w_t - y_in_arr) ** 2))
            mult_200_E_out[error_record_index] += (np.mean((X_out_mat @ w_t - y_out_arr) ** 2))

```

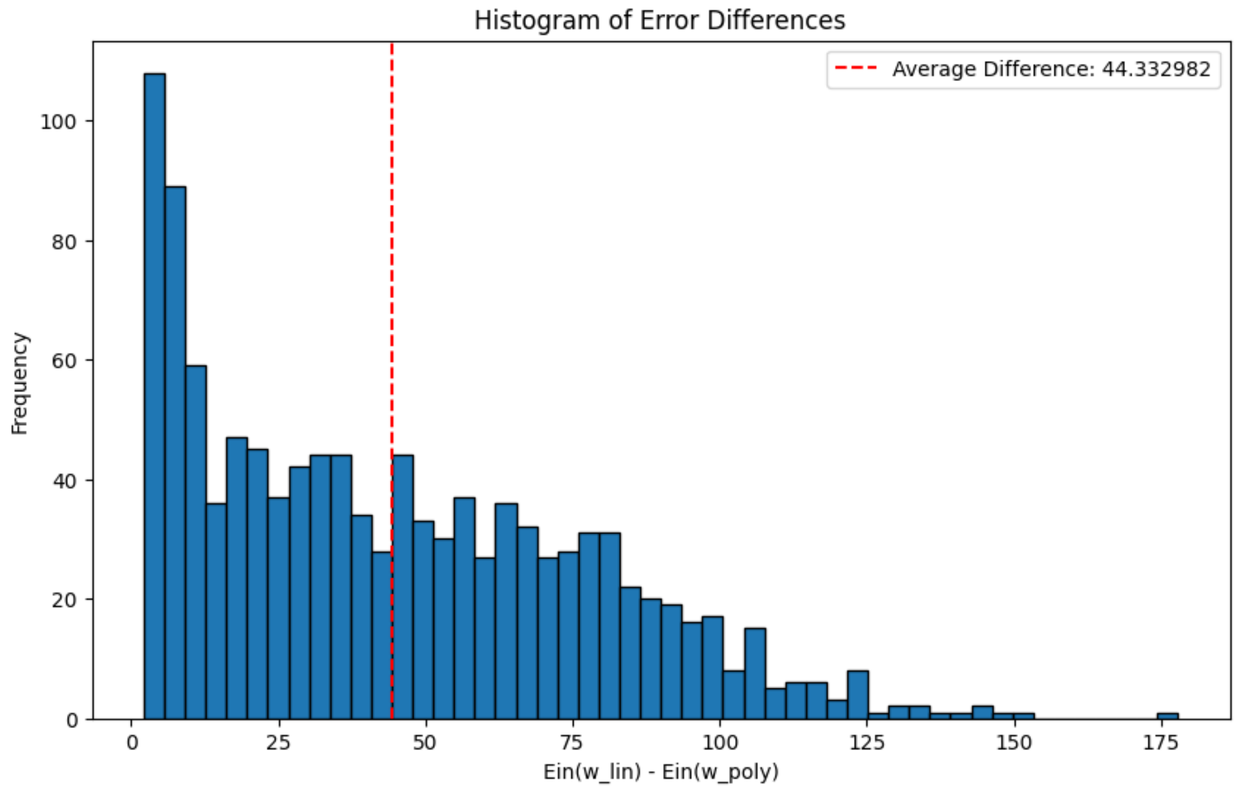
✓ 0.0s

## main function

```python
for experiment in tqdm(range(1126), leave = True):

    seed = experiment
    random_sample_indices = generate_random_sample(seed)
    X_sample = [X[i] for i in random_sample_indices]
    y_sample = [y[i] for i in random_sample_indices]

    out_ind = generate_out_of_sample_ind(random_sample_indices)
    X_out_of_sample = [X[i] for i in out_ind]
    y_out_of_sample = [y[i] for i in out_ind]

    X_sample_mat = convert_dtype(X_sample)
    X_out_of_sample_mat = convert_dtype(X_out_of_sample)

    y_sample_array = np.array(y_sample)
    y_out_of_sample_array = np.array(y_out_of_sample)

    linear_regression(X_sample_mat, y_sample_array, X_out_of_sample_mat, y_out_of_sample_array)
    sgd(random_sample_indices, X_sample_mat, y_sample_array, X_out_of_sample_mat, y_out_of_sample_array)

avg_in_sample = np.mean([error[0] for error in in_out_sample_error])
avg_out_sample = np.mean([error[1] for error in in_out_sample_error])
avg_in_sample_200 = mult_200_E_in / 1126
avg_out_sample_200 = mult_200_E_out / 1126

t_values = np.arange(200, 100200, 200)
plt.axhline(y=avg_in_sample, color='r', linestyle='--', label='Ein(wlin)')
plt.axhline(y=avg_out_sample, color='b', linestyle='--', label='Eout(wlin)')
plt.plot(t_values, avg_in_sample_200, label='Ein(wt)')
plt.plot(t_values, avg_out_sample_200, label='Eout(wt)')
plt.xlabel('t (number of iterations)')
plt.ylabel('error value')
plt.legend()
plt.show()

```

[518] ✓ 59m 20.1s

# ML homework 4: question 11




## Meaning:

From this plot we can see that the differences are mostly small positive values, which means that the error we got using linear regression is slightly greater than the error we got using polynomial transform.

This is because by using polynomial transform, we can use more parameters to fit our data, making our model more flexible to capture the nonlinear relationships among the features, thus, we can draw a function that is nearer to fitting all the examples over the 64 points.

However, the difference between the two approaches are not large, owing to our small dataset and small number of features, since under this situation, it is possible to find a line that nearly fits the points in a low dimensional space.

## Code:

### transform function $\Phi()$

We use this function to map the orignal data into a higher dimensional space, and save the augmented input vectors in `X_aug_arr`. Which means that each of the element in `X_aug_arr` is a 37-dimensional vector (12 features + 1 constant + 12 squared features + 12 cubed features).

```
def Phi(X_arr):
    X_aug_arr = []
    for arr in X_arr:
        Phi_arr = np.concatenate((np.array([1]), arr, arr**2, arr**3))
        X_aug_arr.append(Phi_arr)
    return np.array(X_aug_arr)

```

[64] ✓ 0.0s

### get weight vector

Use the pseudo inverse to get the weight vector.

```
def weight_vector(X_in_sample_mat_lin, y_in_sample_array, X_in_sample_mat_poly):

    w_lin = np.linalg.pinv(X_in_sample_mat_lin) @ y_in_sample_array
    w_poly = np.linalg.pinv(X_in_sample_mat_poly) @ y_in_sample_array
    return w_lin , w_poly
```

[26] ✓ 0.0s

### in sample error

```
def in_sample_error(X_sample_mat_lin, X_sample_mat_poly, y_sample_array, w_lin, w_poly):

    in_sample_error_lin = np.mean((X_sample_mat_lin @ w_lin - y_sample_array) ** 2)
    in_sample_error_poly = np.mean((X_sample_mat_poly @ w_poly - y_sample_array) ** 2)
    return in_sample_error_lin, in_sample_error_poly
```

[27] ✓ 0.0s

### Main function

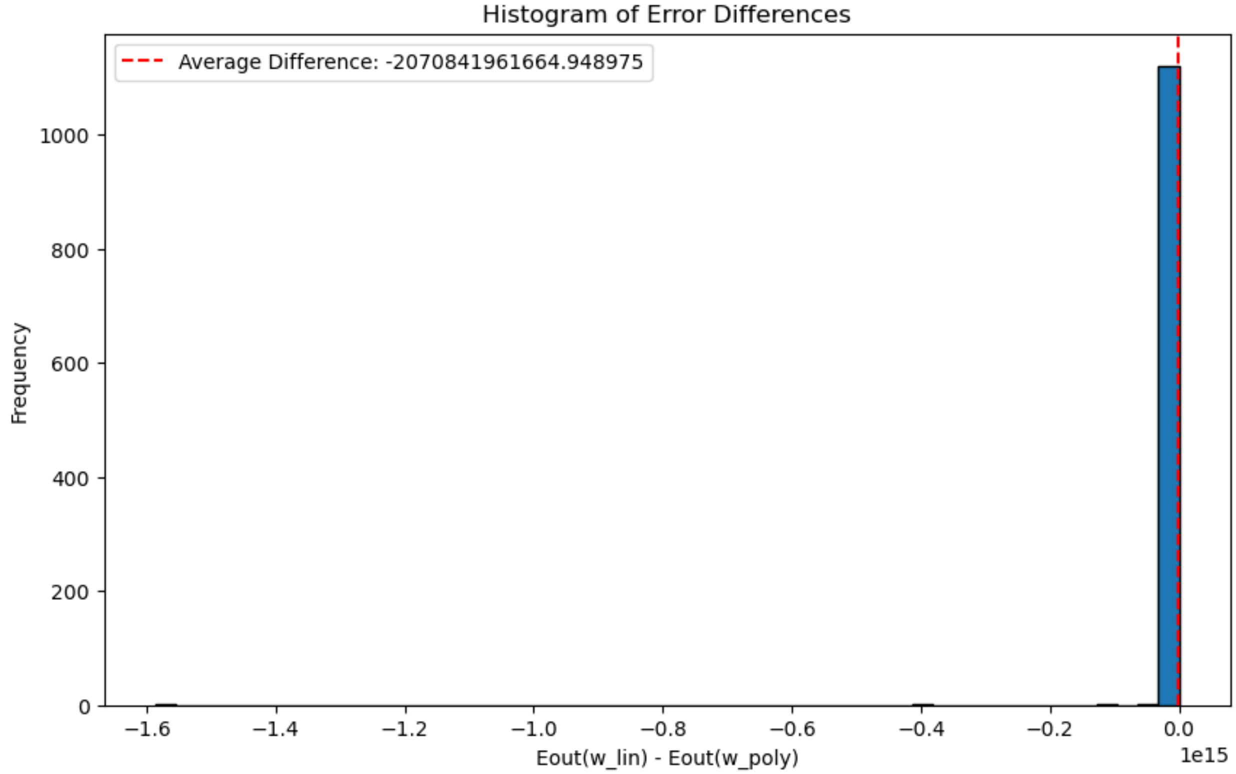
```
for experiment in range(1126):

    seed = experiment
    random_sample_indices = generate_random_sample(seed)
    X_sample = [X[i] for i in random_sample_indices]
    y_sample = [y[i] for i in random_sample_indices]

    X_sample_mat_lin = np.array(convert_dtype(X_sample))
    X_sample_mat_poly = Phi(convert_dtype(X_sample))

    y_sample_array = np.array(y_sample)

    w_lin, w_poly = weight_vector(X_sample_mat_lin, y_sample_array, X_sample_mat_poly)
    in_sample_error_lin, in_sample_error_poly = in_sample_error(X_sample_mat_lin, X_sample_mat_poly, y_sample_array, w_lin, w_poly)

    lin_sub_poly_error.append(in_sample_error_lin - in_sample_error_poly)
    avg_difference = np.mean(lin_sub_poly_error)



```

[28] ✓ 0.7s

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10, 6))
plt.hist(lin_sub_poly_error, bins=50, edgecolor='black')
plt.axvline(x=avg_difference, color='r', linestyle='--',
            label=f'Average Difference: {avg_difference:.6f}')
plt.xlabel('Ein(w_lin) - Ein(w_poly)')
plt.ylabel('Frequency')
plt.title('Histogram of Error Differences')
plt.legend()
plt.show()
```

[29] ✓ 0.1s

# ML homework 4: question 12




## Meaning:

From the plot we can see that the differences are all negative, indicating that we're getting way more bigger $\mathrm{E_{out}}(\mathrm{w_{poly}})$ than $\mathrm{E_{out}}(\mathrm{w_{LIN}})$, this may due to we're using too many features to fit our data. The original data we have might lie in a smaller space, so overfitting occurs, hence generating enormous out of sample error.

## Code:

### transform function $\Phi()$

We use this function to map the orignal data into a higher dimensional space, and save the augmented input vectors in `X_aug_arr`. Which means that each of the element in `X_aug_arr` is a 37-dimensional vector (12 features + 1 constant + 12 squared features + 12 cubed features).

```
def Phi(X_arr):
    X_aug_arr = []
    for arr in X_arr:
        Phi_arr = np.concatenate((np.array([1]), arr, arr**2, arr**3))
        X_aug_arr.append(Phi_arr)
    return np.array(X_aug_arr)

```

[41] ✓ 0.0s

R13922136 周洛君

## get weight vector

+ Code + Markdown

```python
def weight_vector(X_in_sample_mat_lin, y_in_sample_array, X_in_sample_mat_poly):

    w_lin = np.linalg.inv(X_in_sample_mat_lin.T @ X_in_sample_mat_lin) @ X_in_sample_mat_lin.T @ y_in_sample_array
    w_poly = np.linalg.inv(X_in_sample_mat_poly.T @ X_in_sample_mat_poly) @ X_in_sample_mat_poly.T @ y_in_sample_array
    return w_lin , w_poly

```

[44] ✓ 0.0s

## out of sample error

```python
def out_of_sample_error(X_out_of_sample_mat_lin, X_out_of_sample_mat_poly, y_out_of_sample_array, w_lin, w_poly):

    out_of_sample_error_lin = np.mean((X_out_of_sample_mat_lin @ w_lin - y_out_of_sample_array) ** 2)
    out_of_sample_error_poly = np.mean((X_out_of_sample_mat_poly @ w_poly - y_out_of_sample_array) ** 2)
    return out_of_sample_error_lin, out_of_sample_error_poly


```

[45] ✓ 0.0s

```python
for experiment in range(1126):

    seed = experiment
    random_sample_indices = generate_random_sample(seed)

    X_sample = [X[i] for i in random_sample_indices]
    y_sample = [y[i] for i in random_sample_indices]

    X_in_sample_mat_lin = np.array(convert_dtype(X_sample))
    X_in_sample_mat_poly = Phi(convert_dtype(X_sample))

    y_in_sample_array = np.array(y_sample)

    out_ind = generate_out_of_sample_ind(random_sample_indices)

    X_out_of_sample = [X[i] for i in out_ind]
    y_out_of_sample = [y[i] for i in out_ind]

    X_out_of_sample_mat_lin = np.array(convert_dtype(X_out_of_sample))
    X_out_of_sample_mat_poly = Phi(convert_dtype(X_out_of_sample))

    y_out_of_sample_array = np.array(y_out_of_sample)

    w_lin, w_poly = weight_vector(X_in_sample_mat_lin, y_in_sample_array, X_in_sample_mat_poly)
    out_of_sample_error_lin, out_of_sample_error_poly = out_of_sample_error(X_out_of_sample_mat_lin, X_out_of_sample_mat_poly, y_out_of_sample_array, w_lin, 

    out_sample_error_lin.append(out_of_sample_error_lin)
    out_sample_error_poly.append(out_of_sample_error_poly)
    lin_sub_poly_error.append(out_of_sample_error_lin - out_of_sample_error_poly)
    avg_difference = np.mean(lin_sub_poly_error)


plt.figure(figsize=(10, 6))
plt.hist(lin_sub_poly_error, bins=50, edgecolor='black')
plt.axvline(x=avg_difference, color='r', linestyle='--',
            label=f'Average Difference: {avg_difference:.6f}')
plt.xlabel('Eout(w_lin) - Eout(w_poly)')
plt.ylabel('Frequency')
plt.title('Histogram of Error Differences')
plt.legend()
plt.show()
```