6140 Machine Learning Assignment 1

Shreeti Shrestha

Due Date: 30 September 2025

1. In the class, we showed that for two vectors $a \in R^n$ and $b \in R^n$, the complexity of computing the inner product $a^\top b$ is $O(n)$. Let $A \in R^{m \times n}$ be a matrix. Show that the complexity of computing Ab is $O(mn)$.

Let $A \in R^{m \times n}$ be a matrix. We have:

$$\text{A} = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \vdots & \vdots & & \\ A_{m1} & A_{m2} & \dots & A_{mn} \end{bmatrix}_{mxn} \quad \text{and b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}_{nx1}$$

Given the dimensions of A = m x n and b = n x 1, we have:

$$\text{Ab} = \begin{bmatrix} A_{11}b_1 + A_{12}b_2 + \ldots + A_{1n}b_n \\ A_{21}b_1 + A_{22}b_2 + \ldots + A_{2n}b_n \\ \vdots \\ A_{m1}b_1 + A_{m2}b_2 + \ldots + A_{mn}b_n \end{bmatrix}_{mxn}$$

For each row (m) in matrix A:

We compute n products

Then, we sum up all the n products, which takes (n-1) operations

So the total number of operations is:
m * {n+ (n-1)}
= m(2n -1)
= 2mn - m (ignoring coefficients and the lower order terms for big O notation)
= $\boldsymbol{O(mn)}$.
Therefore, complexity for computing Ab is O(mn).

2. For vectors $\theta \in R^n$, $a \in R^n$ and the matrix $X \in R^{n \times n}$, show the following (provide all steps of your derivations)

1. $\frac{\partial a^\top X\theta}{\partial \theta} = X^\top a$

   We have vectors $\theta \in R^n$, $a \in R^n$ and matrix $X \in R^{n \times n}$
   $\frac{\partial (a^\top X\theta)}{\partial \theta}$

   $= \frac{\partial ((a^\top X)\theta)}{\partial \theta}$ ($a^T X$is a $1 \times n$ vector)
   $= \frac{\partial ((X^T a)^T \theta)}{\partial \theta}((X^T a)^T = a^T X)$
   =$X^T a$ (gradient of a vector z^T with respect to θ is z)

2. $\|X\theta - a\|_2^2 = \theta^\top X^\top X\theta - 2a^\top X\theta + a^\top a$
   Solving for $\|X\theta - a\|_2^2$ (LHS) :
   $= (X\theta - a)^\top (X\theta - a)$ (expansion of the squared L2 norm. $\|b\|_2^2 = b^\top b$ for a vector b)
   $= (X^\top \theta^\top - a^\top)(X\theta - a)$
   $= X^\top \theta^\top X\theta - X^\top \theta^\top a - a^\top X\theta + a^\top a$
   $\theta^T X^T a$ and $a^T X\theta$ are both scalars and scalars are unaffected by transpose. So, $\theta^T X^T a = (\theta^T X^T a)^T = a^T X\theta$
   $= X^\top \theta^\top X\theta - a^\top X\theta - a^\top X\theta + a^\top a$
   $= X^\top \theta^\top X\theta - 2a^\top X\theta + a^\top a$
   $= \boldsymbol{\theta^\top X^\top X\theta - 2a^\top X\theta + a^\top a}$ (RHS)
   Therefore, $\|X\theta - a\|_2^2 = \theta^\top X^\top X\theta - 2a^\top X\theta + a^\top a$.

3. $\frac{\partial \|X\theta - a\|_2^2}{\partial \theta} = 2X^\top (X\theta - a)$.
   From no. 2, we have: $\|X\theta - a\|_2^2 = \theta^\top X^\top X\theta - 2a^\top X\theta + a^\top a$
   $\frac{\partial \|X\theta - a\|_2^2}{\partial \theta} = \frac{\partial (\theta^\top X^\top X\theta - 2a^\top X\theta + a^\top a)}{\partial \theta}$
   $= \frac{\partial (\theta^\top X^\top X\theta)}{\partial \theta} - \frac{\partial (2a^\top X\theta)}{\partial \theta} + \frac{\partial (a^\top a)}{\partial \theta}$ (splitting the terms)

   $= 2X^\top X\theta - 2a^\top X$ ($\frac{\partial (\theta^T A\theta)}{\partial \theta} = (A + A^T)\theta$ where A = $X^\top X$)
   $\frac{\partial (\theta^T X^T X\theta)}{\partial \theta} = (X^T X + (X^T X)^T)\theta = (X^T X + X^T X)\theta = 2X^T X\theta$
   $= \boldsymbol{2X^\top (X\theta - a)}$ (taking $2X^\top$ common)
   Therefore, $\frac{\partial \|X\theta - a\|_2^2}{\partial \theta} = 2X^\top (X\theta - a)$.

3. Consider the function $f(x_1, x_2) = (x_1 - 2x_2)^2 + (x_1 - 2)^2 + (x_2 - 1)^2$.

1. Compute the derivative of f with respect to (x_1, x_2). What is the dimension of this derivative?

$$f(x_1, x_2) = (x_1 - 2x_2)^2 + (x_1 - 2)^2 + (x_2 - 1)^2$$
$$= x_1^2 - 2x_1.2x_2 + 4{x_2}^2 + x_1^2 - 2x_1.2 + 4 + x_2^2 - 2x_2.1 + 1$$
$$= x_1^2 - 4x_1x_2 + 4{x_2}^2 + x_1^2 - 4x_1 + 4 + {x_2}^2 - 2x_2 + 1$$
$$= 2x_1^2 - 4x_1x_2 - 4x_1 - 2x_2 + 5{x_2}^2 + 5$$

Taking partial derivative of $f(x_1, x_2)$ with respect to x_1:

$$\frac{\partial(f(x_1,x_2))}{\partial(x_1)} = \frac{\partial(2x_1^2 - 4x_1x_2 - 4x_1 - 2x_2 + 5{x_2}^2 + 5)}{\partial(x_1)} = 4x_1 - 4x_2 - 4$$

Taking partial derivative of $f(x_1, x_2)$ with respect to x_2:

$$\frac{\partial(f(x_1,x_2))}{\partial(x_2)} = \frac{\partial(2x_1^2 - 4x_1x_2 - 4x_1 - 2x_2 + 5{x_2}^2 + 5)}{\partial(x_2)} = -4x_1 - 2 + 10x_2$$

$$\frac{\partial(f(x_1,x_2))}{\partial(x_1,x_2)} = \begin{pmatrix} \frac{\partial(f(x_1,x_2))}{\partial(x_1)} \\ \\ \frac{\partial(f(x_1,x_2))}{\partial(x_2)} \end{pmatrix} = \begin{pmatrix} 4x_1 - 4x_2 - 4 \\ \\ -4x_1 + 10x_2 - 2 \end{pmatrix}_{2\times 1}$$

The result is a 2×1 (dimension) vector.

Starting with $(x_1, x_2) = (0, 0)$, write down the 3 first iterations of the Gradient Descent (GD) algorithm for the cases below. For each case, at each iteration of the GD, write down the updated value of (x_1, x_2) and the value of the function at the updated (x_1, x_2). Check for convergence at each iteration and if converged (in this case the gradient is zero), then stop.

For gradient descent:
t = number of iterations
$\theta = (x_1, x_2)$
l = loss function
ρ = learning rate
Until convergence: $\theta^{t+1} = \theta^t - \rho \frac{\partial(l_\theta)}{\partial\theta}\Big|_{\theta^t}$

$f(x_1, x_2) = (x_1 - 2x_2)^2 + (x_1 - 2)^2 + (x_2 - 1)^2$

$$\frac{\partial(f(x_1,x_2))}{\partial(x_1,x_2)} = \begin{pmatrix} 4x_1 - 4x_2 - 4 \\ \\ -4x_1 + 10x_2 - 2 \end{pmatrix}$$

At t = 0,

$(x_1, x_2) = (0, 0)$:

$f(0,0) = (0-2.0)^2 + (0-2)^2 + (0-1)^2 = 5$

$\frac{\partial(f(0,0))}{\partial(0,0)} = \begin{pmatrix} -4 \\ \\ -2 \end{pmatrix}$

At t = 1:

$\theta^1 = \theta^0 - \rho \frac{\partial(l_\theta)}{\partial\theta}\Big|_{\theta^0} = \begin{pmatrix} 0 \\ \\ 0 \end{pmatrix} - \rho \begin{pmatrix} -4 \\ \\ -2 \end{pmatrix} = \begin{pmatrix} \boldsymbol{4\rho} \\ \\ \boldsymbol{2\rho} \end{pmatrix}$

$$\begin{aligned} f(4\rho, 2\rho) &= (4\rho - 4\rho)^2 + (4\rho - 2)^2 + (2\rho - 1)^2 \\ &= (4\rho - 2)^2 + (2\rho - 1)^2 \\ &= 16\rho^2 - 16\rho + 4 + 4\rho^2 - 4\rho + 1 \\ &= 20\rho^2 - 20\rho + 5 \end{aligned}$$

$\frac{\partial(f(4\rho,2\rho))}{\partial(4\rho,2\rho)} = \begin{pmatrix} 4(4\rho) - 4(2\rho) - 4 \\ -4(4\rho) + 10(2\rho) - 2 \end{pmatrix} = \begin{pmatrix} 8\rho - 4 \\ 4\rho - 2 \end{pmatrix}$

At t = 2:

$\theta^2 = \theta^1 - \rho \frac{\partial(l_\theta)}{\partial\theta}\Big|_{\theta^1} = \begin{pmatrix} 4\rho \\ \\ 2\rho \end{pmatrix} - \rho \begin{pmatrix} 8\rho - 4 \\ \\ 4\rho - 2 \end{pmatrix} = \begin{pmatrix} 4\rho - 8\rho^2 + 4\rho \\ 2\rho - 4\rho^2 + 2\rho \end{pmatrix} = \begin{pmatrix} \boldsymbol{8\rho - 8\rho^2} \\ \boldsymbol{4\rho - 4\rho^2} \end{pmatrix}$

$$\begin{aligned} f(8\rho - 8\rho^2, 4\rho - 4\rho^2) &= (8\rho - 8\rho^2 - 2(4\rho - 4\rho^2))^2 + (8\rho - 8\rho^2 - 2)^2 + (4\rho - 4\rho^2 - 1)^2 \\ &= (8\rho - 8\rho^2 - 8\rho + 8\rho^2))^2 + (8\rho - 8\rho^2 - 2)^2 + (4\rho - 4\rho^2 - 1)^2 \\ &= (8\rho - 8\rho^2 - 2)^2 + (4\rho - 4\rho^2 - 1)^2 \\ &= (8\rho - 8\rho^2 - 2)^2 + (4\rho - 4\rho^2 - 1)^2 \end{aligned}$$

$$\begin{aligned} \frac{\partial(f(8\rho-8\rho^2,4\rho-4\rho^2))}{\partial(8\rho-8\rho^2,4\rho-4\rho^2)} &= \begin{pmatrix} 4(8\rho - 8\rho^2) - 4(4\rho - 4\rho^2) - 4 \\ -4(8\rho - 8\rho^2) + 10(4\rho - 4\rho^2) - 2 \end{pmatrix} \\ &= \begin{pmatrix} 32\rho - 32\rho^2 - 16\rho + 16\rho^2 - 4 \\ -32\rho + 32\rho^2 + 40\rho - 40\rho^2 - 2 \end{pmatrix} \\ &= \begin{pmatrix} 16\rho - 16\rho^2 - 4 \\ 8\rho - 8\rho^2 - 2 \end{pmatrix} \\ &= \begin{pmatrix} 16\rho(1-\rho) - 4 \\ 8\rho(1-\rho) - 2 \end{pmatrix} \end{aligned}$$

At t = 3:

$$\begin{aligned} \theta^3 = \theta^2 - \rho \frac{\partial(l_\theta)}{\partial\theta}\Big|_{\theta^2} &= \begin{pmatrix} 8\rho(1-\rho) \\ 4\rho(1-\rho) \end{pmatrix} - \rho \begin{pmatrix} 16\rho(1-\rho) - 4 \\ 8\rho(1-\rho) - 2 \end{pmatrix} \\ &= \begin{pmatrix} 8\rho(1-\rho) - 16\rho^2(1-\rho) + 4\rho \\ 4\rho(1-\rho) - 8\rho^2(1-\rho) + 2\rho \end{pmatrix} \\ &= \begin{pmatrix} 8\rho - 8\rho^2 - 16\rho^2 + 16\rho^3 + 4\rho \\ 4\rho - 4\rho^2 - 8\rho^2 + 8\rho^3 + 2\rho \end{pmatrix} \\ &= \begin{pmatrix} \boldsymbol{16\rho^3 - 24\rho^2 + 12\rho} \\ \boldsymbol{8\rho^3 - 12\rho^2 + 6\rho} \end{pmatrix} \end{aligned}$$

$f(16\rho^3 - 24\rho^2 + 12\rho, 8\rho^3 - 12\rho^2 + 6\rho) = (16\rho^3 - 24\rho^2 + 12\rho - 2(8\rho^3 - 12\rho^2 + 6\rho))^2 + (16\rho^3 - 24\rho^2 + 12\rho - 2)^2 + (8\rho^3 - 12\rho^2 + 6\rho - 1)^2$

$= (16\rho^3 - 24\rho^2 + 12\rho - 16\rho^3 + 24\rho^2 - 12\rho)^2 + (16\rho^3 -$

$$24\rho^2 + 12\rho - 2)^2 + (8\rho^3 - 12\rho^2 + 6\rho - 1)^2$$
$$= (16\rho^3 - 24\rho^2 + 12\rho - 2)^2 + (8\rho^3 - 12\rho^2 + 6\rho - 1)^2$$

$$\frac{\partial(f(\theta^{(3)})}{\partial\theta^{(3)}} = \begin{pmatrix} 4(16\rho^3 - 24\rho^2 + 12\rho) - 4(8\rho^3 - 12\rho^2 + 6\rho) - 4 \\ -4(16\rho^3 - 24\rho^2 + 12\rho) + 10(8\rho^3 - 12\rho^2 + 6\rho) - 2 \end{pmatrix}$$
$$= \begin{pmatrix} 64\rho^3 - 96\rho^2 + 48\rho - 32\rho^3 + 48\rho^2 - 24\rho - 4 \\ -64\rho^3 + 96\rho^2 - 48\rho + 80\rho^3 - 120\rho^2 + 60\rho - 2 \end{pmatrix}$$
$$= \begin{pmatrix} 32\rho^3 - 48\rho^2 + 24\rho - 4 \\ 16\rho^3 - 24\rho^2 + 12\rho - 2 \end{pmatrix}$$

2. Use the learning rate $\rho = 2$. Has GD converged in 3 iterations? If not, will it converge if you keep running it beyond 3 iterations? Explain what is happening.

   For $\rho = 2$:

   $\mathbf{t = 0}$:

   $\frac{\partial(f(x_1,x_2))}{\partial(x_1,x_2)} = \begin{pmatrix} -4 \\ -2 \end{pmatrix}$

   $\mathbf{t = 1}$: $\theta^{(1)} = \begin{pmatrix} 4\rho \\ 2\rho \end{pmatrix} = \begin{pmatrix} 8 \\ 4 \end{pmatrix}$

   $\frac{\partial(f(x_1,x_2))}{\partial(x_1,x_2)} = \begin{pmatrix} 8\rho - 4 \\ 4\rho - 2 \end{pmatrix} = \begin{pmatrix} 12 \\ 6 \end{pmatrix}$

   $\mathbf{t = 2}$: $\theta^{(2)} = \begin{pmatrix} 8\rho - 8\rho^2 \\ 4\rho - 4\rho^2 \end{pmatrix} = \begin{pmatrix} 16 - 32 \\ 8 - 16 \end{pmatrix} = \begin{pmatrix} -16 \\ -8 \end{pmatrix}$

   $\frac{\partial(f(x_1,x_2))}{\partial(x_1,x_2)} = \begin{pmatrix} 16\rho(1-\rho) - 4 \\ 8\rho(1-\rho) - 2 \end{pmatrix} = \begin{pmatrix} -36 \\ -18 \end{pmatrix}$

   $\mathbf{t = 3}$: $\theta^{(3)} = \begin{pmatrix} 16\rho^3 - 24\rho^2 + 12\rho \\ 8\rho^3 - 12\rho^2 + 6\rho \end{pmatrix} = \begin{pmatrix} 128 - 96 + 24 \\ 64 - 48 + 12 \end{pmatrix} = \begin{pmatrix} 56 \\ 28 \end{pmatrix}$

   $\frac{\partial(f(x_1,x_2))}{\partial(x_1,x_2)} = \begin{pmatrix} 32\rho^3 - 48\rho^2 + 24\rho - 4 \\ 16\rho^3 - 24\rho^2 + 12\rho - 2 \end{pmatrix} = \begin{pmatrix} 108 \\ 54 \end{pmatrix}$

   At $\rho = 2$, it diverges because the learning rate (ρ) is very large. Gradient absolute values are exploding (alternating between positive and negative values, but increasing in magnitude), so it will not converge even if we were to run it beyond 3 iterations.

3. Use the learning rate $\rho = 1$. Has GD converged in 3 iterations? If not, will it converge if you keep running it beyond 3 iterations? Explain what is happening.

   For $\rho = 1$:

   $\mathbf{t = 0}$:

   $\frac{\partial(f(x_1,x_2))}{\partial(x_1,x_2)} = \begin{pmatrix} -4 \\ -2 \end{pmatrix}$

   $\mathbf{t = 1}$:$\theta^{(1)} = \begin{pmatrix} 4\rho \\ 2\rho \end{pmatrix} = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$

$\frac{\partial(f(x_1,x_2))}{\partial(x_1,x_2)} = \begin{pmatrix} 8\rho - 4 \\ 4\rho - 2 \end{pmatrix} = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$

t = 2: $\theta^{(2)} = \begin{pmatrix} 8\rho - 8\rho^2 \\ 4\rho - 4\rho^2 \end{pmatrix} = \begin{pmatrix} 8 - 8 \\ 4 - 4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

$\frac{\partial(f(x_1,x_2))}{\partial(x_1,x_2)} = \begin{pmatrix} 16\rho(1-\rho) - 4 \\ 8\rho(1-\rho) - 2 \end{pmatrix} = \begin{pmatrix} -4 \\ -2 \end{pmatrix}$

t = 3: $\theta^{(3)} = \begin{pmatrix} 16\rho^3 - 24\rho^2 + 12\rho \\ 8\rho^3 - 12\rho^2 + 6\rho \end{pmatrix} = \begin{pmatrix} 16 - 24 + 12 \\ 8 - 12 + 6 \end{pmatrix} = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$

$\frac{\partial(f(x_1,x_2))}{\partial(x_1,x_2)} = \begin{pmatrix} 32\rho^3 - 48\rho^2 + 24\rho - 4 \\ 16\rho^3 - 24\rho^2 + 12\rho - 2 \end{pmatrix} = \begin{pmatrix} 4 \\ 2 \end{pmatrix}$

At $\rho = 1$, the learning rate is still large. The gradient is not yet zero. It stays almost the same (very slow progression across a lot of iterations) and hence, unlikely to converge.

4. Use the learning rate $\rho = 0.5$. Has GD converged in 3 iterations? If not, will it converge if you keep running it beyond 3 iterations? Explain what is happening.

   For $\rho = 0.5 = \frac{1}{2}$:

   t = 0:
   $\frac{\partial(f(x_1,x_2))}{\partial(x_1,x_2)} = \begin{pmatrix} -4 \\ -2 \end{pmatrix}$

   t = 1 : $\theta^{(1)} = \begin{pmatrix} 4\rho \\ 2\rho \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$

   $\frac{\partial(f(x_1,x_2))}{\partial(x_1,x_2)} = \begin{pmatrix} 8\rho - 4 \\ 4\rho - 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

   t = 2: $\theta^{(2)} = \begin{pmatrix} 8\rho - 8\rho^2 \\ 4\rho - 4\rho^2 \end{pmatrix} = \begin{pmatrix} 4 - 2 \\ 2 - 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$

   $\frac{\partial(f(x_1,x_2))}{\partial(x_1,x_2)} = \begin{pmatrix} 16\rho(1-\rho) - 4 \\ 8\rho(1-\rho) - 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

   t = 3: $\theta^{(3)} = \begin{pmatrix} 16\rho^3 - 24\rho^2 + 12\rho \\ 8\rho^3 - 12\rho^2 + 6\rho \end{pmatrix} = \begin{pmatrix} 2 - 6 + 6 \\ 1 - 3 + 3 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$

   $\frac{\partial(f(x_1,x_2))}{\partial(x_1,x_2)} = \begin{pmatrix} 32\rho^3 - 48\rho^2 + 24\rho - 4 \\ 16\rho^3 - 24\rho^2 + 12\rho - 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

   At $\rho = 0.5$, the gradient has decreased to zero in the 1st iteration, so we stop after the first iteration since the GD has converged.

4. Consider the training dataset $\{(x_1, y_1), \ldots, (x_{100}, y_{100})\}$, where $x_i \in R^d$ and $y_i \in R$. Consider the modified regression problem

$$\min_{\theta \in R^d} \sum_{i=1}^{10} 4(y_i - \theta^\top x_i)^2 + \sum_{i=11}^{100} (y_i - \theta^\top x_i)^2.$$

Notice that for the first 10 samples, we are emphasizing more on minimizing the error than the remaining samples (by putting a coefficient 4 in front of the error). In other words, we believe the first 10 samples are more important.

1. Write down the closed-form solution of the above regression problem. Provide all steps of your derivations. Hint: try to turn the above optimization into the form we learned in the class, i.e., $\|A\theta - b\|_2^2$, for which you know how to minimize and solve for θ.

   Let D be a 100×100 diagonal matrix, that has 4 as a value in the first 10 diagonal entries and 1 as a value in the latter 90 diagonal entries (the other entries are 0 by default in a diagonal matrix).

   $$\min_{\theta \in R^d} \sum_{i=1}^{10} 4(y_i - \theta^\top x_i)^2 + \sum_{i=11}^{100} (y_i - \theta^\top x_i)^2.$$

   Now, we can generalize this equation to:

   $$\min_{\theta \in R^d} \sum_{i=1}^{100} D(y_i - \theta^\top x_i)^2$$

   (since the least squared error remains the same except the weight of the first 10 samples, that are now in the diagonal matrix).
   Since D has constant values, we will take it out of the summation, to get:
   $\min_{\theta \in R^d} D \sum_{i=1}^{100} (y_i - \theta^\top x_i)^2$

   $= \min_{\theta \in R^d} D[(y_1 - \theta^\top x_1)^2 + (y_2 - \theta^\top x_2)^2 + ... + (y_{100} - \theta^\top x_{100})^2]$

   $= \min_{\theta \in R^d} D[(\theta^\top x_1 - y_1)^2 + (\theta^\top x_2 - y_2)^2 + ... + (\theta^\top x_{100} - y_{100})^2]$ (since it's squared)

   $= \min_{\theta \in R^d} D[(x_1^\top \theta - y_1)^2 + (x_2^\top \theta - y_2)^2 + ... + (x_{100}^\top \theta - y_{100})^2]$ (dot product of vectors)

   $$= \min_{\theta \in R^d} D \left|\left| \begin{matrix} (x_1^\top \theta - y_1) \\ (x_2^\top \theta - y_2) \\ ... \\ (x_{100}^\top \theta - y_{100}) \end{matrix} \right|\right|_2^2 \text{ (sum of squares (L2 norm))}$$

   $$= \min_{\theta \in R^d} D \left|\left| \begin{pmatrix} x_1^\top \theta \\ x_2^\top \theta \\ ... \\ x_{100}^\top \theta \end{pmatrix} - \begin{pmatrix} y_1 \\ y_2 \\ ... \\ y_{100} \end{pmatrix} \right|\right|_2^2 \text{ (splitting)}$$

$= \mathbf{min}_{\boldsymbol{\theta} \in \boldsymbol{R^d}} \boldsymbol{D} \, \boldsymbol{||X\theta - Y||_2^2}$(as proven in class for regression loss)

Let $f(\theta) = D \, ||X\theta - Y||_2^2$
$= (\mathrm{X}_i\theta - Y)^\top D(X_i\theta - Y)$(as proven for $||X||_2^2 = X^\top X$)

Now, to solve for a closed form solution, we can take the derivative and set it to 0:
$\frac{\partial f(\theta)}{\partial(\theta)} = 0$
$or, \frac{\partial (X\theta - Y)^\top D(X\theta - Y)}{\partial \theta} = 0$

$or, \frac{\partial ((X^T\theta^T - Y^\top)(DX\theta - DY)}{\partial \theta} = 0$

$or, \frac{\partial (X^T\theta^T DX\theta - X^\top\theta^T DY - Y^T DX\theta + Y^T DY)}{\partial \theta} = 0$

$or, \frac{\partial (\theta^T X^T DX\theta - 2\theta^T X^T DY + Y^T DY)}{\partial \theta} = 0$

$or, 2X^T DX\theta - 2X^T DY = 0$

$or, 2X^T DX\theta = 2X^T DY$
$or, X^T DX\theta = X^T DY$
$\theta^* = (X^T DX)^{-1} X^T DY$

$$\boldsymbol{\theta^* = (X^T DX)^{-1} X^T DY}$$

2. Write down the steps of the gradient-descent algorithm to solve for θ for the above loss.
We have $f(\theta) = D \, ||X\theta - Y||_2^2$
$= (\mathrm{X}\theta - Y)^\top D(X\theta - Y)$(as proven for $||X||_2^2 = X^\top X$)

Gradient Descent algorithm to solve for θ:
1. Initialize t = 0, θ^0 = random or zero value
2. Convergence criterion: $\left|\left| \frac{\partial (f_\theta)}{\partial \theta}\Big|_{\theta^t} \right|\right|_2 \leq \epsilon$ (where ϵ is a predefined threshold)
2. Until convergence, do:

$\theta^{t+1} = \theta^t - \rho \frac{\partial (f_\theta)}{\partial \theta}\Big|_{\theta^t}$ for some learning rateρ and $\frac{\partial f(\theta)}{\partial \theta} = 2X^T D(X\theta - y)$ (computed above)

5. In the linear regression problem, assume we are given a training set $\{(x_1, y_1), \ldots, (x_N, y_N)\}$, where $x_i \in R^d$ and $y_i \in R^k$. In other words, each response/output is a vector of dimension k instead of a scalar. Propose an approach to generalize what you have already learned in the class to vectors for responses/outputs (instead of scalars). Write down the closed-form solution of your proposed approach.

For responses as scalars, we had the loss function: $\sum_{i=1}^{N}(x_i^T\theta - y_i)^2 = \|X\theta - y\|_2^2$

When responses are vectors, our training dataset looks like:

$$\theta \in R^{(d+1)\times k},\ X = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_N^T \end{pmatrix}_{N\times(d+1)},\ Y = \begin{pmatrix} y_1^T \\ y_2^T \\ \vdots \\ y_N^T \end{pmatrix}_{N\times k}$$

The prediction is given by some function: $X\theta$

The difference is prediction is then: $X\theta - Y$, which is also an $N \times k$ matrix.

The squared error loss is then given by:

$\sum_{i=1}^{N} \|x_i^T\Theta - y_i^T\|_2^2$

We can write this as the squared Frobenius norm (since that is simply the l_2 norm applied to the vectorization of a matrix and allows us to stack columns):

$\|X\Theta - Y\|_F^2 = \|\text{vec}(X\Theta - Y)\|_2^2$

So, to find the optimal paramters, $\hat{\theta}$, we use the ordinary least squares (like for the scalar responses) and minimize the loss function

$\hat{\Theta} = \arg\min_\Theta \|X\Theta - Y\|_F^2 = \arg\min_\Theta \text{tr}((X\Theta - Y)^T(X\Theta - Y))$

Now, for closed form solution, we take the derivative and set it to 0:

$\frac{\partial(l_\theta)}{\partial\theta} = 0$

$or, \frac{\partial(\|X\Theta - Y\|_F^2)}{\partial\theta} = 0$

$or, \frac{\partial((X\Theta - Y)^T(X\theta - Y))}{\partial\theta} = 0$

$or, 2X^T(X\theta - Y) = 0$

$or, X^TX\theta - X^TY = 0$

$or, X^TX\theta = X^TY$

$or, \hat{\theta} = (X^TX)^{-1}X^TY$

Closed form solution:$\boldsymbol{\hat{\theta} = (X^TX)^{-1}X^TY}$

6. **Regularized Linear Regression:** Consider the modified linear regression problem

$$\hat{\theta} = \arg\min_{\theta} \sum_{i=1}^{N} \left(\theta^\top \phi(x_i) - y_i\right)^2 + \lambda \|\theta - a\|_2^2, \tag{1}$$

where a is a given (known) vector.

1. Derive the closed-form solution for (1). Provide all steps of the derivation.
   Closed form solution:
   Let $f(\theta) = \sum_{i=1}^{N} (\theta^T \phi x_i - y_i)^2 + \lambda \, ||\theta - a \, ||_2^2$
   $= \, ||\phi\theta - Y \, ||_2^2 + \lambda \, ||\theta - a \, ||_2^2$
   $\frac{\partial f(\theta)}{\partial \theta} = 0$
   $or, \frac{\partial(\, ||\phi\theta - Y \, ||_2^2 + \lambda \, ||\theta - a \, ||_2^2)}{\partial \theta} = 0$

   $or, \frac{\partial((\phi\theta - Y)^T(\phi\theta - Y) + \lambda(\theta - a)^T(\theta - a))}{\partial \theta} = 0$

   $or, \frac{\partial(\phi^T\theta^T - Y^T)(\phi\theta - Y) + \lambda(\theta^T - a^T)(\theta - a))}{\partial \theta} = 0$

   $or, \frac{\partial(\phi^T\theta^T\phi\theta - \phi^T\theta^T Y - Y^T\phi\theta + Y^T Y + \lambda(\theta^T\theta - \theta^T a - a^T\theta + a^T a))}{\partial \theta} = 0$
   $or, 2\phi^T\phi\theta - 2Y^T\phi + 2\lambda\theta - 2\lambda a^T = 0$
   $or, 2\phi(\phi^T\theta - Y^T) + 2\lambda(\theta - a) = 0$
   $or, 2\{\phi(\phi^T\theta - Y^T) + \lambda(\theta - a)\} = 0$
   $or, \phi\phi^T\theta - \phi^T Y + \lambda\theta - \lambda a = 0$
   (Adding an identity matrix I since λ is a scalar)
   $or, \phi\phi^T\theta - \phi^T Y + \lambda I\theta - \lambda a = 0$
   $or, (\phi^T\phi + \lambda I)\theta = \phi^T Y + \lambda a$
   $\boldsymbol{\theta^* = (\phi^T\phi + \lambda I)^{-1}(\phi^T Y + \lambda a)}$

2. What is the solution $\hat{\theta}$ when $\lambda \to \infty$?
   Solution
   In $eqn1$, the first term $\sum_{i=1}^{N} \left(\theta^\top \phi(x_i) - y_i\right)^2$ tries to fit the data and the second term $\lambda\|\theta - a\|_2^2$ penalizes how far θ is from a. λ here becomes the weight on the penalty and a large λ prioritizes making the predicted value closer to a. When $\lambda \to \infty$, the second term of equation 1 dominates (minimizing $\lambda\|\theta - a\|_2^2$).

   We have:
   $\boldsymbol{\hat{\theta} = (\phi^T\phi + \lambda I)^{-1}(\phi^T Y + \lambda a)}$
   $or, \hat{\theta} = (\lambda(\frac{1}{\lambda}\phi^T\phi + I))^{-1}(\phi^T Y + \lambda a)$

   $or, \hat{\theta} = (\lambda(I + \frac{1}{\lambda}\phi^T\phi))^{-1}\lambda(a + \frac{1}{\lambda}\phi^T Y)$

   $or, \hat{\theta} = \lambda^{-1}\lambda(I + \frac{1}{\lambda}\phi^T\phi)^{-1}(a + \frac{1}{\lambda}\phi^T Y)$

$or, \hat{\theta} = (I + \frac{1}{\lambda}\phi^T\phi)^{-1}(a + \frac{1}{\lambda}\phi^T Y)$

As $\lambda \to \infty$,
$=> \frac{1}{\lambda}\phi^T\phi \to 0$, so $I + \frac{1}{\lambda}\phi^T\phi \to I$
$=> \frac{1}{\lambda}\phi^T Y \to 0$, so $a + \frac{1}{\lambda}\phi^T Y \to a$
So, the final product becomes $\boldsymbol{I} \times \boldsymbol{a} = \boldsymbol{a}$.

7. In this assignment, you will write Python codes to implement Linear Regression in two ways: i) closed-form solution, and ii) gradient descent. You are provided with two CSV files, `q7-train.csv` and `q7-test.csv`, each containing columns `x` and `y`. You can download the files via this link.

1. Load the `q7-train.csv` and `q7-test.csv` files. Extract the feature x and the target y. Plot the data points (x_i, y_i) using `plt.scatter`. Plot the training points in one color, and plot the testing points in another color.

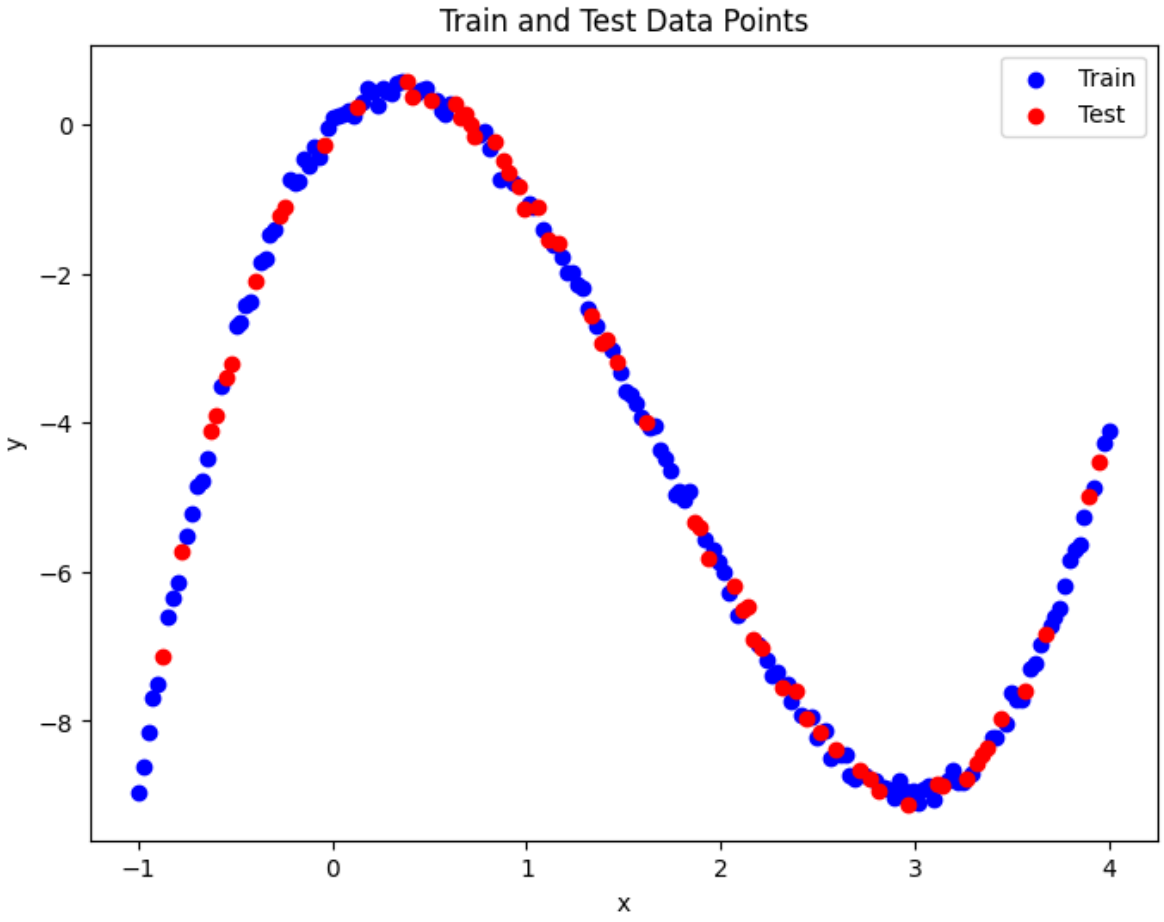

Figure 1: Plotting training and testing points.

2. Assume a model of the form:

$$y = \theta_0 + \theta_1 x.$$

(a) **Closed-Form Solution:** Use the closed-form solution to find the optimal θ_0 and θ_1. First, construct the matrix

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix}, \quad \text{and the target} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

from the training set. Then solve

$$\theta = (X^T X)^{-1} X^T y, \quad \text{where} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}.$$

Finally, calculate the Mean Squared Error (MSE) on the training set and the testing set. Specifically, define

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - (\theta_0 + \theta_1 x_i) \right)^2.$$

Report the values of θ_0 and θ_1 and the MSE on the training set and test set.

2.a CLosed Form Solution Stats

Based on the closed-form solution, the optimal values are:

$\boldsymbol{\theta_0}$ = -2.4461 (rounded to 4 decimal places)

$\boldsymbol{\theta_1}$ = -1.3619 (rounded to 4 decimal places)

MSE on training set: 6.9620 (rounded to 4 decimal places)

MSE on test set: 5.8611 (rounded to 4 decimal places)

Figure 2: Results of the closed form solution

(b) **Gradient Descent:** Implement gradient descent to find the optimal θ_0 and θ_1. The loss function is given by

$$l(\theta) = \|X\theta - y\|_2^2.$$

Using matrix calculus, the gradient of $l(\theta)$ with respect to θ is:

$$\nabla_\theta l(\theta) = 2X^T(X\theta - y).$$

Thus, each gradient descent iteration updates θ as

$$\theta \leftarrow \theta - \eta X^T(X\theta - y),$$

where η is the learning rate.

Initialize θ_0 and θ_1 (for example, to zero or small random values), choose a learning rate η, and then iteratively update the parameters according to the update rule above. Continue until convergence, which you can detect by limiting the number of iterations or checking that changes in θ_0, θ_1 (or the loss value) fall below a small threshold.

After convergence, report the final values of θ_0 and θ_1. Also compute the MSE on both the training set and the test set.

2.b Gradient Descent Analysis

Initial run gave overflow error and NaN values for theta parameters, which suggested that the gradient was diverging due to large learning rate. Decreasing the learning rate helped curb that issue. Based on the gradient descent solution, the optimal values are:

$\boldsymbol{\theta_0}$ = -2.4461 (rounded to 4 decimal places)

$\boldsymbol{\theta_1}$ = -1.3619 (rounded to 4 decimal places)

MSE on training set: 6.9620 (rounded to 4 decimal places)

MSE on test set: 5.8611 (rounded to 4 decimal places)

Figure 3: Results of the gradient descent solution where convergence happened on 93 iterations for the stated optimal parameter values

(c) **Compare and Discuss:**

- **Parameter Comparison:** Compare θ_0 and θ_1 from the closed-form solution (part (a)) with the ones from gradient descent (part (b)). Are the values close to each other? You could compute the absolute or relative differences. If they differ noticeably, discuss possible reasons (e.g., choice of learning rate or number of iterations).

```
In [17]:
# theta_cfand theta_gd are 2x1 arrays
def compare_values(theta_cf, theta_gd, mse_train_cf, mse_train_gd, mse_test_cf, mse_test_gd):
    diff = theta_gd.flatten() - theta_cf.flatten()
    print("theta difference:", diff)
    print("L2 norm difference:", np.linalg.norm(diff))
    print("relative difference:", np.linalg.norm(diff) / (np.linalg.norm(theta_cf) + 1e-12))

    # MSE differences
    mse_diff_train = mse_train_gd - mse_train_cf
    mse_diff_test  = mse_test_gd  - mse_test_cf
    print("MSE diff (train):", mse_diff_train)
    print("MSE diff (test):", mse_diff_test)

In [18]:
compare_values(theta_cf, theta_gd, mse_train_cf, mse_train_gd, mse_test_cf, mse_test_gd)

theta difference: [ 7.23009637e-06 -2.69443028e-06]
L2 norm difference: 7.715843964842476e-06
relative difference: 2.7559525909112712e-06
```

Figure 4: The difference in between the gradient descent vs the closed solution param values are $7.27.23009637e-06$ for θ_0 and $-2.69443028e-06$ for θ_1, which explains the identical values when rounded to the nearest 4 digits. As the GD converged in 93 iterations and produced similar param values, the learning rate of $1e-3$ is considered ideal.

- **Error Comparison:** Compare the training and testing MSE values from both methods. Ideally, they should be very similar if gradient descent converged successfully and everything is implemented correctly.

```
MSE diff (train): 2.6749269466108672e-11
MSE diff (test): -4.855979693019208e-06
```

Figure 5: The difference in MSE values for both training and testing data between the GD and closed form solution is relatively small. These similar MSE values suggest that the GD converged successfully.

- **Plotting:** On your scatter plot of training and testing data, plot the line

$$y = \theta_0 + \theta_1 x$$

for each method (i.e., closed-form and gradient descent) to confirm visually whether they match. Label each line in the legend so that you can see if there is any notable difference between the two solutions.

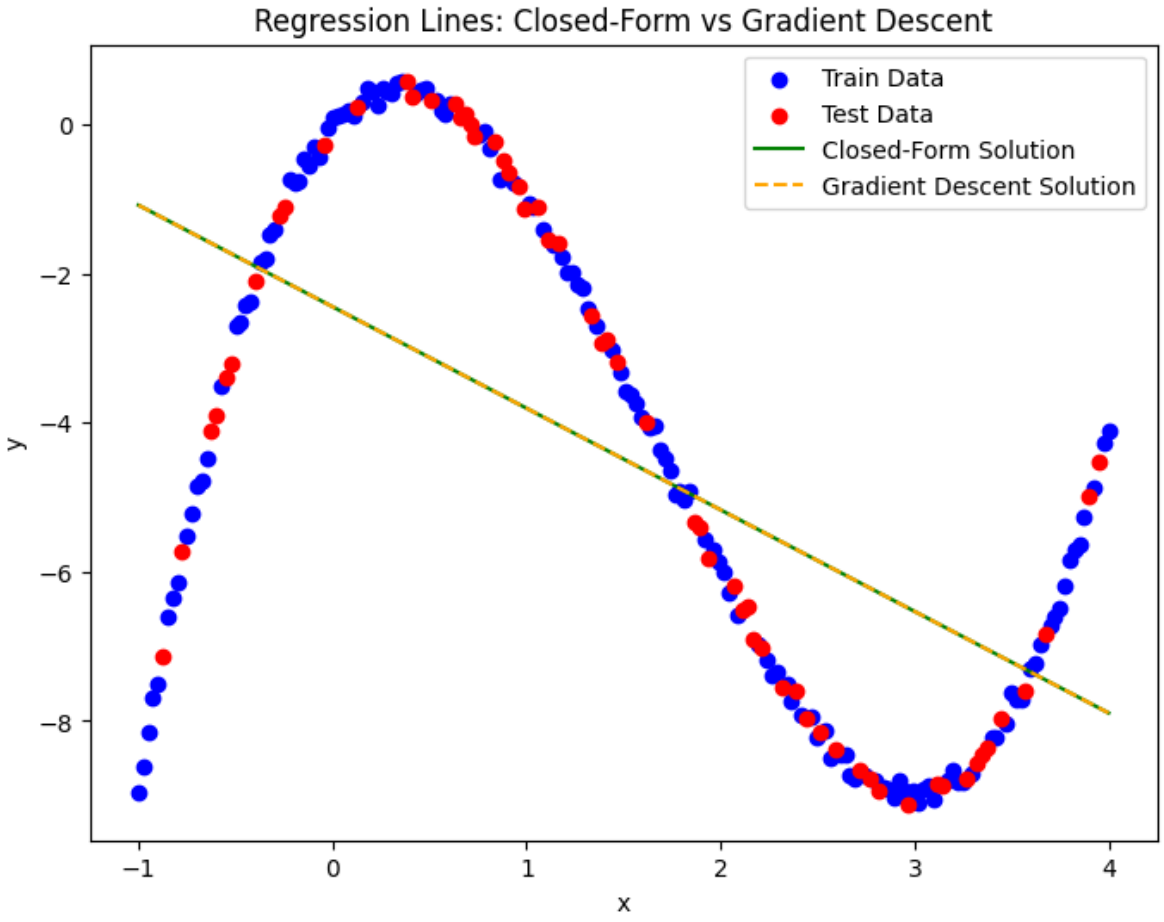

Figure 6: The equation for the line for both solutions match where the orange represents the gradient descent and the green represents the closed form.

3. Now assume a quadratic model of the form:

$$y = \theta_0 + \theta_1 x + \theta_2 x^2.$$

Repeat steps (a)–(c) from Part 2, but note the changes: construct the matrix

$$X = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 \end{bmatrix}.$$

(a) Use the closed-form solution to find the values of θ_0, θ_1, θ_2. Report their values and the MSE on the training set and test set.

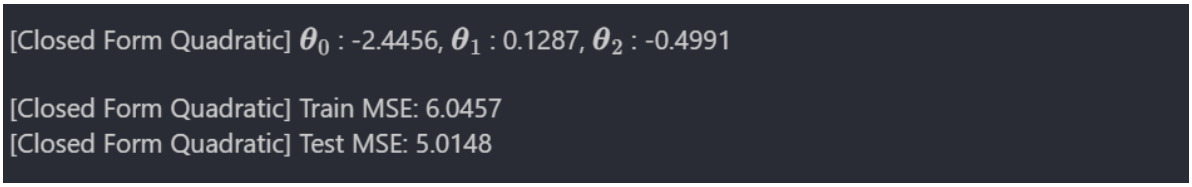

Figure 7: Results of closed form solution for quadratic model (rounded to 4 digits).

(b) Use gradient descent to find the values of θ_0, θ_1, θ_2. Report their values and the MSE on the training set and test set.

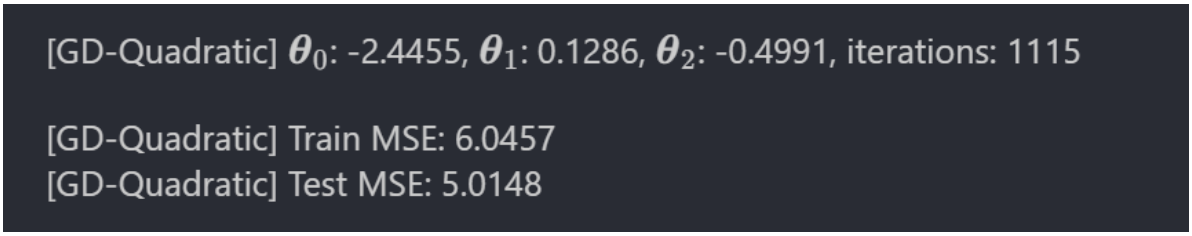

Figure 8: Results of gradient descent solution for quadratic model (rounded to 4 digits). Converged with learning rate 1e-4 within 1115 iterations

(c) Compare the values of θ, training/testing MSEs, and plot the curves $y = \theta_0 + \theta_1 x + \theta_2 x^2$ obtained from the two methods (closed-form vs. gradient descent).

```
theta difference: [ 4.99716630e-05 -1.06819021e-04 2.77742143e-05]
L2 norm difference: 0.00012115641668485732
relative difference: 4.8476599776236206e-05

MSE diff (train): 4.325894309431533e-09
MSE diff (test): -2.7716230618146653e-05
```

Figure 9: Differences between gradient descent - closed form values. These smaller differences in θ values and MSEs across the 2 solutions reflect that they are similar, which is reinforced by the plot below.

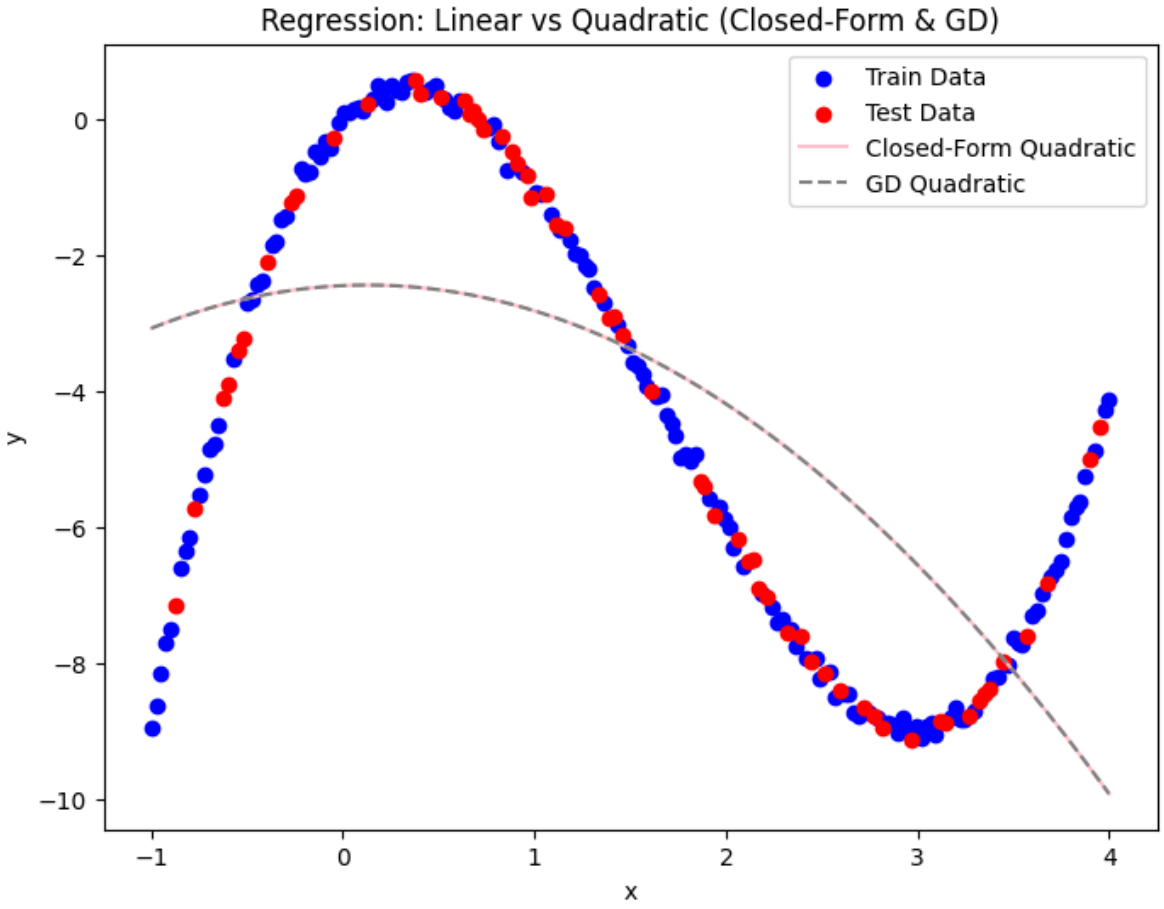

Figure 10: The equation for the line for both solutions match where the grey represents the gradient descent and pink represents the closed form.

4. Now assume a cubic model of the form:

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3.$$

Repeat steps (a)–(c) from Part 2, but note the changes: construct the matrix

$$X = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 & x_N^3 \end{bmatrix}$$

(a) Use the closed-form solution to find the values of θ_0, θ_1, θ_2, θ_3. Report these values and the MSE on the training set and test set.

[Closed-form Cubic] $\boldsymbol{\theta}_0$: -0.0181, $\boldsymbol{\theta}_1$: 2.9846, $\boldsymbol{\theta}_2$: -4.9782, $\boldsymbol{\theta}_3$: 0.9954
[Closed-form Cubic] Train MSE: 0.0088
[Closed-form Cubic] Test MSE: 0.0080

Figure 11: Results of closed form solution for cubic model (rounded to 4 digits).

(b) Use gradient descent to find the values of θ_0, θ_1, θ_2, θ_3. Report these values and the MSE on the training set and test set.

[GD-Cubic] $\boldsymbol{\theta}_0$: -0.0628, $\boldsymbol{\theta}_1$: 2.8987, $\boldsymbol{\theta}_2$: -4.8910, $\boldsymbol{\theta}_3$: 0.9788, iterations: 10000
[GD-Cubic] Train MSE: 0.0111
[GD-Cubic] Test MSE: 0.0115

Figure 12: Results of gradient descent solution for cubic model (rounded to 4 digits). Converged with learning rate 1e-5 within 1000 iterations

(c) Compare the values of θ, the training/testing MSEs, and plot the curves $y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$ obtained from the two methods (closed-form vs. gradient descent).

```
theta difference: [-0.04473305 -0.08591811 0.08722987 -0.01663108]
L2 norm difference: 0.13141008110509933
relative difference: 0.022314151103569334

MSE diff (train): 0.002288422615401846
MSE diff (test): 0.0034865284443234574
```

Figure 13: Differences between gradient descent - closed form values. These smaller differences in θ values and MSEs across the 2 solutions reflect that they are similar, which is reinforced by the plot below.

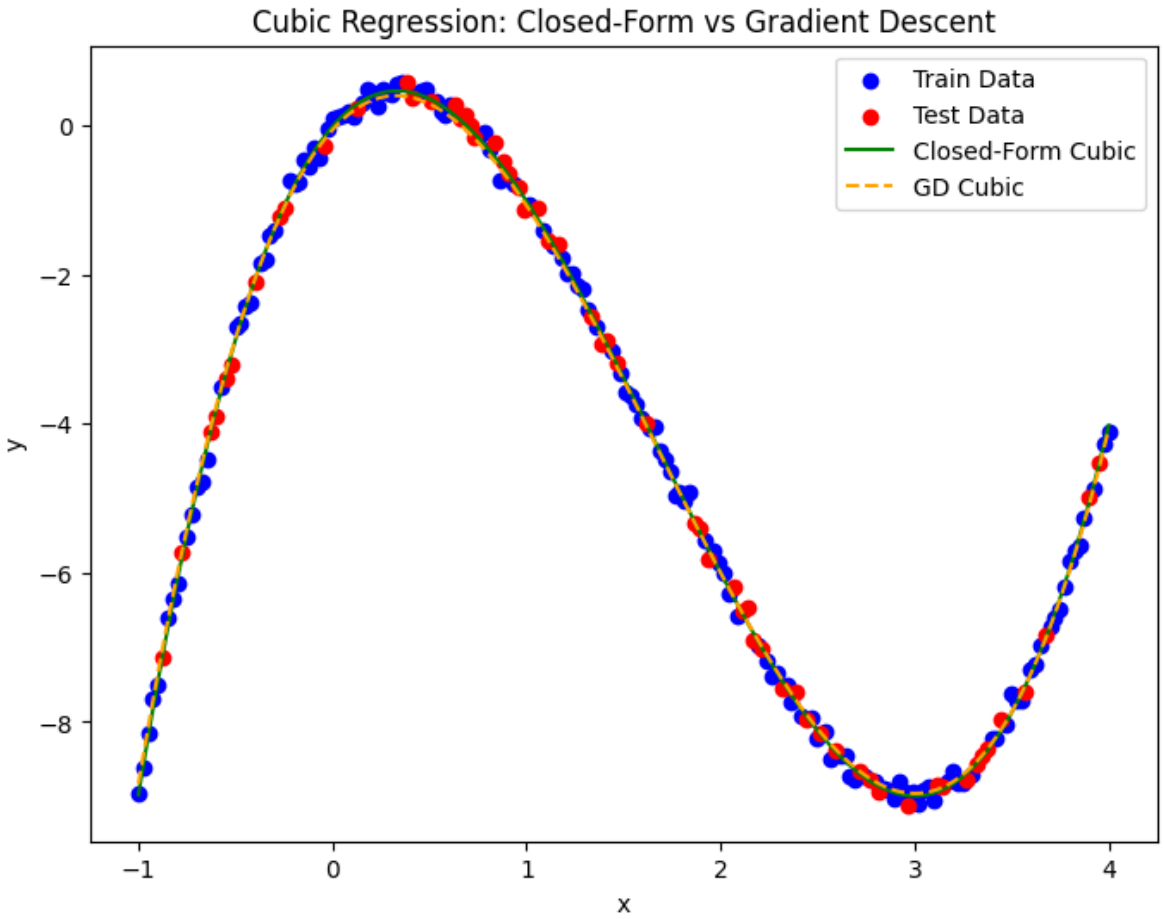

Figure 14: The equation for the line for both solutions match where the orange represents the gradient descent and the green represents the closed form.

8. In this assignment, you will use Linear Regression to analyze the correlation among wine quality and several key factors of wines, such as pH and density.

1. Download the `q8-train.csv`, `q8-val.csv` and `q8-test.csv` from this link. In the files, "quality" column will be the target (Y) to predict and other columns are input features (X).

8.1 Load Data

```python
# Load q8-train.csv, q8-val.csv, q8-test.csv and extract features/target
import pandas as pd

# Load data
train_data = pd.read_csv('q8-train.csv')
val_data = pd.read_csv('q8-val.csv')
test_data = pd.read_csv('q8-test.csv')

target_column = 'quality'

# Split features and target
X_train = train_data.drop(columns=[target_column])
y_train = train_data[target_column]

X_val = val_data.drop(columns=[target_column])
y_val = val_data[target_column]

X_test = test_data.drop(columns=[target_column])
y_test = test_data[target_column]

print('Train shape:', X_train.shape, y_train.shape)
print('Validation shape:', X_val.shape, y_val.shape)
print('Test shape:', X_test.shape, y_test.shape)
```

```
Train shape: (959, 12) (959,)
Validation shape: (400, 12) (400,)
Test shape: (240, 12) (240,)
```

Figure 15: Loading data from csv files

2. You will write a python function to train linear regressor using training data in `q8-train.csv`. To build a good model, you need to first compute quadratic features from input features (You are also encouraged to try linear/cubic features). This can be done with scipy.

8.2 Training Linear Regressor

```python
def train_linear_regression(X_train, y_train, X_val, y_val, degree):
    # Create polynomial features
    poly = PolynomialFeatures(degree=degree, include_bias=False)

    # Transform features
    X_train_poly = poly.fit_transform(X_train)
    X_val_poly = poly.transform(X_val)

    # Train linear regression model
    model = LinearRegression()
    model.fit(X_train_poly, y_train)

    # Predict on train, val, test
    y_pred_train = model.predict(X_train_poly)
    y_pred_val = model.predict(X_val_poly)

    # Compute MSE
    mse_train = mean_squared_error(y_train, y_pred_train)
    mse_val = mean_squared_error(y_val, y_pred_val)

    print(f"Train MSE: {mse_train:.4f}")
    print(f"Validation MSE: {mse_val:.4f}")

    return X_train_poly, X_val_poly, y_pred_train, y_pred_val
```

Figure 16: Python function to train linear regressor training data on features. Linear, quadratic and cubic features called in the notebook file.

3. To evaluate your model, you should compute mean square errors of your models on training data and validation data (validation data is in val.csv). Please refer to the definition of mean square error given in Problem (7.a).

	Model	Train MSE	Validation MSE
0	Linear	0.401759	0.406257
1	Quadratic	0.330480	0.412984
2	Cubic	0.173259	3.096138

Figure 17: MSE on training and validation data across linear, quadratic, cubic feature models. For quadratic, we have an error of 0.4129 on validation data, higher than that on training data.

4. You will notice the low errors on training data and higher errors on validation data. This issue is called over-fitting. To put it simply, models learn certain feature-target correlations that only exist in training data and do not generalize to test data. This can be mitigated by L2 regularization, which prevents models from over-relying on certain features. You can implement L2 regularization by replacing `scipy LinearRegression` with `Ridge`. You should try different value of alpha and observe the changes in validation error.

	alpha	Train MSE	Validation MSE
0	0.001	0.340193	0.404614
1	0.100	0.346472	0.402224
2	1.000	0.352262	0.400351
3	8.000	0.359463	0.393523
4	10.000	0.360764	0.392847
5	25.000	0.367452	0.391268
6	50.000	0.373349	0.391129
7	75.000	0.376922	0.391197
8	100.000	0.379497	0.391229
9	250.000	0.388056	0.391016
10	260.000	0.388440	0.391000
11	275.000	0.388994	0.390978
12	280.000	0.389172	0.390971
13	300.000	0.389861	0.390945
14	350.000	0.391423	0.390896
15	500.000	0.395168	0.390880
16	750.000	0.399612	0.391122
17	1000.000	0.402808	0.391498
18	1500.000	0.407194	0.392293

Figure 18: Trying different alpha values to see effect on validation error

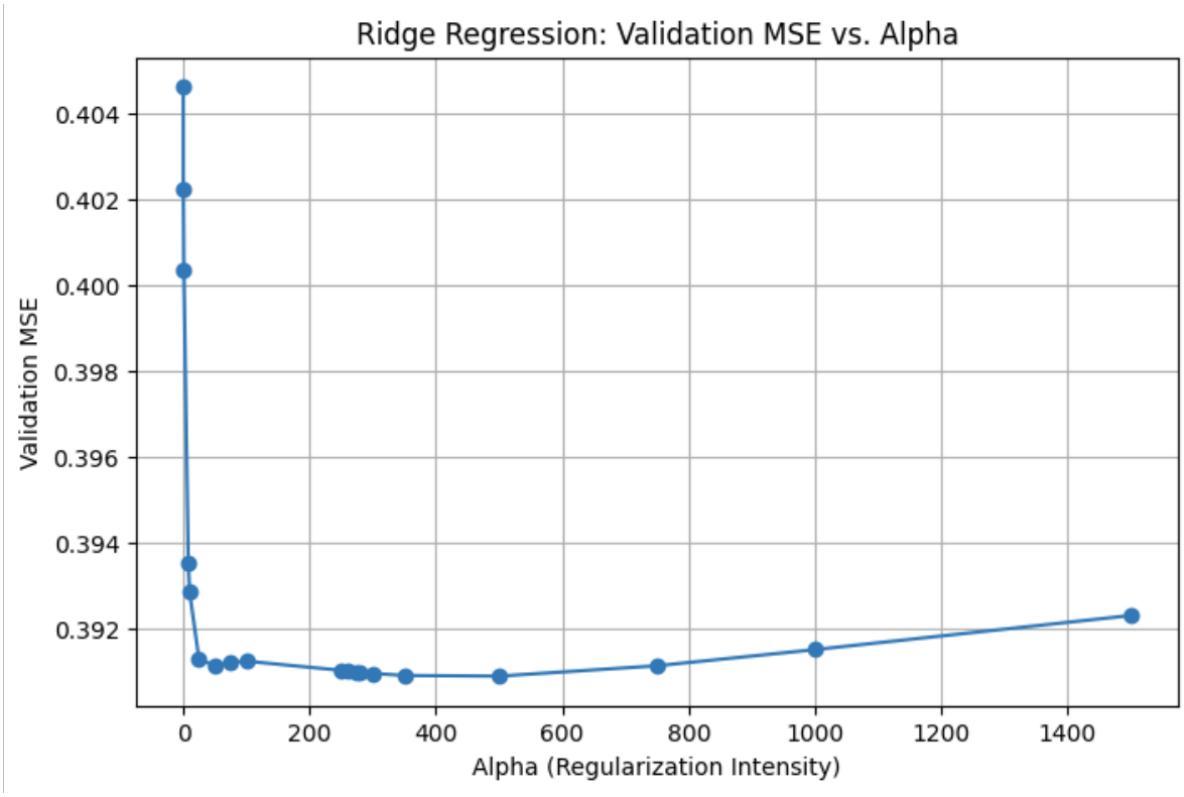

Figure 19: As we see, the validation error drops from 0.40 to 0.392 as alpha is increased from 0 to 200. After that, it stays roughly the same for an interval and starts increasing as the alpha value gets greater than 500. An optimal alpha would be somewhere between 10 and 500. We'll take the value 50 (0.391129) for our testing.

5. Based on validation errors, choose the optimal L2 regularization intensity. Then report the performance of your best model on test data in `q8-test.csv`.

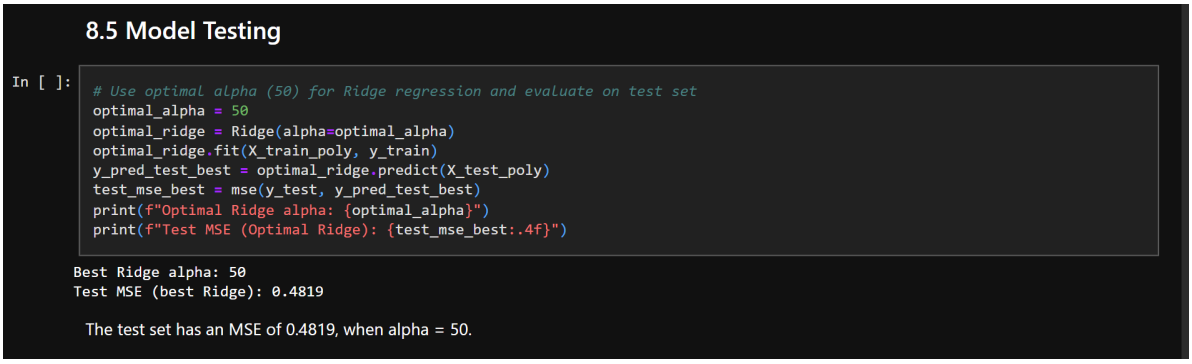

Figure 20: Testing optimal alpha value = 50 with test data shows an MSE of 0.4819.