CIS 419/519: Applied Machine Learning	Fall 2024
Homework 1	
Handed Out:	*Due: 7:59 pm October 2*

Name: Richard Zhang

PennKey: zhank24

PennID: 19331985

1 Declaration

- **Person(s) discussed with:** *N/A*
- **Affiliation to the course: student, TA, prof etc.** *student*
- **Which question(s) in coding / written HW did you discuss? *N/A***
- **Briefly explain what was discussed.** *N/A*

2 Question 1

1. Bias: Stay the same
   Variance: Increases
2. Bias: Increases
   Variance: Decreases
3. Bias: Increases
   Variance: Decreases
4. Bias: Stay the same
   Variance: Stay the same
5. Bias: Stay the same
   Variance: Stay the same
6. n: Increase it to tackle over-fitting
   λ: Increase it to tackle over-fitting
   d: Decrease it to tackle over-fitting
   c: No effect
   α: No effect

3 Question 2

1. We know the ℓ_1 regularization term is $\lambda \sum_j |\beta_j|$, so we take the derivative of it with respect to β_j:

$$\frac{\partial}{\partial \beta_j}(\lambda|\beta_j|) = \lambda \cdot \text{signum}(\beta_j)$$

signum is a function defined as follows:

$$\text{signum}(\beta_j) = \begin{cases} +1 & \text{if } \beta_j > 0 \\ -1 & \text{if } \beta_j < 0 \end{cases}$$

2. In gradient descent, the update rule for β_j is:

$$\beta_j new = \beta_j old - \alpha \left(\frac{\partial}{\partial \beta_j} \text{Loss} + \frac{\partial}{\partial \beta_j} \text{Regularization Term} \right)$$

Plugging in the gradients of the linear regression loss term and the ℓ_1 term, we get:

$$\beta_j new = \beta_j old - \alpha(\nabla_{\beta_j}(\sum_{i=1}^{N}(y_i - \beta^\top x_i)^2) + \lambda \cdot \text{signum}(\beta_j old))$$

Gradient of the Loss Term: When λ is sufficiently large, the ℓ_1 term dominates the update, so the impact of the linear regression loss term becomes trivial. For non-important features j, the gradient of ∇_{β_j} is too small to be useful because the changes in β_j have little impact. Therefore, the gradient of the loss term wouldn't contribute to this behavior.

Gradient of the ℓ_1 Term: When λ is sufficiently large, the ℓ_1 term dominates the update and makes the gradient of the loss term have trivial influence on sparsity. For non-important features j, the gradient of ℓ_1 can either be λ or $-\lambda$. This implies that even if β_j is already close to zero, it will still be reduced by a fixed amount in each step and eventually pushed exactly to zero. For example, if $\beta_j = 0.0001$, which is pretty small, then in each future step, it will still be decreased by λ until it becomes zero. Thus, the sparsity is achieved, and we can decrease the model complexity and simplify the model. In other words, the gradient of the regularization term $\lambda \cdot \text{signum}(\beta_j)$ is constant (either λ or $-\lambda$). The ℓ_1 term adds a constant shrinkage force ($\lambda \cdot \text{signum}(\beta_j)$) to the model, pushing β_j to exactly zero when β_j is very small. This causes β_j to be exactly zero.

3. We know $l2$ regularization is:

$$\lambda \sum_j \beta_j^2$$

Its gradient with respect to β_j is:

$$\frac{\partial}{\partial \beta_j}\left(\lambda \sum_j \beta_j^2\right) = 2\lambda\beta_j$$

ℓ_2 Doesn't encourage sparsity. As the above equation shows, the gradient of ℓ_2 regularization is proportional to β_j. This means that the penalty on β_j decreases as β_j becomes smaller, and ℓ_2 doesn't have the constant shrinkage force as ℓ_1. Small β_j would receive small shrinkage, so they are difficult to become exactly zero. Therefore, ℓ_2 doesn't encourage sparsity. It shrinks β_j toward zero but does not push them to exactly zero.

4 Question 3

1. Derivative with respect to w_0:

$$\frac{\partial J(\mathbf{w})}{\partial w_0} = \frac{1}{n}\sum_{i=1}^{n} -2(y_i - w_0 - w_1 x_i) = \frac{-2}{n}\sum_{i=1}^{n}(y_i - w_0 - w_1 x_i)$$

Derivative with respect to w_1:

$$\frac{\partial J(\mathbf{w})}{\partial w_1} = \frac{1}{n}\sum_{i=1}^{n} -2x_i(y_i - w_0 - w_1 x_i) = \frac{-2}{n}\sum_{i=1}^{n} x_i(y_i - w_0 - w_1 x_i)$$

2. We know $w^* = [w_0^*, w_1^*]^T$ is the least squares solution. Thus, at the minimum point, these derivatives are zero:

$$\left.\frac{\partial J}{\partial w_0}\right|_{w_0=w_0^*,\ w_1=w_1^*} = 0 \quad \text{and} \quad \left.\frac{\partial J}{\partial w_1}\right|_{w_0=w_0^*,\ w_1=w_1^*} = 0$$

Neglecting the constants in the derivative, we get:

$$\sum_{i=1}^{n}(y_i - w_0^* - w_1^* x_i) = 0$$

$$\sum_{i=1}^{n} x_i\,(y_i - w_0^* - w_1^* x_i) = 0$$

Now, let's expand the equation we want to prove:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - w_0^* - w_1^* x_i)(x_i - \bar{x}) = \frac{1}{n}[\sum_{i=1}^{n}(y_i - w_0^* - w_1^* x_i)x_i - \bar{x}\sum_{i=1}^{n}(y_i - w_0^* - w_1^* x_i)]$$

Since we know $\sum_{i=1}^{n} x_i\,(y_i - w_0^* - w_1^* x_i) = 0$ and $\sum_{i=1}^{n}(y_i - w_0^* - w_1^* x_i) = 0$, we can rewrite the equation as follows:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - w_0^* - w_1^* x_i)(x_i - \bar{x}) = \frac{1}{n}\,(0 - \bar{x} \times 0) = 0$$

Therefore,

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - w_0^* - w_1^* x_i)(x_i - \bar{x}) = 0$$

3. Linear regression is not guaranteed to have a unique solution for any dataset.

   In linear regression, the function we are minimizing ($J(\mathbf{w})$) is convex because of its quadratic nature. Because the function is quadratic, it has no local optima that are not global optima. This characteristic ensures that for any solution we find, if we take the derivatives and set them to zero, we can find the global optimum.

   On the contrary, suppose all the x values are the same (i.e., no variation in x), the objective function $J(\mathbf{w})$ becomes flat in the w_1 direction. Under this scenario, the quadratic function loses its convexity, implying the solution is not unique. This happens because the parameter w_1 can take any value without affecting the value of the objective function, leading to multiple optimal solutions (or, no optimal solution at all).

   Therefore, we can conclude that:

   Unique solution exists when the input feature x_i has variability (not all values are the same).

   Unique solution doesn't exist when x_i has no variability (i.e., all of them are identical, then $\sum(x_i - \overline{x})^2 = 0$. In this case, the solution wouldn't be unique).

   Therefore, linear regression is not guaranteed to have a unique solution for any dataset.

5 Question 4

1. With initial weights $\mathbf{w} = [0, 0]^T$, we can calculate the predicted values:

$$\hat{y}_1 = \mathbf{w}^T\mathbf{x}_1 = 0, \quad \hat{y}_2 = \mathbf{w}^T\mathbf{x}_2 = 0$$

   The errors are:

$$e_1 = y_1 - \hat{y}_1 = 0 - 0 = 0, \quad e_2 = y_2 - \hat{y}_2 = 1 - 0 = 1$$

   The loss function is:

$$L = \frac{1}{2}(e_1^2 + e_2^2) + \lambda\|\mathbf{w}\|^2 = \frac{1}{2}(0^2 + 1^2) + 0 = 0.5$$

   Thus, the loss is 0.5 at the beginning.

2. We know the gradient equation is:

$$\nabla L = -\frac{2}{N}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)\mathbf{x}_i + 2\lambda\mathbf{w}$$

We have already known the initial errors e_1 and e_2, so we can rewrite the gradient equation using the following from:

$$\nabla L = -\frac{2}{N}\sum_{i=1}^{N} e_i \mathbf{x}_i + 2\lambda \mathbf{w}$$

Compute the gradient at $\mathbf{w}^{(0)} = [0, 0]^T$:

$$\nabla L = -1(e_1\mathbf{x}_1 + e_2\mathbf{x}_2) + 2\lambda\mathbf{w} = -(0 \cdot \mathbf{x}_1 + 1 \cdot \mathbf{x}_2) + 0 = -\mathbf{x}_2 = [1, 1]^T$$

$$\mathbf{w}^{(1)} = \mathbf{w}^{(0)} - \alpha\nabla L = [0, 0]^T - 1 \cdot [1, 1]^T = [-1, -1]^T$$

Then, we can calculate errors with the updated weights:

$$\begin{aligned}
\hat{y}_1 &= \mathbf{w}^{(1)T}\mathbf{x}_1 \\
&= [-1, -1]^T \cdot [1, -1] \\
&= (-1) \times 1 + (-1) \times (-1) \\
&= -1 + 1 \\
&= 0
\end{aligned}$$

Thus, $e_1 = y_1 - \hat{y}_1 = 0 - 0 = 0$

$$\begin{aligned}
\hat{y}_2 &= \mathbf{w}^{(1)T}\mathbf{x}_2 \\
&= [-1, -1]^T \cdot [-1, -1] \\
&= (-1) \times (-1) + (-1) \times (-1) \\
&= 1 + 1 \\
&= 2
\end{aligned}$$

Thus, $e_2 = y_2 - \hat{y}_2 = 1 - 2 = -1$

Compute the gradient at $\mathbf{w}^{(1)} = [-1, -1]^T$:

$$\nabla L = -1(e_1\mathbf{x}_1 + e_2\mathbf{x}_2) + 2\lambda\mathbf{w}^{(1)} = -[-\mathbf{x}_2] + 2 \cdot 1 \cdot [-1, -1]^T = -[1, 1]^T + [-2, -2]^T = [-3, -3]^T$$

$$\mathbf{w}^{(2)} = \mathbf{w}^{(1)} - \alpha\nabla L = [-1, -1]^T - 1 \cdot [-3, -3]^T = [2, 2]^T$$

Then, we can calculate errors with the updated weights:

$$\begin{aligned}
\hat{y}_1 &= \mathbf{w}^{(2)T}\mathbf{x}_1 \\
&= [2, 2]^T \cdot [1, -1] \\
&= -2 + 2 \\
&= 0
\end{aligned}$$

Thus, $e_1 = y_1 - \hat{y}_1 = 0 - 0 = 0$

$$\begin{aligned}\hat{y}_2 &= \mathbf{w}^{(2)T}\mathbf{x}_2 \\ &= [2,2]^T \cdot [-1,-1] \\ &= -4\end{aligned}$$

Thus, $e_2 = y_2 - \hat{y}_2 = 1 - (-4) = 5$

Compute the final loss function value:

$$L = \frac{1}{2}(e_1^2 + e_2^2) + \lambda\|\mathbf{w}^{(2)}\|^2 = \frac{1}{2}(0^2 + 5^2) + 1(2^2 + 2^2) = 12.5 + 8 = 20.5$$

$\mathbf{w} = [2,2]^T$, and the loss function value is 20.5.

3. A linear regression model that implements ℓ_2 norm for regularisation is called ridge regression.

   The matrix form of the loss function is:

$$L(\mathbf{w}) = \frac{1}{N}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda\mathbf{w}^T\mathbf{w}.$$

   Calculating its gradient:

$$\begin{aligned}\nabla_w L(w) &= \nabla_w \frac{1}{N}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + 2\lambda\mathbf{w} \\ &= \nabla_w \frac{1}{N}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) + 2\lambda\mathbf{w} \\ &= \frac{2}{N}\left[\nabla_w(\mathbf{y} - \mathbf{X}\mathbf{w})^\top\right](\mathbf{y} - \mathbf{X}\mathbf{w}) + 2\lambda\mathbf{w} \\ &= -\frac{2}{N}\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + 2\lambda\mathbf{w}\end{aligned}$$

   To find the minimum, we set $\nabla_w L(w) = 0$:

$$\nabla L = -\frac{2}{N}\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + 2\lambda\mathbf{w} = 0$$

$$\left(\frac{2}{N}\mathbf{X}^T\mathbf{X} + 2\lambda\mathbf{I}\right)\mathbf{w} = \frac{2}{N}\mathbf{X}^T\mathbf{y}$$

$$\left(\frac{1}{N}\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)\mathbf{w} = \frac{1}{N}\mathbf{X}^T\mathbf{y}$$

   Always invertible if $\lambda > 0$ (according to the lecture slides), so we have:

$$\mathbf{w} = \left(\frac{1}{N}\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\left(\frac{1}{N}\mathbf{X}^T\mathbf{y}\right)$$

Multiplying both numerator and denominator by N:

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

Thus, the closed-form solution is $\mathbf{w} = \left(\mathbf{X}^T\mathbf{X} + N\lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{y}$.

6 Python Programming Question 1.3

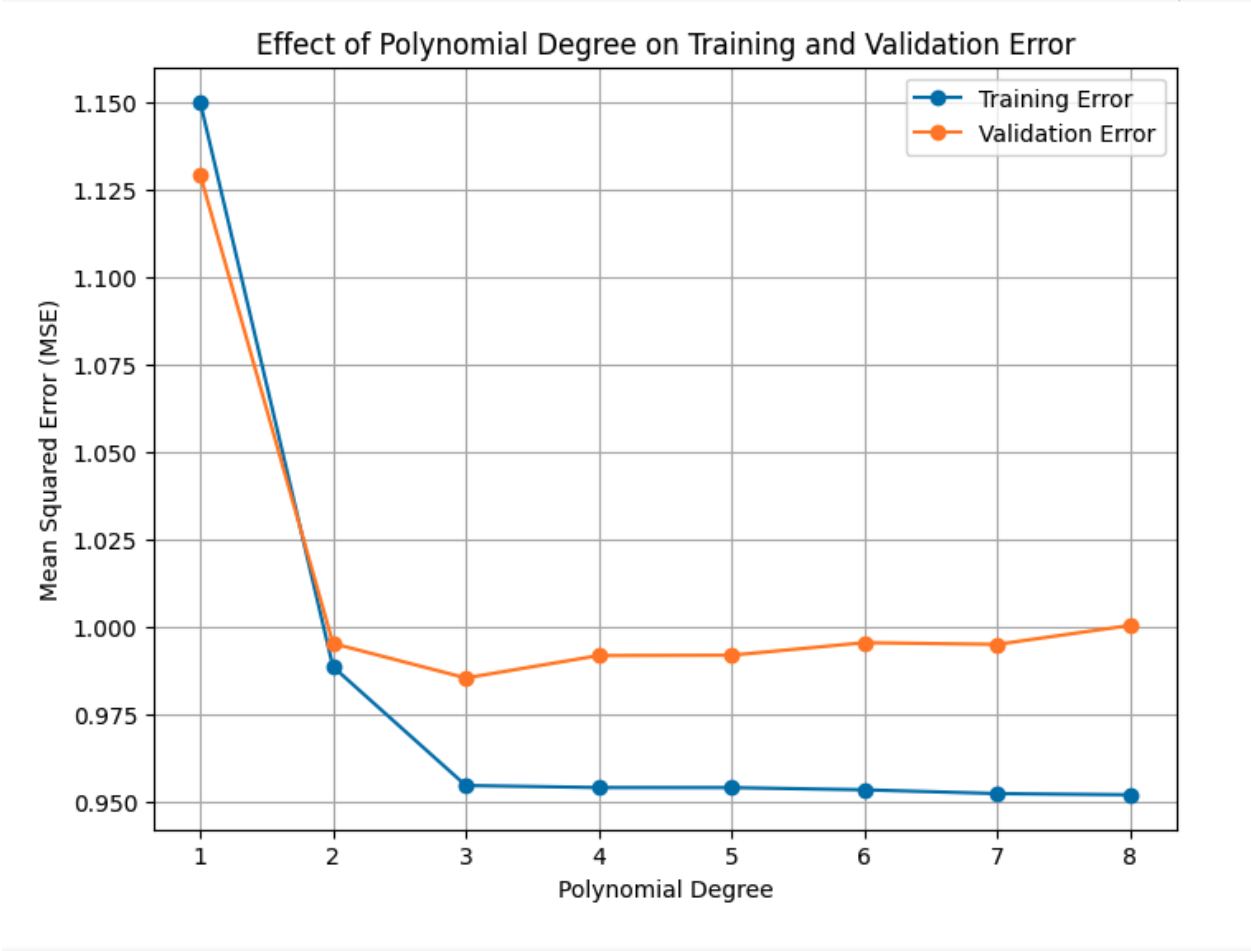

Figure 1: Effect of Polynomial Degree on Training and Validation Error

Trends

Initially, for lower polynomial degrees, both the training error and the validation error drop quickly. The training error and the validation error decrease as the model becomes more complex, implying that the model becomes better when it comes to fitting the underlying pattern of the dataset. During this stage, the model is under-fitted.

From degree 2 to 8, the training error remains low and becomes stable as the polynomial degree increases. Obviously, this means the model can fit the training model pretty well at this stage. Also, this could also means the problem of over-fitting begins to arise.

From degree 3 onwards, the validation error slightly increases or remains relatively stable. This implies over-fitting. Although the model fits the training data pretty well, its performance on unseen data decreases as the degree of the polynomial increases. This can be attributed to the model fits lots of small fluctuations or noises in the training data. Thoes noises make the model less accurate.

Explanation

I think this plot makes sense: When the polynomial degrees is small, the model lacks complexity and cannot capture the underlying patterns in the dataset. This lead to underfitting.

At degree 3, the model complexity seems to match both the training data and the validation data well. Both training and validation errors are low, suggesting this point represents a good balance between bias and variance. The validation error is the lowest at this point.

As polynomial degree continues to increase (from 3 to 8), the model becomes more complex and starts to over-fit the training data. Even though the training error becomes lower, the validation error increases. This means the model considers lots of noises and may not be able to generalize to other data.

7 Python Programming Question 1.4

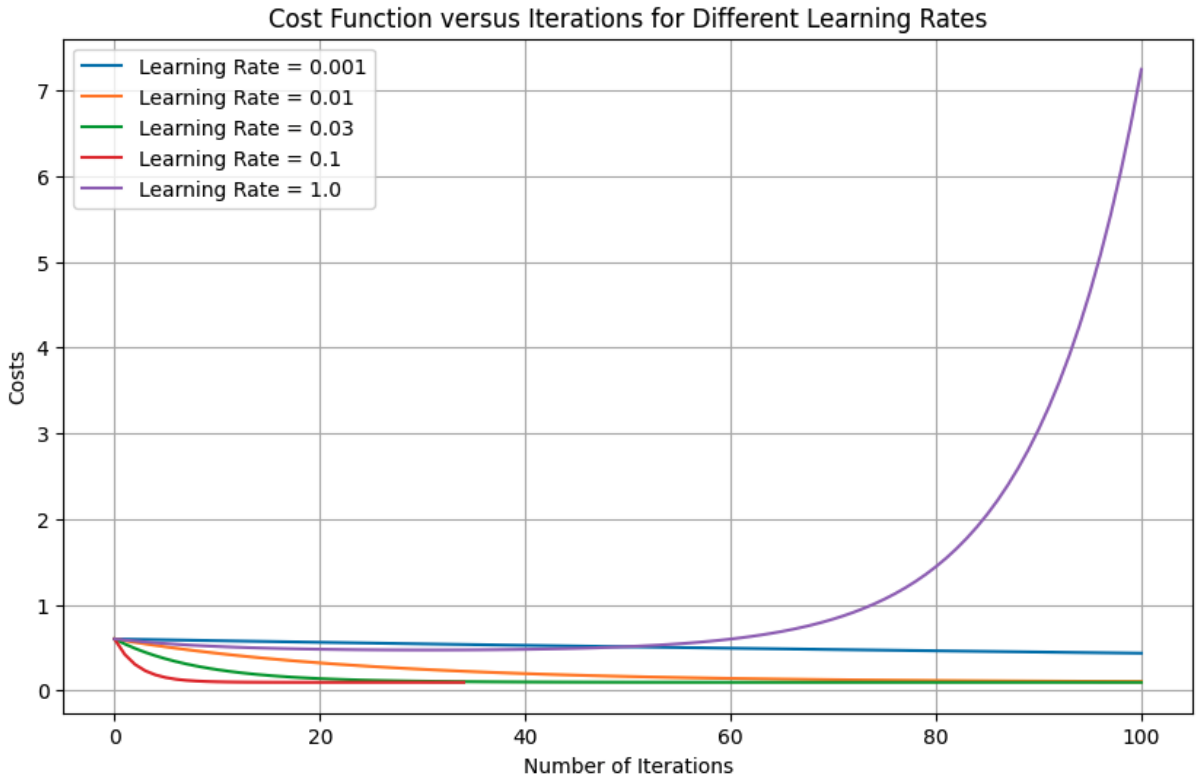

Figure 2: Cost Function versus Iterations for Different Learning Rates

The purple line (1.0) increase exponentially after 60 iterations. This implies the learning rate might be too high, and the model obviously diverges. Thus, we cannot use this.

The blue line (0.001) converges slowly but steadily, but its final cost (100 iterations) is not as good as the orange, red, and green lines. So, we have better choices than the blue line. It would take much more iterations to reach the optimal point (because its learning rate is so small), so we won't consider it.

The red line (0.1) drops very quickly until it reaches to a pretty low cost point. It converge pretty quickly. The overall path is smooth and stable. This is an ideal choice. Some may concern, however, that its convergence speed is a little bit too fast.

The orange (0.01) and green (0.03) lines are smooth and stable. Their cost decreases gradually and eventually stabilizes, suggesting that these rates are neither too slow nor too fast. However, the green line performs faster than the orange line (it reaches a lower cost even earlier).

Thus, the green line (learning rate = 0.03) and the red line (0.1) are the best.