Homework 1 Report

1.

Problem 1

$$y = f_0(x) + \varepsilon \quad \text{—— (1)}$$

$$\hat{y} = \sum_{k=0}^{K} \alpha_K x_i^K \quad \text{—— (2)}$$

$$SSE = \sum_{i=1}^{N} \left(\hat{y}_i - y_i\right)^2$$

$$= \sum_{i=1}^{N} \left(\hat{y}_i - y_i\right)\left(\hat{y}_i - y_i\right)$$

In matrix notation,

$$SSE = (\hat{Y} - Y)^T (\hat{Y} - Y) \quad - \text{(3)}$$

Where

$$\hat{y} = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_N]$$

$$y = [y_1, y_2, \ldots, y_N)$$

But

$$\hat{y} = \sum_{k=0}^{K} \alpha_K x_i^K, \text{ so that in matrix form,}$$

$$\hat{y} = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^K \\ 1 & x_2 & x_2^2 & \cdots & x_2^K \\ \vdots & \vdots & \vdots & & \\ 1 & x_N & x_N^2 & \cdots & x_N^K \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_K \end{pmatrix}$$

$$\hat{y} = XA \quad \text{—} (4)$$

Eqn (3) becomes

$$SSE = (XA - Y)^T (XA - Y)$$

$$= \left((XA)^T - Y^T\right)(XA - Y)$$

$$= A^T X^T X A - 2(XA)^T Y + Y^T Y$$

$$= A^T X^T X A - 2A^T X^T Y + Y^T Y$$

So that the gradient of SSE wrt to A is

$$\nabla_A SSE = \nabla_A \left(A^T X^T X A\right) - 2\nabla_A \left(A^T X^T Y\right) + \nabla_A \left(Y^T Y\right)$$

$$= 2X^T X A - 2X^T Y + 0$$

$$= 2X^T X A - 2X^T Y$$

setting $\nabla_A SSE = 0$

$$2X^T X A - 2X^T Y = 0$$

$$2X^T X A = 2X^T Y$$

$$A = X^T Y / X^T X$$

$$\therefore \quad A = (X^T X)^{-1} X^T Y \quad \text{—} (5)$$

$$\therefore \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_K \end{pmatrix} = (X^T X)^{-1} X^T Y \quad - \; 6)$$

Which gives the optimal coefficients $(\alpha_0, \alpha_1, \cdots \alpha_K)$ when solved.

$$\therefore \hat{Y} = X (X^T X)^{-1} X^T Y$$

By Gradient Descent,

$$\nabla_A SSE = -2X^T (Y - XA)$$

$$A\_ = A - \cancel{\beta} \nabla_A SSE$$

$$\begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_K \end{pmatrix}_{new} = \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_K \end{pmatrix} - \beta \left(-2X^T (Y - XA) \right)$$

$$\begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_K \end{pmatrix}_{new} = \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_K \end{pmatrix}_{old} - \beta \left(2X^T (XA - Y) \right)$$

$$\bar{\alpha}_{new} = \bar{\alpha}_{old} - \beta \left(2X^T (X (X^T X)^{-1} X^T Y - Y) \right)$$

Observations for 1f.

For **K = 1,**

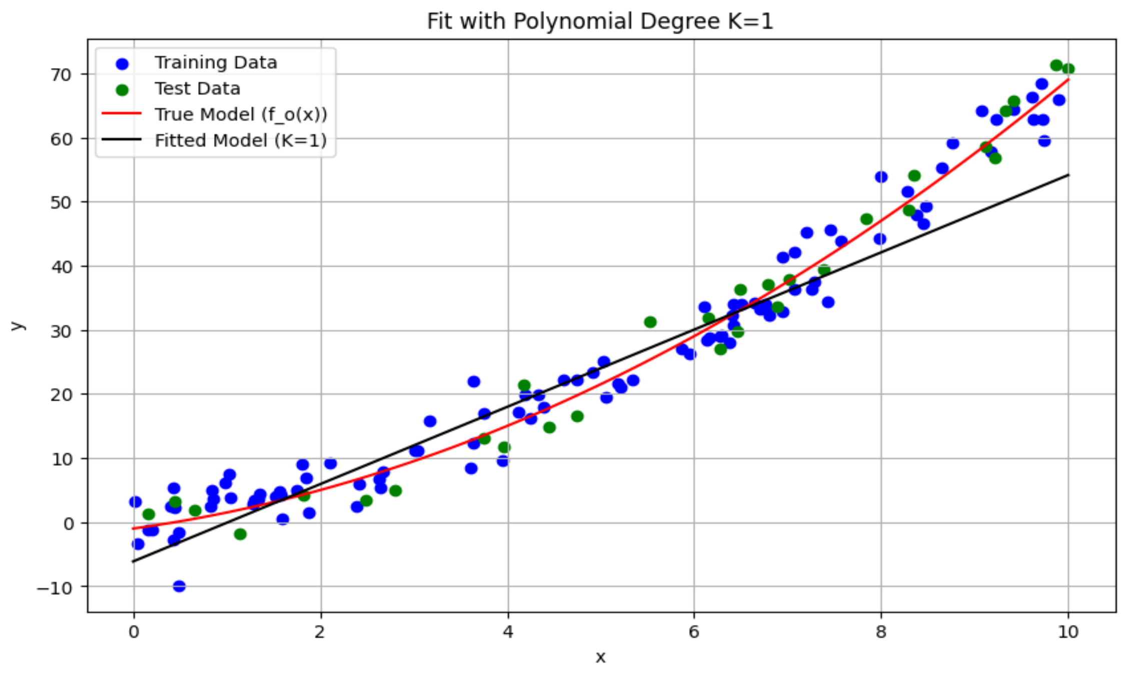

Figure 1: Plot of Test and Train Data (K = 1)

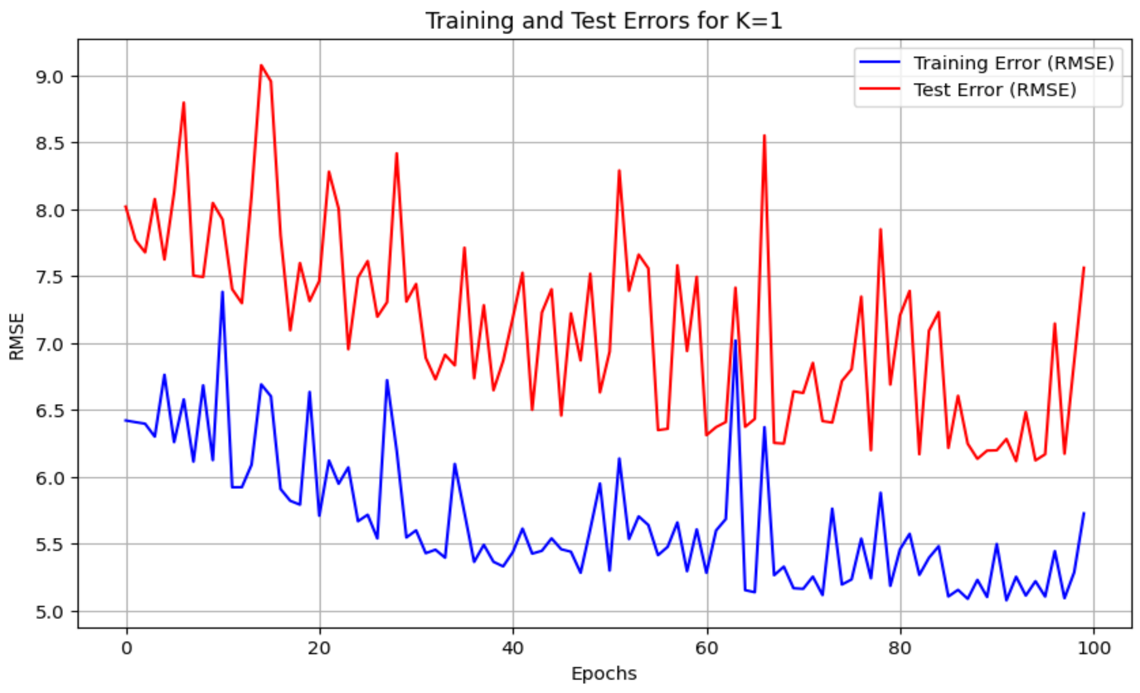

Figure 2: Plot of Test and Testing error (K=1)

The model is underfitting the data. This is evident from the consistently higher and more volatile test error compared to the training error. The test error, fluctuating between 7.0 and 9.0 RMSE, suggests that the model is unable to generalize well to unseen data. The lack of a significant improvement in the test error indicates that the model is too simple to capture the underlying patterns in the data. The linear model is not flexible enough to capture the complexity of the quadratic data, so it struggles to fit both the training and test datasets.

For **K = 5**,

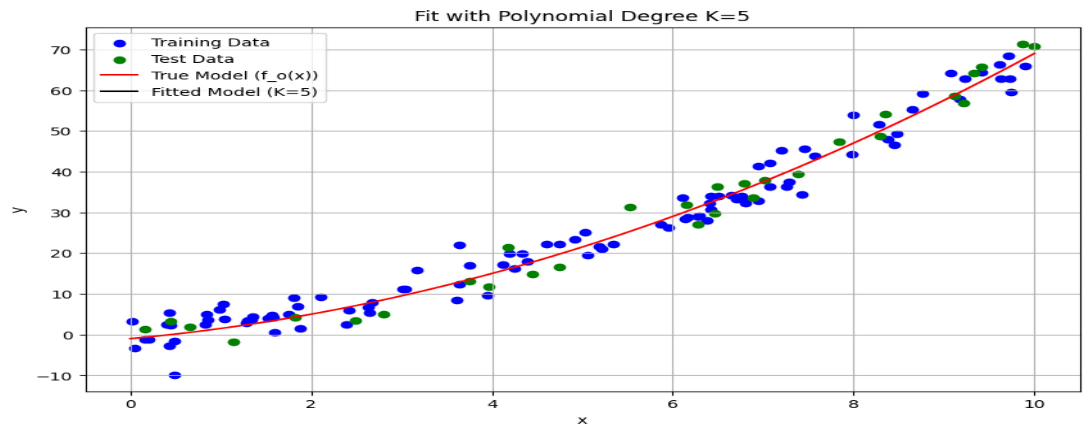

Figure 3: Plot of Test and Train Data (K = 5)

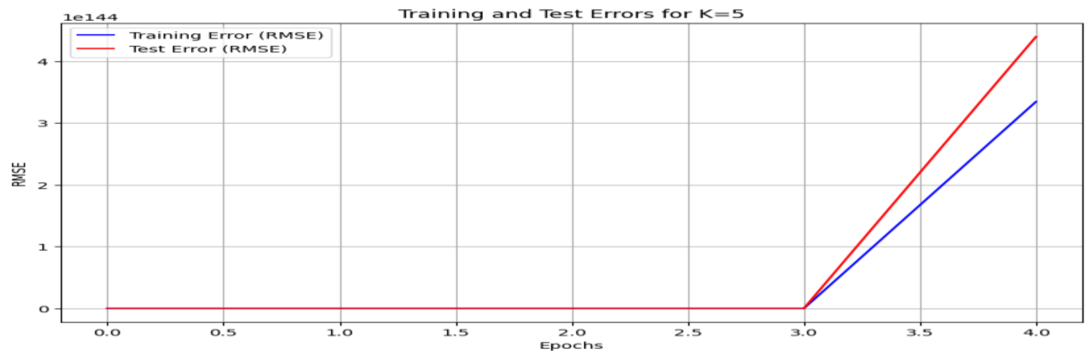

Figure 4: Plot of Test and Testing error (K=5)

With respect to the above plots, both the training and test errors remain close to zero for the first few epochs and then rapidly escalate starting around epoch 3. The sudden rise in RMSE suggests that the model initially fits the data well but starts diverging. This model also exhibits signs of overfitting. Early stopping should be considered, as continuing to train after epoch 3 is leading to a worse performance.

If $\boldsymbol{K}$ is set to be very large, say $\boldsymbol{K = N + 5}$**,** In such cases, while a higher degree polynomial might give a near-perfect fit to the training data, the test error would increase significantly due to overfitting. This is because the model becomes too complex and captures the noise in the training data, leading to poor generalization on new, unseen data.

For $\boldsymbol{K = 1}$ (Linear Model):

Underfitting Issue: Likely. A linear model may not be flexible enough to capture the true relationship if the data has a nonlinear trend (e.g., quadratic). Therefore, the training error will be relatively high, and the model will generalize poorly because it cannot represent the underlying complexity of the data.

Mitigation: Introduce feature engineering and increase model complexity.

For $\boldsymbol{K = 2}$ (Quadratic Model):

Overfitting Issue: Unlikely. The model still has a relatively low degree and is less likely to fit the noise in the data unless the true relationship is very simple.

Mitigation: Introduce regularization, which penalizes large coefficient values.

For $\boldsymbol{K = 5}$ (Higher Degree Polynomial):

Overfitting Issue: Possible. The model may start overfitting, especially if the data has some noise. Since the degree is relatively high, it might fit the noise and result in poor generalization to test data.

Cause: Higher complexity introduces more flexibility, but it also leads to higher variance, making the model more sensitive to fluctuations in the training data.

Mitigation: Tune the hyperparameters, especially for the unseen test data

For $\boldsymbol{K = N + 5}$ (Higher Degree Polynomial):

Overfitting Issue: Very Likely. This is a classic case of overfitting. With a very high degree, the model fits the noise and the random variations in the training data, resulting in high variance. It will not generalize well to unseen test data.

Cause: The model is excessively complex, capturing noise and spurious patterns in the data.

Mitigation: Reduce the Polynomial Degree.

When N increases, how does it affect the training and test errors? What is the computational complexity order with respect to N?

As N increases, the model is exposed to more data, which helps it learn the underlying pattern better. As for the computation complexity, as N grows, the time to compute the solutions grows cubically, making it computationally expensive for large datasets.

2

a.

<u>Problem 2</u>

$$P(y|x) = \begin{cases} \hat{y}, & \text{if } y = 1 \\ 1-\hat{y}, & \text{if } y = 0 \end{cases} \quad \text{———} \; ①$$

$$P(y=1|x) = \hat{y}, \quad P(y=0|x) = 1-\hat{y} \quad \text{—} \; ②$$

$y \in [0, 1]$

$\therefore$ ① becomes

$$P(y|x) = \hat{y}^{y} \cdot (1-\hat{y})^{1-y} \quad \text{—} \; ③$$

Because when $y = 0$

$$P(y=0|x) = \hat{y}^{0} \cdot (1-\hat{y})^{1} = (1-\hat{y})$$

When $y = 1$

$$P(y=1|x) = \hat{y}^{1} \cdot (1-\hat{y})^{1-1} = \hat{y}$$

$$\therefore \quad P(y|x) = \hat{y}^{y} \cdot (1-\hat{y})^{1-y}$$

Recall Maximum Likelihood Estimation

$$L(\theta) = P(\vec{y} \,|\, x; \theta)$$

$$L(\theta) = \prod_{i=1}^{m} P(y^{(i)} \,|\, x^{(i)}; \theta)$$

$$L(\theta) = \prod_{i=1}^{m} P(y|x) \qquad \left\{ x \text{ is parameterized by } \theta \right.$$

$$L(\theta) = \prod_{i=1}^{m} \hat{y}^{y} \cdot (1-\hat{y})^{1-y}$$

log likelihood

$l(\theta) = \operatorname{Log} L(\theta)$

$\therefore\ l(\theta) = \operatorname{Log}\left(\prod_{i=1}^{m} \hat{y}^{y} \cdot (1-\hat{y})^{1-y}\right)$

$$\log P(y|x) = \log\left(\hat{y}^{y} \cdot (1-\hat{y})^{1-y}\right)$$

$$\log P(y|x) = \log \hat{y}^{y} + \log (1-\hat{y})^{1-y}$$

$$\log P(y|x) = y \log \hat{y} + (1-y) \log (1-\hat{y})$$

Negative log likelihood is then

$$-\log P(y|x) = -\left(y \log \hat{y} + (1-y) \log (1-\hat{y})\right)$$

NB: The log likelihood is maximized to estimate the best model Parameters, while NLL is minimized using gradient descent optimization

b,c. Steps

The steps taken into the logistic regression for the binary classification problem on the Spam dataset included the following.

1. Import necessary python modules for data preprocessing, visualization, and normalization.
2. Load the dataset using the provided URL.
3. Inspect the contents of the datasets using the pandas functionalities.
4. Split the data into training set and a test set (a random state of value, 42 was used).
5. Define the following functions
    a. Sigmoid function (Activation function)
    b. Cost function (losses)
    c. Gradient Descent (Optimization Algorithm)
    d. Prediction (comparison with true values)
    e. Logistic regression (Putting it altogether).

Findings and Observations

The findings are summarized in the table below.

Without Normalization	**With Normalization**
Learning rate = 0.1	Learning rate = 0.1
Number of Iterations = 1000	Number of Iterations = 1000
Training accuracy = 52.5%	Training accuracy = 92.1%
Test accuracy = 50.9%	Test accuracy = 92.3%

By Normalizing, the accuracy of the model was significantly improved for both the training and testing sets. The model also converged faster, and the features contributed more equally to the model. It also facilitated the visualization and interpretation of the model performance.

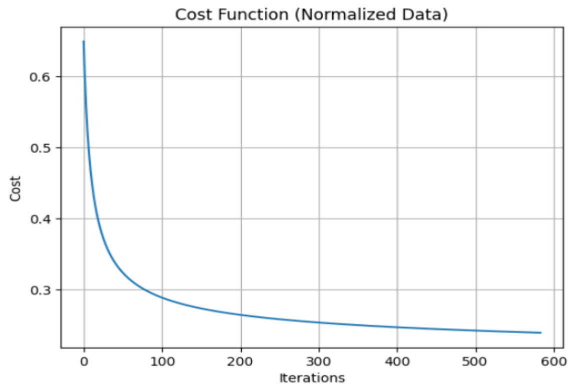

Figure 5: With Normalization

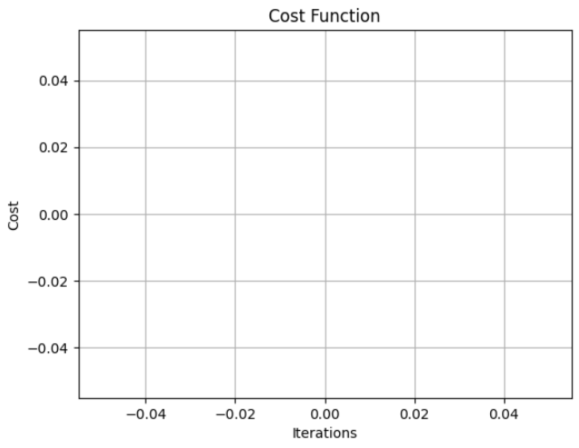

Figure 6: Without Normalization

Machine Learning algorithms are sensitive to the scale of the input data. If input features have different scales, the optimization process can take longer and may fail to find the optimal solution.