Machine Learning Assignment-S-1

Murray
202300460030

August 15, 2024

School of Cyber Science and Technology
Shandong University

Problem 1: Linear Regression Model Design

Given data for 15 students with hours spent in lectures, hours spent on homework, and their corresponding marks, your task is to design a linear regression model that minimizes the mean squared error (MSE). The data provided is as follows:

- Hours spent in lectures:

$$\text{Hrs\_Lec} = [15, 18, 33, 22, 5, 29, 8, 5, 30, 8, 16, 1, 24, 20, 2]$$

- Hours spent on homework:

$$\text{Hrs\_HW} = [7, 10, 7, 6, 8, 10, 6, 8, 4, 10, 6, 5, 10, 5, 2]$$

- Corresponding marks:

$$\text{Marks} = [57, 66, 88, 64, 48, 100, 49, 46, 77, 50, 64, 29, 86, 59, 26]$$

1. Write X and y

The design matrix X and target vector y for this linear regression problem are:

$$X = \begin{bmatrix} 1 & \text{Hrs\_Lec}_1 & \text{Hrs\_HW}_1 \\ 1 & \text{Hrs\_Lec}_2 & \text{Hrs\_HW}_2 \\ \vdots & \vdots & \vdots \\ 1 & \text{Hrs\_Lec}_{15} & \text{Hrs\_HW}_{15} \end{bmatrix}, \quad y = \begin{bmatrix} \text{Marks}_1 \\ \text{Marks}_2 \\ \vdots \\ \text{Marks}_{15} \end{bmatrix}$$

Here, X is a 15×3 matrix where the first column consists of ones (bias term), the second column represents hours spent in lectures, and the third column represents hours spent on homework. y is a 15×1 column vector representing the marks of the students.

That is:

$$X^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 15 & 18 & 33 & 22 & 5 & 29 & 8 & 5 & 30 & 8 & 16 & 1 & 24 & 20 & 2 \\ 7 & 10 & 7 & 6 & 8 & 10 & 6 & 8 & 4 & 10 & 6 & 5 & 10 & 5 & 2 \end{bmatrix}$$

$$y^T = \begin{bmatrix} 57 & 66 & 88 & 64 & 48 & 100 & 49 & 46 & 77 & 50 & 64 & 29 & 86 & 59 & 26 \end{bmatrix}$$

2. Python Code for Calculating Optimal Parameters

Using the formula for the optimal parameters $w = (X^T X)^{-1} X^T y$, the Python code to calculate these parameters is as follows:

```python
import numpy as np

# Data
hrs_lec = np.array([15, 18, 33, 22, 5, 29, 8, 5, 30, 8, 16, 1, 24, 20, 2])
hrs_hw = np.array([7, 10, 7, 6, 8, 10, 6, 8, 4, 10, 6, 5, 10, 5, 2])
marks = np.array([57, 66, 88, 64, 48, 100, 49, 46, 77, 50, 64, 29, 86, 59, 26])

# Construct X and y
X = np.column_stack((np.ones(hrs_lec.shape[0]), hrs_lec, hrs_hw))
y = marks

# Calculate optimal parameters w
w = np.linalg.inv(X.T @ X) @ X.T @ y

# Output result
print(f"Optimal␣parameters:␣{w}")
```

Listing 1: Python Code for Optimal Parameters

3. Results

The optimal parameters obtained from the Python code are:

$$w = \begin{bmatrix} 15.396 \\ 1.711 \\ 2.638 \end{bmatrix}$$

Problem 2: Concrete Compressive Strength Prediction

You are given a dataset to predict concrete compressive strength based on various features. Follow the instructions below:

1. Make a Pandas DataFrame by reading the .csv file and print five rows of the DataFrame from the top

Cement	BlastFurnaceSlag	FlyAsh	Water	CoarseAggregate	FineAggregate	Age	CompressiveStrength
540.0	0.0	0.0	162.0	1040.0	676.0	28	79.99
540.0	0.0	0.0	162.0	1055.0	676.0	28	61.89
332.5	142.5	0.0	228.0	932.0	594.0	270	40.27
332.5	142.5	0.0	228.0	932.0	594.0	365	41.05
198.6	132.4	0.0	192.0	978.4	825.5	360	44.30

Table 1: First five rows of the concrete dataset.

2. How many features (N) do you have for this data set?

The dataset contains 7 features.

3. Write down the names of all the features

The names of all the features in the dataset are:

- Cement
- BlastFurnaceSlag
- FlyAsh
- Water
- CoarseAggregate
- FineAggregate
- Age

4. Is your problem a supervised learning problem or an unsupervised learning problem?

This is a supervised learning problem because we are predicting concrete compressive strength (target variable) based on input features.

5. Plot feature maps by considering one feature at a time (you can plot feature map for any three features).

Here are the feature maps for three selected features:

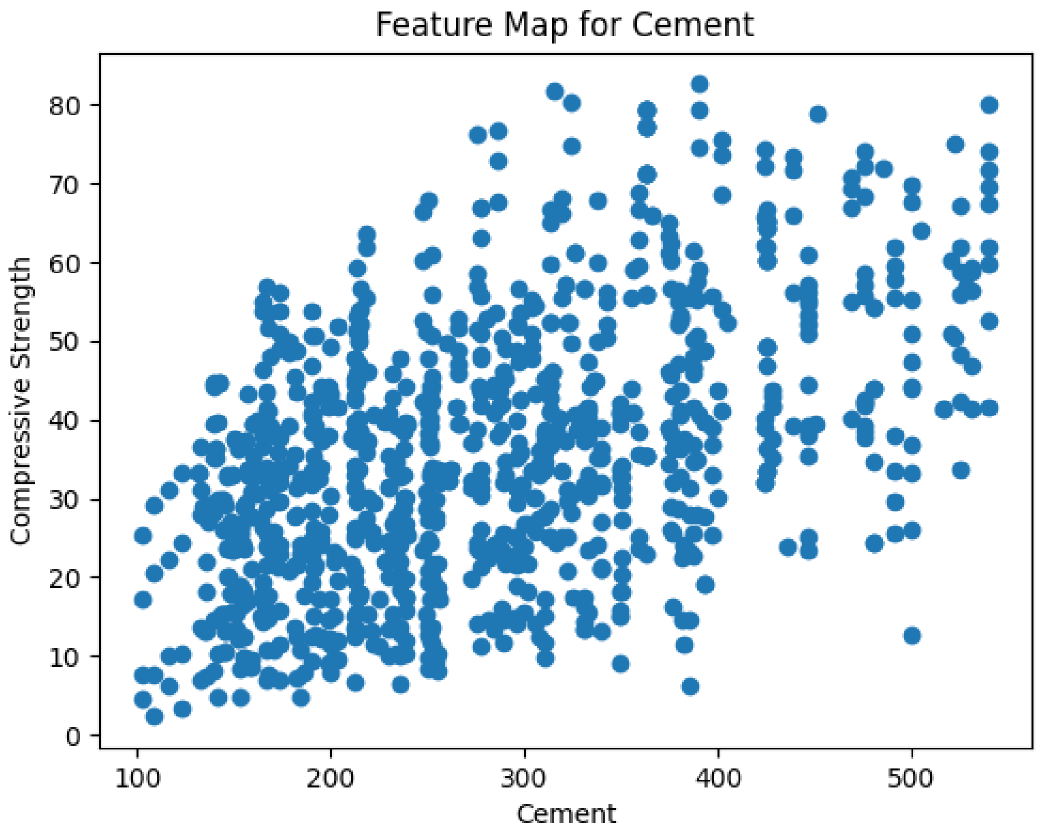

Figure 1: cement map

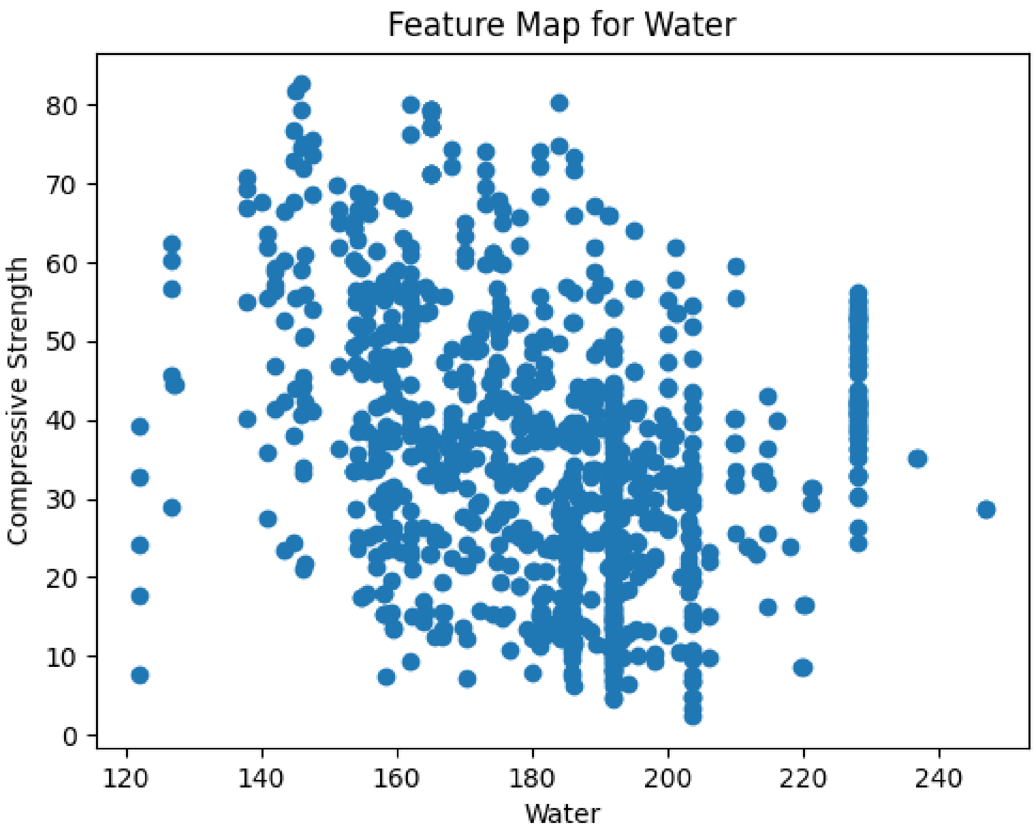

Figure 2: water map

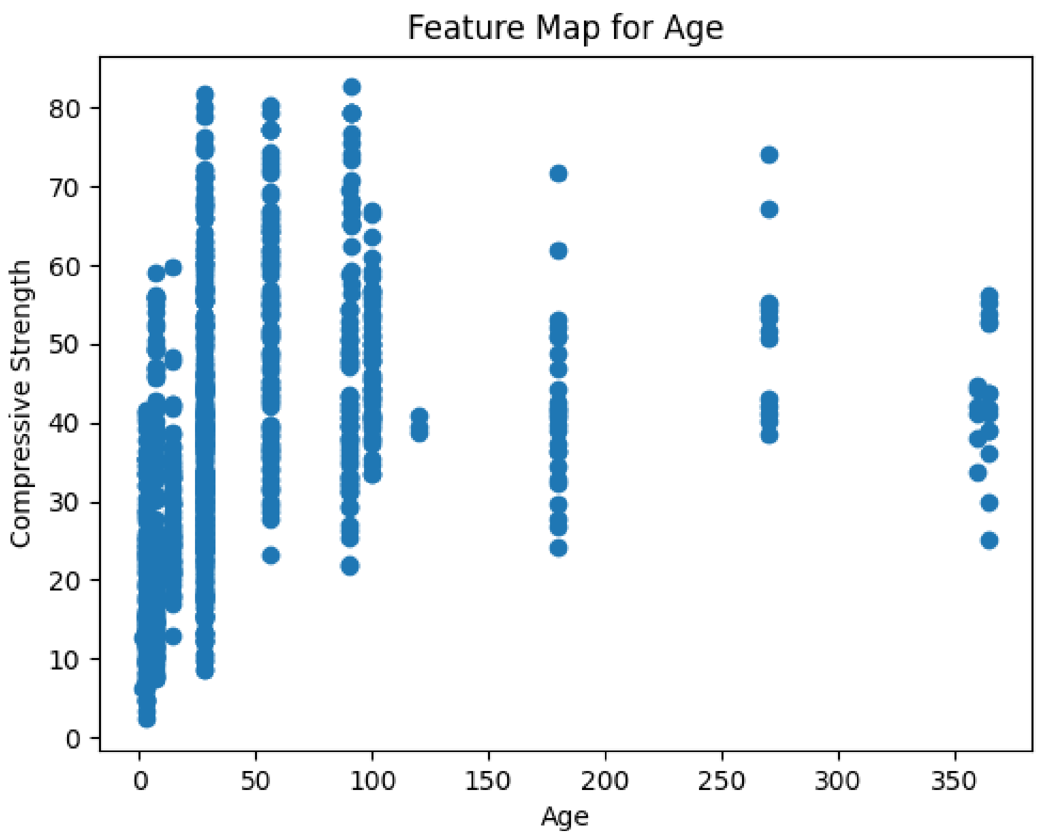

Figure 3: age map

6. Report the test MSE of your linear regression model with only one input feature.

Below is the Python code to build a linear regression model using one feature (e.g., ‘water‘) and report the test MSE:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Load and prepare the dataset
data = pd.read_csv('concrete.csv')
X = data[['Water']]  # Use 'Water' as the input feature
y = data['concrete_compressive_strength']  # Target variable

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42)

# Define the training set size
Train_set_size = 800
X_train_set = X_train.iloc[:Train_set_size]
y_train_set = y_train.iloc[:Train_set_size]

# Create and fit the linear regression model
linear_model = LinearRegression()
linear_model.fit(X_train_set, y_train_set)

```

```
# Predict and calculate MSE
y_pred = linear_model.predict(X_test)
validation_MSE = mean_squared_error(y_test, y_pred)

# Output result
print(f"Number␣of␣training␣data␣samples␣is␣{Train_set_size}␣and␣the␣Validation␣
    MSE␣is␣{validation_MSE}")
```

Listing 2: Python Code for Linear Regression and MSE Calculation

Test MSE Result

The result from the python code above:

```
Number of training data samples is 800 and the Validation MSE is 232.7491698027472
```

Problem 3: Free Fall Regression Analysis

The distance traveled in free fall is given by $d = \frac{g}{2}t^2$, where g is the acceleration due to gravity, d denotes the distance, and t denotes the time. You are given the following data for distance d and t^2:

$$d\ (\text{cm}) : [100, 100, 100, 127, 127, 127, 152, 152, 152, 178, 178, 178]$$

$$t^2\ (\text{s}^2) : [0.36, 0.38, 0.46, 0.46, 0.49, 0.51, 0.50, 0.53, 0.56, 0.55, 0.58, 0.61]$$

1. Scatter Plot of the Data

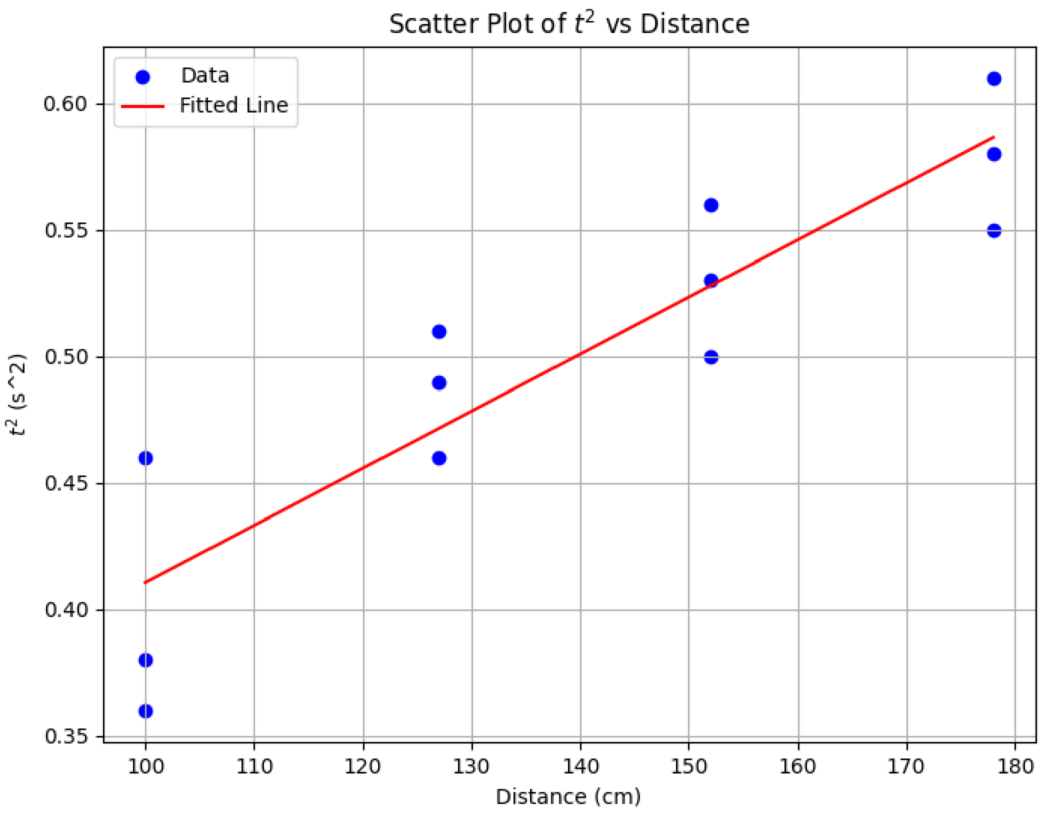

Figure 4: scatter plot

2. Calculating Parameters

To find the optimal parameters that minimize the mean squared error (MSE), the Python code below calculates the slope and intercept for the linear regression model:

```
import numpy as np
import matplotlib.pyplot as plt

d_cm = np.array([100, 100, 100, 127, 127, 127, 152, 152, 152, 178, 178, 178])
t_squared = np.array([0.36, 0.38, 0.46, 0.46, 0.49, 0.51, 0.50, 0.53, 0.56,
    0.55, 0.58, 0.61])

d_m = d_cm * 0.01

A = np.vstack([d_m, np.ones(len(d_m))]).T
slope, intercept = np.linalg.lstsq(A, t_squared, rcond=None)[0]

```

```python
g = 2 / slope

print(f"Slope: {slope:.4f}")
print(f"Intercept: {intercept:.4f}")
print(f"Estimated g: {g:.4f}")
```
Listing 3: Python Code for Regression Parameters

The calculated parameters are:

- Slope: 0.2257
- Intercept: 0.1849

3. Estimating g

Using the calculated slope m from the linear regression, estimate g with the formula $d = \frac{g}{2}t^2$. Rearranging this, we get $g = 2/\text{slope}$. Compare this with the typical value of g (approximately 9.81 m/s^2).

- Estimated g: 8.8623 m/s^2

Problem 4: Polynomial Regression Model Validation

Given are the parameters for polynomial regression models with different orders. The polynomial regression model with order $D = 2$ has parameters $w_0 = 5$, $w_1 = 1.5$, and $w_2 = 0.03$. The polynomial regression model with order $D = 3$ has parameters $w_0 = 2$, $w_1 = 0.5$, $w_2 = 0.01$, and $w_3 = -0.001$. You are given 4 data points for validation:

Input feature x	Output label y
4	10
8	20
9	25
14	35

1. Calculate Validation MSE for Both Models

To calculate the Mean Squared Error (MSE) for the validation data for both models, we use the following Python code:

```python
import numpy as np
from sklearn.metrics import mean_squared_error

# Data points for validation
X = np.array([4, 8, 9, 14])
y = np.array([10, 20, 25, 35])

# Model parameters for D=2
w0_d2, w1_d2, w2_d2 = 5, 1.5, 0.03

# Predict using polynomial of order 2
X_d2 = np.vstack((np.ones(X.shape[0]), X, X**2)).T
y_pred_d2 = X_d2 @ np.array([w0_d2, w1_d2, w2_d2])
mse_d2 = mean_squared_error(y, y_pred_d2)

# Model parameters for D=3
w0_d3, w1_d3, w2_d3, w3_d3 = 2, 0.5, 0.01, -0.001

# Predict using polynomial of order 3
X_d3 = np.vstack((np.ones(X.shape[0]), X, X**2, X**3)).T
y_pred_d3 = X_d3 @ np.array([w0_d3, w1_d3, w2_d3, w3_d3])
mse_d3 = mean_squared_error(y, y_pred_d3)

print(f"MSE␣for␣D=2␣model:␣{mse_d2:.4f}")
print(f"MSE␣for␣D=3␣model:␣{mse_d3:.4f}")
```

Listing 4: Python Code for Linear Regression and MSE Calculation

The calculated MSE values are:

$$\text{MSE for } D = 2 \text{ model: } 7.4140$$

$$\text{MSE for } D = 3 \text{ model: } 320.9830$$

2. Model Comparison

Comparing the validation MSEs of the two models:

- MSE for $D = 2$ model: 7.4140
- MSE for $D = 3$ model: 320.9830

Based on the MSE values, the polynomial regression model with $D = 2$ has a lower MSE compared to the model with $D = 3$. Therefore, the model with $D = 2$ is more appropriate for this regression problem.

Problem 5: Gradient Descent Algorithm

1. Calculate w_0 for the next two iterations with step size $\alpha = 0.1$

Given the function $f(w_0) = w_0^4 - 5w_0^2 - 3w_0$, the gradient is:

$$\text{Gradient} = 4w_0^3 - 10w_0 - 3$$

For iteration 0:

$$w_0^{(0)} = -2.0$$
$$\text{Gradient} = 4(-2.0)^3 - 10(-2.0) - 3 = -32 + 20 - 3 = -15$$
$$w_0^{(1)} = -2.0 - 0.1 \times (-15) = -2.0 + 1.5 = -0.5$$

For iteration 1:

$$w_0^{(1)} = -0.5$$
$$\text{Gradient} = 4(-0.5)^3 - 10(-0.5) - 3 = -0.5 + 5 - 3 = 1.5$$
$$w_0^{(2)} = -0.5 - 0.1 \times 1.5 = -0.5 - 0.15 = -0.65$$

2. Calculate the optimal w_0 using gradient descent for different learning rates

The following Python code was used to find the optimal w_0 and the minimum value of the function for various learning rates:

```python
import numpy as np

def f(w0):
    return w0**4 - 5*w0**2 - 3*w0

def gradient(w0):
    return 4*w0**3 - 10*w0 - 3

def gradient_descent(alpha, w0_init, iterations):
    w0 = w0_init
    for i in range(iterations):
        grad = gradient(w0)
        w0 = w0 - alpha * grad
        w0 = np.clip(w0, -10, 10)
        if i % 10 == 0:
            print(f"Iteration␣{i}:␣w0␣=␣{w0:.4f},␣Gradient␣=␣{grad:.4f}")
    return w0, f(w0)

w0_init = -2.0
iterations = 50

alphas = [0.2, 0.1, 0.01, 0.001]
results = {}

for alpha in alphas:
    optimal_w0, min_value = gradient_descent(alpha, w0_init, iterations)
    results[alpha] = (optimal_w0, min_value)

for alpha, (optimal_w0, min_value) in results.items():
    print(f"Learning␣rate␣{alpha}:␣Optimal␣w0␣=␣{optimal_w0:.4f},␣Minimum␣value␣
    =␣{min_value:.4f}")
```

Listing 5: Python Code for Optimal Parameters

Results:

- Learning rate $\alpha = 0.2$: Optimal $w_0 = 10.0000$, Minimum value = 9470.0000
- Learning rate $\alpha = 0.1$: Optimal $w_0 = -1.4018$, Minimum value = -1.7584
- Learning rate $\alpha = 0.01$: Optimal $w_0 = -1.4020$, Minimum value = -1.7584
- Learning rate $\alpha = 0.001$: Optimal $w_0 = -1.6160$, Minimum value = -1.3894

3. Did you obtain the global optimal value of w_0?

For the learning rates tested, the optimal w_0 values are as follows. The values for $\alpha = 0.1$ and $\alpha = 0.01$ yield the same minimum function value, indicating a potential local minimum. To determine if this is the global optimum, further investigation with different learning rates and more advanced optimization techniques may be necessary.

4. Analysis of Optimization Results

For the learning rate $\alpha = 0.1$ and initial point $w_0 = -2.0$, the gradient descent algorithm yielded the following results:

```
Learning rate: 0.1
  Initial point: -2.0
Iteration 0: w0 = 1.0000, Gradient = -15.0000
Iteration 1: w0 = -0.6500, Gradient = 1.5000, Function Value = 0.0160
Iteration 2: w0 = -0.8902, Gradient = 2.4015, Function Value = -0.6635
Iteration 3: w0 = -1.1982, Gradient = 3.0802, Function Value = -1.5226
Iteration 4: w0 = -1.4083, Gradient = 2.1013, Function Value = -1.7581
Iteration 5: w0 = -1.3994, Gradient = -0.0893, Function Value = -1.7584
Iteration 6: w0 = -1.4026, Gradient = 0.0326, Function Value = -1.7584
...
Iteration 98: w0 = -1.4018, Gradient = 0.0000, Function Value = -1.7584
Iteration 99: w0 = -1.4018, Gradient = 0.0000, Function Value = -1.7584
Optimal w0: -1.4018, Minimum value: -1.7584
```

The output shows that the gradient descent algorithm has converged to an optimal w_0 value of -1.4018 with a minimum function value of -1.7584. Therefore, it is highly likely that this result represents the global minimum.

To verify if this result is indeed the global minimum, it is essential to test different learning rates and potentially employ advanced optimization techniques.