Lab1_report

student ID :313512065

Code:

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


# Load dataset
x_train, x_test, y_train, y_test = np.load('regression_data.npy', allow_pickle=True)

# Reshape targets
y_train = y_train.reshape(-1,)
y_test = y_test.reshape(-1,)

# Add bias term (column of ones)
train_data = np.hstack((x_train, np.ones((x_train.shape[0], 1))))
test_data = np.hstack((x_test, np.ones((x_test.shape[0], 1))))

# ==============================
# Task 1: Standard Linear Regression (Gradient Descent)
# ==============================
def linear_regression_train(x_train, y_train, lr=1e-3, iterations=7000):
    weight = np.random.randn(2)
    loss = np.zeros(iterations)

    for i in range(iterations):
        y_pred = x_train @ weight  # Compute predicted values

        loss[i] = np.mean((y_pred - y_train) ** 2)  # Compute MSE

        # Compute gradients
        m_gradient = (2 / x_train.shape[0]) * np.sum((y_pred - y_train) * x_train[:, 0])  # Compute
gradient for weight
        b_gradient = (2 / x_train.shape[0]) * np.sum(y_pred - y_train)
         # Compute gradient for bias

        # Update weights
        weight[0] -= lr * m_gradient  # Apply gradient descent for weight
        weight[1] -= lr * b_gradient # Apply gradient descent for bias

    return weight, loss

weight_standard, loss_standard = linear_regression_train(train_data, y_train)

print("\n=== Standard Linear Regression Parameters ===")
print(f'Weight (m): {weight_standard[0]}')  # Print weight[0]
print(f'Bias (b): {weight_standard[1]}')  # Print weight[1]
```

```
# ==============================
# Task 2: Compute MSE
# ==============================
def compute_mse(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)
  # Compute MSE formula

y_pred_standard = train_data @ weight_standard.reshape(-1, 1)
y_pred_standard = y_pred_standard.flatten()


print("\n=== Mean Squared Error (Standard Regression) ===")
print(f'MSE: {compute_mse(y_train, y_pred_standard)}')

# ==============================
# Task 3: Ridge Regression (Gradient Descent)
# ==============================
def ridge_regression_train(x_train, y_train, lr=1e-3, iterations=7000, lambda_reg=0.1):
    weight = np.random.randn(2)
    loss = np.zeros(iterations)

    for i in range(iterations):
        y_pred = x_train @ weight  # Compute predicted values

        loss[i] = np.mean((y_pred - y_train) ** 2) + lambda_reg * np.sum(weight ** 2)
  # Compute MSE with regularization term

        # Compute gradients with regularization
        m_gradient = (2 / x_train.shape[0]) * np.sum((y_pred - y_train) * x_train[:, 0]) + 2 * lambda_reg *
weight[0]  # Compute weight gradient with regularization
        b_gradient = (2 / x_train.shape[0]) * np.sum(y_pred - y_train) + 2 * lambda_reg * weight[1] #
Compute bias gradient

        # Update weights
        weight[0] -= lr * m_gradient  # Apply gradient descent for weight
        weight[1] -= lr * b_gradient
  # Apply gradient descent for bias

    return weight, loss

weight_ridge, loss_ridge = ridge_regression_train(train_data, y_train)

print("\n=== Ridge Regression Parameters ===")
print(f'Weight (m): {weight_ridge[0]}')
print(f'Bias (b): {weight_ridge[1]}')

y_pred_ridge = test_data @ weight_ridge.reshape(-1, 1)  # Compute predictions for test data
y_pred_ridge = y_pred_ridge.flatten()
```

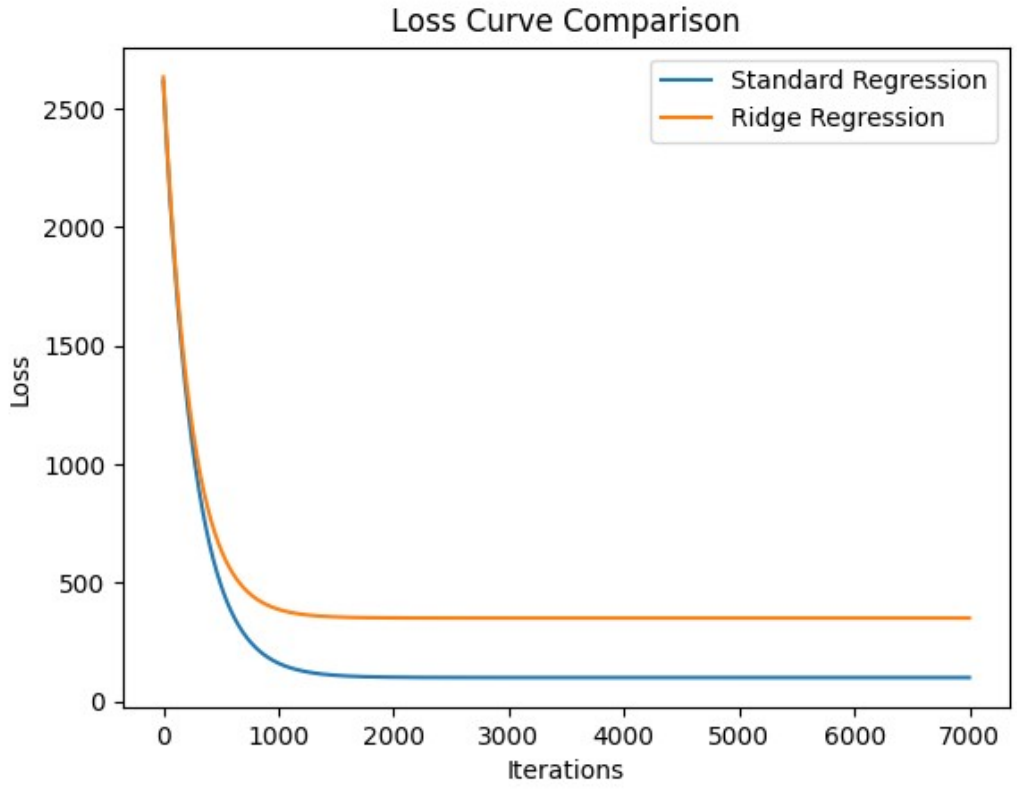
```
mse_ridge = compute_mse(y_test, y_pred_ridge)

print("\n=== Mean Squared Error (Ridge Regression) ===")
print(f'MSE: {mse_ridge}')

# ==============================
# Task 4: Plot Loss Curve
# ==============================
plt.plot(loss_standard, label='Standard Regression')
plt.plot(loss_ridge, label='Ridge Regression')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.title('Loss Curve Comparison')
plt.legend()
plt.show()

# ==============================
# Task 5: Closed-form Ridge Regression
# ==============================
def closed_form_ridge(x_train, y_train, lambda_reg=0.1):
    I = np.eye(x_train.shape[1])
    I[-1, -1] = 0
    w_closed_form = np.linalg.inv(x_train.T @ x_train + lambda_reg * I) @ x_train.T @ y_train  #
Compute closed-form solution (Equation 4.27)
    return w_closed_form

weight_closed_form = closed_form_ridge(train_data, y_train)
y_pred_closed_form = test_data @ weight_closed_form.reshape(-1, 1)
y_pred_closed_form = y_pred_closed_form.flatten()
mse_closed_form = compute_mse(y_test, y_pred_closed_form)

print("\n=== Closed-form Ridge Regression Parameters ===")
print(f'Weight (m): {weight_closed_form[0]}')
print(f'Bias (b): {weight_closed_form[1:]}')
print("\n=== Mean Squared Error (Closed-form Ridge Regression) ===")
print(f'MSE: {mse_closed_form}')

# ==============================
# Task 6: Predictive Distribution
# ==============================
# Compute posterior covariance
lambda_reg=0.1
I = np.eye(x_train.shape[1])
sigma_n = lambda_reg * np.eye(x_train.shape[1])

 # Compute posterior covariance
S_N_inv = x_train.T @ x_train + sigma_n
S_N = np.linalg.inv(S_N_inv)
```

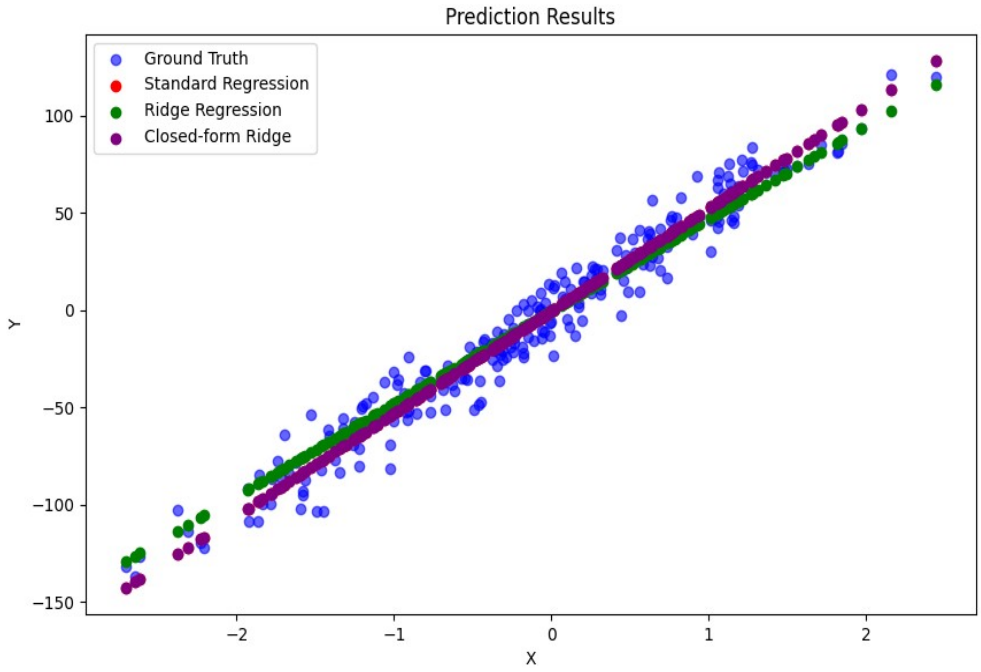
```
# Compute posterior mean
w_mean = S_N @ x_train.T @ y_train

# Predictive mean
predictive_mean = x_test @ w_mean

# Predictive variance
predictive_variance = np.sum(x_test @ S_N * x_test, axis=1)



# ==============================
# Task 7: Plot Predictions
# ==============================
plt.figure(figsize=(10, 6))


# Prediction lines

plt.scatter(x_test[:, 0], y_test, label='Ground Truth', color='blue', alpha=0.6)

# Standard Regression
y_pred_standard = test_data @ weight_standard
plt.scatter(x_test[:, 0], y_pred_standard, label='Standard Regression', color='red')

# Ridge Regression
y_pred_ridge = test_data @ weight_ridge
plt.scatter(x_test[:, 0], y_pred_ridge, label='Ridge Regression', color='green')

# Closed-form Ridge Regression
y_pred_closed_form = test_data @ weight_closed_form
plt.scatter(x_test[:, 0], y_pred_closed_form, label='Closed-form Ridge', color='purple')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Prediction Results')
plt.legend()
plt.show()

# Confidence Interval Shading


# ==============================
# Plot Confidence Intervals
# ==============================
plt.scatter(x_test[:, 0], y_test, label='Ground Truth', color='blue', alpha=0.6)

# Ridge Regression
```

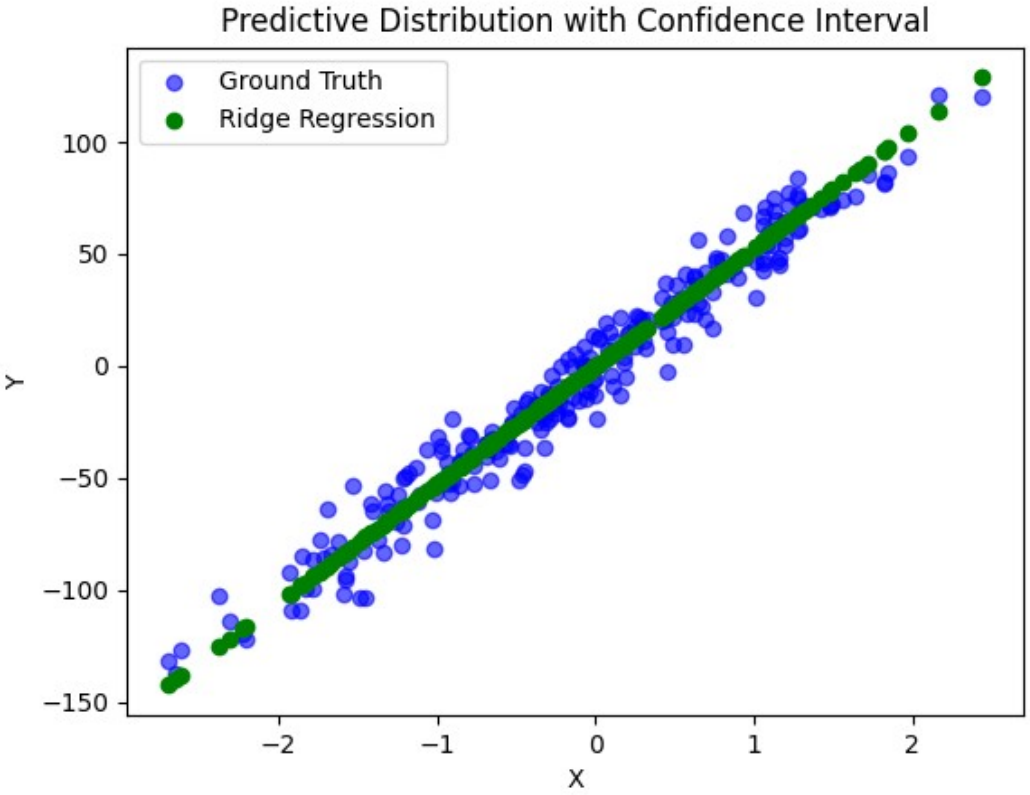
```
plt.scatter(x_test[:, 0], predictive_mean, label='Ridge Regression', color='green')


plt.xlabel('X')
plt.ylabel('Y')
plt.title('Predictive Distribution with Confidence Interval')
plt.legend()
plt.show()
```

Output result:

```
=== Standard Linear Regression Parameters ===
Weight (m): 52.74342253045384
Bias (b): -0.33378000891502935

=== Mean Squared Error (Standard Regression) ===
MSE: 99.34237310127232

=== Ridge Regression Parameters ===
Weight (m): 47.62796542012239
Bias (b): -0.3949129745921893

=== Mean Squared Error (Ridge Regression) ===
MSE: 122.25191150505415

=== Closed-form Ridge Regression Parameters ===
Weight (m): 52.73598698626496
Bias (b): [-0.3339075]

=== Mean Squared Error (Closed-form Ridge Regression) ===
MSE: 110.41535937243981
```




Prediction Results
Ground Truth
Standard Regression
Ridge Regression
Closed-form Ridge
100
50
0
−50
−100
−150
Y
−2
−1
0
1
2
X


Predictive Distribution with Confidence Interval
Ground Truth
Ridge Regression
100
50
0
−50
−100
−150
Y
−2
−1
0
1
2
X