Machine Learning
LABORATORY: REGRESSION

NAME: **STUDENT ID#:**

Objectives:

- This assignment aims to develop a deeper understanding of linear regression by exploring their mathematical foundations, implementation, and evaluation using gradient descent.
- Understand the concepts and mathematics behind regression models.
- Implement and Evaluate regression models from scratch.
- Understand the key concepts of regression, including least squares, likelihood estimation, and bias–variance trade-off.

Part 1. Background

Here we will know the basics concepts that we will use for the implementation of this algorithm.

What is Regression? Regression is a statistical approach to model the relationship between a dependent variable (output) and one or more independent variables (inputs). Linear Regression is used to solve problems where the relationship between variables can be reasonably approximated by a straight line.

Tasks: In this assignment, you are required to implement linear and logistic regression models using only NumPy. You will get no points by simply calling *sklearn.linear _model.LinearRegression*. Your task is to train these models with gradient Descent on a provided dataset, evaluate their performance, and test them on unseen data. The dataset and sample code can be found here: https://github.com/Satriosnjya/ML-Labs.git

Part 2. Arithmetic Instructions.

Step	Procedure
1	**Define the Model:** Linear regression predicts the output y using a **linear function**: $\boldsymbol{y}(\boldsymbol{x},\boldsymbol{w}) = \boldsymbol{w_0} + \boldsymbol{w_1}\boldsymbol{x_1} + \cdots + \boldsymbol{w_D}\boldsymbol{x_D}$, In simple linear regression with only one feature *x*: $\boldsymbol{y} = \boldsymbol{mx} + \boldsymbol{b}$
2	**Mean Square Error (MSE) Loss Calculation** From the textbook, ***Equation (4.11)*** defines the sum-of-squares error function: $E_D(w) = \frac{1}{2}\sum_{n=1}^{N}\left(t_n - w^T\phi(x_n)\right)^2$ which directly follows the equation, computing the squared difference between predicted and actual values.
3	**Gradient Descent for Weight Updates** From the textbook, ***Equation (4.12)*** defines the gradient of the log-likelihood: $\nabla_w \ln\, p(t \mid X, w, \sigma^2) = \frac{1}{\sigma^2}\sum_{n=1}^{N}\left(t_n - w^T\phi(x_n)\right)\phi(x_n)^T$ This follows the principle of **gradient descent**, adjusting weights by subtracting a scaled version of the gradient.
4	**Model Training Using Gradient Descent** Equation Reference: From the textbook, ***Equation (4.22)*** for sequential learning (LMS Algorithm):

$$w^{(\tau+1)} = w^{(\tau)} + \eta\big(t_n - w^T\phi(x_n)\big)\phi_n$$

Follows this equation by iteratively updating w based on the gradient.

5 **Mean Square Error (MSE) as Evaluation Metric**

From the textbook, ***Equation (4.20)*** defines the MLE estimate for variance:

$$\sigma_{ML}^2 = \frac{1}{N}\sum_{n=1}^{N}\left(t_n - w_{ML}^T\phi(x_n)\right)^2$$

Part 3. Data Transfer Instructions.

Step **Procedure**

1 **Download dataset and example code from https://github.com/Satriosnjya/ML-Labs.git**

2 **Open colab.research.google.com or you can run the python code in your own computer**

3 **Colab : Make a new Notebook, connect the runtime, then upload regression_data.npy**

4 **Load Libraries**

```
Load Libraries and Generate Data
import numpy as np
import matplotlib.pyplot as plt
```

5 **Generate Data**

```
# 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))))
```

Grading & Submission Instructions

Hands-on Tasks:

1. (10%) Implement Standard Linear Regression using Gradient Descent
    a. Compute gradients for weight (m) and bias (b).
    b. Update weights.
    c. Output : Model parameters (weight and bias)
2. (10%) Evaluate the model using MSE for standard regression
    a. Complete compute_mse() function
    b. Output : MSE for standard regression
3. (10%) Implement Ridge Regression with L2 Regularization
    a. Modify loss function to include L2
    b. Compute updated gradients for weight and bias.
    c. Output : Model parameters and MSE for Ridge Regression
4. (10%) Plot the training loss curve.
    a. Store the loss at each iteration and plot it using matplotlib.
    b. Try different values for: Learning Rate (*lr*) and Number of Iterations (*iterations*)
    c. Output : Loss curve comparison of Standard regression and ridge regression

Assignment:

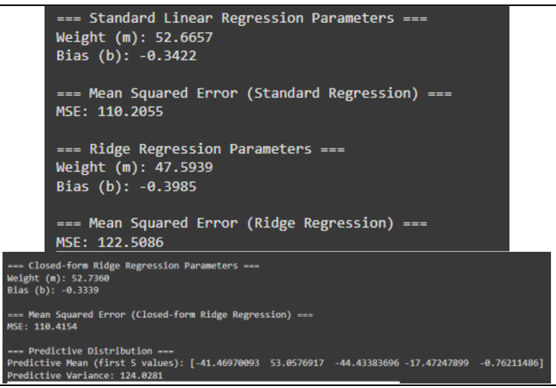
5. **(20%) Implement Closed-form Ridge Regression (Refer to Equation 4.27)**

   a. **Fill in closed_form_ridge() function**
   b. **Compute w_closed_form**
   c. **Output : Model parameters and MSE for closed-form Ridge Regression**
6. **(20%) Implement predictive distribution (Refer to Equation 4.33)**
   a. **Fill in predictive_mean and predictive_variance**
   b. **Print 5 values of predictive_mean**
   c. **Output : Predictive variance & example predictions**
7. **(20%) Visualize predictions and confidence intervals**
   a. **Fill in missing values in scatter plot (plt.scatter)**
   b. **Implement confidence interval shading using plt.fill_between().**
   c. **Output :**
      i. **Prediction plot of standard regression, ridge regression and closed-form regression.**
      ii. **Predictive distribution with confidence interval.**

Submission:

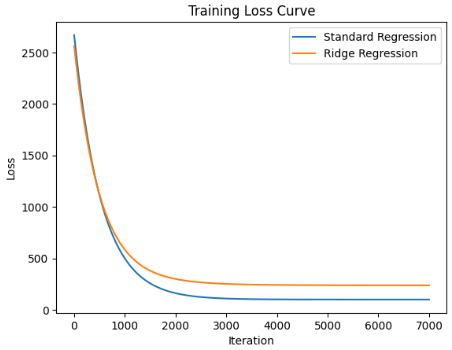
1. Report: Include screenshots of your results for each task (model training, evaluation, and plots) in the last pages of this PDF file.
2. Code: Submit your complete Python script (.py or .ipynb notebook).
3. Upload both your report and code to the E3 system. Name your files correctly:
   a. Report: StudentID_Lab1.pdf
   b. Code: StudentID_Lab1.py or StudentID_Lab1.ipynb
4. 1 day late: 10% deduction from total score. (Due Date : Sunday 9:00 PM)
5. Plagiarism is **strictly prohibited**. Submitting copied work from other students will result in penalties.

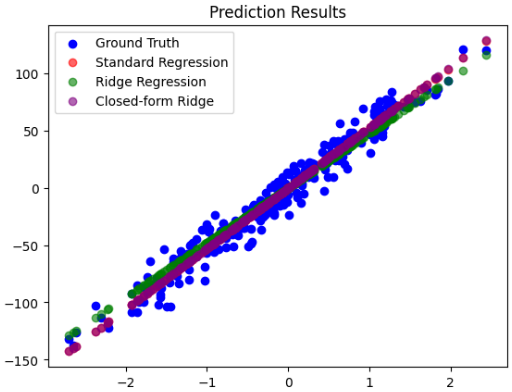
Sample Output: Please note that it is for reference only

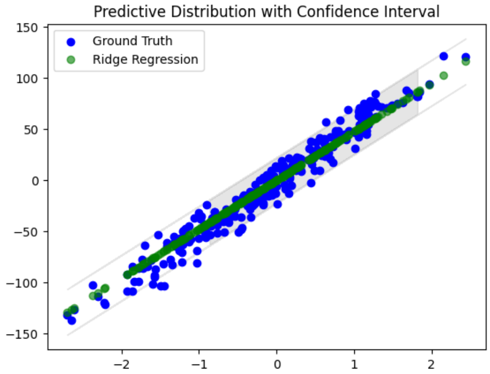


```
=== Standard Linear Regression Parameters ===
Weight (m): 52.6657
Bias (b): -0.3422

=== Mean Squared Error (Standard Regression) ===
MSE: 110.2055

=== Ridge Regression Parameters ===
Weight (m): 47.5939
Bias (b): -0.3985

=== Mean Squared Error (Ridge Regression) ===
MSE: 122.5086
```

```
=== Closed-form Ridge Regression Parameters ===
Weight (m): 52.7360
Bias (b): -0.3339

=== Mean Squared Error (Closed-form Ridge Regression) ===
MSE: 110.4154

=== Predictive Distribution ===
Predictive Mean (first 5 values): [-41.46970093  53.0576917  -44.43383696 -17.47247899  -0.76211486]
Predictive Variance: 124.0281
```

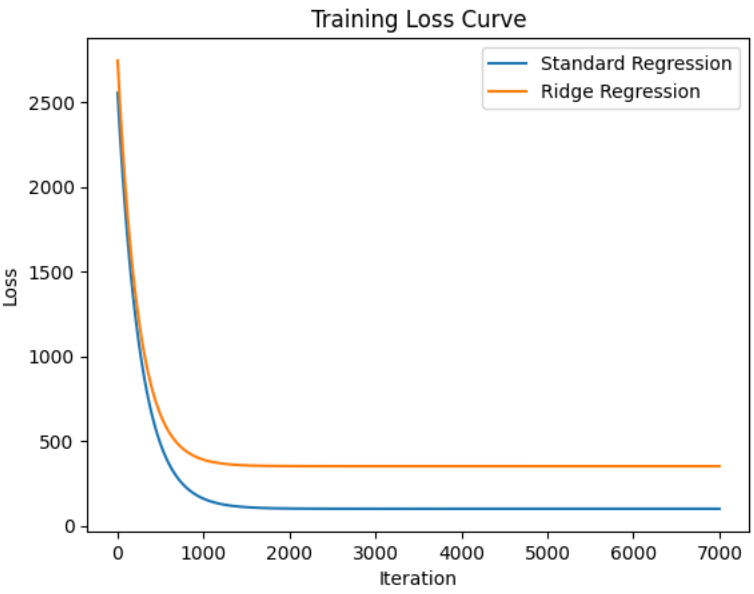




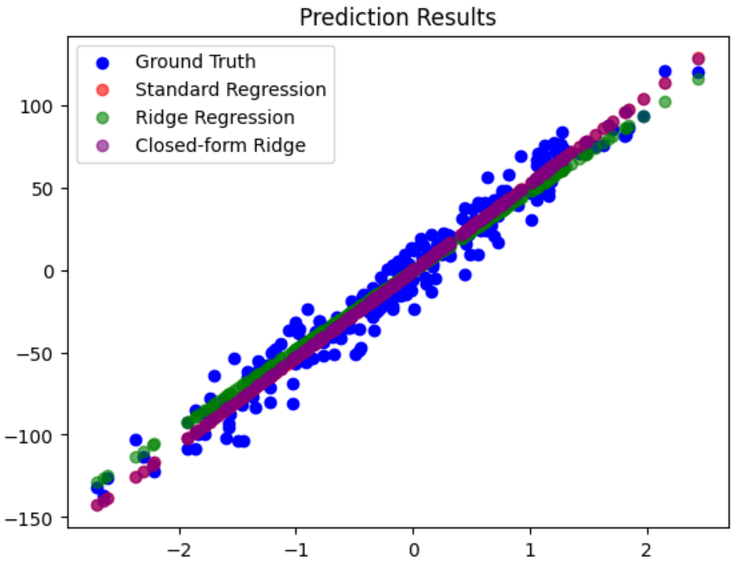


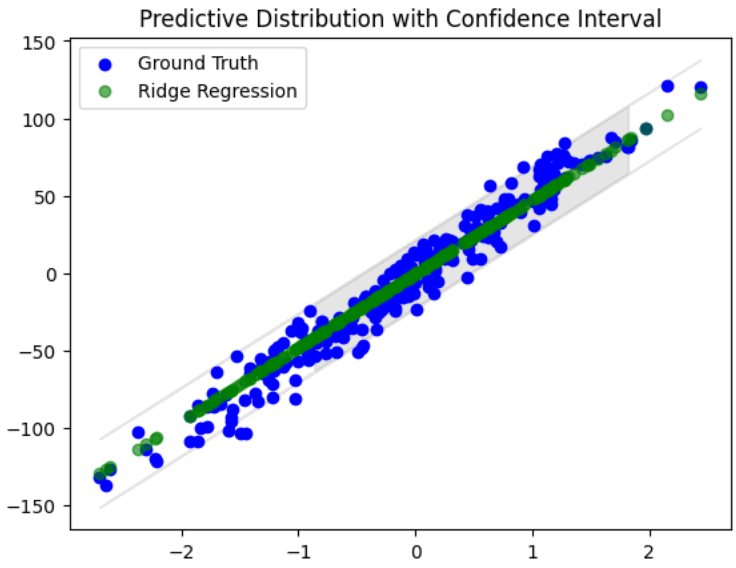
Put Your Code Results Here:

```
=== Standard Linear Regression Parameters ===
Weight (m): 52.743426614647376
Bias (b): 2545.7158438235733

=== Mean Squared Error (Standard Regression) ===
MSE: 110.43783964770469

=== Ridge Regression Parameters ===
Weight (m): 47.627963689867656
Bias (b): -0.39491310131103347
```



```
=== Closed-form Ridge Regression Parameters ===
Weight (m): 52.7359879269174
Bias (b): -0.33386296536821425

=== Mean Squared Error (Closed-form Ridge Regression) ===
MSE: 110.41538156212286

=== Predictive Distribution ===
Predictive Mean (first 5 values): [-4.14954578e+01  5.30995030e+01 -4.44617126e+01 -1.74810827e+01
```





Code

```python
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_train - y_pred) ** 2)  # Compute MSE

        # Compute gradients
        m_gradient, b_gradient= -2 * x_train.T @ (y_train - y_pred) / len(y_train)  # Compute gradient for weight
        # b_gradient = None  # 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)

```

```python
print("\n=== Standard Linear Regression Parameters ===")
print(f'Weight (m): {weight_standard[0]}')  # Print weight[0]
print(f'Bias (b): {loss_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 = test_data @ weight_standard  # Compute predictions for test data
mse_standard = compute_mse(y_test, y_pred_standard)

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

# ================================
# Task 3: Ridge Regression (Gradient Descent)
# ================================
```

```python
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_train - y_pred) ** 2) + lambda_reg * np.sum(weight ** 2)  # Compute MSE with regularization term

        # Compute gradients with regularization
        m_gradient, b_gradient = -2 * x_train.T @ (y_train - y_pred) / len(y_train) + 2 * lambda_reg * weight  # Compute weight gradient with regularization
        # b_gradient = None  # 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  # Compute predictions for test data
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")  # Plot loss_standard
plt.plot(loss_ridge, label="Ridge Regression")  # Plot loss_ridge
```

```
plt.xlabel("Iteration")
plt.ylabel("Loss")
plt.legend()
plt.title("Training Loss Curve")
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])
    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  # Compute predictions for test data
```

```
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
# ==============================
predictive_mean = y_pred_ridge  # Compute predictive mean
predictive_variance = np.var(y_test - y_pred_ridge)  # Compute predictive variance

print("\n=== Predictive Distribution ===")
print(f'Predictive Mean (first 5 values): {predictive_mean}')
print(f'Predictive Variance: {predictive_variance}')



```

```
# ================================
# Task 7: Plot Predictions
# ================================
plt.scatter(x_test, y_test, label='Ground Truth', color='blue')
plt.scatter(x_test, y_pred_standard, label='Standard Regression', color='red', alpha=0.6)  # y_pred_standard
plt.scatter(x_test, y_pred_ridge, label='Ridge Regression', color='green', alpha=0.6)  # y_pred_ridge
plt.scatter(x_test, y_pred_closed_form, label='Closed-form Ridge', color='purple', alpha=0.6)  # y_pred_closed_form
plt.legend()
plt.title("Prediction Results")
plt.show()

# ================================
# Plot Confidence Intervals
# ================================
plt.fill_between(x_test.flatten(), predictive_mean - 2*np.sqrt(predictive_variance),  # Predictive mean - 2 std dev
```

```
                 predictive_mean + 2*np.sqrt(predictive_variance), alpha=0.2, color='gray')  # Predictive mean + 2 std dev

plt.scatter(x_test, y_test, label='Ground Truth', color='blue')
plt.scatter(x_test, y_pred_ridge, label='Ridge Regression', color='green', alpha=0.6)  # y_pred_ridge
plt.legend()
plt.title("Predictive Distribution with Confidence Interval")
plt.show()
```