Lab1: Implementing Linear Regression with Gradient Descent in Python

Objectives

By the end of this lab, students will be able to:

- Understand the mathematical foundation of linear regression.
- Implement gradient descent **step by step** in Python.
- Apply regression to both **univariate** and **multivariate** datasets.
- Visualize cost function convergence and regression lines.

Lab Decomposition

Part A: Univariate Linear Regression

Step 1 – Data Preparation

- Provide or generate a simple dataset (e.g., house size vs price).
- Students load the dataset using **NumPy / Pandas**.
- Task: Plot the data points using **Matplotlib**.

Step 2 – Hypothesis Function

- Introduce $h_\theta(x) = \theta_0 + \theta_1 x$.
- Students implement a Python function:

```python
def hypothesis(theta0, theta1, x):
    return theta0 + theta1 * x
```

Step 3 – Cost Function

- Define the Mean Squared Error (MSE):

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

- Students implement `compute_cost`.

Step 4 – Gradient Descent Update Rule

- Update formulas:

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum (h_\theta(x^{(i)}) - y^{(i)})$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

- Students implement `gradient_descent` to update θ.

Step 5 – Training Loop

- Initialize θ_0, θ_1 randomly.
- Iterate updates until convergence.
- Track and plot cost reduction.

Step 6 – Results & Visualization

- Print final θ_0, θ_1.
- Plot regression line vs data points.
- Plot cost function vs iterations.

Part B: Multivariate Linear Regression

Step 1 – Data Preparation

- Use a dataset with multiple features (e.g., housing dataset: size, rooms, age → price).
- Normalize features (mean normalization or z-score scaling).

Step 2 – Vectorized Hypothesis

- Define:

$$h_\theta(X) = X\theta$$

- Students implement:

python Copy code

```python
def hypothesis(X, theta):
    return np.dot(X, theta)
```

Step 3 – Vectorized Cost Function

$$J(\theta) = \frac{1}{2m}(X\theta - y)^T(X\theta - y)$$

- Implement `compute_cost(X, y, theta)`.

Step 4 – Vectorized Gradient Descent

$$\theta := \theta - \alpha\frac{1}{m}X^T(X\theta - y)$$

- Students implement gradient descent in matrix form.

Step 5 – Experimentation

- Vary learning rates (α) and plot convergence speed.
- Compare results with Scikit-Learn's `LinearRegression`.

Step 6 – Visualization & Analysis

- If features are 2D, plot regression plane.
- Otherwise, report final cost and coefficients.

Deliverables

1. Jupyter Notebook with:
   - Code for each step.
   - Plots (data, regression line, cost convergence).
   - Comments explaining implementation.
2. A short report:
   - Observations about learning rate effect.
   - Comparison with Scikit-Learn.
   - Lessons learned from univariate → multivariate transition.

Extensions (Optional, for stronger students)

- Implement **stochastic gradient descent (SGD)**.
- Visualize **contour plots** of the cost function in 2D.
- Compare gradient descent with the **normal equation** solution.

Jupyter Notebook: Linear Regression with Gradient Descent

python

Copy code

```python
# ============================================================
# LAB: Linear Regression with Gradient Descent
# ============================================================

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

Part A – Univariate Linear Regression

Step 1: Data Preparation

python

```python
# Generate a simple dataset (house size vs. price)
X = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])  # house size in 100 m²
y = np.array([3, 4, 2, 5, 6, 7, 8, 9, 10]) # price in 100k $

plt.scatter(X, y, color="blue", marker="o")
plt.xlabel("House Size (100 m²)")
plt.ylabel("Price (100k $)")
plt.title("Training Data")
plt.show()
```

Step 2: Hypothesis Function

python

```python
# TODO: Implement hypothesis function h(x) = theta0 + theta1 * x
def hypothesis(theta0, theta1, x):
    pass  # replace with your code
```

Step 3: Cost Function

python

```python
# TODO: Implement the cost function (Mean Squared Error)
def compute_cost(theta0, theta1, X, y):
    m = len(y)
    # J(theta0, theta1) = (1/2m) * Σ(h(xi) - yi)^2
    pass  # replace with your code
```

Step 4: Gradient Descent Update Rule

python

```python
# TODO: Implement gradient descent update rule
def gradient_descent(X, y, theta0, theta1, alpha, iterations):
    m = len(y)
    cost_history = []

    for _ in range(iterations):
        # Compute predictions
        pass

        # Update rules for theta0, theta1
        pass

        # Save cost for plotting
        cost_history.append(compute_c(theta0, theta1, X, y))

    return theta0, theta1, cost_history
```

Step 5: Train the Model

python Copy code

```python
theta0, theta1, cost_history = gradient_descent(X, y, theta0=0, theta1=0, alpha=0.01, iterations=1

print("Final theta0:", theta0)
print("Final theta1:", theta1)

# Plot cost convergence
plt.plot(cost_history)
plt.xlabel("Iteration")
plt.ylabel("Cost J")
plt.title("Cost Function Convergence")
plt.show()
```

Step 6: Results & Visualization

python

```python
# Plot regression line
plt.scatter(X, y, color="blue", label="Training Data")
plt.plot(X, hypothesis(theta0, theta1, X), color="red", label="Regression Line")
plt.xlabel("House Size (100 m²)")
plt.ylabel("Price (100k $)")
plt.legend()
plt.show()
```

Step 1: Data Preparation

python

```python
# Example dataset (size, rooms → price)
data = {
    "size": [2100, 1600, 2400, 1416, 3000],
    "rooms": [3, 2, 4, 2, 4],
    "price": [400, 330, 369, 232, 540]
}
df = pd.DataFrame(data)

X = df[["size", "rooms"]].values
y = df["price"].values

# Feature normalization
X = (X - X.mean(axis=0)) / X.std(axis=0)

# Add intercept column
X = np.c_[np.ones(X.shape[0]), X]

print("X shape:", X.shape)
print("y shape:", y.shape)
```

Step 2: Hypothesis (Vectorized)

python

```python
# TODO: Implement hypothesis in vectorized form
def hypothesis(X, theta):
    pass  # replace with your code
```

Step 3: Cost Function (Vectorized)

python

```python
# TODO: Implement cost function in vectorized form
def compute_cost(X, y, theta):
    m = len(y)
    pass  # replace with your code
```

Step 4: Gradient Descent (Vectorized)

python

```python
# TODO: Implement gradient descent in vectorized form
def gradient_descent(X, y, theta, alpha, iterations):
    m = len(y)
    cost_history = []

    for _ in range(iterations):
        # Predictions
        pass

        # Gradient update
        pass

        # Save cost
        cost_history.append(compute_cost(X, y, theta))

    return theta, cost_history
```

Step 5: Train and Compare

python

```python
theta = np.zeros(X.shape[1])  # initialize
theta, cost_history = gradient_descent(X, y, theta, alpha=0.01, iterations=1000)

print("Learned parameters:", theta)

plt.plot(cost_history)
plt.xlabel("Iteration")
plt.ylabel("Cost J")
plt.title("Multivariate Cost Convergence")
plt.show()
```

Step 6: Compare with Scikit-Learn

```python
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(df[["size", "rooms"]], df["price"])

print("Sklearn coefficients:", model.intercept_, model.coef_)
```

Deliverables

- Complete all TODOs.
- Submit notebook with:
  - Plots of regression line (univariate).
  - Cost convergence plots.
  - Learned parameters (univariate + multivariate).
  - Comparison with Scikit-Learn.

Grading Rubric: Linear Regression with Gradient Descent Lab

🔹 Part A – Univariate Linear Regression (40 points)

Step 1 – Data Preparation & Visualization (5 pts)

- **5 pts**: Correct dataset loaded/generated; scatter plot shown with proper labels.
- **3 pts**: Dataset prepared but plot missing/poorly labeled.
- **0 pts**: Dataset missing or incorrect.

Step 2 – Hypothesis Function (5 pts)

- **5 pts**: Correct implementation `h(x) = θ0 + θ1 * x`.
- **3 pts**: Function exists but incorrect formula.
- **0 pts**: No function provided.

Step 3 – Cost Function (10 pts)

- **10 pts**: Correct MSE implementation with vectorized or loop approach.
- **7 pts**: Minor mistakes but generally correct.
- **3 pts**: Wrong formula but attempt made.
- **0 pts**: Not implemented.

Step 4 – Gradient Descent Update Rule (10 pts)

- **10 pts**: Correct update for θ0 and θ1 with loop iteration.
- **7 pts**: Small errors (e.g., missing learning rate, wrong sign).
- **3 pts**: Wrong idea but shows attempt.
- **0 pts**: Not implemented.

Step 5 & 6 – Training Loop & Visualization (10 pts)

- **5 pts**: Correctly trains until convergence, cost decreases.
- **3 pts**: Training works but cost does not consistently decrease.
- **2 pts**: Attempt made but incorrect loop.
- **5 pts**: Regression line plotted correctly on data.

◆ Part B – Multivariate Linear Regression (40 points)

Step 1 – Data Preparation & Normalization (5 pts)

- **5 pts**: Features normalized, intercept column added.
- **3 pts**: Data loaded but missing normalization.
- **0 pts**: Data not prepared correctly.

Step 2 – Vectorized Hypothesis (5 pts)

- **5 pts**: Correctly implements `h(X) = Xθ`.
- **3 pts**: Wrong formula but attempt shown.
- **0 pts**: Not implemented.

Step 3 – Vectorized Cost Function (10 pts)

- **10 pts**: Correct cost implementation in vectorized form.
- **7 pts**: Minor mistakes (extra division, wrong parentheses).
- **3 pts**: Attempt but fundamentally wrong.
- **0 pts**: Not implemented.

Step 4 – Vectorized Gradient Descent (10 pts)

- **10 pts**: Correct update rule $\theta := \theta - \alpha \, (1/m) \, X^T \, (X\theta - y)$.
- **7 pts**: Minor issues but shows cost decreasing.
- **3 pts**: Wrong formula but attempt made.
- **0 pts**: Not implemented.

Step 5 – Training & Convergence Plot (5 pts)

- **5 pts**: Correct cost curve plotted showing convergence.
- **3 pts**: Curve plotted but does not converge.
- **0 pts**: Missing.

Step 6 – Comparison with Scikit-Learn (5 pts)

- **5 pts**: Correct use of `LinearRegression()`, reports intercept & coefficients.
- **3 pts**: Implemented but results not compared.
- **0 pts**: Missing

◆ Deliverables & Report (20 points)

- **10 pts**: Jupyter Notebook clean, well-commented, all TODOs filled.
- **5 pts**: Plots properly labeled, readable.

- **5 pts**: Short written discussion (learning rate effects, comparison with Scikit-Learn).

◆ Total: 100 Points

- Part A (Univariate) → 40 pts
- Part B (Multivariate) → 40 pts
- Deliverables & Report → 20 pts