National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Electrical Engineering

Faculty Member: Ma'am Neelma Naz **Date:** October 9, 2025

Semester: 7th **Group:** Gp-02

CS471 Machine Learning

Lab 6: Linear Regression I - Feature Scaling, Cost Function and Gradient Descent

		PLO4	PLO5	PLO5	PLO8	PLO9
		CLO4	CLO5	CLO5	CLO6	CLO7
Student Name	**Reg. No**	**Viva / Quiz / Demo** 5 Marks	**Analysis of Data in Report** 5 Marks	**Modern Tool Usage** 5 Marks	**Ethics** 5 Marks	**Individual and Teamwork** 5 Marks
Hanzla Sajjad	403214g-					
Irfa Farooq	412564					

Introduction

This laboratory exercise will focus on the implementation of linear regression in python. Linear regression is a basic supervised learning technique which serves as a good starting point for learning supervised learning and also sets the fundamental basis for learning other machine learning techniques. In linear regression, a dataset with various features and a label is trained. It consists of weighted parameters that are trained to fit a model that best approximates the dataset.

Objectives

The following are the main objectives of this lab:

- Extract and prepare the training dataset
- Use feature scaling to ensure uniformity among the feature columns
- Implement cost function to get the overall loss
- Implement gradient descent algorithm to train the weight parameters
- Plot the training loss
- Use a prediction function to use the trained model

Lab Conduct

- Respect faculty and peers through speech and actions
- The lab faculty will be available to assist the students. In case some aspect of the lab experiment is not understood, the students are advised to seek help from the faculty.
- In the tasks, there are commented lines such as #YOUR CODE STARTS HERE# where you have to provide the code. You must put the code/screenshot/plot between the #START and #END parts of these commented lines. Do NOT remove the commented lines.
- Use the tab key to provide the indentation in python.
- When you provide the code in the report, keep the font size at 12

Theory

Linear Regression is a very basic supervised learning technique. To calculate the loss in each training example, the difference between a hypothesis and the label (y) is calculated. The hypothesis is a linear equation of the features (x) in the dataset with the coefficients acting as the weight parameters. These weight parameters are initialized to random values at the start but are then trained over time to learn the model.

The cost function is used to calculated the error between the predicted y^ and the actual y. This cost is used to determine how the weights are to be adjusted in what is called the gradient descent algorithm. The gradient descent uses a step size (alpha) as a hyperparameter which can be tuned. This hyperparameter is varied to determine the model that best fits the dataset.

A brief summary of the relevant keywords and functions in python is provided below:

print()	output text on console
input()	get input from user on console
range()	create a sequence of numbers
len()	gives the number of characters in a string
if	contains code that executes depending on a logical condition
else	connects with **if** and **elif**, executes when conditions are not met
elif	equivalent to **else if**
while	loops code as long as a condition is true
for	loops code through a sequence of items in an iterable object
break	exit loop immediately
continue	jump to the next iteration of the loop
def	used to define a function

Lab Task 1 - Dataset Preparation _____

Download a dataset containing several columns. You will need to select any 3 of the columns as features and one of the columns as label of the dataset. Ensure that the label is of continuous values. Load the dataset into your python program as NumPy arrays (X_{train} ,y_{train}). Print the dataset (you need to show any 5 rows of the dataset).

Code

```
# Task 1
import pandas as pd
import numpy as np

# Loading the dataset
df = pd.read_csv('ML_Lab5_Dataset_ver5.csv')
df = df.drop_duplicates()
print(df.head())

# Customixing the dataset
df = df[['MSSubClass', 'LotArea', 'YrSold', 'SalePrice']]
df = df.interpolate()

# Creating numpy arrays
x1_train = np.array(df['MSSubClass'])
x2_train = np.array(df['LotArea'])
x3_train = np.array(df['YrSold'])
y_train = np.array(df['SalePrice'])
print(df.head())
```

Console Output

```
   MSSubClass MSZoning  LotFrontage  LotArea LotShape LandContour Utilities  \
0          60       RL         65.0     8450      Reg         Lvl    AllPub
1          20       RL         80.0     9600      Reg         Lvl    AllPub
2          60       RL         68.0    11250      IR1         Lvl    AllPub
3          70       RL         60.0     9550      IR1         Lvl    AllPub
4          60       RL         84.0    14260      IR1         Lvl    AllPub

  LotConfig Neighborhood Condition1  ... PoolArea PoolQC Fence  MiscFeature  \
0    Inside      CollgCr       Norm  ...        0    NaN   NaN          NaN
1       FR2      Veenker      Feedr  ...        0    NaN   NaN          NaN
2    Inside      CollgCr       Norm  ...        0    NaN   NaN          NaN
3    Corner      Crawfor       Norm  ...        0    NaN   NaN          NaN
4       FR2      NoRidge       Norm  ...        0    NaN   NaN          NaN

   MiscVal  MoSold  YrSold SaleType SaleCondition SalePrice
0        0       2    2008       WD        Normal    208500
1        0       5    2007       WD        Normal    181500
2        0       9    2008       WD        Normal    223500
3        0       2    2006       WD       Abnorml    140000
4        0      12    2008       WD        Normal    250000

[5 rows x 77 columns]
   MSSubClass  LotArea  YrSold  SalePrice
0          60     8450    2008     208500
1          20     9600    2007     181500
2          60    11250    2008     223500
3          70     9550    2006     140000
4          60    14260    2008     250000
```

Lab Task 2 - Feature Scaling

In the input matrix (X_{train}), use feature scaling to rescale the feature columns so that their values range from 0 to 1:

$$x_{j\,\text{scaled}}[\text{i}] = \frac{x_j[\text{i}] - x_j\_\text{min}}{x_j\_\text{max} - x_j\_\text{min}}$$

You will use these rescaled values in the upcoming tasks. Print the rescaled dataset (you need to show any 5 rows of the dataset).

Code

```
# Task 2
# Extracting minimum and maximum values
x1_min = np.min(x1_train)
x1_max = np.max(x1_train)
x2_min = np.min(x2_train)
x2_max = np.max(x2_train)
x3_min = np.min(x3_train)
x3_max = np.max(x3_train)

# Function for feature Scaling
def feature_scaling(x, x_min, x_max):
  return (x - x_min) / (x_max - x_min)

# Converting feature arrays to scaled values
x1_scaled = feature_scaling(x1_train, x1_min, x1_max)
x2_scaled = feature_scaling(x2_train, x2_min, x2_max)
x3_scaled = feature_scaling(x3_train, x3_min, x3_max)

# Printing the first 5 values with scaled values
print('X1 scaled: ', x1_scaled[:5])
print('X2 scaled: ', x2_scaled[:5])
print('X3 scaled: ', x3_scaled[:5])
```

Console Ouput

```
X1 scaled:  [0.23529412 0.         0.23529412 0.29411765 0.23529412]
X2 scaled:  [0.0334198  0.03879502 0.04650728 0.03856131 0.06057632]
X3 scaled:  [0.5  0.25 0.5  0.   0.5 ]
```

Lab Task 3 - Cost Function _____

For linear regression, you will implement the following hypothesis:

$$h(x) = b + w_1x_1 + w_2x_2 + w_3x_3$$

The w_j and b represent the weights while the x_j represents the features. The feature number is denoted by j. The linear hypothesis h(x) is to be calculated for each training example and its difference with the label y of that training example will represent the loss. Initialize the weights and bias to random values between 0 and 1.

In this task, you will write a cost function that calculates the overall loss across a set of training examples:

```
cost_function(X, y)
```

The X and y are the features and labels of the training dataset. The function will return the cost value. The cost function is given by:

$$J(w, b) = \frac{1}{2m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2$$

The m is the number of the training examples in the dataset. Write the code for the cost function and implement it to print out the cost. Provide the code and all relevant screenshots of the final output.

Code

```
# Task 3
# Defining Hypothesis
def hypothesis(x1, x2, x3, w1, w2, w3, b):
  return w1 * x1 + w2 * x2 + w3 * x3 + b
```

```
# Writing cost function
def cost_function(x1, x2, x3, y, w1, w2, w3, b):
  m = len(y)
  cost = 0

  for i in range(m):
    cost += (hypothesis(x1[i], x2[i], x3[i], w1, w2, w3, b) - y[i]) ** 2

  return cost / (2 * m)

# Printing the cost of the current dataset
cost = cost_function(x1_scaled, x2_scaled, x3_scaled, y_train, 0.1, 0.1, 0.1,
0.1)
print('Cost: ', cost)
```

Output Console

```
Cost:  19519602987.904007
```

Lab Task 4 - Gradient Descent Algorithm _____

In this task, you will write a function that uses gradient descent to update the weight parameters:

```
gradient_descent(X, y, alpha)
```

The X and y are the features and labels of the training dataset, *alpha* is the learning rate which is a tuning hyperparameter. The gradient descent algorithm is given as follows:

$$dw_j = \frac{\partial \mathrm{J}}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m}(\mathrm{h}(\mathrm{x}^{(i)}) - \mathrm{y}^{(i)})x_j{}^{(i)}$$

$$db = \frac{\partial \mathrm{J}}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(\mathrm{h}(\mathrm{x}^{(i)}) - \mathrm{y}^{(i)})$$

$$w_j := w_j - \alpha\frac{\partial \mathrm{J}}{\partial w_j}$$

$$b := b - \alpha\frac{\partial \mathrm{J}}{\partial w_j}$$

For the submission, you will need to run the gradient descent algorithm once to update the weights. You will need to print the weights and cost both before and after the weight update. Provide the code and all relevant screenshots of the final output.

Code

```
# Task 4
w1 = 0.1
w2 = 0.1
w3 = 0.1
b = 0.1
alpha = 0.01

# Creating Gradient Descent
def gradient_descent(x1, x2, x3, y, w1, w2, w3, b, alpha):
  weight1 = 0
  weight2 = 0
  weight3 = 0
  bias = 0
  m = len(x1)

  for i in range(m):
    weight1 += (hypothesis(x1[i], x2[i], x3[i], w1, w2, w3, b) - y[i]) *
x1[i]
    weight2 += (hypothesis(x1[i], x2[i], x3[i], w1, w2, w3, b) - y[i]) *
x2[i]
    weight3 += (hypothesis(x1[i], x2[i], x3[i], w1, w2, w3, b) - y[i]) *
x3[i]
    bias += (hypothesis(x1[i], x2[i], x3[i], w1, w2, w3, b) - y[i])

  return (w1 - (alpha / m) * weight1), (w2 - (alpha / m) * weight2), (w3 -
(alpha / m) * weight3), (b - (alpha / m) * bias)

# Printing values of weights, bias, and cost before and after a single run
print("W1 before run: ", w1)
print("W2 before run: ", w2)
print("W3 before run: ", w3)
print("b before run: ", b)
print("Cost before run: ", cost_function(x1_scaled, x2_scaled, x3_scaled,
y_train, w1, w2, w3, b))

w1, w2, w3, b = gradient_descent(x1_scaled, x2_scaled, x3_scaled, y_train,
w1, w2, w3, b, alpha)
```

```
print("W1 after run: ", w1)
print("W2 after run: ", w2)
print("W3 after run: ", w3)
print("b after run: ", b)
print("Cost after run: ", cost_function(x1_scaled, x2_scaled, x3_scaled,
y_train, w1, w2, w3, b))
```

Output Console

```
W1 before run:  0.1
W2 before run:  0.1
W3 before run:  0.1
b before run:  0.1
Cost before run:  19519602987.904007
W1 after run:  376.1264349230752
W2 after run:  87.81346592300623
W3 after run:  813.7461890880768
b after run:  1809.3102448426855
Cost after run:  19113770819.982124
```

Lab Task 5 – Training across Epochs _____

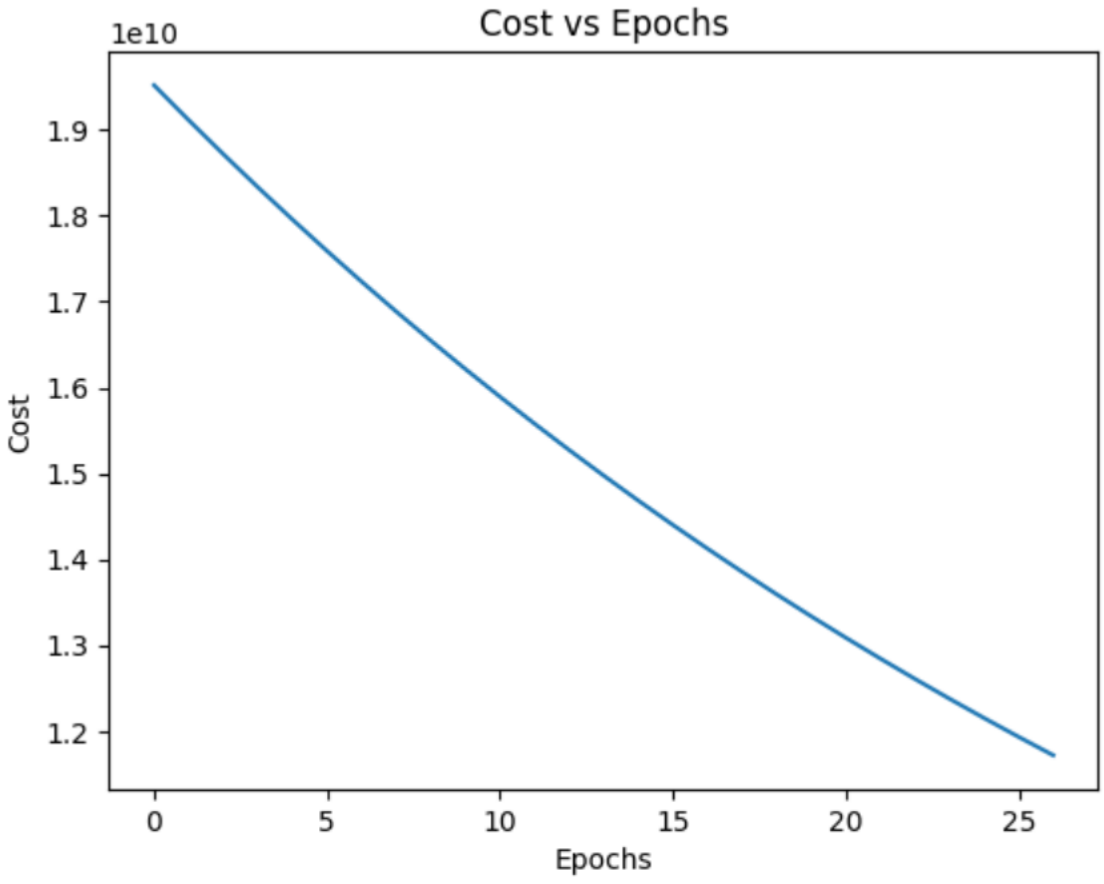
In this task, you will use the functions from the previous two tasks to write a "main" function that performs the actual training. Use the cost function on the entire training dataset to determine the training loss. You will need to store this training loss for later plotting. Next, use the gradient descent function to update the weights and bias. This iteration over the entire dataset is called an "epoch". You will need to perform the training over several epochs (the epoch number is a hyperparameter you must select at the start of the training). Thus, you will compute training loss and weight update at each epoch. At the last epoch, note down the final weight values and plot the training loss (y-axis) over the epochs (x-axis). Provide the code (excluding function definitions of tasks 2 and 3) as well as all relevant screenshots of the final output.

Code

```
# Task 5
import matplotlib.pyplot as plt

# Initial values of parameters
w1_history = [0.1]
w2_history = [0.1]
w3_history = [0.1]
b_history = [0.1]
cost = []
alpha = 0.01
epochs = 25

# Initial Cost
cost.append(cost_function(x1_scaled, x2_scaled, x3_scaled, y_train,
w1_history[0], w2_history[0], w3_history[0], b_history[0]))

# Calculating new values of weights, bias, and cost to store
for i in range(epochs + 1):
  w1, w2, w3, b = gradient_descent(x1_scaled, x2_scaled, x3_scaled, y_train,
w1_history[i], w2_history[i], w3_history[i], b_history[i], alpha)
```

```
  w1_history.append(w1)
  w2_history.append(w2)
  w3_history.append(w3)
  b_history.append(b)
  cost.append(cost_function(x1_scaled, x2_scaled, x3_scaled, y_train, w1, w2,
w3, b))

# Plotting cost over each epoch
plt.plot(cost)
plt.xlabel('Epochs')
plt.ylabel('Cost')
plt.title('Cost vs Epochs')
plt.show()
```

Output Console



Lab Task 6 – Hyperparameter Tuning _____

In this task, you will use your code from the previous task. Tune the alpha hyperparameter at different values to get various plots. You will need to provide at least 3 plots. Mention the alpha value in the plot titles. Ensure all the axes are labeled appropriately. Note down the weights at the final epochs.

Code

```
# Task 6
alpha = [0.1, 0.01, 0.001]

# Using multiple values of alpha to tune weights
for a in alpha:
  # Initial values of parameters
  w1_history = [0.1]
  w2_history = [0.1]
  w3_history = [0.1]
  b_history = [0.1]
  cost = []
  epochs = 25

  # Initial Cost
  cost.append(cost_function(x1_scaled, x2_scaled, x3_scaled, y_train,
w1_history[0], w2_history[0], w3_history[0], b_history[0]))

  # Calculating new values of weights, bias, and cost to store
  for i in range(epochs + 1):
    w1, w2, w3, b = gradient_descent(x1_scaled, x2_scaled, x3_scaled,
y_train, w1_history[i], w2_history[i], w3_history[i], b_history[i], a)
    w1_history.append(w1)
    w2_history.append(w2)
    w3_history.append(w3)
    b_history.append(b)
    cost.append(cost_function(x1_scaled, x2_scaled, x3_scaled, y_train, w1,
w2, w3, b))
```

```
# Plotting cost over each epoch
plt.plot(cost)
plt.xlabel('Epochs')
plt.ylabel('Cost')
plt.title('Cost vs Epochs for alpha = '+ str(a))
plt.show()
```

Output Console