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 16, 2025

**Semester:** 7th **Group:** 02

## CS471 Machine Learning

## Lab 7: Linear Regression II – Train-Validation Split and Regularization

| | | 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 | 403214 | | | | | |
| Irfa Farooq | 412564 | | | | | |

## Introduction

This laboratory exercise will extend the python implementation of linear regression performed in the previous lab. Linear regression is a basic supervised learning technique in which parameters are trained on a dataset to fit a model that best approximates that dataset. The problem with using simple linear regression is that the trained models can overfit the dataset at which point regularization must be used to prevent overfitting. This lab will focus on integrating the regularization concept into the gradient descent algorithm.

## Objectives

The following are the main objectives of this lab:

- Extract and prepare the training and cross-validation datasets
- Use feature scaling to ensure uniformity among the feature columns
- Implement cost function on both training and cross-validation datasets
- Implement gradient descent algorithm
- Plot the training and cross-validation losses
- Use L2 regularization to counter overfitting

## 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.

A major problem in the training is that the weights that are trained may fit the model for only the data it is given. This means that the model will not generalize to examples outside the dataset and is referred to as "overfitting". Such overfitting makes the machine learning implementation very impractical for real-life applications where data has high variation. To prevent overfitting of the model, a modification in the cost function and gradient descent is implemented. This modification is called regularization and is itself controlled by a hyperparameter (lambda).

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, Feature Scaling _____

You have been provided with a dataset containing several feature columns. You will need to select any 3 of the feature columns to make your own dataset. The "Sale Price" is the label column that your model will predict. The dataset examples are to be divided into 2 separate portions: training and cross-validation datasets (choose from 80-20 to 70-30 ratios). Save the prepared datasets as CSV files. Next, load the datasets into your python program and store them as NumPy arrays ($X_{train}$ , $y_{train}$, $X_{val}$, $y_{val}$,). Next, use feature scaling to rescale the feature columns of both datasets so that their values range from 0 to 1. Finally, print both of the datasets (you need to show any 5 rows of the datasets).

**Code**

```
# Task 1
import pandas as pd
import numpy as np
import sklearn as sk
from sklearn.model_selection import train_test_split

# Loading the dataset
df = pd.read_csv('ML_Lab5_6_Dataset_ver5.csv')
df = df.drop_duplicates()
print(df.head())

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

# Creating numpy arrays and splitting data
x1 = np.array(df['MSSubClass'])
x1_train, x1_test = train_test_split(x1, test_size = 0.2, random_state = 42)
x2 = np.array(df['LotArea'])
x2_train, x2_test = train_test_split(x2, test_size = 0.2, random_state = 42)
x3 = np.array(df['YrSold'])
x3_train, x3_test = train_test_split(x3, test_size = 0.2, random_state = 42)
```

```
y = np.array(df['SalePrice'])
y_train, y_test = train_test_split(y, test_size = 0.2, random_state = 42)
print(df.head())

# Saving the data set
df.to_csv('Customized_Lab6.csv', index = False)

# Feature Scaling
# Extracting minimum and maximum values from the training data
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('\nScaled Features:')
print('X1 scaled: ', x1_scaled[:5])
print('X2 scaled: ', x2_scaled[:5])
print('X3 scaled: ', x3_scaled[:5])

# Feature SCaled dataset
df_scaled = pd.DataFrame({
    'x1_scaled': x1_scaled,
    'x2_scaled': x2_scaled,
    'x3_scaled': x3_scaled,
    'y': y_train
})
df_scaled.to_csv('Scaled_Lab6.csv', index = False)
```

```
# Importing and printing the customized dataset
df = pd.read_csv('Customized_Lab6.csv')
print('\nCustomized Dataset:')
print(df.head())

df = pd.read_csv('Scaled_Lab6.csv')
print('\nScaled Dataset:')
print(df.head())
```

**Output Console**

```
   MSSubClass MSZoning  LotFrontage  ...  SaleType SaleCondition SalePrice
0          60       RL         65.0  ...        WD        Normal    208500
1          20       RL         80.0  ...        WD        Normal    181500
2          60       RL         68.0  ...        WD        Normal    223500
3          70       RL         60.0  ...        WD       Abnorml    140000
4          60       RL         84.0  ...        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

Scaled Features:
X1 scaled:  [0.         0.23529412 0.05882353 0.17647059 0.17647059]
X2 scaled:  [0.0331861  0.03055458 0.03494823 0.02757718 0.01729416]
X3 scaled:  [1.   0.75 0.5  0.25 1.  ]

Customized Dataset:
   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

Scaled Dataset:
   x1_scaled  x2_scaled  x3_scaled       y
0   0.000000   0.033186       1.00  145000
1   0.235294   0.030555       0.75  178000
2   0.058824   0.034948       0.50   85000
3   0.176471   0.027577       0.25  175000
4   0.176471   0.017294       1.00  127000
```

## Lab Task 2 - Cost Function with Regularization _____

For linear regression, you will implement the following hypothesis:

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

The $w_j$ and b represent the weights while the $x_j$ represents the $j^{th}$ feature. 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. In this task, you will write a cost function that calculates the overall loss across a set of examples. This cost function will be useful to calculate the losses in both the training and cross-validation phases of the program.

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

The X and y are the features and labels of either the training or the cross-validation datasets. This is useful as it can be used for either the training examples or the cross-validation examples of the dataset. The *lambd* is the regularization parameter (Note that ***lambda*** is a keyword reserved in python). The function will calculate the losses to return the overall cost value. The cost function is given by:

$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2 + \frac{1}{n}\lambda\sum_{j=1}^{n}(w_j)^2$$

The m is the number of the examples in the dataset and n is the total number of features (or non-bias weights) in the hypothesis. Write the code for the cost function and implement it for your training and cross-validation datasets to print out the cost. Provide the code and all relevant screenshots of the final output.

**Code**

```
# Task 2
# 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, lambda_):
  m = len(y)
  n =  3 # Total number of features
  cost = 0

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

  cost = cost / (2 * m)
  cost += (lambda_ / (2 * n)) * (w1 ** 2 + w2 ** 2 + w3 ** 2)

  return cost

# 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, lambda_ = 0.1)
print('Cost: ', cost)
```

**Output Console**

```
Cost:  19442760730.707497
```

## Lab Task 3 –Gradient Descent with Regularization _____

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

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

The *alpha* is the learning rate (hyperparameter 1) and *lambd* is the regularization parameter (hyperparameter 2). The gradient descent algorithm is given as follows:

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

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

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

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

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

## Code

```
# Task 3
w1 = 0.1
w2 = 0.1
w3 = 0.1
b = 0.1
alpha = 0.01
lambda_ = 0.1

# Creating Gradient Descent
def gradient_descent(x1, x2, x3, y, w1, w2, w3, b, alpha, lambda_):
  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])

  weight1 += (lambda_ / m) * w1
  weight2 += (lambda_ / m) * w2
  weight3 += (lambda_ / m) * w3
  bias += (lambda_ / m) * b

  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, lambda_))

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

**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:  19442760730.707497
W1 after run:  376.3815885349084
W2 after run:  90.06596996997378
W3 after run:  822.6519708653461
b after run:  1814.5137042483982
Cost after run:  19033563311.227943
```

## Lab Task 4 – Training and Validation Program _____

In this task, you will use the functions from the previous two tasks to write a "main" function that performs the actual training and validation. Use the cost function and gradient descent function on the training examples to determine the training loss and update the weights respectively. Then, use the cost function on the cross-validation examples to determine the cross-validation loss. This single iteration over the entire dataset (both training and cross-validation) marks the completion of one epoch. You will need to perform the training and cross-validation over several epochs (the epoch number is another hyperparameter that must be chosen). Ensure that at the end of each epoch, the training and cross-validation losses are stored for plotting purposes. When the final epoch is performed, note down the trained parameters (weights and bias) and make plot of the training and cross-validation losses (y-axis) over the epochs (x-axis). Ensure that both of the losses appear on the same graph. You only need to show a single plot for this task. Provide the code (excluding function definitions) and all relevant screenshots of the final output.

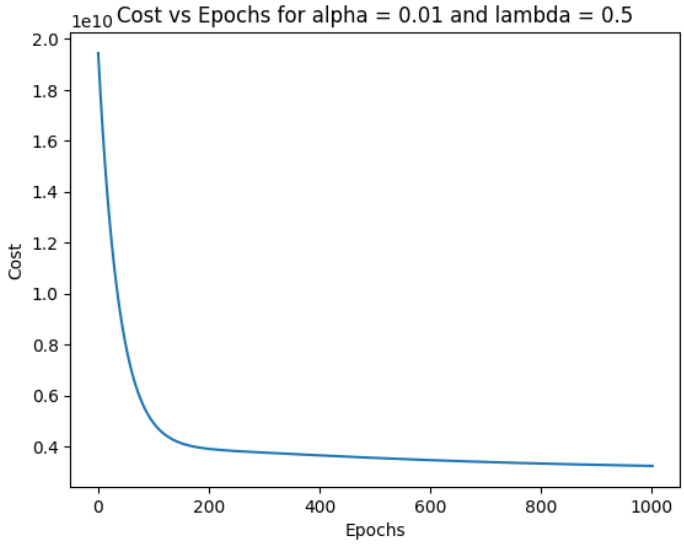
**Code**

```
# Task 4
import matplotlib.pyplot as plt

# Initials for model
lambda_ = 0.5
alpha = 0.01
epochs = 1000

def Linear_Regression(lambda_, alpha, epochs):
  # Initial values of parameters
  w1_history = [0.1]
  w2_history = [0.1]
  w3_history = [0.1]
  b_history = [0.1]
  cost = []
```

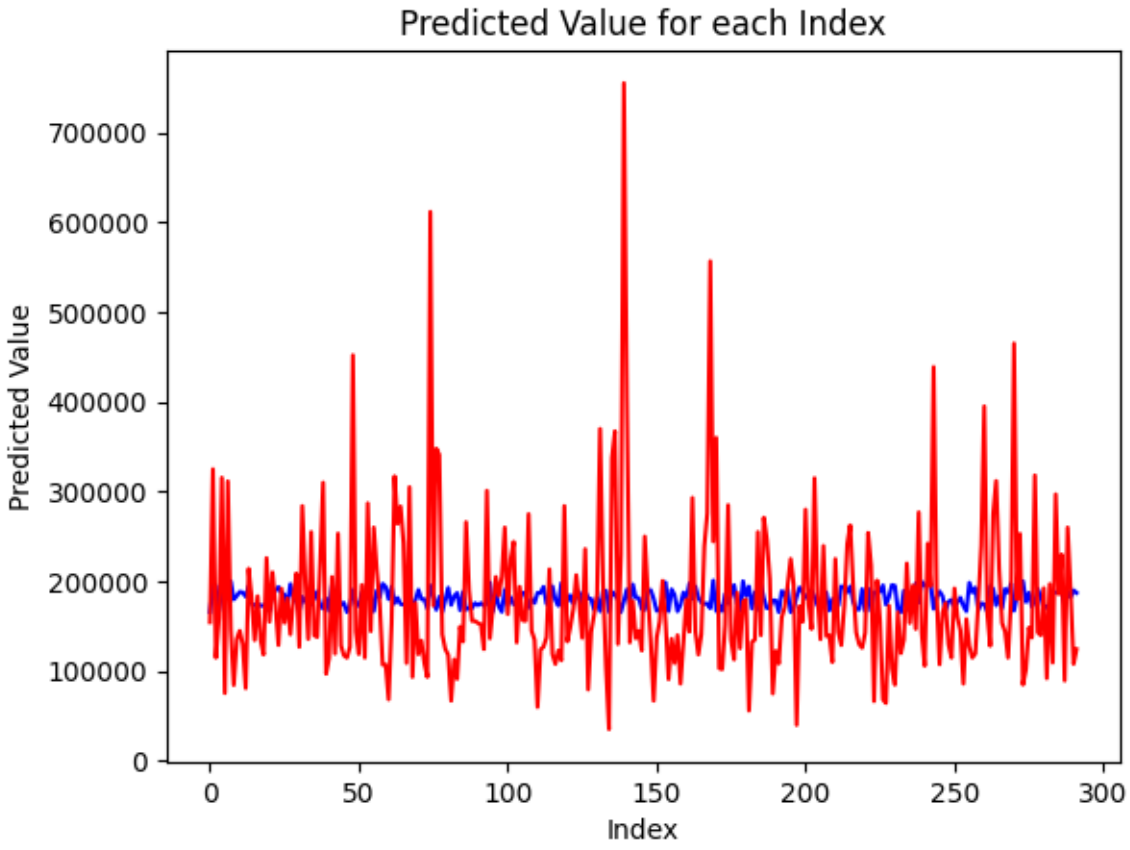
```
  # Initial Cost
  cost.append(cost_function(x1_scaled, x2_scaled, x3_scaled, y_train,
w1_history[0], w2_history[0], w3_history[0], b_history[0], lambda_))

  # 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,
lambda_)
    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, lambda_))

  # Noting the final trained parameters
  w1 = w1_history[-1]
  w2 = w2_history[-1]
  w3 = w3_history[-1]
  b = b_history[-1]

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

# Calling the model
Linear_Regression(lambda_, alpha, epochs)

# Scaling test data
x1_min = np.min(x1_test)
x1_max = np.max(x1_test)
x2_min = np.min(x2_test)
x2_max = np.max(x2_test)
x3_min = np.min(x3_test)
x3_max = np.max(x3_test)
```

```
x1_test_scaled = feature_scaling(x1_test, x1_min, x1_max)
x2_test_scaled = feature_scaling(x2_test, x2_min, x2_max)
x3_test_scaled = feature_scaling(x3_test, x3_min, x3_max)

# Applying the new parameters on test data to see the difference
test_hypothesis = []

for i in range(len(y_test)):
  test_hypothesis.append(hypothesis(x1_test_scaled[i], x2_test_scaled[i],
x3_test_scaled[i], w1, w2, w3, b))

print('Initial Test Cost: ', cost_function(x1_test_scaled, x2_test_scaled,
x3_test_scaled, y_test, w1, w2, w3, b, lambda_))

# Plotting data
plt.plot(test_hypothesis, color = 'b')
plt.plot(y_test, color = 'r')
plt.xlabel('Index')
plt.ylabel('Predicted Value')
plt.title('Predicted Value for each Index')
plt.show()
```

**Output Console**



```
Initial Test Cost:  4139873410.8585916
```



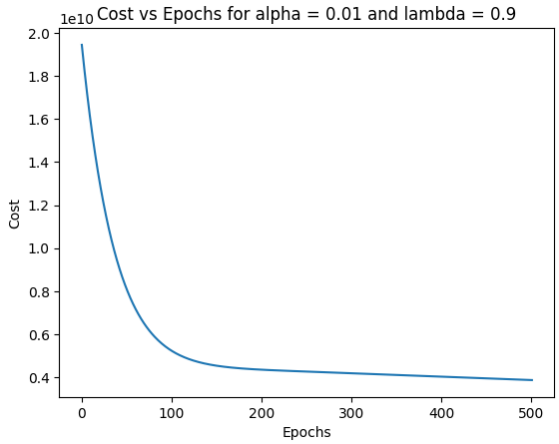
## Lab Task 5 – Tuning Alpha and Lambda

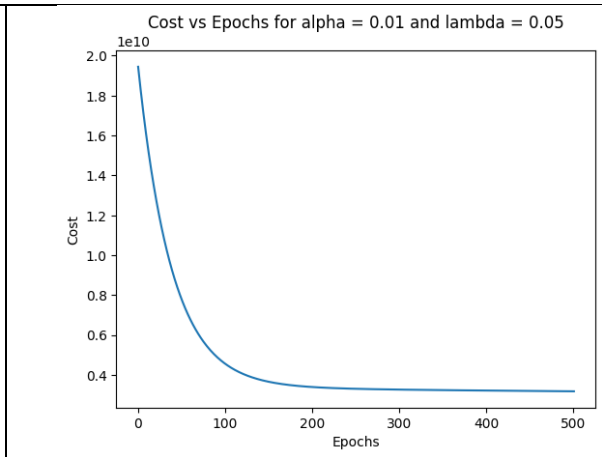
In this task, you will use your linear regression code from the previous task. Tune the alpha and lambda hyperparameters at different values to get several plots. You need to get at least 6 plots. Mention the alpha and lambda values in the plot titles. Ensure all axes are labeled appropriately.

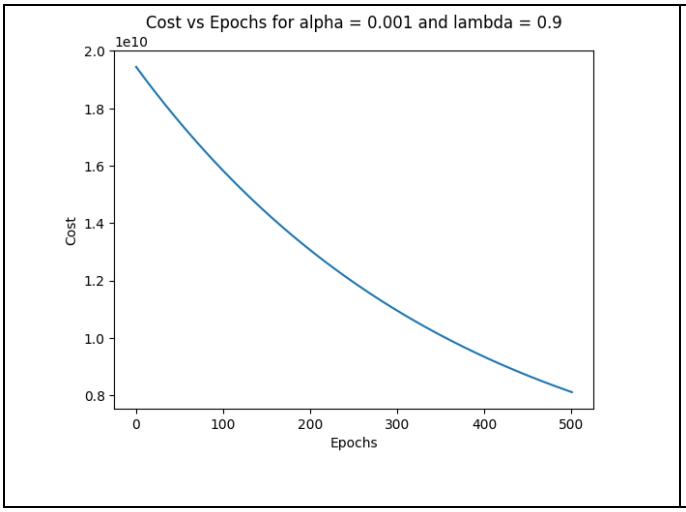
**Code**

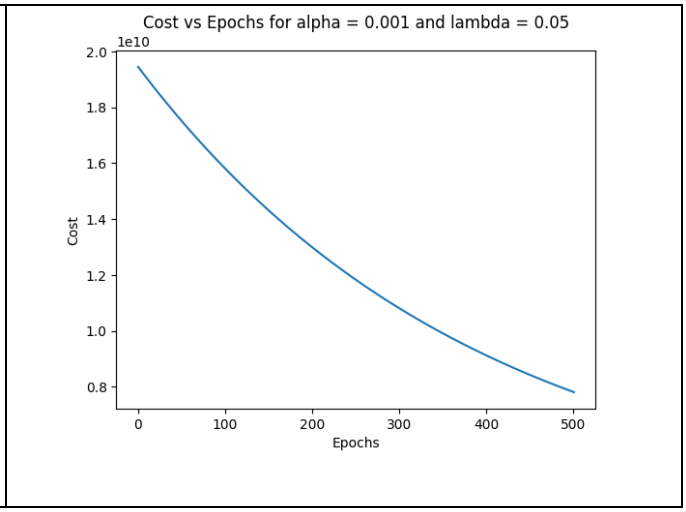
```
# Task 5
alpha = [0.01, 0.001]
lambda_ = [0.9, 0.05]

for a in alpha:
  for l in lambda_:
    Linear_Regression(l, a, epochs = 500)
```

**Output Console**




Cost vs Epochs for alpha = 0.001 and lambda = 0.9
1e10
2.0
1.8
1.6
1.4
1.2
1.0
0.8
Cost
0
100
200
300
400
500
Epochs


Cost vs Epochs for alpha = 0.001 and lambda = 0.05
1e10
2.0
1.8
1.6
1.4
1.2
1.0
0.8
Cost
0
100
200
300
400
500
Epochs