**Department of Electrical Engineering and   
Computer Science**

**Faculty Member:** Dr. Ahmad Salman **Dated:** 25/10/2023

**Semester:** 7th **Section:** BEE 12C

**CS-471 Machine Learning**

Lab 7: Linear Regression II

*Train – Test Split and Regularization*

**Group Members**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **PLO4 - CLO4** | | **PLO5 -CLO5** | **PLO8 -CLO6** | **PLO9 -CLO7** |
| **Name** | **Reg. No** | **Viva / Quiz / Lab Performance** | **Analysis of Data in Lab Report** | **Modern Tool Usage** | **Ethics and Safety** | **Individual and Teamwork** |
|  |  | **5 Marks** | **5 Marks** | **5 Marks** | **5 Marks** | **5 Marks** |
| Afif Arif Siddiqi | 344504 |  |  |  |  |  |
| Muhammad Ali Farooq | 331879 |  |  |  |  |  |
| Muhammad Ahmed Mohsin | 333060 |  |  |  |  |  |
| Danial Ahmad | 331388 |  |  |  |  |  |
| Ehtishaam Tanveer | 333074 |  |  |  |  |  |
| Muhammad Umer | 345834 |  |  |  |  |  |

**Table of Contents**

[2 Linear Regression II 3](#_Toc149249023)

[2.1 Introduction 3](#_Toc149249024)

[2.2 Objectives 3](#_Toc149249025)

[2.3 Theory 3](#_Toc149249026)

[3 Lab Tasks 4](#_Toc149249027)

[3.1 Task 1 – Dataset Preparation, Feature Scaling 4](#_Toc149249028)

[3.2 Task 2 – Cost Function with Regularization 5](#_Toc149249029)

[3.3 Task 3 – Gradient Descent with Regularization 6](#_Toc149249030)

[3.4 Task 4 – Training and Testing Program 7](#_Toc149249031)

[3.5 Task 5 – Tuning Alpha and Lambda 9](#_Toc149249032)

[4 Conclusion 11](#_Toc149249033)

# Linear Regression II

## 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 test datasets
* Use feature scaling to ensure uniformity among the feature columns
* Implement cost function on both training and test datasets
* Implement gradient descent algorithm
* Plot the training and test losses
* Use L2 regularization to counter overfitting

## 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 calculate 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).

# Lab Tasks

## 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 test 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 (Xtrain, ytrain, Xtest, ytest,). 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).

### TASK 1 CODE STARTS HERE ###

*# Load the dataset into your python program as NumPy arrays (Xtrain ,ytrain).*

dataset = pd.read\_csv(path\_data)

features = ["BsmtFinSF1", "BsmtUnfSF", "TotalBsmtSF"]

label = "SalePrice"

X = dataset[features].values

y = dataset[label].values

*def* feature\_scaling(*X*, *axis*=0):

    return (X - X.min(*axis*=axis)) / (X.max(*axis*=axis) - X.min(*axis*=axis))

*# Split the dataset into training and test datasets*

*def* random\_split(*X*, *y*, *ratio*=0.8, *scaling*=feature\_scaling):

    m = len(X)

    X = np.random.permutation(X)

    split = *int*(m \* ratio)

    return feature\_scaling(X[:split]), feature\_scaling(y[:split]), feature\_scaling(X[split:]), feature\_scaling(y[split:])

X\_train, y\_train, X\_test, y\_test = random\_split(X, y)

print("X train:")

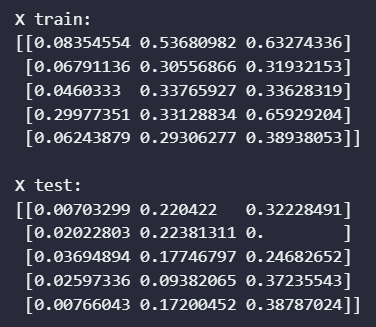
print(X\_train[:5])

print("\nX test:")

print(X\_test[:5])

### TASK 1 CODE ENDS HERE ###

### TASK 1 OUTPUT SCREENSHOT STARTS HERE ###



### TASK 1 OUTPUT SCREENSHOT ENDS HERE ###

## Task 2 – Cost Function with Regularization

For linear regression, you will implement the following hypothesis:

h(x) = w0 + w1x1 + w2x2 + w3x3 + …

The wj and b represent the weights while the xj represents the jth 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 test phases of the program.

cost\_function(X, y, lambd)

The X and y are the features and labels of either the training or the test datasets. This is useful as it can be used for either the training examples or the test 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:

The m is the number of 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 test datasets to print out the cost. Provide the code and all relevant screenshots of the final output.

### TASK 2 CODE STARTS HERE ###

*def* hypothesis(*X*, *w*, *b*):

    return b + np.dot(X, w)

*def* cost\_function(*X*, *y*, *w*, *b*, *lambda\_*):

    assert len(X) == len(y), "X and y must have the same length"

    m = len(X)

    n = X.shape[1]

    J = 1 / (2 \* m) \* np.sum((hypothesis(X, w, b) - y) \*\* 2) + 1 / n \* (

        lambda\_

    ) \* np.sum(w\*\*2)

    return J

*# Initialize the weights and bias to random values between 0 and 1.*

w = np.random.rand(3)

b = np.random.rand(1)

lambda\_ = 0.2

*# Verify the cost function*

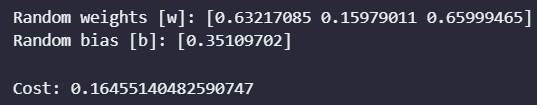
print("Random weights [w]: ", w)

print("Random bias [b]: ", b)

print("\nCost: ", cost\_function(X\_train, y\_train, w, b, lambda\_))

### TASK 2 CODE ENDS HERE ###

### TASK 2 OUTPUT SCREENSHOTS START HERE ###



### TASK 2 OUTPUT SCREENSHOTS END HERE ###

## 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:

Provide the code and all relevant screenshots of the final output.

### TASK 3 CODE STARTS HERE ###

*def* gradient\_descent(*X*, *y*, *w*, *b*, *alpha*, *lambda\_*):

    m = len(X)

    w = w - (

        alpha \* (1 / m) \* np.dot(X.T, (hypothesis(X, w, b) - y))

        + (1 / m) \* (lambda\_ \* w)

    )

    b = b - alpha \* (1 / m) \* np.sum(hypothesis(X, w, b) - y)

    return w, b

*# Initialize the weights and bias to random values between 0 and 1.*

w = np.random.rand(3)

b = np.random.rand(1)

lambda\_ = 0.2

alpha = 0.01

*# Verify the gradient descent function*

print("Gradient descent: ")

print(

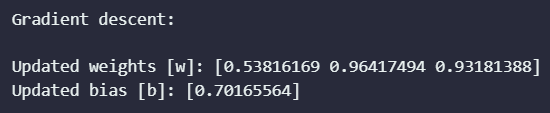
    "\nUpdated weights [w]: ", gradient\_descent(X\_train, y\_train, w, b, alpha, lambda\_)[0]

)

print("Updated bias [b]: ", gradient\_descent(X\_train, y\_train, w, b, alpha, lambda\_)[1])

### TASK 3 CODE ENDS HERE ###

### TASK 3 OUTPUT SCREENSHOT STARTS HERE ###



### TASK 3 OUTPUT SCREENSHOT ENDS HERE ###

## Task 4 – Training and Testing 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 testing. 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 test examples to determine the test loss. This single iteration over the entire dataset (both training and test) marks completion of one epoch. You will need to perform the training and testing over several epochs (the epoch number is another hyperparameter that must be chosen). Ensure that at the end of each epoch, the training and test losses are stored for plotting purposes. When the final epoch is performed, note down the trained parameters (weights and bias) and makes plot of the training and test losses (y-axis) over the epochs (x-axis). Ensure that both 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.

### TASK 4 CODE STARTS HERE ###

*def* main(*X\_train*, *y\_train*, *X\_test*, *y\_test*, *alpha*, *lambda\_*, *epochs*):

*# Initialize the weights and bias to random values between 0 and 1.*

    w = np.random.rand(3)

    b = np.random.rand(1)

*# Initialize the train and test cost arrays*

    train\_cost = []

    test\_cost = []

    for epoch in range(epochs):

*# Calculate the training and test cost*

        train\_cost.append(cost\_function(X\_train, y\_train, w, b, lambda\_))

        test\_cost.append(cost\_function(X\_test, y\_test, w, b, lambda\_))

*# Update the weights and bias*

        w, b = gradient\_descent(X\_train, y\_train, w, b, alpha, lambda\_)

*# Plot the training and test cost*

    plt.suptitle(

*f*"Training and Test Cost for $\\alpha$ = {alpha}, Epochs = {epochs}, $\\lambda$ = {lambda\_}"

    )

    plt.plot(train\_cost)

    plt.plot(test\_cost, *marker*="o", *linestyle*="none", *color*="red", *markevery*=50)

    plt.title("Training and Test Cost")

    plt.xlabel("Epochs")

    plt.ylabel("Cost")

    plt.legend(["Training Cost", "Test Cost"])

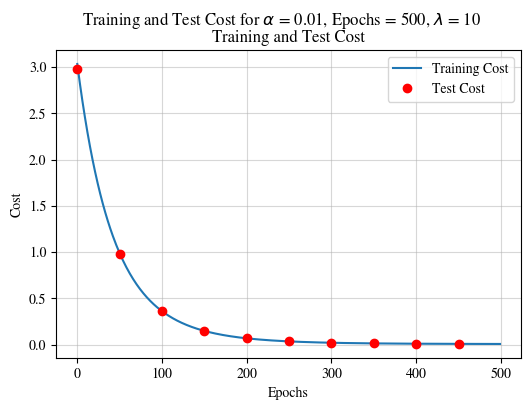
    plt.grid(*alpha*=0.5)

    plt.show()

main(X\_train, y\_train, X\_test, y\_test, 0.01, 10, 500)

### TASK 4 CODE ENDS HERE ###

### TASK 4 OUTPUT SCREENSHOTS START HERE ###



### TASK 4 OUTPUT SCREENSHOTS START HERE ###

## 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.

### TASK 5 CODE STARTS HERE ###

alpha = [0.01, 0.005, 0.001]

lambda\_ = [0.2, 1]

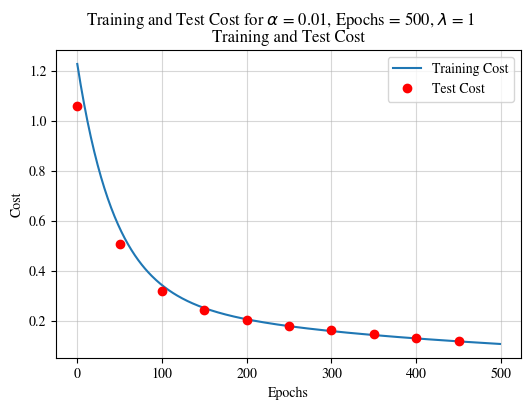
for a in alpha:

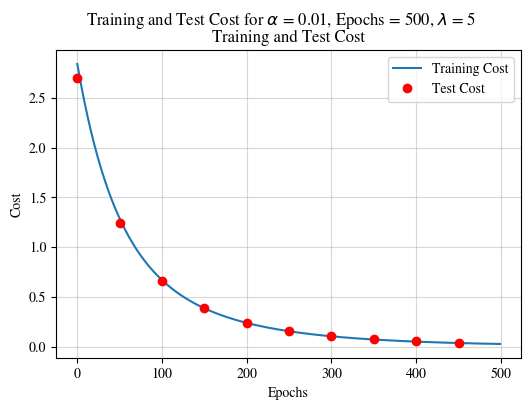
    for l in lambda\_:

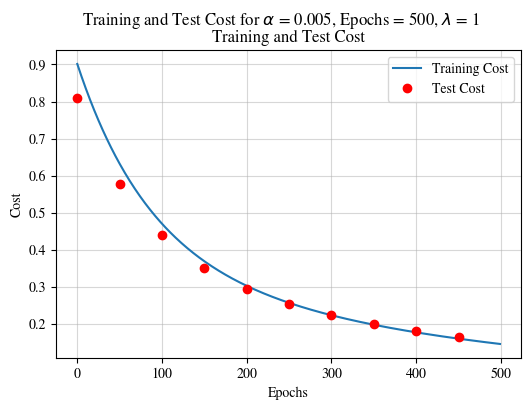
        main(X\_train, y\_train, X\_test, y\_test, a, l, 500)

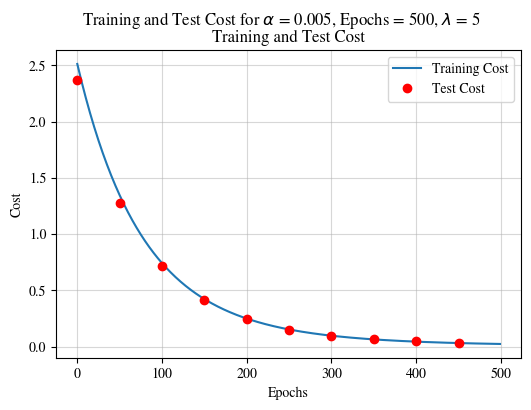
### TASK 5 CODE ENDS HERE ###

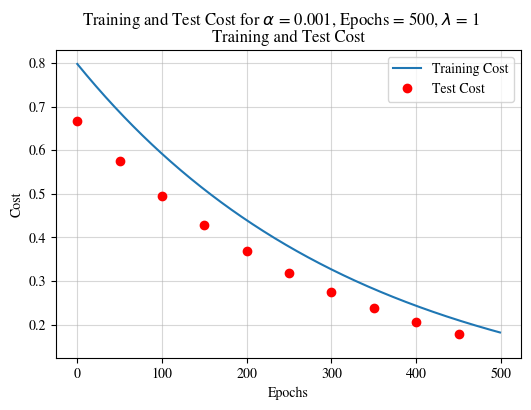
### TASK 5 OUTPUT SCREENSHOTS START HERE ###

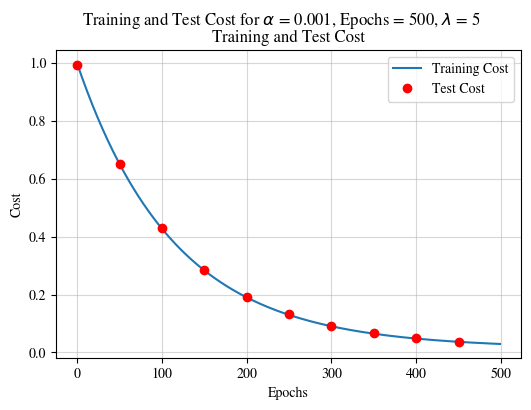












### TASK 5 OUTPUT SCREENSHOTS START HERE ###

# Conclusion

In this laboratory exercise, we extended the Python implementation of linear regression performed in the previous lab to integrate the regularization concept into the gradient descent algorithm. Regularization helps to prevent overfitting of the trained model on the training dataset. We implemented two common regularization techniques: L1 (Lasso) and L2 (Ridge) regularization. We evaluated the performance of the regularized models on a held-out test set and found that both regularization techniques improved the model's generalization performance.