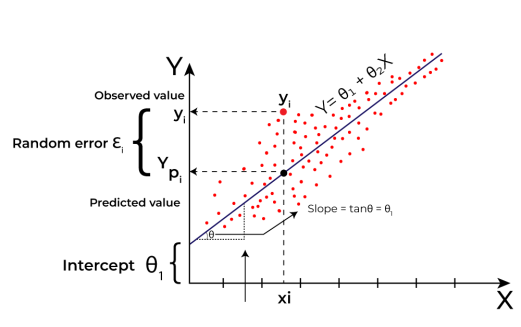
Experiment-4

Exercises to solve the real-world problems using the following machine learning methods: a) Linear Regression b) Logistic Regression c) Binary Classifier

1. Linear Regression:

Linear regression is a type of [supervised machine learning](https://www.geeksforgeeks.org/supervised-machine-learning/) algorithm that computes the linear relationship between a dependent variable and one or more independent features. When the number of the independent feature is 1 then it is known as Univariate Linear regression, and in the case of more than one feature, it is known as multivariate linear regression.



Code:

**# Import the necessary libraries:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.axes as ax

from matplotlib.animation import FuncAnimation

#### #Load the dataset and separate input and Target variables

data = pd.read\_csv(r"F:\EDUCATION\ISTS\B.Tech\ML\Lab programs\Exp4\data\_for\_lr.csv")

data

# Drop the missing values

data = data.dropna()

# training dataset and labels

train\_input = np.array(data.x[0:500]).reshape(500, 1)

train\_output = np.array(data.y[0:500]).reshape(500, 1)

# valid dataset and labels

test\_input = np.array(data.x[500:700]).reshape(199, 1)

test\_output = np.array(data.y[500:700]).reshape(199, 1)

#### # Build the Linear Regression Model and Plot the regression line

class LinearRegression:

def \_\_init\_\_(self):

self.parameters = {}

def forward\_propagation(self, train\_input):

m = self.parameters['m']

c = self.parameters['c']

predictions = np.multiply(m, train\_input) + c

return predictions

def cost\_function(self, predictions, train\_output):

cost = np.mean((train\_output - predictions) \*\* 2)

return cost

def backward\_propagation(self, train\_input, train\_output, predictions):

derivatives = {}

df = (train\_output - predictions) \* -1

dm = np.mean(np.multiply(train\_input, df))

dc = np.mean(df)

derivatives['dm'] = dm

derivatives['dc'] = dc

return derivatives

def update\_parameters(self, derivatives, learning\_rate):

self.parameters['m'] = self.parameters['m'] - \

learning\_rate \* derivatives['dm']

self.parameters['c'] = self.parameters['c'] - \

learning\_rate \* derivatives['dc']

def train(self, train\_input, train\_output, learning\_rate, iters):

# Initialize random parameters

self.parameters['m'] = np.random.uniform(0, 1) \* -1

self.parameters['c'] = np.random.uniform(0, 1) \* -1

# Initialize loss

self.loss = []

# Initialize figure and axis for animation

fig, ax = plt.subplots()

x\_vals = np.linspace(min(train\_input), max(train\_input), 100)

line, = ax.plot(x\_vals, self.parameters['m'] \* x\_vals +

self.parameters['c'], color='red', label='Regression Line')

ax.scatter(train\_input, train\_output, marker='o',

color='green', label='Training Data')

# Set y-axis limits to exclude negative values

ax.set\_ylim(0, max(train\_output) + 1)

def update(frame):

# Forward propagation

predictions = self.forward\_propagation(train\_input)

# Cost function

cost = self.cost\_function(predictions, train\_output)

# Back propagation

derivatives = self.backward\_propagation(

train\_input, train\_output, predictions)

# Update parameters

self.update\_parameters(derivatives, learning\_rate)

# Update the regression line

line.set\_ydata(self.parameters['m']

\* x\_vals + self.parameters['c'])

# Append loss and print

self.loss.append(cost)

print("Iteration = {}, Loss = {}".format(frame + 1, cost))

return line,

# Create animation

ani = FuncAnimation(fig, update, frames=iters, interval=200, blit=True)

# Save the animation as a video file (e.g., MP4)

ani.save('linear\_regression\_A.mp4', writer='ffmpeg')

plt.xlabel('Input')

plt.ylabel('Output')

plt.title('Linear Regression')

plt.legend()

plt.show()

return self.parameters, self.loss

#Example usage & out put

linear\_reg = LinearRegression ()

Parameters, loss = linear\_reg.train (train\_input, train\_output, 0.0001, 20)

1. **Logistic Regression:**

## What is Logistic Regression?

Logistic regression is used for binary [classification](https://www.geeksforgeeks.org/getting-started-with-classification/) where we use [sigmoid function](https://www.geeksforgeeks.org/derivative-of-the-sigmoid-function/), that takes input as independent variables and produces a probability value between 0 and 1.For example, we have two classes Class 0 and Class 1 if the value of the logistic function for an input is greater than 0.5 (threshold value) then it belongs to Class 1 it belongs to Class 0. It’s referred to as regression because it is the extension of[linear regression](https://www.geeksforgeeks.org/ml-linear-regression/) but is mainly used for classification problems.

### Key Points:

* Logistic regression predicts the output of a categorical dependent variable. Therefore, the outcome must be a categorical or discrete value.
* It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
* In Logistic regression, instead of fitting a regression line, we fit an “S” shaped logistic function, which predicts two maximum values (0 or 1).

