ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

DAY – 13

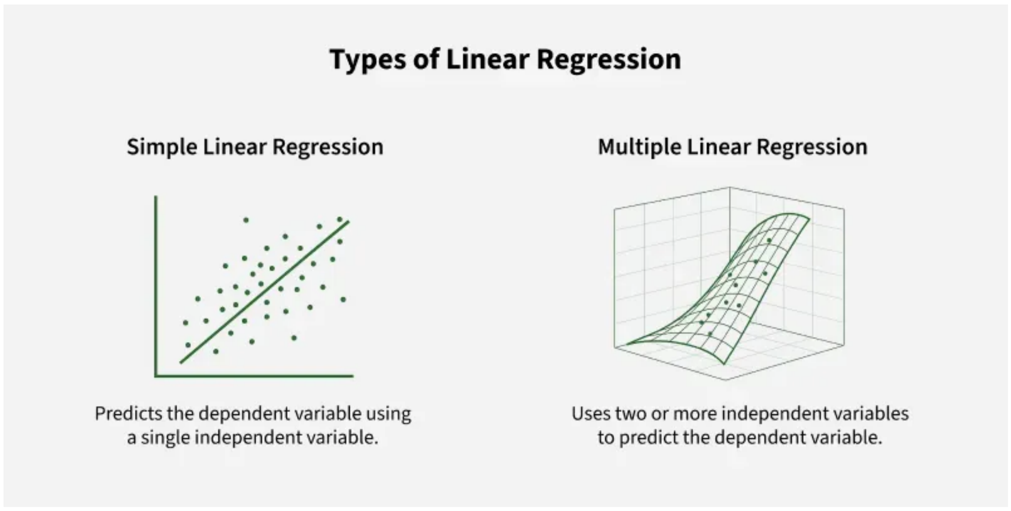
Date: Jul 09, 2025

Linear Regression

Linear regression is a type of supervised machine-learning algorithm that learns from the labelled datasets and maps the data points with most optimized linear functions which can be used for prediction on new datasets. It assumes that there is a linear relationship between the input and output, meaning the output changes at a constant rate as the input changes. This relationship is represented by a straight line.

For example we want to predict a student's exam score based on how many hours they studied. We observe that as students study more hours, their scores go up. In the example of predicting exam scores based on hours studied. Here

- **Independent variable (input):** Hours studied because it's the factor we control or observe.
- **Dependent variable (output):** Exam score because it depends on how many hours were studied.


Types of Linear Regression
Simple Linear Regression
Multiple Linear Regression
Predicts the dependent variable using a single independent variable.
Uses two or more independent variables to predict the dependent variable.

Types of Linear Regression

When there is only one independent feature it is known as Simple Linear Regression or Univariate Linear Regression and when there are more than one feature it is known as Multiple Linear Regression or Multivariate Regression.

1. Simple Linear Regression

Simple linear regression is used when we want to predict a target value (dependent variable) using only one input feature (independent variable). It assumes a straight-line relationship between the two.

Formula:

$y\hat{}=\theta_0+\theta_1 x$

Where: $y\hat{}$ is the predicted value, x is the input (independent variable), θ_0 is the intercept (value of $y\hat{}$ when x=0), θ_1 is the slope or coefficient (how much $y\hat{}$ changes with one unit of x)
Example: Predicting a person's salary (y) based on their years of experience (x).

2. Multiple Linear Regression

Multiple linear regression involves more than one independent variable and one dependent variable. The equation for multiple linear regression is:

$y\hat{}=\theta_0+\theta_1 x_1+\theta_2 x_2+\cdots+\theta_n x_n$

where: $y\hat{}$ is the predicted value, $x_1,x_2,\ldots,x_n x_1 ,x_2 ,\ldots,x_n$ are the independent variables
$\theta_1,\theta_2,\ldots,\theta_n\theta_1 ,\theta_2 ,\ldots,\theta_n$ are the coefficients (weights) corresponding to each predictor.
θ_0 is the intercept.
The goal of the algorithm is to find the best Fit Line equation that can predict the values based on the independent variables.

Use Case of Multiple Linear Regression

Multiple linear regression allows us to analyze relationship between multiple independent variables and a single dependent variable. Here are some use cases:

- **Real Estate Pricing:** In real estate MLR is used to predict property prices based on multiple factors such as location, size, number of bedrooms, etc. This helps buyers and sellers understand market trends and set competitive prices.
- **Financial Forecasting:** Financial analysts use MLR to predict stock prices or economic indicators based on multiple influencing factors such as interest rates, inflation rates and market trends. This enables better investment strategies and risk management24.
- **Agricultural Yield Prediction:** Farmers can use MLR to estimate crop yields based on several variables like rainfall, temperature, soil quality and fertilizer usage. This information helps in planning agricultural practices for optimal productivity
- **E-commerce Sales Analysis:** An e-commerce company can utilize MLR to assess how various factors such as product price, marketing promotions and seasonal trends impact sales.

Python Implementation of Linear Regression

1. 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
```

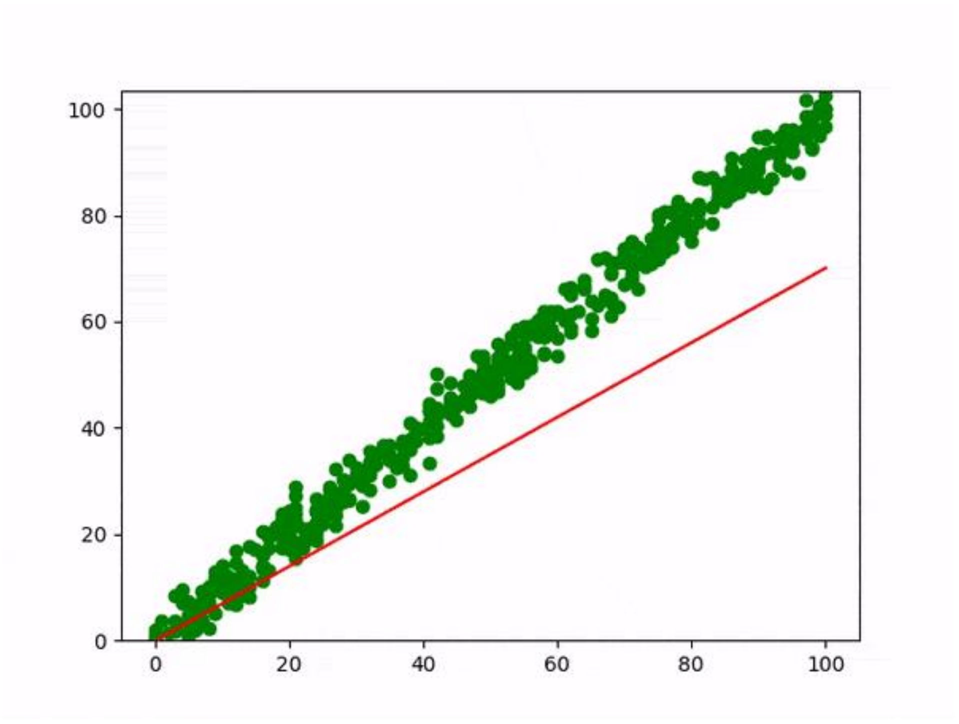
2. Load the dataset and separate input and Target variables

```
url = 'https://media.geeksforgeeks.org/wp-content/uploads/20240320114716/data_for_lr.csv'
data = pd.read_csv(url)

data = data.dropna()

train_input = np.array(data.x[0:500]).reshape(500, 1)
train_output = np.array(data.y[0:500]).reshape(500, 1)

test_input = np.array(data.x[500:700]).reshape(199, 1)
test_output = np.array(data.y[500:700]).reshape(199, 1)
```

3. Build the Linear Regression Model and Plot the regression line

In forward propagation Linear regression function Y=mx+cY=mx+c is applied by initially assigning random value of parameter (m and c). The we have written the function to finding the cost function i.e the mean

```
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 = (predictions-train_output)
        dm = 2 * np.mean(np.multiply(train_input, df))
        dc = 2 * 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):
        self.parameters['m'] = np.random.uniform(0, 1) * -1
        self.parameters['c'] = np.random.uniform(0, 1) * -1

        self.loss = []

        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')

        ax.set_ylim(0, max(train_output) + 1)

        def update(frame):
            predictions = self.forward_propagation(train_input)
            cost = self.cost_function(predictions, train_output)
            derivatives = self.backward_propagation(train_input, train_output, predictions)
            self.update_parameters(derivatives, learning_rate)
            line.set_ydata(self.parameters['m'] * x_vals + self.parameters['c'])
            self.loss.append(cost)
            print("Iteration = {}, Loss = {}".format(frame + 1, cost))
            return line,

        ani = FuncAnimation(fig, update, frames=iters, interval=200, blit=True)
        ani.save('linear_regression_A.gif', writer='ffmpeg')

        plt.xlabel('Input')

        plt.ylabel('Output')
        plt.title('Linear Regression')
        plt.legend()
        plt.show()

        return self.parameters, self.loss
```



The linear regression line provides valuable insights into the relationship between the two variables. It represents the best-fitting line that captures the overall trend of how a dependent variable (Y) changes in response to variations in an independent variable (X).

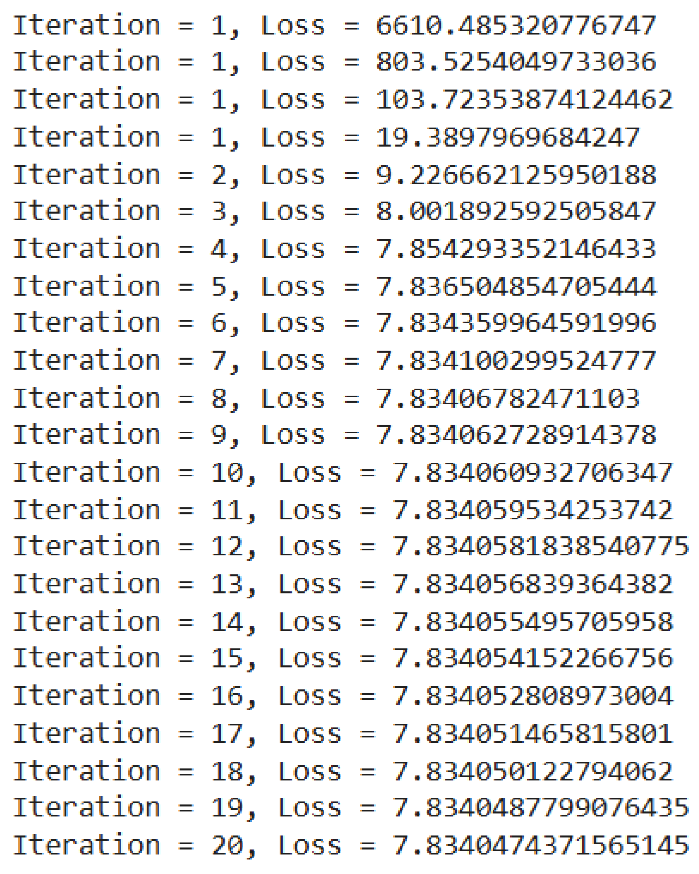
Positive Linear Regression Line: A positive linear regression line indicates a direct relationship between the independent variable (XX) and the dependent variable (YY). This means that as the value of X increases, the value of Y also increases. The slope of a positive linear regression line is positive, meaning that the line slants upward from left to right.

Negative Linear Regression Line: A negative linear regression line indicates an inverse relationship between the independent variable (XX) and the dependent variable (YY). This means that as the value of X increases, the value of Y decreases. The slope of a negative linear regression line is negative, meaning that the line slants downward from left to right.

4. Trained the model and Final Prediction

```
linear_reg = LinearRegression()
parameters, loss = linear_reg.train(train_input, train_output, 0.0001, 20)
```

Output:



```
Iteration = 1, Loss = 6610.485320776747
Iteration = 1, Loss = 803.5254049733036
Iteration = 1, Loss = 103.72353874124462
Iteration = 1, Loss = 19.3897969684247
Iteration = 2, Loss = 9.226662125950188
Iteration = 3, Loss = 8.001892592505847
Iteration = 4, Loss = 7.854293352146433
Iteration = 5, Loss = 7.836504854705444
Iteration = 6, Loss = 7.834359964591996
Iteration = 7, Loss = 7.834100299524777
Iteration = 8, Loss = 7.83406782471103
Iteration = 9, Loss = 7.834062728914378
Iteration = 10, Loss = 7.834060932706347
Iteration = 11, Loss = 7.834059534253742
Iteration = 12, Loss = 7.8340581838540775
Iteration = 13, Loss = 7.834056839364382
Iteration = 14, Loss = 7.834055495705958
Iteration = 15, Loss = 7.834054152266756
Iteration = 16, Loss = 7.834052808973004
Iteration = 17, Loss = 7.834051465815801
Iteration = 18, Loss = 7.834050122794062
Iteration = 19, Loss = 7.8340487799076435
Iteration = 20, Loss = 7.8340474371565145
```