Linear Regression in Machine learning

Machine Learning is a branch of Artificial intelligence that focuses on the development of algorithms and statistical models that can learn from and make predictions on data. Linear regression is also a type of machine-learning algorithm more specifically a supervised machine-learning algorithm that learns from the labeled datasets and maps the data points to the most optimized linear functions. which can be used for prediction on new datasets.

First of we should know what is supervised machine learning algorithms. It is a type of machine learning where the algorithm learns from labeled data. Labeled data means the dataset whose respective target value is already known. Supervised learning has two types:

- **Classification**: It predicts the class of the dataset based on the independent input variable. Class is the categorical or discrete values. like the image of an animal is a cat or dog?
- **Regression**: It predicts the continuous output variables based on the independent input variable. like the prediction of house prices based on different parameters like house age, distance from the main road, location, area, etc.

Here, we will discuss one of the simplest types of regression i.e **Linear Regression.**

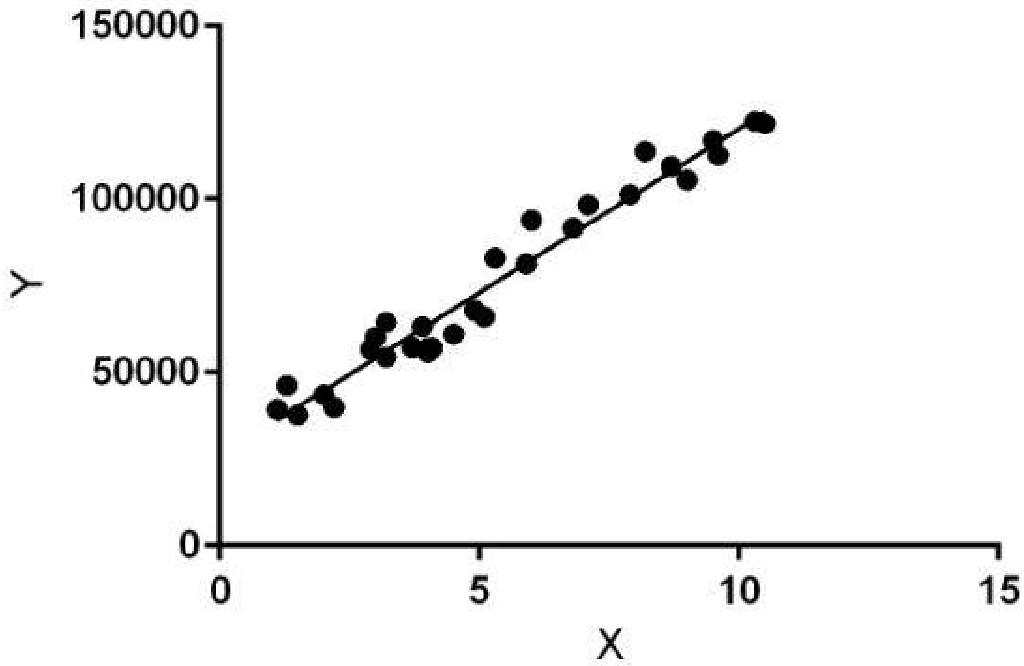
Linear Regression

Linear regression is a type of 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 algorithm is to find the best linear equation that can predict the value of the dependent variable based on the independent variables. The equation provides a straight line that represents the relationship between the dependent and independent variables. The slope of the line indicates how much the dependent variable changes for a unit change in the independent variable(s).

Linear regression is used in many different fields, including finance, economics, and psychology, to understand and predict the behavior of a particular variable. For example, in finance, linear regression might be used to understand the relationship between a company's stock price and its earnings or to predict the future value of a currency based on its past performance.

One of the most important supervised learning tasks is regression. In regression set of records are present with X and Y values and these values are used to learn a function so if you want to predict Y from an unknown X this learned function can be used. In regression we have to find the value of Y, So, a function is required that predicts continuous Y in the case of regression given X as independent features.

Here Y is called a dependent or target variable and X is called an independent variable also known as the predictor of Y. There are many types of functions or modules that can be used for regression. A linear function is the simplest type of function. Here, X may be a single feature or multiple features representing the problem.



Linear Regression

Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x)). Hence, the name is Linear Regression. In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best-fit line for our model.

Assumption for Linear Regression Model

Linear regression is a powerful tool for understanding and predicting the behavior of a variable, however, it needs to meet a few conditions in order to be accurate and dependable solutions.

1. **Linearity**: The independent and dependent variables have a linear relationship with one another. This implies that changes in the dependent variable follow those in the independent variable(s) in a linear fashion.
2. **Independence**: The observations in the dataset are independent of each other. This means that the value of the dependent variable for one observation does not depend on the value of the dependent variable for another observation.
3. **Homoscedasticity**: Across all levels of the independent variable(s), the

independent variable(s) has no impact on the variance of the errors.

4. **Normality**: The errors in the model are normally distributed.
5. **No multicollinearity**: There is no high correlation between the independent variables. This indicates that there is little or no correlation between the independent variables.

Hypothesis function for Linear Regression :

As we have assumed earlier that our independent feature is the experience i.e X and the respective salary Y is the dependent variable. Let's assume there is a linear relationship between X and Y then the salary can be predicted using:

$$\hat{Y} = \theta_1 + \theta_2 X$$

OR

$$\hat{y}_i = \theta_1 + \theta_2 x_i$$

Here,

- $y_i \epsilon Y \ \ (i = 1, 2, \cdots, n)$ are labels to data (Supervised learning)
- $x_i \epsilon X \ \ (i = 1, 2, \cdots, n)$ are the input independent training data (univariate – one input variable(parameter))
- $\hat{y}_i \epsilon \hat{Y} \ \ (i = 1, 2, \cdots, n)$ are the predicted values.

The model gets the best regression fit line by finding the best θ_1 and θ_2 values.

- **θ_1:** intercept
- **θ_2:** coefficient of x

Once we find the best θ_1 and θ_2 values, we get the best-fit line. So when we are finally using our model for prediction, it will predict the value of y for the input value of x.

Cost function

The cost function or the loss function is nothing but the error or difference

Error (MSE) between the predicted value and the true value. The cost function (J) can be written as:

$$\text{Cost function}(J) = \frac{1}{n}\sum_{n}^{i}(\hat{y}_i - y_i)^2$$

How to update θ_1 and θ_2 values to get the best-fit line?

To achieve the best-fit regression line, the model aims to predict the target value $\hat{Y}$ such that the error difference between the predicted value $\hat{Y}$ and the true value Y is minimum. So, it is very important to update the θ_1 and θ_2 values, to reach the best value that minimizes the error between the predicted y value (pred) and the true y value (y).

$$minimize\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

Gradient Descent:

A linear regression model can be trained using the optimization algorithm gradient descent by iteratively modifying the model's parameters to reduce the mean squared error (MSE) of the model on a training dataset. To update θ_1 and θ_2 values in order to reduce the Cost function (minimizing RMSE value) and achieve the best-fit line the model uses Gradient Descent. The idea is to start with random θ_1 and θ_2 values and then iteratively update the values, reaching minimum cost.

A gradient is nothing but a derivative that defines the effects on outputs of the function with a little bit of variation in inputs.

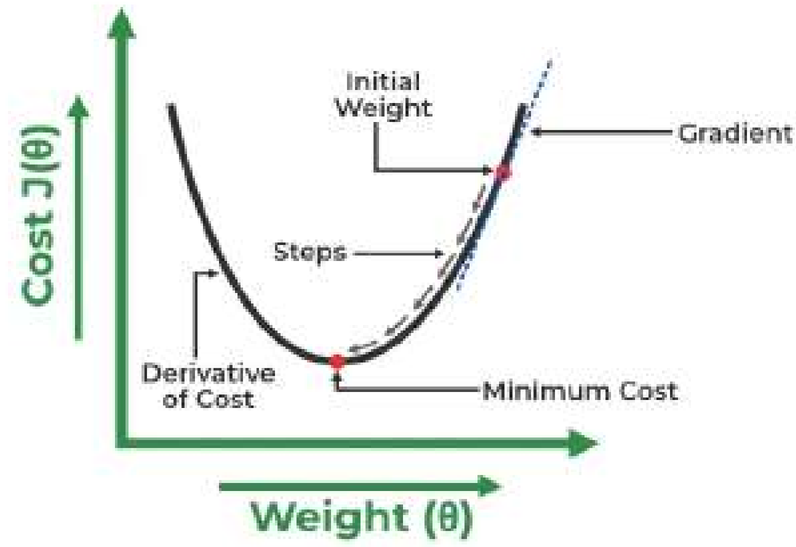
Let's differentiate the cost function(J) with respect to θ_1

$$\begin{aligned}
J'_{\theta_1} &= \frac{\partial J(\theta_1,\theta_2)}{\partial \theta_1} \\
&= \frac{\partial}{\partial \theta_1}\left[\frac{1}{n}\left(\sum_{i=1}^{n}(\hat{y}_i - y_i)^2\right)\right] \\
&= \frac{1}{n}\left[\sum_{i=1}^{n}2(\hat{y}_i - y_i)\left(\frac{\partial}{\partial \theta_1}(\hat{y}_i - y_i)\right)\right] \\
&= \frac{1}{n}\left[\sum_{i=1}^{n}2(\hat{y}_i - y_i)\left(\frac{\partial}{\partial \theta_1}(\theta_1 + \theta_2 x_i - y_i)\right)\right] \\
&= \frac{1}{n}\left[\sum_{i=1}^{n}2(\hat{y}_i - y_i)\left(1 + 0 - 0\right)\right] \\
&= \frac{1}{n}\left[\sum_{i=1}^{n}(\hat{y}_i - y_i)\left(2\right)\right] \\
&= \frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)
\end{aligned}$$

Let’s differentiate the cost function(J) with respect to θ_2

$$\begin{aligned}
J'_{\theta_2} &= \frac{\partial J(\theta_1,\theta_2)}{\partial \theta_2} \\
&= \frac{\partial}{\partial \theta_2}\left[\frac{1}{n}\left(\sum_{i=1}^{n}(\hat{y}_i - y_i)^2\right)\right] \\
&= \frac{1}{n}\left[\sum_{i=1}^{n}2(\hat{y}_i - y_i)\left(\frac{\partial}{\partial \theta_2}(\hat{y}_i - y_i)\right)\right] \\
&= \frac{1}{n}\left[\sum_{i=1}^{n}2(\hat{y}_i - y_i)\left(\frac{\partial}{\partial \theta_2}(\theta_1 + \theta_2 x_i - y_i)\right)\right] \\
&= \frac{1}{n}\left[\sum_{i=1}^{n}2(\hat{y}_i - y_i)\left(0 + x_i - 0\right)\right] \\
&= \frac{1}{n}\left[\sum_{i=1}^{n}(\hat{y}_i - y_i)\left(2x_i\right)\right] \\
&= \frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)\cdot x_i
\end{aligned}$$

Finding the coefficients of a linear equation that best fits the training data is the objective of linear regression. By moving in the direction of the Mean Squared Error negative gradient with respect to the coefficients, the coefficients can be changed. And the respective intercept and coefficient of X will be if α is the learning rate.



Gradient Descent

$$\begin{aligned}\theta_1 &= \theta_1 - \alpha \left(J'_{\theta_1}\right) \\ &= \theta_1 - \alpha \left(\frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)\right)\end{aligned}$$

$$\begin{aligned}\theta_2 &= \theta_2 - \alpha \left(J'_{\theta_2}\right) \\ &= \theta_2 - \alpha \left(\frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)\cdot x_i\right)\end{aligned}$$

Build the Linear Regression model from Scratch

Import the necessary libraries:

Python3

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.axes as ax
```

Load the dataset and separate input and Target variables

Dataset Link: [https://github.com/AshishJangra27/Machine-Learning-with-Python-GFG/tree/main/Linear%20Regression]

```
data = pd.read_csv('data_for_lr.csv')

# 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

Steps:

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

Python3

```
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
```

```
    def update_parameters(self, derivatives, learning_rate):
        self.parameters['m'] = self.parameters['m'] - learning_rate * derivatives
        self.parameters['c'] = self.parameters['c'] - learning_rate * derivatives

    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 = []

        #iterate
        for i in range(iters):
            #forward propagation
            predictions = self.forward_propagation(train_input)

            #cost function
            cost = self.cost_function(predictions, train_output)

            #append loss and print
            self.loss.append(cost)
            print("Iteration = {}, Loss = {}".format(i+1, cost))

            #back propagation
            derivatives = self.backward_propagation(train_input, train_output, pre

            #update parameters
            self.update_parameters(derivatives, learning_rate)

        return self.parameters, self.loss
```

Trained the model

Python3

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

```
Iteration = 1, Loss = 5363.981028641572
Iteration = 2, Loss = 2437.9165904342512
Iteration = 3, Loss = 1110.3579137897523
Iteration = 4, Loss = 508.043071737168
Iteration = 5, Loss = 234.7721607488976
Iteration = 6, Loss = 110.78884574712548
Iteration = 7, Loss = 54.53747840152165
Iteration = 8, Loss = 29.016170730218153
Iteration = 9, Loss = 17.43712517102535
Iteration = 10, Loss = 12.183699375121314
Iteration = 11, Loss = 9.800214272338595
Iteration = 12, Loss = 8.718824440889573
Iteration = 13, Loss = 8.228196676299069
Iteration = 14, Loss = 8.005598315794709
Iteration = 15, Loss = 7.904605192804647
Iteration = 16, Loss = 7.858784500769819
Iteration = 17, Loss = 7.837995601770647
Iteration = 18, Loss = 7.828563654998014
Iteration = 19, Loss = 7.824284370030002
Iteration = 20, Loss = 7.822342853430061
```

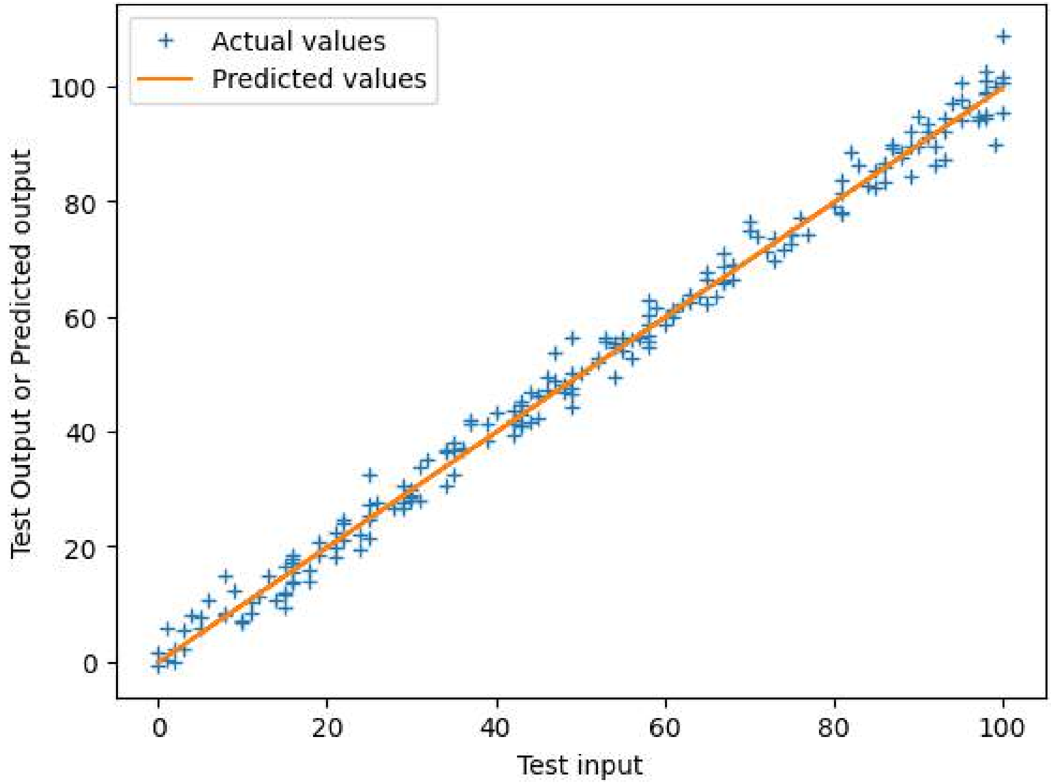
Final Prediction and Plot the regression line

Python3

```
#Prediction on test data
y_pred = test_input*parameters['m'] + parameters['c']

# Plot the regression line with actual data pointa
plt.plot(test_input, test_output, '+', label='Actual values')
plt.plot(test_input, y_pred, label='Predicted values')
plt.xlabel('Test input')
plt.ylabel('Test Output or Predicted output')
plt.legend()
plt.show()
```

Output:



Best fit Linear regression line with actual values

Whether you're preparing for your first job interview or aiming to upskill in this ever-evolving tech landscape, [GeeksforGeeks Courses] are your key to success. We provide top-quality content at affordable prices, all geared towards accelerating your growth in a time-bound manner. Join the millions we've already empowered, and we're here to do the same for you. Don't miss out - [check it out now!]

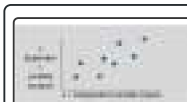
Last Updated : 08 May, 2023

Similar Reads

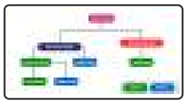
ML | Linear Regression vs Logistic Regression

Support Vector Regression (SVR) using Linear and Non-Linear Kernels in Scikit Learn

Logistic Regression in Machine Learning



Regression and Classification | Supervised Machine Learning


CART (Classification And Regression Tree) in Machine Learning

Robust Regression for Machine Learning in Python


Classification vs Regression in Machine Learning


Locally Linear Embedding in machine learning

Related Tutorials

OpenAI Python API - Complete Guide

Computer Vision Tutorial


Computer Science and Programming For Kids


Pandas AI: The Generative AI Python Library


Top Computer Vision Projects (2023)

Previous

Ordinary Least Squares (OLS) using statsmodels

Next

Ordinary Least Squares (OLS) using statsmodels

Article Contributed By :

M

Mohit Gupta_OMG :)
Mohit Gupta_OMG :)

Vote for difficulty

Current difficulty : Medium

Easy Normal Medium Hard Expert

Article Tags : Computer Subject , Machine Learning , Python

Practice Tags : Machine Learning, python

Improve Article Report Issue