```
[ ] Start coding or generate with AI.
```

Double-click (or enter) to edit

Introduction

What is a Regression?

Types of Regression models

Linear Regression

Gradient descent

Impact of different values for learning rate

Use case

Steps to implement Linear regression model

What is a Regression?

In Regression, we plot a graph between the variables which best fit the given data points. The machine learning model can deliver predictions regarding the data. In naïve words, "Regression shows a line or curve that passes through all the data points on a target-predictor graph in such a way that the vertical distance between the data points and the regression line is minimum." It is used principally for prediction, forecasting, time series modeling, and determining the causal-effect relationship between variables.

Types of Regression models

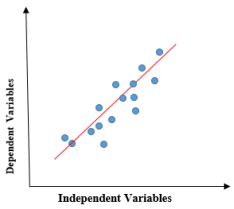
Linear Regression

Polynomial Regression

Logistics Regression

Linear Regression

Linear regression is a quiet and simple statistical regression method used for predictive analysis and shows the relationship between the continuous variables. Linear regression shows the linear relationship between the independent variable (X-axis) and the dependent variable (Y-axis), consequently called linear regression. If there is a single input variable (x), such linear regression is called simple linear regression. And if there is more than one input variable, such linear regression is called multiple linear regression. The linear regression model gives a sloped straight line describing the relationship within the variables.



The above graph presents the linear relationship between the dependent variable and independent variables. When the value of x (independent variable) increases, the value of y (dependent variable) is likewise increasing. The red line is referred to as the best fit straight line. Based on the given data points, we try to plot a line that models the points the best.

To calculate best-fit line linear regression uses a traditional slope-intercept form.

$$y = mx+b \quad \Longrightarrow \quad y = a_0+a_1x$$

y= Dependent Variable.

x= Independent Variable.

a0= intercept of the line.

a1 = Linear regression coefficient.

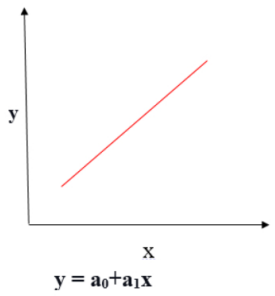
As mentioned above, Linear regression estimates the relationship between a dependent variable and an independent variable. Let's understand this with an easy example:

Let's say we want to estimate the salary of an employee based on year of experience. You have the recent company data, which indicates that the relationship between experience and salary. Here year of experience is an independent variable, and the salary of an employee is a dependent variable, as the salary of an employee is dependent on the experience of an employee. Using this insight, we can predict the future salary of the employee based on current & past information.

A regression line can be a Positive Linear Relationship or a Negative Linear Relationship.

Positive Linear Relationship

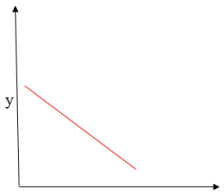
If the dependent variable expands on the Y-axis and the independent variable progress on X-axis, then such a relationship is termed a Positive linear relationship.



$y = a_0 + a_1 x$

Negative Linear Relationship

If the dependent variable decreases on the Y-axis and the independent variable increases on the X-axis, such a relationship is called a negative linear relationship.



$y = -a_0 + a_1 x$

The goal of the linear regression algorithm is to get the best values for a0 and a1 to find the best fit line. The best fit line should have the least error means the error between predicted values and actual values should be minimized.

Cost function

The cost function helps to figure out the best possible values for a0 and a1, which provides the best fit line for the data points.

Cost function optimizes the regression coefficients or weights and measures how a linear regression model is performing. The cost function is used to find the accuracy of the mapping function that maps the input variable to the output variable. This mapping function is also known as the Hypothesis function.

In Linear Regression, Mean Squared Error (MSE) cost function is used, which is the average of squared error that occurred between the predicted values and actual values.

By simple linear equation y=mx+b we can calculate MSE as:

Let's y = actual values, yi = predicted values

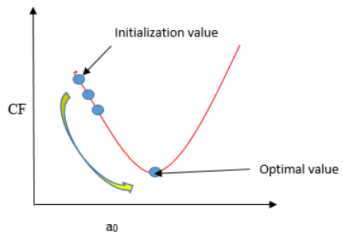
$$MSE = \frac{1}{N}\sum_{i=1}^{n}(y_i - (mx_i + b))^2$$

Double-click (or enter) to edit

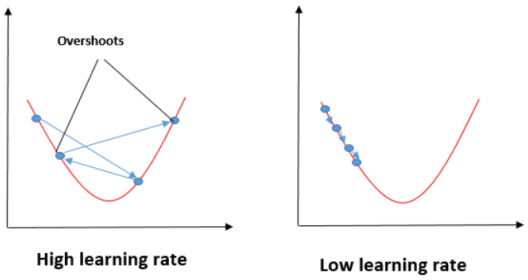
Using the MSE function, we will change the values of a0 and a1 such that the MSE value settles at the minima. Model parameters xi, b (a0,a1) can be manipulated to minimize the cost function. These parameters can be determined using the gradient descent method so that the cost function value is minimum.

Gradient descent

Gradient descent is a method of updating a0 and a1 to minimize the cost function (MSE). A regression model uses gradient descent to update the coefficients of the line (a0, a1 => xi, b) by reducing the cost function by a random selection of coefficient values and then iteratively update the values to reach the minimum cost function.



Imagine a pit in the shape of U. You are standing at the topmost point in the pit, and your objective is to reach the bottom of the pit. There is a treasure, and you can only take a discrete number of steps to reach the bottom. If you decide to take one footstep at a time, you would eventually get to the bottom of the pit but, this would take a longer time. If you choose to take longer steps each time, you may get to sooner but, there is a chance that you could overshoot the bottom of the pit and not near the bottom. In the gradient descent algorithm, the number of steps you take is the learning rate, and this decides how fast the algorithm converges to the minima.



To update a0 and a1, we take gradients from the cost function. To find these gradients, we take partial derivatives for a0 and a1.

Use case

In this, I will take random numbers for the dependent variable (salary) and an independent variable (experience) and will predict the impact of a year of experience on salary.

import some required libraries

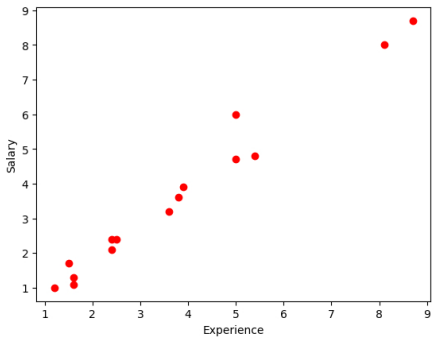
```
[1] import matplotlib.pyplot as plt
    import pandas as pd
    import numpy as np
```

Define the dataset

```
[2] x= np.array([2.4,5.0,1.5,3.8,8.7,3.6,1.2,8.1,2.5,5,1.6,1.6,2.4,3.9,5.4])
    y = np.array([2.1,4.7,1.7,3.6,8.7,3.2,1.0,8.0,2.4,6,1.1,1.3,2.4,3.9,4.8])
    n = np.size(x)
```

Plot the data points

```
[3] import matplotlib.pyplot as plt
    import pandas as pd
    import numpy as np

    x= np.array([2.4,5.0,1.5,3.8,8.7,3.6,1.2,8.1,2.5,5,1.6,1.6,2.4,3.9,5.4])
    y = np.array([2.1,4.7,1.7,3.6,8.7,3.2,1.0,8.0,2.4,6,1.1,1.3,2.4,3.9,4.8])
    n = np.size(x)

    plt.scatter(x, y, color = 'red')
    plt.xlabel("Experience")
    plt.ylabel("Salary")
    plt.show()
```



The main function to calculate values of coefficients

1. Initialize the parameters.
2. Predict the value of a dependent variable by given an independent variable.
3. Calculate the error in prediction for all data points.
4. Calculate partial derivative w.r.t a0 and a1.
5. Calculate the cost for each number and add them. Update the values of a0 and a1.

```
#initialize the parameters
a0 = 0                  #intercept
a1 = 0                  #Slop
lr = 0.0001             #Learning rate
iterations = 1000       # Number of iterations
error = []              # Error array to calculate cost for each iterations.
for itr in range(iterations):
    error_cost = 0
    cost_a0 = 0
    cost_a1 = 0
    for i in range(len(x)):
        y_pred = a0+a1*x[i]   # predict value for given x
        error_cost = error_cost +(y[i]-y_pred)**2
        for j in range(len(x)):
            partial_wrt_a0 = -2 *(y[j] - (a0 + a1*x[j]))               #partial derivative w.r.t a0
            partial_wrt_a1 = (-2*x[j])*(y[j]-(a0 + a1*x[j]))   #partial derivative w.r.t a1
            cost_a0 = cost_a0 + partial_wrt_a0      #calculate cost for each number and add
            cost_a1 = cost_a1 + partial_wrt_a1      #calculate cost for each number and add
        a0 = a0 - lr * cost_a0    #update a0
        a1 = a1 - lr * cost_a1    #update a1
        print(itr,a0,a1)          #Check iteration and updated a0 and a1
    error.append(error_cost)      #Append the data in array
```

```
987 -0.21354150071690245 1.0247464287610857
987 -0.21354150071690242 1.0247464287610857
987 -0.21354150071690242 1.0247464287610857
987 -0.21354150071690242 1.0247464287610857
987 -0.21354150071690242 1.0247464287610857
```

```
987 -0.21354150071690242 1.0247464287610857
987 -0.21354150071690242 1.0247464287610857
987 -0.21354150071690242 1.0247464287610857
988 -0.21354150071690242 1.0247464287610857
988 -0.21354150071690242 1.0247464287610857
988 -0.21354150071690242 1.0247464287610857
988 -0.21354150071690245 1.0247464287610857
988 -0.21354150071690248 1.0247464287610857
988 -0.2135415007169025 1.024746428761085
988 -0.21354150071690253 1.0247464287610852
988 -0.2135415007169026 1.024746428761085
988 -0.213541500716926 1.0247464287610848
988 -0.21354150071690262 1.0247464287610848
988 -0.21354150071690262 1.0247464287610848
988 -0.21354150071690262 1.0247464287610848
988 -0.21354150071690262 1.0247464287610848
988 -0.21354150071690262 1.0247464287610848
988 -0.21354150071690262 1.0247464287610848
989 -0.21354150071690262 1.0247464287610848
989 -0.21354150071690262 1.0247464287610848
989 -0.2135415007169026 1.024746428761085
989 -0.213541500716926 1.0247464287610848
989 -0.21354150071690253 1.0247464287610852
989 -0.2135415007169025 1.024746428761085
989 -0.21354150071690248 1.0247464287610857
989 -0.21354150071690245 1.0247464287610857
989 -0.21354150071690242 1.0247464287610857
989 -0.21354150071690242 1.0247464287610857
989 -0.21354150071690242 1.0247464287610857
989 -0.21354150071690242 1.0247464287610857
989 -0.21354150071690242 1.0247464287610857
989 -0.21354150071690242 1.0247464287610857
989 -0.21354150071690242 1.0247464287610857
990 -0.21354150071690242 1.0247464287610857
990 -0.21354150071690242 1.0247464287610857
990 -0.21354150071690242 1.0247464287610857
990 -0.21354150071690245 1.0247464287610857
990 -0.21354150071690248 1.0247464287610857
990 -0.2135415007169025 1.024746428761085
990 -0.21354150071690253 1.0247464287610852
990 -0.21354150071690256 1.024746428761085
990 -0.2135415007169026 1.024746428761085
990 -0.21354150071690262 1.0247464287610848
990 -0.21354150071690262 1.0247464287610848
990 -0.21354150071690262 1.0247464287610848
990 -0.21354150071690262 1.0247464287610848
990 -0.21354150071690262 1.0247464287610848
991 -0.21354150071690262 1.0247464287610848
991 -0.21354150071690262 1.0247464287610848
991 -0.2135415007169026 1.024746428761085
991 -0.21354150071690256 1.024746428761085
991 -0.21354150071690253 1.0247464287610852
991 -0.2135415007169025 1.0247464287610855
```

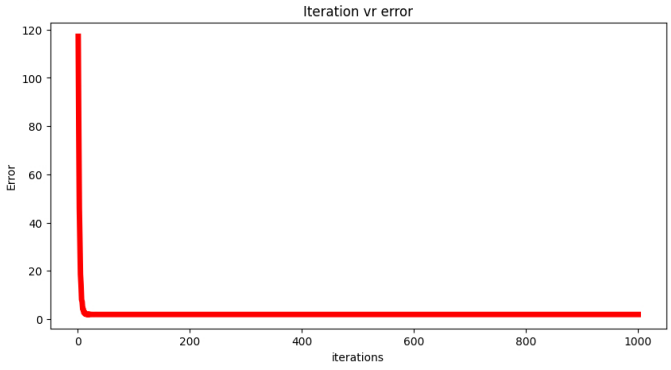
[6]
```
#At approximate iteration 50- 60, we got the value of a0 and a1.

print(a0)
print(a1)
```

```
-0.21354150071690242
1.0247464287610857
```

Plotting the error for each iteration.

[7]
```
plt.figure(figsize=(10,5))
plt.plot(np.arange(1,len(error)+1),error,color='red',linewidth = 5)
plt.title("Iteration vr error")
plt.xlabel("iterations")
plt.ylabel("Error")
```

```
Text(0, 0.5, 'Error')
```



Predicting the values.

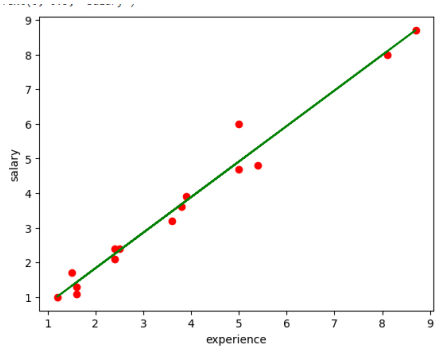
[9]
```
pred = a0+a1*x
print(pred)
```

```
[2.24584993 4.91019064 1.32357814 3.68049493 8.70175243 3.47554564
 1.01615421 8.08690457 2.34832457 4.91019064 1.42605279 1.42605279
 2.24584993 3.78296957 5.32008921]
```

Plot the regression line.

```
plt.scatter(x,y,color = 'red')
plt.plot(x,pred, color = 'green')
plt.xlabel("experience")
plt.ylabel("salary")
```

```
Text(0, 0.5, 'salary')
```



Analyze the performance of the model by calculating the mean squared error.

```
[13] error1 = y - pred
     se = np.sum(error1 ** 2)
     mse = se/n
     print("mean squared error is", mse)
```

```
mean squared error is 0.12785817711928918
```

Use the scikit library to confirm the above steps.

```
[16] from sklearn.linear_model import LinearRegression
     from sklearn.metrics import mean_squared_error
     x= x.reshape(-1,1)
     model = LinearRegression()
     model.fit(x,y)
     salary_pred = model.predict(x)
     Mse = mean_squared_error(x, salary_pred)
     print('slop', model.coef_)
     print("Intercept", model.intercept_)
     print("MSE", Mse)
```

```
slop [1.02474643]
Intercept -0.21354150071690547
MSE 0.01747515621404453
```

```
#Summary
#In Regression, we plot a graph between the variables which best fit the given data points. Linear regression shows the linear relationship between the independent variable (X-axis) and the dependent variable (Y-axis).To calculate best-fit line linear regression uses a traditional slope-intercept form. A regression line can be a Positive Linear Relationship or a Negative Linear Relationship.

#The goal of the linear regression algorithm is to get the best values for a0 and a1 to find the best fit line and the best fit line should have the least error. In Linear Regression, Mean Squared Error (MSE) cost function is used, which helps to figure out the best possible values for a0 and a1, which provides the best fit line for the data points. Using the MSE function, we will change the values of a0 and a1 such that the MSE value settles at the minima. Gradient descent is a method of updating a0 and a1 to minimize the cost function (MSE)
```

Summary

In Regression, we plot a graph between the variables which best fit the given data points. Linear regression shows the linear relationship between the independent variable (X-axis) and the dependent variable (Y-axis).To calculate best-fit line linear regression uses a traditional slope-intercept form. A regression line can be a Positive Linear Relationship or a Negative Linear Relationship.

The goal of the linear regression algorithm is to get the best values for a0 and a1 to find the best fit line and the best fit line should have the least error. In Linear Regression, Mean Squared Error (MSE) cost function is used, which helps to figure out the best possible values for a0 and a1, which provides the best fit line for the data points. Using the MSE function, we will change the values of a0 and a1 such that the MSE value settles at the minima. Gradient descent is a method of updating a0 and a1 to minimize the cost function (MSE)