Linear Regression: Concepts and Techniques

Linear regression is a fundamental technique in statistics and machine learning used to model the relationship between a dependent variable (y) and one or more independent variables (x). It assumes a linear relationship, where y can be predicted as a weighted sum of x values, plus a bias term.

---

1. Simple Linear Regression

Definition:
Simple linear regression models the relationship between one dependent variable (y) and one independent variable (x).

Equation:

$$y = \theta_0 + \theta_1 x + \epsilon$$

Where:

- y: Dependent variable (output).
- x: Independent variable (input).
- θ_0: Intercept (value of y when $x = 0$).
- θ_1: Slope (rate of change of y with respect to x).
- ϵ: Error term (captures noise or deviation from the linear relationship).

2. Multiple Linear Regression

Definition:

Multiple linear regression models the relationship between one dependent variable (y) and two or more independent variables ($x_1, x_2, \ldots, x_p$). It extends simple linear regression to include multiple predictors. The goal is to predict y using a weighted combination of the independent variables.

Equation:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_p x_p + \epsilon$$

Where:

- y: Dependent variable (output).
- $x_1, x_2, \ldots, x_p$: Independent variables (inputs).
- θ_0: Intercept (value of y when all $x_i = 0$).
- $\theta_1, \theta_2, \ldots, \theta_p$: Coefficients (rate of change of y with respect to each x_i).
- ϵ: Error term (captures noise or deviation from the linear relationship).

Example:

Predicting House Prices:

- y: House price.
- x_1: Square footage.
- x_2: Number of bedrooms.
- x_3: Location rating.

Equation:

$$y = 50000 + 200x_1 + 10000x_2 + 5000x_3 + \epsilon$$

This means the base price of a house is 50,000, and for every additional square foot, the price increases by 200. Similarly, adding one bedroom increases the price by 10,000, and each improvement in location rating adds 5,000.

Techniques Used to fit linear regression model

1. Ordinary Least Squares(OLS)
2. Ridge Regression
3. Lasso Regression

Ordinary Least Squares (OLS) in Detail

Ordinary Least Squares (OLS) is a method used for estimating the coefficients (θ_0 and θ_1) of a linear regression model. It minimizes the sum of squared residuals (errors) between the actual values and the predicted values to find the best-fitting line. This method is used to determine the relationship between the dependent variable (y) and the independent variable (x).

Formula for OLS Linear Regression Model

The simple linear regression model is:

$$y = \theta_0 + \theta_1 x + \epsilon$$

Where:

- y = Dependent variable (output)
- x = Independent variable (input)
- θ_0 = Intercept (the value of y when $x = 0$)
- θ_1 = Slope (the rate of change of y with respect to x)
- ϵ = Error term (captures noise or deviations)

OLS Estimation of θ_0 and θ_1

To estimate the coefficients θ_0 and θ_1, OLS minimizes the sum of squared residuals (errors). The residuals are the differences between the actual and predicted values of y:

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

Where $\hat{y}_i$ is the predicted value for y_i.

The closed-form solutions for θ_0 and θ_1 are:

$$\theta_1 = \frac{n \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}$$

$$\theta_0 = \bar{y} - \theta_1 \bar{x}$$

Where:

- n = Number of data points
- $\sum_{i=1}^{n} x_i$ = Sum of x-values
- $\sum_{i=1}^{n} y_i$ = Sum of y-values
- $\sum_{i=1}^{n} x_i y_i$ = Sum of the product of x and y
- $\sum_{i=1}^{n} x_i^2$ = Sum of squared x-values
- $\bar{x}$ = Mean of x-values
- $\bar{y}$ = Mean of y-values

Step-by-Step Example

Let's take an example with some simple data to illustrate how OLS works.

Dataset Example:

Consider a small dataset of house sizes and their corresponding prices:

Size (Square feet)	Price (USD)
500	100,000
800	150,000
1000	200,000
1200	250,000
1500	300,000

Let's calculate the values for θ_0 and θ_1.

Step 1: Calculate the Necessary Sums

- $\sum x_i = 500 + 800 + 1000 + 1200 + 1500 = 5000$
- $\sum y_i = 100000 + 150000 + 200000 + 250000 + 300000 = 1000000$
- $\sum x_i y_i = (500 \times 100000) + (800 \times 150000) + (1000 \times 200000) + (1200 \times 250000) + (1500 \times 300000) = 50000000 + 120000000 + 200000000 + 300000000 + 450000000 = 1120000000$
- $\sum x_i^2 = (500^2) + (800^2) + (1000^2) + (1200^2) + (1500^2) = 250000 + 640000 + 1000000 + 1440000 + 2250000 = 6335000$
- $n = 5$

Step 2: Calculate θ_1

Using the formula for θ_1:

$$\theta_1 = \frac{5(1120000000) - (5000)(1000000)}{5(6335000) - (5000)^2}$$

$$\theta_1 = \frac{5600000000 - 5000000000}{31675000 - 25000000}$$

$$\theta_1 = \frac{600000000}{6675000} = 89.91$$

So, the slope θ_1 is **89.91**.

Step 3: Calculate θ_0

Now, calculate θ_0 using the formula:

$$\theta_0 = \bar{y} - \theta_1 \bar{x}$$

Where $\bar{x}$ and $\bar{y}$ are the means of x and y:

$$\bar{x} = \frac{5000}{5} = 1000, \quad \bar{y} = \frac{1000000}{5} = 200000$$

Now:

$$\theta_0 = 200000 - (89.91)(1000)$$

$$\theta_0 = 200000 - 89910 = 110090$$

So, the intercept θ_0 is **110090**.

Step 4: Final Model

Thus, the linear regression equation is:

$$\text{Price} = 110090 + 89.91 \times \text{Size}$$

This means:

- The base price of a house (when the size is 0) is **$110,090**.
- For every additional square foot, the price increases by **$89.91**.

Example Scenarios:

1. **Scenario 1: Price of a 1200 sq ft house**

Using the model, we can calculate the price of a house with a size of 1200 square feet:

$$\text{Price} = 110090 + 89.91 \times 1200$$

$$\text{Price} = 110090 + 107892$$

$$\text{Price} = 217982$$

So, the predicted price for a 1200 square foot house is **$217,982**.

2. **Scenario 2: Price of a 2000 sq ft house**

Now, let's calculate the price of a 2000 square foot house:

$$\text{Price} = 110090 + 89.91 \times 2000$$

$$\text{Price} = 110090 + 179820$$

$$\text{Price} = 289910$$

So, the predicted price for a 2000 square foot house is **$289,910**.

3. **Scenario 3: Price of a 3000 sq ft house**

Let's calculate the price of a 3000 square foot house:

$$\text{Price} = 110090 + 89.91 \times 3000$$

$$\text{Price} = 110090 + 269730$$

$$\text{Price} = 379820$$

So, the predicted price for a 3000 square foot house is **$379,820**.

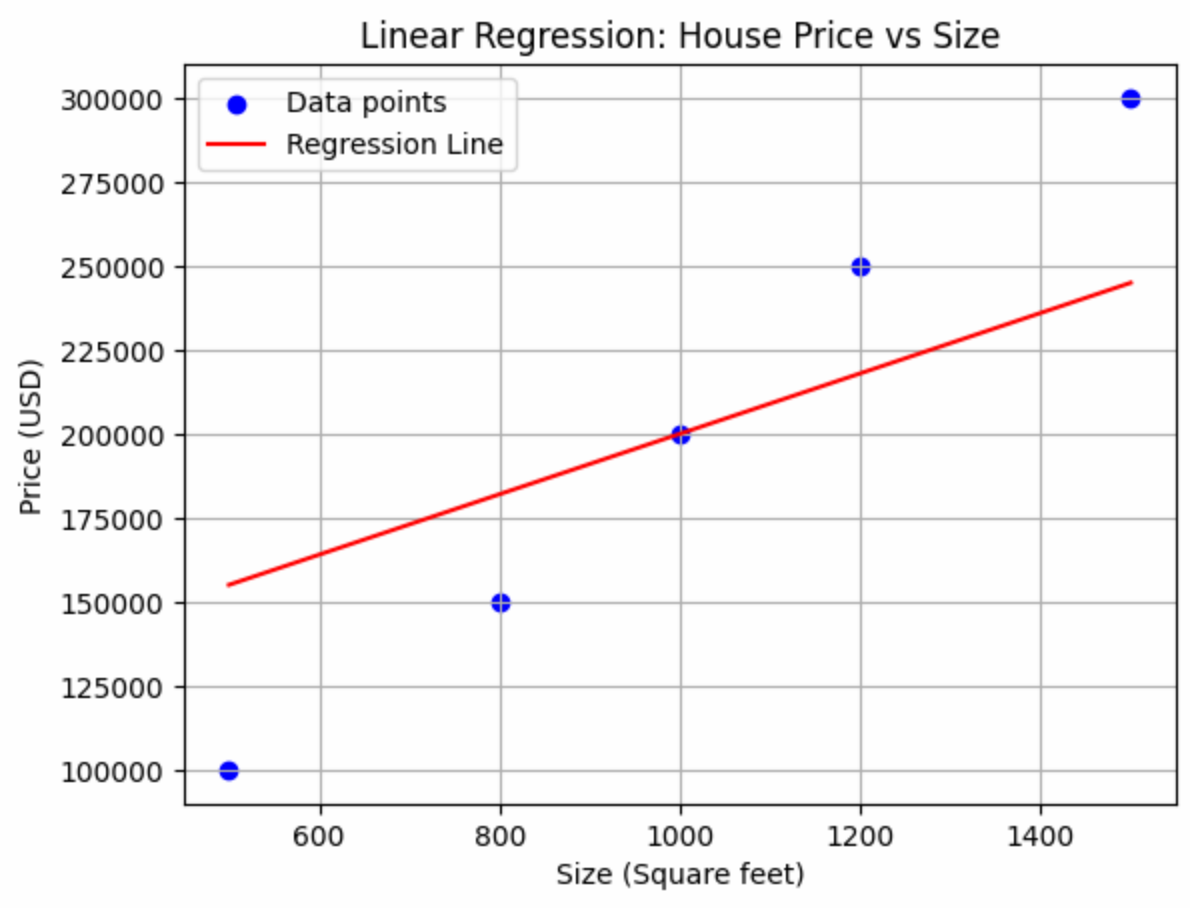
Step 5: Plotting the Line

Let's plot this data and the regression line on a graph.

```python
import matplotlib.pyplot as plt
import numpy as np

# Data points
x = np.array([500, 800, 1000, 1200, 1500])
y = np.array([100000, 150000, 200000, 250000, 300000])

# Calculated values for θ0 and θ1
theta_0 = 110090
theta_1 = 89.91

# Regression line
y_pred = theta_0 + theta_1 * x

# Plotting the data points and the regression line
plt.scatter(x, y, color='blue', label='Data points')
plt.plot(x, y_pred, color='red', label='Regression Line')
plt.xlabel('Size (Square feet)')
plt.ylabel('Price (USD)')
plt.title('Linear Regression: House Price vs Size')
plt.legend()
plt.grid(True)
plt.show()
```

Explanation of Plot:

- **Blue dots** represent the actual data points (house size vs price).
- **Red line** represents the best-fitting regression line, calculated using OLS, which shows how the price increases as the size of the house increases.


Linear Regression: House Price vs Size
Data points
Regression Line
300000
275000
250000
225000
200000
175000
150000
125000
100000
Price (USD)
600
800
1000
1200
1400
Size (Square feet)

What is a Cost Function?

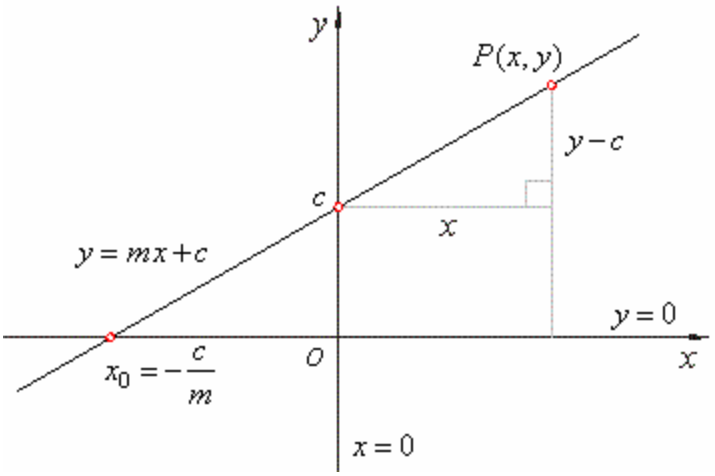
A **cost function** is a mathematical function that measures the performance of a machine learning model. It quantifies the error (or "cost") between the predicted outputs ($\hat{y}$) of the model and the actual target outputs (y). The goal of training a model is to minimize the cost function, thereby improving the model's accuracy.

Linear Regression with Gradient Descent

Linear Regression

In statistics, linear regression is a linear approach to modelling the relationship between a dependent variable and one or more independent variables. Let **X** be the independent variable and **Y** be the dependent variable. The linear relationship between these two variables is as follows:

$$Y = mX + c$$



Source: http://www.nabla.hr/SlopeInterceptLineEqu.gif

This is the equation for a line that you studied in high school. **m** is the slope of the line and **c** is the y intercept. Today we will use this equation to train our model with a given dataset and predict the value of **Y** for any given value of **X**. Our challenge today is to determine the value of **m** and **c**, such that the line corresponding to those values is the best fitting line or gives the minimum error.

Loss Function in Linear regression

The loss is the error in our predicted value of **m** and **c**. Our goal is to minimize this error to obtain the most accurate value of **m** and **c**.

We will use the Mean Squared Error function to calculate the loss.

There are three steps in this function:

1. Find the difference between the actual y and predicted y value(y = mx + c), for a given x.
2. Square this difference.
3. Find the mean of the squares for every value in X.

$$E = \frac{1}{n}\sum_{i=0}^{n}(y_i - \bar{y}_i)^2$$

Mean Squared Error Equation

Here y_i is the actual value and $\bar{y}_i$ is the predicted value. Lets substitute the value of $\bar{y}_i$:

$$E = \frac{1}{n}\sum_{i=0}^{n}(y_i - (mx_i + c))^2$$

Substituting the value of $\bar{y}_i$

So we square the error and find the mean. hence the name Mean Squared Error. Now that we have defined the loss function, lets get into the interesting part — minimizing it and finding **m** and **c.**

The Gradient Descent Algorithm

Gradient descent is an iterative optimization algorithm to find the minimum of a function. Here that function is our Loss Function.

Understanding Gradient Descent

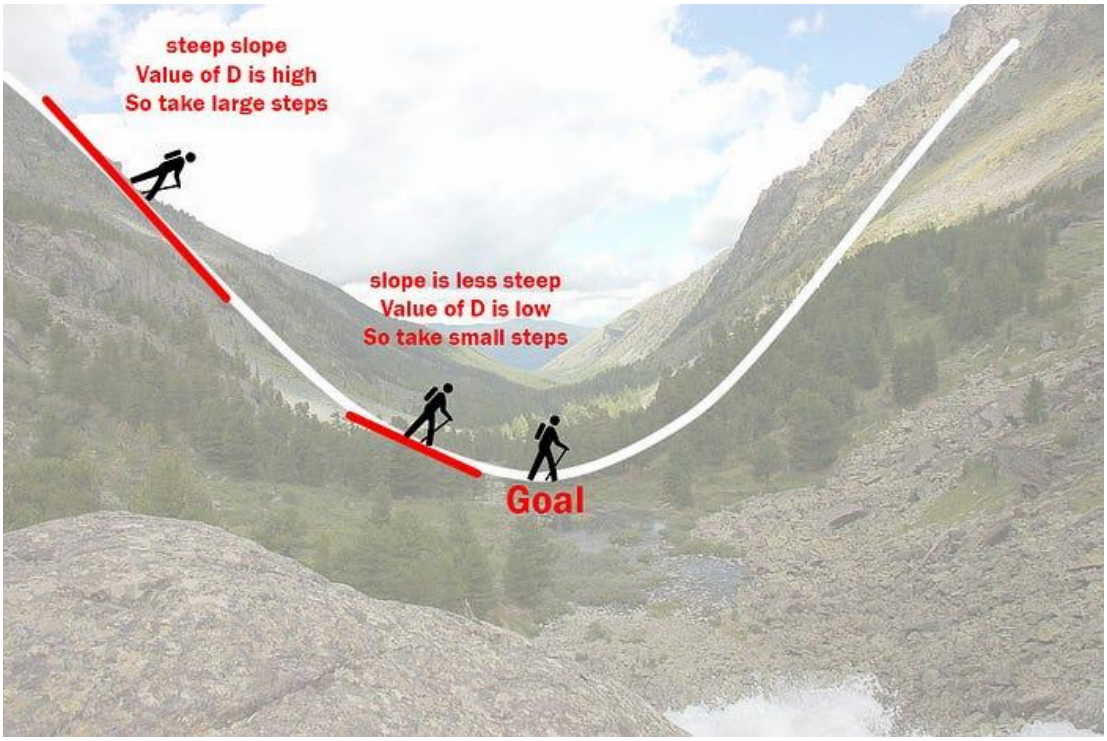


Illustration of how the gradient descent algorithm works

Imagine a valley and a person with no sense of direction who wants to get to the bottom of the valley. He goes down the slope and takes large steps when the slope is steep and small steps when the slope is less steep. He decides his next position based on his current position and stops when he gets to the bottom of the valley which was his goal.

Let's try applying gradient descent to **m** and **c** and approach it step by step:

1. Initially let m = 0 and c = 0. Let L be our learning rate. This controls how much the value of **m** changes with each step. L could be a small value like 0.0001 for good accuracy.
2. Calculate the partial derivative of the loss function with respect to m, and plug in the current values of x, y, m and c in it to obtain the derivative value **D**.

$$D_m = \frac{1}{n}\sum_{i=0}^{n} 2(y_i - (mx_i + c))(-x_i)$$

$$D_m = \frac{-2}{n}\sum_{i=0}^{n} x_i(y_i - \bar{y}_i)$$

Derivative with respect to m

D is the value of the partial derivative with respect to **m**. Similarly lets find the partial derivative with respect to **c**, Dc :

$$D_c = \frac{-2}{n}\sum_{i=0}^{n} (y_i - \bar{y}_i)$$

Derivative with respect to c

3. Now we update the current value of **m** and **c** using the following equation:

$$m = m - L \times D_m$$

$$c = c - L \times D_c$$

4. We repeat this process until our loss function is a very small value or ideally 0 (which means 0 error or 100% accuracy). The value of **m** and **c** that we are left with now will be the optimum values.

Now going back to our analogy, **m** can be considered the current position of the person. **D** is equivalent to the steepness of the slope and **L** can be the speed with which he moves. Now the new value of **m** that we calculate using the above equation will be his next position, and **L×D** will be the size of the steps he will take. When the slope is more steep (**D** is more) he takes longer steps and when it is less steep (**D** is less), he takes smaller steps. Finally he arrives at the bottom of the valley which corresponds to our loss = 0.

Now with the optimum value of **m** and **c** our model is ready to make predictions !

4. Example

Dataset:

We have 5 data points:

$$x = [1, 2, 3, 4, 5], \quad y = [1, 2, 3, 4, 5]$$

Clearly, the best-fit line is $y = x$, where $m = 1$ and $c = 0$. Let's use gradient descent to find m and c.

Initialization:

- Start with $m = 0, c = 0$.
- Learning rate $L = 0.01$.

Iteration 1:

1. Compute the predicted values:
$$\hat{y} = mx + c = 0 \cdot x + 0 = [0, 0, 0, 0, 0]$$
2. Compute the gradients:
   - Gradient for m:
   $$\frac{\partial \text{MSE}}{\partial m} = -\frac{2}{5} \sum (x_i \cdot (y_i - \hat{y}_i)) = -\frac{2}{5}(1 + 4 + 9 + 16 + 25) = -22$$
   - Gradient for c:
   $$\frac{\partial \text{MSE}}{\partial c} = -\frac{2}{5} \sum (y_i - \hat{y}_i) = -\frac{2}{5}(1 + 2 + 3 + 4 + 5) = -6$$
3. Update m and c:
   - $m = m - L \cdot \frac{\partial \text{MSE}}{\partial m} = 0 - 0.01 \cdot (-22) = 0.22$
   - $c = c - L \cdot \frac{\partial \text{MSE}}{\partial c} = 0 - 0.01 \cdot (-6) = 0.06$

Iteration 2:

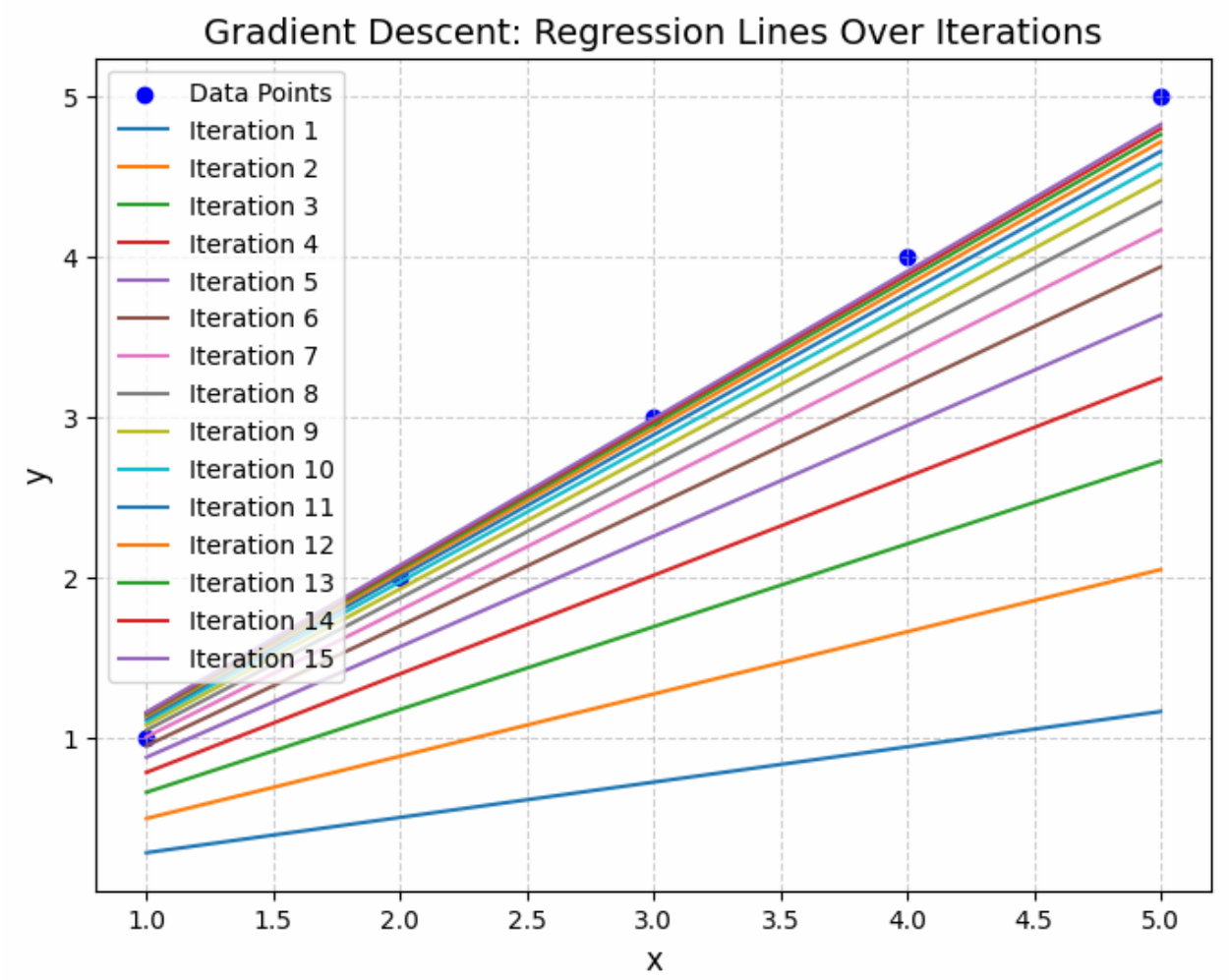
1. Compute the predicted values:
$$\hat{y} = mx + c = 0.22x + 0.06 = [0.28, 0.5, 0.72, 0.94, 1.16]$$
2. Compute the gradients:
   - Gradient for m:
   $$\frac{\partial \text{MSE}}{\partial m} = -\frac{2}{5}\sum(x_i \cdot (y_i - \hat{y}_i)) = -\frac{2}{5}(0.72 + 1.5 + 2.28 + 3.06 + 3.84) = -19.8$$
   - Gradient for c:
   $$\frac{\partial \text{MSE}}{\partial c} = -\frac{2}{5}\sum(y_i - \hat{y}_i) = -\frac{2}{5}(0.72 + 1.5 + 2.28 + 3.06 + 3.84) = -5.4$$
3. Update m and c:
   - $m = m - L \cdot \frac{\partial \text{MSE}}{\partial m} = 0.22 - 0.01 \cdot (-19.8) = 0.418$
   - $c = c - L \cdot \frac{\partial \text{MSE}}{\partial c} = 0.06 - 0.01 \cdot (-5.4) = 0.114$

Convergence:

Repeat these steps until m and c converge to $m = 1$ and $c = 0$.

```python
import numpy as np
import matplotlib.pyplot as plt

# Dataset
x = np.array([1, 2, 3, 4, 5])
y = np.array([1, 2, 3, 4, 5])

# Initialize parameters
m, c = 0, 0
learning_rate = 0.01
iterations = 15

# Function to calculate predictions
def predict(x, m, c):
    return m * x + c
```

```
# Function to compute gradients
def compute_gradients(x, y, m, c):
    n = len(y)
    y_pred = predict(x, m, c)
    gradient_m = -(2/n) * np.sum((y - y_pred) * x)
    gradient_c = -(2/n) * np.sum(y - y_pred)
    return gradient_m, gradient_c

# Store parameters for visualization
lines = []

# Gradient Descent
for _ in range(iterations):
    gradient_m, gradient_c = compute_gradients(x, y, m, c)
    m -= learning_rate * gradient_m
    c -= learning_rate * gradient_c
    lines.append((m, c))

# Plot the dataset and regression lines
plt.figure(figsize=(8, 6))
plt.scatter(x, y, color='blue', label='Data Points')

# Plot regression lines for each iteration
for i, (m, c) in enumerate(lines):
    plt.plot(x, predict(x, m, c), label=f'Iteration {i + 1}')

# Labels, legend, and grid
plt.title('Gradient Descent: Regression Lines Over Iterations',
fontsize=14)
plt.xlabel('x', fontsize=12)
plt.ylabel('y', fontsize=12)
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.show()
```


Gradient Descent: Regression Lines Over Iterations
Data Points
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
Iteration 11
Iteration 12
Iteration 13
Iteration 14
Iteration 15
y
x
1
2
3
4
5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0