Multiple Linear Regression

- Extension of Simple linear Regression.
- multiple independent variables ($X_1, X_2, \dots X_n$)
- Single dependent variable (Y)
- Statistical model

$$[\ Y = b_0 + b_1X_1 + b_2X_2 + \cdots b_nX_n + \epsilon\ ]$$

$b_0 \rightarrow$ intercept

$b_1, b_2, \dots \rightarrow$ coefficients

$\epsilon \rightarrow$ Error term

Least Square

Ex

X_1	X_2	Y
1	3	2
2	4	5
3	7	9

Soln $Y = b_0 + b_1X_1 + b_2X_2$

To find : b_0, b_1, b_2

$$\hat{b} = ((X^TX)^{-1}X^T)Y$$

$\hat{b} \rightarrow$ vector

$$\left.\begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}\right\} \quad \hat{b} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

$$X = \begin{matrix} b_0 & b_1 & b_2 \end{matrix} \begin{bmatrix} 1 & 1 & 3 \\ 1 & 2 & 4 \\ 1 & 3 & 7 \end{bmatrix} \qquad X^T = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 3 & 4 & 7 \end{bmatrix}$$

$$y = \begin{bmatrix} 2 \\ 5 \\ 9 \end{bmatrix}$$

$$X^T X = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 3 & 4 & 7 \end{bmatrix} \begin{bmatrix} 1 & 1 & 3 \\ 1 & 2 & 4 \\ 1 & 3 & 7 \end{bmatrix}$$

$$= \begin{bmatrix} 3 & 6 & 14 \\ 6 & 14 & 32 \\ 14 & 32 & 74 \end{bmatrix}$$

$$(X^T X)^{-1} X^T = \begin{bmatrix} 1 & 1 & -1 \\ -1.5 & 2 & -0.5 \\ 0.5 & -1 & 0.5 \end{bmatrix}$$

use numpy :

```
original_matrix = np.array([[2,1],
                            [4,3]])

inv_matrix = np.linalg.inv(matrix)
inv_matrix
```

$$((X^TX)^{-1}X^T)Y = \begin{bmatrix} 1 & 1 & -1 \\ -1.5 & 2 & -0.5 \\ 0.5 & -1 & 0.5 \end{bmatrix} \begin{bmatrix} 2 \\ 5 \\ 9 \end{bmatrix}$$

$$\hat{b} = \begin{bmatrix} -2 \\ 2.5 \\ 0.5 \end{bmatrix} \quad \begin{aligned} b_0 &= -2 \\ b_1 &= 2.5 \\ b_2 &= 0.5 \end{aligned}$$

$X_1 = 1.5$
$X_2 = 3$

$$\begin{aligned} y &= b_0 + b_1X_1 + b_2X_2 \\ y &= -2 + (2.5)(1.5) + (0.5)(3) \\ y &= -2 + 3.75 + 1.5 \\ y &= 3.25 \end{aligned}$$

$pred = X \cdot \hat{b}$ } Final result

If we take 4 input features then, for viz we must fix 2 features (least influencetial)

least influential → Small coefficients

↳ Pearson correlation (less)

Gradient Descent

- Use same approach and reach till finding input matrix "X"
- Then we define the hyperparameters & assume $\hat{b}$ vector coefficients = 0

  $b = np.zeros(X.shape[1])$

- Start the loop:

```
for i in range(steps):
  y_pred = x @ b
  residual = y - y_pred

  loss_mse = (1/n)*np.sum(residual ** 2)
  residual_list.append(loss_mse)

  # gradient
  gradient = (-2/n) * (x.T @ residual)
  # updating the weight
  b = b - (lr * gradient)

  print(f"step {i} | coefficents: {b} | loss: {loss_mse}")

print('\n✅ final coefficents after gradient Descent...')
for i, coefficent in enumerate(b):
  print(f"b{i} : {coefficent:.6f}")
```

- Then for plot we fix the least influential coefficient of $\hat{b}$ except b_0 (intercept).

Finding gradient,

$$\left\{ \text{gradient} = \frac{-2}{n}\left(X^T \cdot \text{residual}\right) \right\}$$

Derivation :

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left[y - \hat{y}\right]^2$$

Actual

predicted

For all data points,

$$\hat{y} = X \cdot b \quad \ldots\ldots\ldots$$

$$MSE = J(b) = \frac{1}{n}\Sigma\left[y_i - (X \cdot b)\right]^2$$

vector version of $\Sigma(y - \hat{y}_i)^2$ is dot product of residual with itself

$$\|y - Xb\|^2 = (y - Xb)^T (y - Xb) \quad \ldots\ldots$$

$$J(b) = \frac{1}{n}(y - Xb)^T (y - Xb)$$

$$\left\{ \frac{d(J(b))}{db} \right\}$$

$$\frac{d}{db}\left(\frac{1}{n}(y-Xb)^T(y-Xb)\right)$$

$$\frac{1}{n}\cdot\frac{d}{db}\left((y-Xb)^T(y-Xb)\right)$$

$$\left\{\begin{array}{l}\text{diff}(a-Ab)^T(a-Ab)\\ \quad = -2A^T(a-Ab)\end{array}\right\}$$

$$\frac{1}{n}\left[-2X^T(y-Xb)\right]$$

$$-\frac{2}{n}X^T\underbrace{(y-Xb)}_{\text{Residual}}$$

$$-\frac{2}{n}(X^T\cdot \text{Residual})$$

$$\nabla J(b) = -\frac{2}{n}(X^T\cdot \text{Residual})$$

↓

[Gradient]