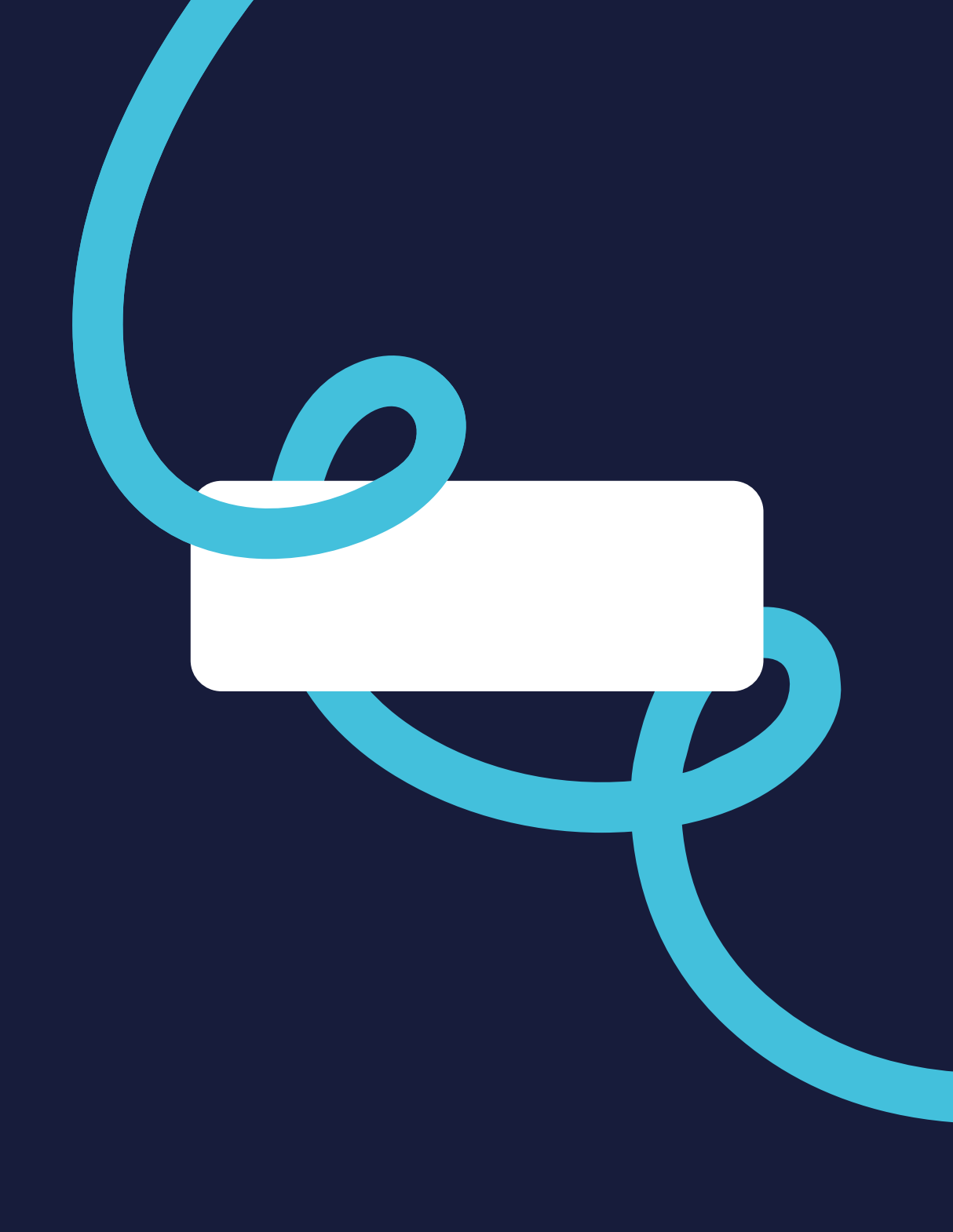
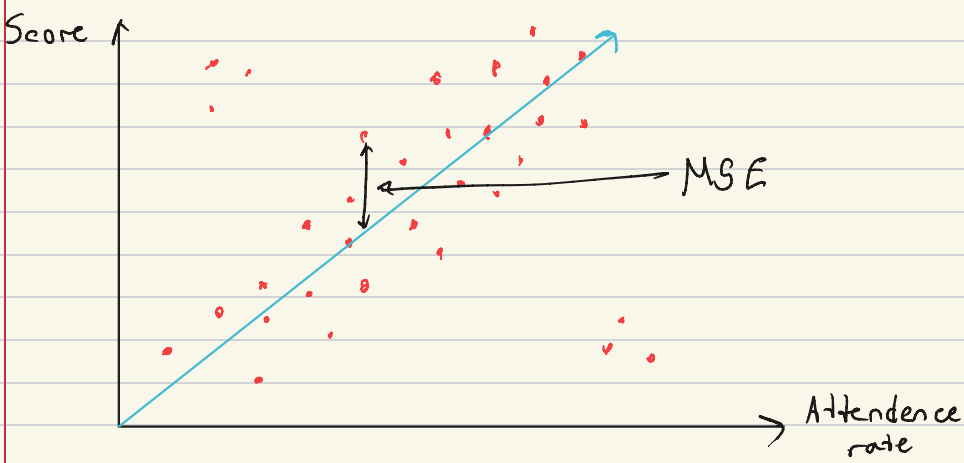


Mini project 00
Linear regression

Part 1 : Linear Regression

Input (feature) ⟶ attendence_rate

Output (Target) ⟶ capstone_score



line = ↗ ⟶ $y = m.x + b$ } $\hat{y}_i$ (Prediction)

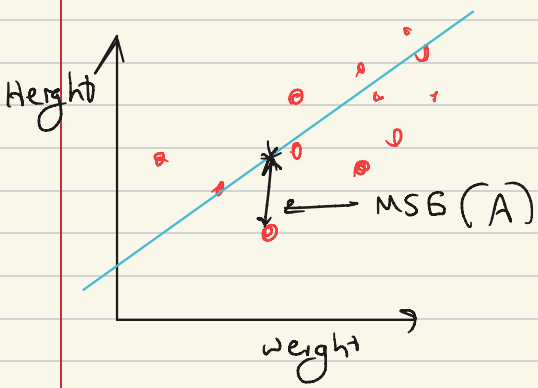
Hypothesis ⟶ $\underbrace{\text{Capstone\_score}}_{y} = \underbrace{B_0}_{b} + \underbrace{B_1 . \text{Attendence}}_{\text{features}}$

MSE ⟶ $\frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$

* We have to find the correct m & b to minimise MSE

Gradient Descent

- We optimize the best line to fit using gradient descent



Predicted height = intercept + slope x W

$$Y = B_0 + B_1 \cdot X$$

(X: input)

Finding the intercept using gradient descent

$$Y = B_0 + B_1 \cdot X$$

Slope

$B_1 \rightarrow 0.64$ (From Least squares)

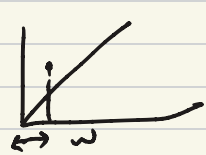
$B_0 \rightarrow 0$ (Any number)

Predicted height (Y)

$Y = 0 + 0.64 \times \text{weight}(0.5)$

$= 0 + 0.64 \times 0.5$

$= 0.32$



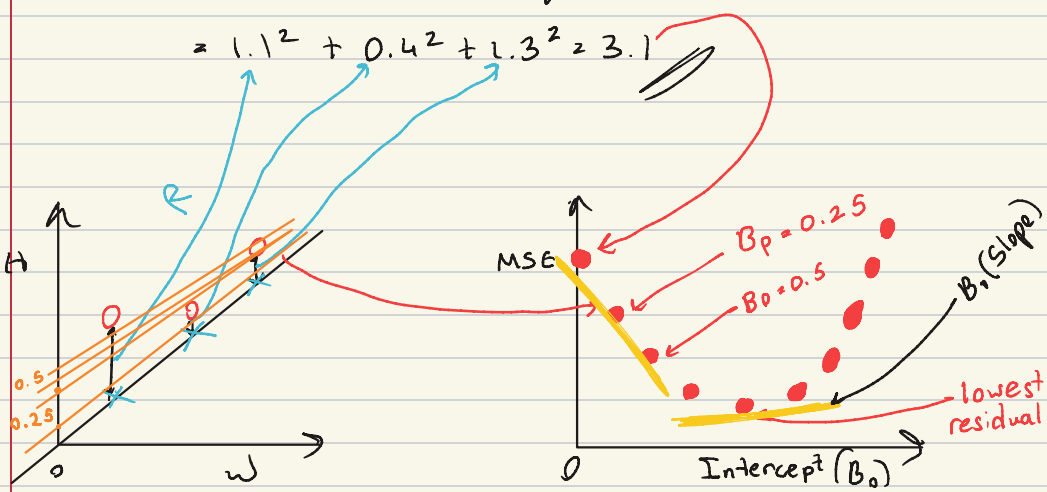
Then residual ⟶ $y - \hat{y}$

$= 1.4 - 0.32 = 1.1$

actual H — predicted H

* Now we need sum of the squared residuals ⟶ MSE Function

$= 1.1^2 + 0.4^2 + 1.3^2 = 3.1$



* Gradient descent only does a few calculations far from the solution

And increases the number of solution closer to to optimal value

Sum of squared residuals = $(1.4 - (\text{intercept} + 0.64 \times 0.5))^2$ + 2nd data point (•) + 3rd + ...

1st data point (•)

actual height

predicted height

This is MSE

* We can give any value for intcept (B_0) and get residual sum

* Now can take the derivate of the function and determine the slope at any value for the intercept

Sum of squared residuals = $(1.4 - (\text{intercept} + 0.64 \times 0.5))^2$

Taking derivatives

$$\frac{\partial}{\partial B_0} \text{Sum of residuals}^2 = \frac{\partial}{\partial B_0} (1.4 - (B_0 + 0.64 \times 0.5))^2$$

1st data point

+ 2nd + 3rd

$$= 2(1.4 - (\text{intercept} + 0.64 \times 0.5)) \times -1$$

$$= \frac{\partial}{\partial B_0} \cdot (1.4 - (\text{intercept}(B_0) + 0.64 \times 0.5))^2$$

$$= -2(1.4 - (B_0 + 0.64 \times 0.5))$$

+ 2nd dev + 3rd

Chain rule for solving derivatives

* We use chain rule method for solving derivatives easily

Slope ⟶ derivate

* multiplying rates of both outer and inner parts of the equation

1) $(1.4 - (B_0 + 0.64 \times 0.5))^2$

$= (1.4 - (B + 0.32))^2$

$= (1.08 - B)^2$

Outer function $(u)^2 \rightarrow 2u$

Inner function :- $1.08 - B \rightarrow -1$

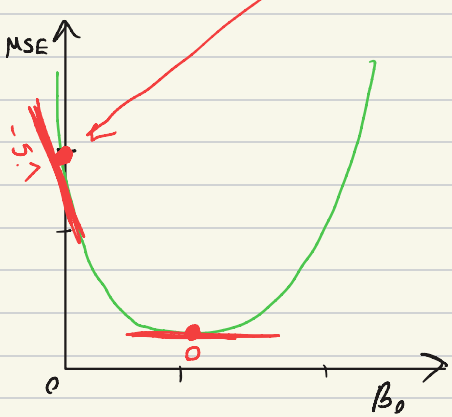
$= 2(1.4 - (B + 0.64 \times 0.5)) \times -1$

$= -2(1.4 - (B + 0.64 \times 0.5))$

Total sum of the derivative is $B_1 = 0$

Sum = 1st + 2nd + 3rd

Slope $\rightarrow$ -5.7 far from 0 $\rightarrow$ maximal value



* When the slope of curve is 0 we find the optimal value for B_0, we
* we should keep getting closer to 0

* Determine a step size to keep getting close to 0

Step size = $-5.7 \times \eta$

$\eta \rightarrow$ learning rate

Step Size = $-5.7 \times 0.1 = -0.57$ (η)

New B_0 = Old 0 (0) − Step size (−0.57)

= 0 − (−0.57)

= 0.57 → This is passed as a new value for B_0 and the derivative is calculated again

* This is done until slope is 0

* Gradient descent will stop after it reaches below 0.001
* Also have a limit of number of steps
  - Max No. of steps → 1000 or greater

Equations

1) Cost function (MSE)

$$J(B) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\beta(x^{(i)}) - y^{(i)}\right)^2$$

3) Gradient with respect B_0 (intercept)

$$\frac{\partial J(\beta)}{\partial \beta_0} = \frac{1}{m} \sum_{i=1}^{m} \left(h_\beta(x^{(i)}) - y^{(i)}\right)$$

2) Gradient with respect to B_j (Slope / weights)

$$\frac{\partial J(\beta)}{\partial \beta_j} = \frac{1}{m} \sum_{i=1}^{m} \left(h_\beta(x^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

Part 2: Polynomial Regression

* Instead of fitting a straight line we fit a polynomial curve by adding powers of x

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \dots\dots\dots \beta_d x^d$$

$d \longrightarrow$ degree

$d = 1 \longrightarrow$ straight line
$d = 2 \longrightarrow$ parabola
$d = 3 \longrightarrow$ cubic curve

Linear regression with multiple features

- We have multiple features

$$h_\beta(x^{(i)}) = \beta_0 + \beta_1 x_1^{(i)} + \beta_2 x_2^{(i)} + \dots\dots\dots$$

* we make in Matrix form

$h_\beta(X^{(i)}) \longrightarrow$ Feature Matrix
shape $(n, m+1)$

$n \longrightarrow$ training examples
$m \longrightarrow$ no. of features

$\beta \rightarrow (m+1, 1)$

$y \rightarrow (n, 1)$

$$\hat{y} = X \cdot \beta$$

Ex:-

$m = 3 + 1$ → +1 for bias column, because to match inner dimensions to multiply

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ 1 & x_{31} & x_{32} \end{bmatrix} \qquad \beta = \begin{bmatrix} B_0 \\ B_1 \\ B_2 \end{bmatrix}$$

$(3,3) \quad (3,1)$

$\times$

$\downarrow$

$$\hat{y} = X \cdot \beta$$

Cost function

MSE → $$J(\beta) = \frac{1}{2n} (X\beta - y)^T (X\beta - y)$$

Gradient rule

$$\nabla J(\beta) = \frac{1}{n} X^T (X\beta - y)$$

Update

$\beta = \beta - \alpha \cdot \nabla J(\beta)$

Feature Scaling

* We have to use feature scaling to standardize value when there are multiple values

Ex :- quize score 0 - 100 vs study hours are different

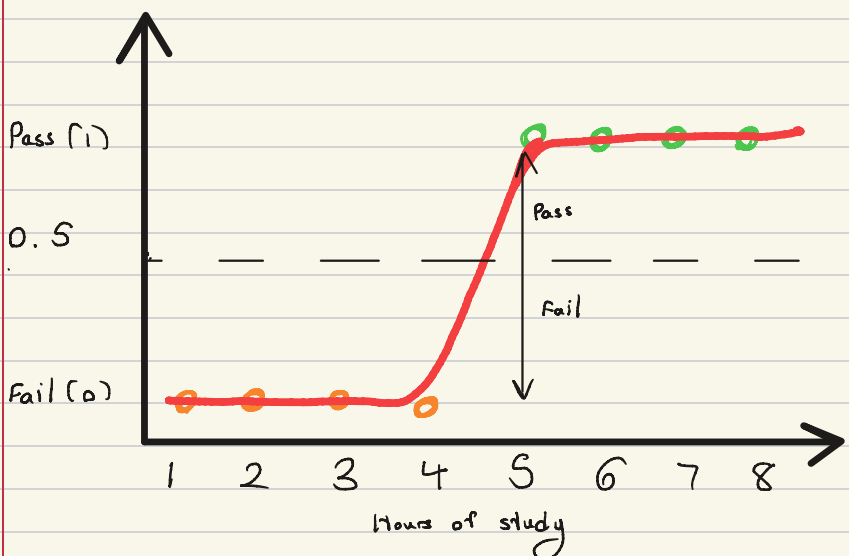
$$x' = \frac{x - \mu}{\sigma}$$

* makes all features have mean 0 and standard deviation 1 ⟶ faster convergence

Part 6: Algorithms in Classifications

Logistic regression

- This predicts if something is true or False



- Instead of fitting a straight line into the data, Logistic Regression fits an S-Shaped curve to the data

S shaped curve ⟶ Sigmoid function

* Sigmoid function takes any real number and makes it into a range between 0 and 1

Essential for interpreting a output as a Probability

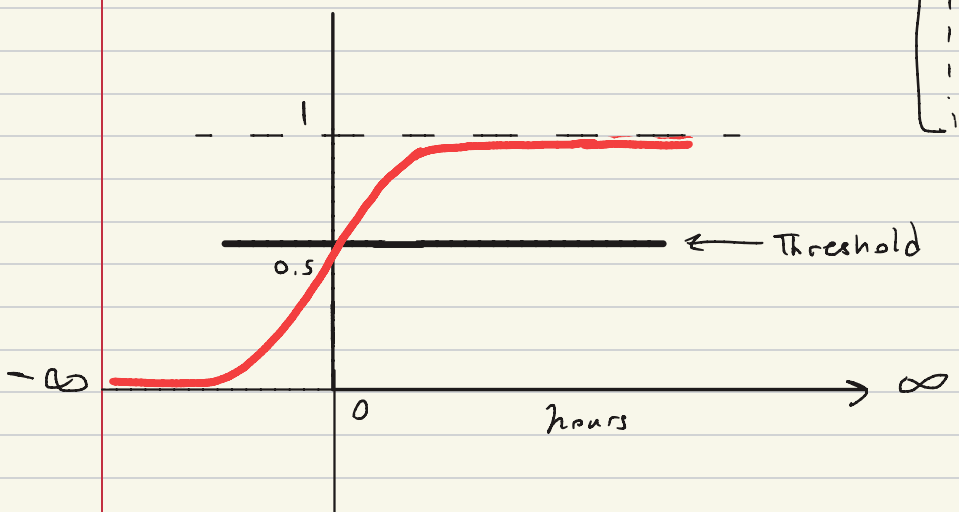
Mathematical Implementations

$\theta^T x^{(i)} = z^{(i)}$ → We use sigmoid function to convert z into a number between 0 & 1

- θ: n parameters
- $x^{(i)}$: n features

θ → theta will include both bias and slopes ⇒ X_bias . $f(w)$

$$\begin{bmatrix} 1 & 10 \\ 1 & 12 \\ 1 & 13 \\ 1 & 14 \end{bmatrix}$$




→ Sigmoid function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Model hypothesis

$$h_\beta(x) = \sigma(\beta^T x) = \frac{1}{1 + e^{-\beta^T x}}$$

$h_\beta(x)$: estimated probability for $x^{(i)}$

$\sigma(\beta^T x)$: Sigmoid function

→ gives the probability

* To optimize parameters sigmoid function is not enough

$$J(\beta) = \sum_{i=1}^{m} \left[h_\beta(x^{(i)}) \right]^{y^{(i)}} \left[1 - h_\beta(x^{(i)}) \right]^{1-y^{(i)}} = 1/0$$

$y^{(i)}$ ← actual value

$h_\beta(x^{(i)})$: predicted prob

* This function returns a value close to 0 and 1

- This function is not very optimal for calculating gradient descent

1) Negatate to minimize gradient descent → $-(-1)$
2) Numerically unstable → $\log(1n)$
3) Scaling the function → $\frac{1}{m}$

Binary Cross-Entropy Loss

$$J(\beta) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(h_\beta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\beta(x^{(i)})) \right]$$

Gradient of Cost function

Derivate of $J(\theta) \longrightarrow \nabla J(\theta) = \frac{1}{m} X^{T}(h_\theta(x) - y)$

Update gradient

$$\theta := \theta - \eta \cdot \nabla J(\theta)$$

* This tell us how much the gradient should change to reduce error