1. Confusion Matrix

A **confusion matrix** is a table used to evaluate the performance of a **classification model** by comparing predicted and actual values.

It helps derive metrics like accuracy, precision, recall, and F1 score.

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

TP : True +ve :-
TN : True -ve :-
FP :-
FN

Predicted → True
" → False
" → True
" → False

Actual → True
" → False
" → False
" → True

2. ☑ Accuracy

Measures how often the model is correct overall.

- Good when classes are balanced.
- Not reliable if one class dominates (e.g., 95% spam, 5% not spam).

Yes	No
50%	50%

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$= \frac{TP + TN}{TP + TN + FP + FN}$$

$$= \frac{TP + TN}{total}$$

3. R-squared Value (R^2) → Regression

Used in **regression models** to measure how well the predictions fit the actual data.

- Range: 0 to 1 (higher is better)
- $R^2 = 1$ → perfect prediction
- $R^2 = 0$ → no better than mean

$$R^2 = 1 - \frac{Sum\ of\ Squared\ Errors}{Total\ Sum\ of\ Squares} \quad ?$$

$$R^2 = \frac{\sum (y_p - \bar{y})^2}{\sum (y - \bar{y})^2}$$

y_p → predicted

$\bar{y}$ → mean of y

y → Actual

4. ⚖️ F1 Measure (F1 Score)

Harmonic mean of **precision** and **recall**. Used when you need a balance between the two.

Best when False Positives and False Negatives are equally costly.

$$F1 = 2.\frac{Precision\ .\ Recall}{Precision + Recall}$$

5. 🎯 Precision

Measures how many predicted positives are **actually correct**.

- High precision = few false positives.
- Useful when false alarms are expensive (e.g., spam filters).

$$Precision = \frac{TP}{TP + FP}$$

5. 🔍 Recall (Sensitivity / True Positive Rate)

Measures how well the model detects **actual positives**.

- High recall = few false negatives.
- Useful in medical tests, fraud detection, etc.

$$= \quad Recall = \frac{TP}{TP + FN}$$

FN :-
predicted → No
Actual → Yes

$$Precision = \frac{TP}{TP + FP}$$

predi → Yes
Actual → No

Metric	Used For	Formula	Key Focus
Confusion Matrix	Classification	—	Overall performance breakdown
Accuracy	Classification	$\frac{TP + TN}{total}$	Overall correctness
R-squared (R^2)	Regression	$\frac{\sum (y_p - \bar{y})^2}{\sum (y - \bar{y})^2}$	Fit of regression line
F1 Score	Classification	$2 \cdot \frac{precision \times Recall}{Precision + Recall}$	Balance between precision & recall
Recall	Classification	$\frac{TP}{TP + FN}$	Catching actual positives
Precision	Classification	$\frac{TP}{TP + FP}$	Reducing false positives

(b) Suppose you have written a classifier for selfie filtering. You have tested your classifier with some data and got the following predictions. Compute the Confusion Matrix from the output and calculate Accuracy, Precision, Recall, and F-1 score from that matrix.

Actual ↓ Predicted

Number	Target	Prediction
1	Selfie	Not Selfie (FN)
2	Not Selfie	Not Selfie (TN)
3	Selfie	Selfie (TP)
4	Not Selfie	Selfie (FP)
5	Not Selfie	Not Selfie (TN)
6	Not Selfie	Not Selfie (FN)
7	Selfie	Selfie (TP)
8	Selfie	Not Selfie (FN)
9	Not Selfie	Not Selfie (TN)
10	Not Selfie	Not Selfie (TN)

	Predicted Yes	Predicted No
Actual Yes	TP	FN
Actual No	FP	TN

TP → Yes | Yes

TN → No | No

FP → Yes | No

FN → No | Yes

	Pred – Selfi	Pred – Not s
Actual Selfi	TP = 2	FN = 2
Actual Not sel	FP = 1	TN = 5

Number	Target	Prediction	Type
1	Selfie	Not Selfie	FN
2	Not Selfie	Not Selfie	TN
3	Selfie	Selfie	TP
4	Not Selfie	Selfie	FP
5	Not Selfie	Not Selfie	TN
6	Not Selfie	Not Selfie	TN
7	Selfie	Selfie	TP
8	Selfie	Not Selfie	FN
9	Not Selfie	Not Selfie	TN
10	Not Selfie	Not Selfie	TN

TP
TN
FP
FN

FP = 1
FN = 2

TP = 2 (row 3, 7)
TN = 5 (row 2, 5, 6, 9, 10)

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{2+5}{10} = \square \times 100\% = \square$$

$$Precision = \frac{TP}{TP + FP} = \frac{2}{2+1} = \square \times 100\% = \square$$

$$Recall = \frac{TP}{TP + FN} = \frac{2}{2+2} = \square \times 100\% = \square$$

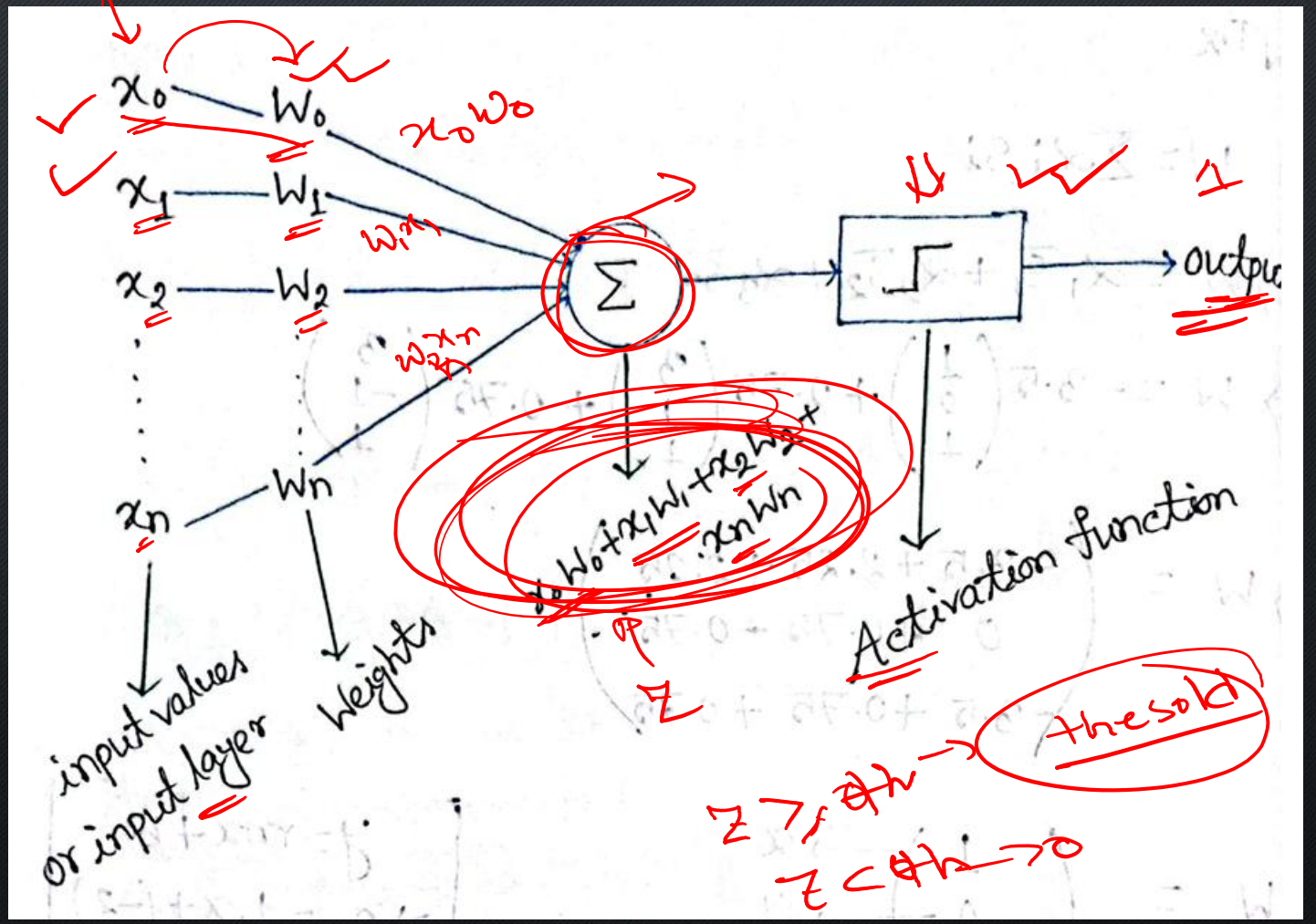
$$F1 = 2.\frac{Precision\,.\,Recall}{Precision + Recall} = \checkmark$$

Perceptron → Neural Networks

- ✓ Perceptron is considered a single-layer neural links with four main parameter.
- ✓ A machine learning based algorithm used for supervised learning of binary classification.
- ✓ Used for binary classification problems.
- ✓ Only works for **linearly separable** data.
- ✓ Cannot solve problems like **XOR**. → multi-layer perceptron
- ✓ Application:
  1. Image recognition (simple tasks) → Cat → Dog
  2. Binary classification problems
  3. Basic pattern recognition →

Yes NO
0 1

Perceptron


x_0
W_0
$x_0 w_0$
x_1
W_1
x_2
W_2
x_n
W_n
Σ
output
$W_0 + x_1 w_1 + x_2 w_2 + \ldots x_n w_n$
Z
Activation function
input values
or input layer
Weights
threshold
$Z > \theta_h \rightarrow 1$
$Z < \theta_h \rightarrow 0$

Perceptron

Structure:

Inputs (x_1, x_2, ..., x_n): Features of the input data.

Weights (w_1, w_2, ..., w_n): Assigned to each input.

Bias (b): Allows the model to shift the decision boundary.

Activation Function: Typically a **step function** that outputs 0 or 1.

Output Function:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

$$f(z) = \begin{cases} 1 & if\ z \geq 0 \\ 0 & if\ z < 0 \end{cases}$$

Perceptron

(c) Apply Perceptron Training Algorithm to the following dataset. Use $\alpha = 0.5$, Th = 1, w1 = 1.2 and w2 = 0.6 for two iterations. Show the final updated weights using the figure of a perceptron.

epoch

$z = w_1 x_1 + w_2 x_2$

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

th = 1
w1 = 1.2
w2 = 0.6
α = 0.5

Perceptron

Perceptron Basics:

- The perceptron predicts using:

$$z = w1 * x1 + w2 * x2$$

If $z \geq Th$, predict 1; else predict 0.

- If the prediction is wrong, update the weights:

$$w1 = w1 + \alpha * (y - prediction) * x1$$
$$w2 = w2 + \alpha * (y - prediction) * x2$$

Perceptron

Initial Setup: α = 0.5 , Th = 1 , w1 = 1.2, w2 = 0.6

Epoch-1

x_1	y_1	y	y_p	$y - y_p$	w_1	w_2	z
0	0	0	0	0	1.2	0.6	0
0	1	1	1	1	1.2	1.1	1.1
1	0	1	1	1	1.7	1.1	1.2
1	1	1	1	0	1.2	1.1	1.8

$w_1 = 1.2 \quad w_2 = 1.1$

$w_1 = w_1 + \alpha (y - y_p) \times x_1$

$= 1.2 + 0.5 \times 1 \times 1 = 1.7$

$w_2 = w_2 + \alpha (y - y_p) \times x_2$

$= 1.1 + 0.5 (1) \times 0$

$= 1.1$

$z = 1.7 \times 1 + 1.1 \times 0$

$= 1.7 > th$

Perceptron

Epoch-2

x_1	y_1	y	y_p	$y - y_p$	w_1	w_2	z

$w_1 = w$

🔁 Activation Function

An **activation function** in a neural network decides whether a neuron should be **activated** (i.e., whether it should pass its signal to the next layer). It introduces **non-linearity** into the network, enabling it to learn and solve complex problems.

Without activation functions, a neural network would behave like a simple linear model, no matter how many layers it has.

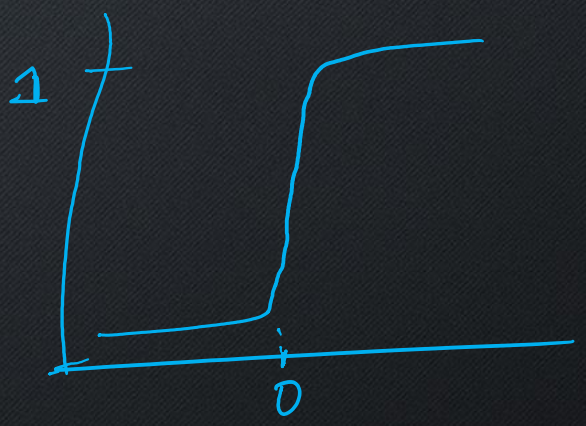
🎯 Role of Activation Functions

- ☑ Introduce **non-linearity** into the model.
- ☑ Help the network learn **complex patterns** in data.
- ☑ Decide whether a neuron should "fire" or stay inactive.
- ☑ Enable deep neural networks to learn **hierarchical representations**.

🔧 Types of Activation Functions

1. Sigmoid Function

$$S(x) = \frac{1}{1 + e^{-x}}$$



- ✓ Function that is s-shaped graph.
- ✓ Value range : (0, 1)
- ✓ Use case: Binary Classification problems.
- ✓ Non-linear Activation function
- ✓ Smooth output, interpretable as probability.
- ✓ Cons: Vanishing gradient problem (output isn't zero-centered), slow learning.

🔧 Types of Activation Functions

2. Tanh (Hyperbolic Tangent) Function

$$T(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- ✓ Function that is s-shaped graph.
- ✓ Zero-centered.
- ✓ Value range : $(-1, 1)$
- ✓ Use case: Hidden layer (better than sigmoid).
- ✓ Zero-centered output, stronger gradients than sigmoid.
- ✓ Still suffers from vanishing gradients.

🔧 Types of Activation Functions

3. ReLU (Rectified Linear Unit)

$$f(x) = \max(0, x)$$

- ✓ Value range : [0, ∞)
- ✓ Use case: Most popular in hidden layers of deep networks.
- ✓ Simple, fast, helps prevent vanishing gradients.
- ✓ Dying ReLU problem (neurons stop learning if struck at 0).

📋 Comparison Table

Function	Output Range	Common Use	Pros	Cons
Sigmoid	(0, 1)	Output layer (binary)	Probabilities	Vanishing gradient
Tanh	(−1, 1)	Hidden layers	Zero-centered	Vanishing gradient
ReLU	[0, ∞)	Hidden layers	Fast, efficient	Dying neurons

[0, ∞)

* Advantage & Disadvantage

Naïve Byes

A	B	C	D	target
				1 / 0

1 = Yes
0 = No

A = 1
B = 0
C = 1
D = 0

$$P(A=1 \mid Yes) = \frac{A=1 \text{ count}}{\text{total Yes}}$$

$$P(C=Yes \mid x = A, B, C, D) =$$

$$= \frac{P(A=1 \mid Yes) * P(B=0 \mid Yes) * P(C=1 \mid Yes) * P(D=0 \mid Yes) * P(Yes)}{P(A) \times P(B) \times P(C) \times P(D)}$$

1. You have data from a telecom company. Each row represents a customer, and the goal is to predict whether a customer will decide (leave the company) or Stay.

ContractType = Postpaid, Bill = High, Method = Credit

Contract Type	Monthly Bill	Support Calls	Payment Method	Decision
Prepaid	Low	Few	Online	Stay
Postpaid	High	Many	Bank Transfer	Leave
Prepaid	Medium	Few	Online	Stay
Postpaid	High	Few	Bank Transfer	Stay
Postpaid	High	Many	Credit Card	Leave
Prepaid	Low	Many	Online	Leave
Postpaid	Medium	Few	Credit Card	Stay

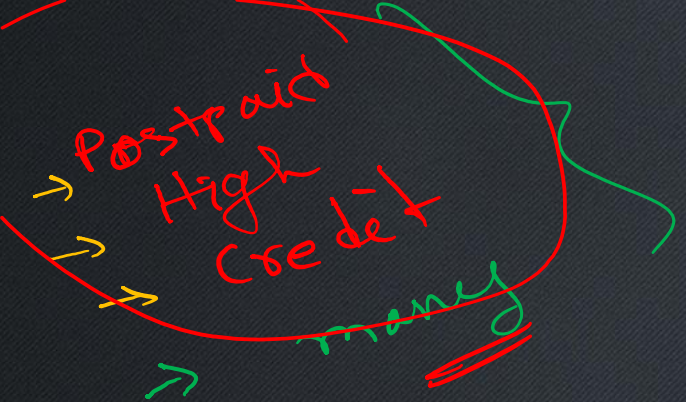
$\therefore P(\text{stay}) = \frac{4}{7}$

$\therefore P(\text{leave}) = \frac{3}{7}$

stay = 4

leave = 3

$= \frac{0}{4}$



$$P(C=\text{stay}\mid x) = \frac{P(\text{postpaid}\mid\text{stay}) * P(\text{high}\mid\text{stay}) * P(\text{credit}\mid\text{stay}) * P(\text{many}\mid\text{stay}) * P(\text{stay})}{P(\text{postpaid}) * P(\text{High}) * P(\text{credit}) * P(\text{many})}$$

$$= \frac{\frac{2}{4} \times \frac{1}{4} \times \frac{0}{4} \times \frac{1}{4} \times \frac{4}{7}}{\frac{4}{7} \times \frac{3}{7} \times \frac{3}{7} \times \frac{2}{7}}$$

$$= 0$$

$$P(\text{leave} \mid x)$$

$$= \frac{P(\text{postpaid} \mid \text{leave}) \times P(\text{high} \mid \text{leave}) \times P(\text{many} \mid \text{leave}) \times P(\text{many credit} \mid \text{leave}) \times P(\text{leave})}{P(\text{postpaid}) * P(\text{high}) * P(\text{credit}) * P(\text{many})}$$

$$= \frac{\frac{2}{3} \times \frac{3}{3} \times \frac{2}{3} \times \frac{1}{3} \cdot \times \frac{3}{7}}{\frac{4}{7} \times \frac{3}{7} \times \frac{2}{7} \times \frac{3}{7}}$$

$$= \boxed{?}$$

KNN

You are given the following 2D dataset representing the coordinates of customers based on their location in a city:

Customer	X (Longitude)	Y (Latitude)
A	1	2
B	2	1
C	2	2
D	1	3
E	2	3
F	8	8
G	9	8
H	9	9
I	8	9
J	7	8

3. **Attempt any TWO** $(2 \times 10 = 20)$

(a) Apply K-means clustering algorithm for the given data: (2, 3), (5, 6), (8, 7), (1, 4), (2, 2), (6, 7), (3, 4), (8, 6).
Divide the datapoints into two clusters C1 and C2, where initial centroids for C1 = (2, 3) and C2 = (5, 6).

C1	C2
2,3	5,6
1,4	8,7

centroid	cents
2,3	~~5,6~~
	6.5, 6.5

For (8,7)

$c_1 = \sqrt{(2-8)^2 + (3-7)^2} = 7.211$

$c_2 = \sqrt{(5-8)^2 + (6-7)^2} = 3.16$

(8,7) $\longrightarrow$ C2

new centroid $= \left(\frac{5+8}{2}, \frac{6+7}{2}\right) = (6.5, 6.5)$

For (1,4)

$c1 = \sqrt{(2-1)^2 + (3-4)^2} = 1.41$

$c2 = \sqrt{(6.5-1)^2 + (6.5-4)^2} = 6.04$

c1	c2
~~2,3~~ 1.5, 1.5	6.5, 6.5

(1,4) $\longrightarrow$ c1

new centroid $= \left(\frac{2+1}{2}, \frac{3+4}{2}\right) = (1.5, 1.5)$

For (2,2)

$$c1 = \sqrt{(1.5-2)^2 + (1.5-2)^2} = 1.58$$

$$c2 = \sqrt{(6.5-2)^2 + (6.5-2)^2} = 6.36$$

c1	c2
2,3	7,6
1,4	8,7
2,2	6,7
~~1.5,1.5~~	6.5,6.5
	1.67,3

$(2,2) \longrightarrow c1$

$$\text{new centroid} = \left(\frac{2+1+2}{3}, \frac{3+4+2}{3}\right)$$

$$= (1.67, 3)$$

(6, 7)

$c_1 = \sqrt{(1.67-6)^2 + (3-7)^2} = 5.89$

$c_2 = \sqrt{(6.5-6)^2 + (6.5-7)^2} = 0.707$

(6, 7) ⟶ c2

2, 4

$c_1 = \sqrt{(1.67-3)^2 + (3-3)^2} =$ ☐

$c_2 = \sqrt{(5.667-3)^2 + (6.667-4)^2} =$ ☐

New = $\left(\frac{5+6+6}{3}, \frac{6+7+7}{3}\right)$

= (5.667, & 6.6667)

Entropy = −

Information Gain