# Báo cáo tuần 2

## 1. Perceptron Learning Algorithm

### 1. Tổng quan thuật toán:

**Perceptron** là một trong những thuật toán *classification* cơ bản nhất, áp dụng cho:

- **Binary classification**: chỉ có **2 class**.
- **Dữ liệu tuyến tính phân tách được** (*linearly separable*)

#### 1.1. Ký hiệu và mô hình toán học:

- **Dữ liệu:**
  $X = [x_1, x_2, \ldots, x_N] \in \mathbb{R}^{d \times N}$
  Mỗi $x_i \in \mathbb{R}^{d \times 1}$ là vector **cột** (ở đây không dùng vector hàng như một số bài trước).
- **Nhãn:**
  $y = [y_1, y_2, \ldots, y_N] \in \mathbb{R}^{1 \times N}$
  Với:

$$y_i = \begin{cases} 1 & \text{nếu thuộc class 1 (xanh)} \\ -1 & \text{nếu thuộc class 2 (đỏ)} \end{cases}$$

- **Boundary (siêu phẳng phân chia):**

$$f_w(x) = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + w_0 = w^T \bar{x} = 0$$

  Trong đó:
  $\bar{x}$ là vector mở rộng từ $x$ bằng cách thêm $x_0 = 1$ (bias).
- **Quy tắc phân loại:**

$$\text{label}(x) = \text{sgn}(w^T x)$$

  với $\text{sgn}(0) = 1$.

## 1.2.Hàm mất mát:

### 1.2.1: Hàm mất mát đơn giản nhất:

$$J_1(w) = \sum_{x_i \in M} (-y_i \cdot \text{sgn}(w^T x_i))$$

-Định nghĩa: Hàm này bằng tổng số tất cả các điểm sai(misclassified)

-Vấn đề: Hàm này rời rạc → **không đạo hàm được** → khó tối ưu bằng giải tích.

### 1.2.2. Hàm mất mát khả vi hơn:

$$J(w) = \sum_{x_i \in M} (-y_i w^T x_i)$$

-Điểm misclassified nằm càng xa đường boundary thì hàm sẽ càng lớn, hàm bằng 0 nếu không có misclassified.

-Hàm mất mát này cũng được cho là tốt hơn vì nó trừng phạt rất nặng những điểm lấn sâu sang lãnh thổ của class kia. Trong khi đó, J1 trừng phạt các điểm misclassified như nhau (đều = 1), bất kể chúng xa hay gần với đường biên giới.

### 1.3. Gradient và tối ưu hàm mất mát:

Với một điểm sai $x_i$:

$$J(w; x_i; y_i) = -y_i w^T x_i$$

Gradient:

$$\nabla_w J(w; x_i; y_i) = -y_i x_i$$

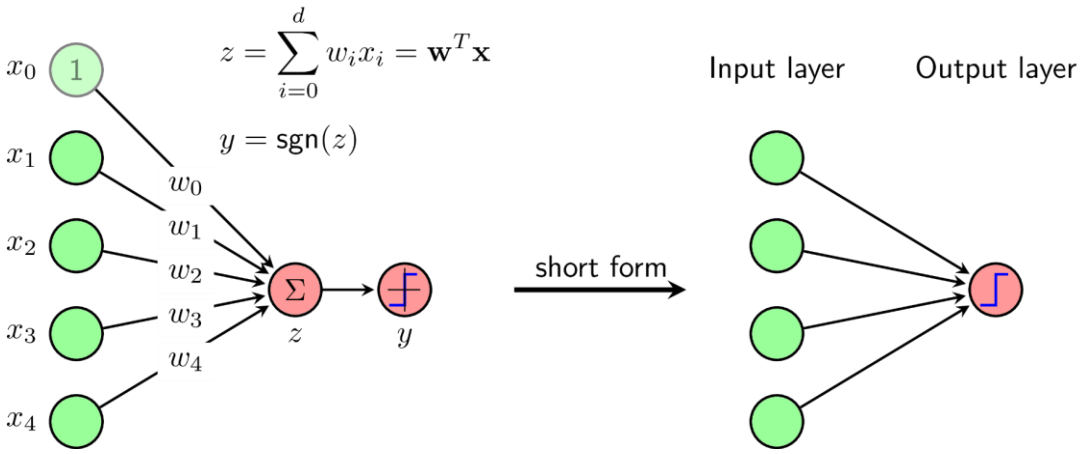
Cập nhật:

$$w \leftarrow w + \eta y_i x_i$$

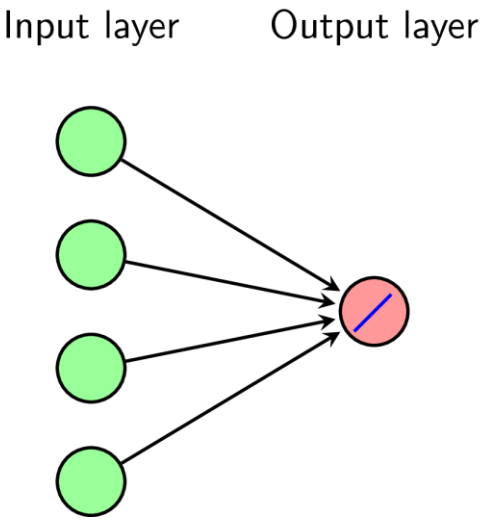
- **Learning rate** $\eta = 1$ (thường dùng trong PLA).
- Tức là:
  - Nếu điểm sai thuộc class $+1$ → đẩy boundary về phía bao trùm nó.
  - Nếu điểm sai thuộc class $-1$ → kéo boundary tránh xa nó.

## 2. Mô hình Neural Network đầu tiên:

Perceptron như một Neural Network đơn giản nhất




-> Đây chính là **một mạng Neural Network 1 tầng (single-layer neural network)**.
-Hàm số **y=sgn(z)** còn được gọi là *activation function*. Để ý rằng Để ý rằng nếu ta thay *activation function* bởi y=z ta sẽ có Neural Network mô tả thuật toán Linear Regression như hình dưới.




→ Điều này mở ra ý tưởng: **thay activation khác nhau → mô hình học máy khác nhau**.

## 2. Logistic Regression

### 1.Tổng quan thuật toán:

Mặc dù tên gọi là "regression" (hồi quy), Logistic Regression thực chất là một **mô hình phân loại**, chứ không phải mô hình hồi quy dự đoán giá trị liên tục.

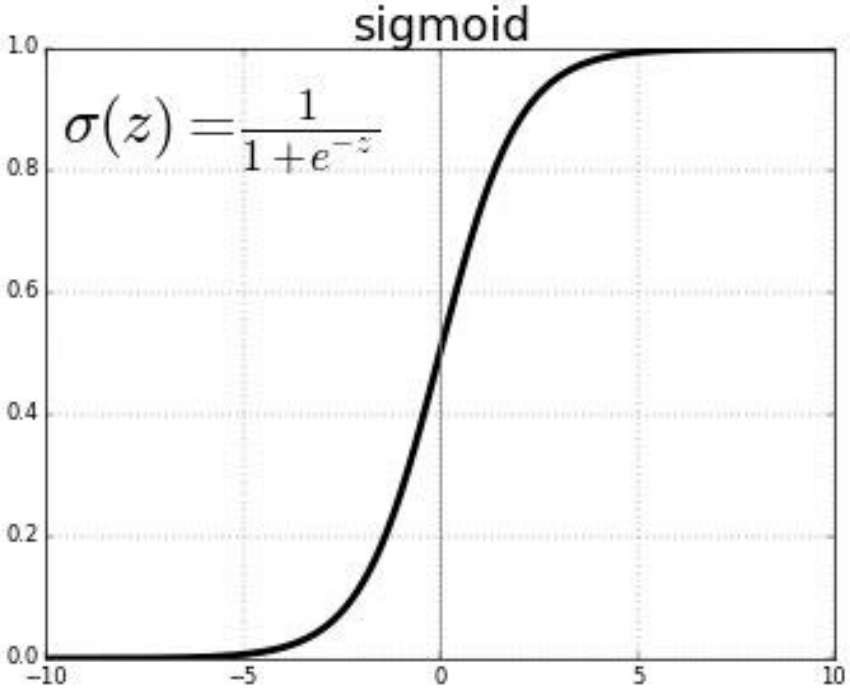
Nó hoạt động bằng cách:

- Áp dụng **hồi quy tuyến tính** để tính tổ hợp tuyến tính của các đặc trưng đầu vào:

$$z=w1x1+w2x2+...+wnxn+w0$$

- Sau đó, sử dụng **hàm sigmoid** để chuyển đổi giá trị z thành xác suất:

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




- Dự đoán cuối cùng:
    - Nếu σ(z)>0.5 → lớp 1
    - Nếu σ(z)≤0.5→ lớp 0

## 2. Hàm mất mát:

### 2.1.Xây dựng hàm mất mát :

-**Likelihood (hàm hợp lý)** trong Logistic Regression chính là tích các xác suất mà mô hình dự đoán đúng cho toàn bộ tập dữ liệu:

$$P(y \mid X; w) = \prod_{i=1}^{N} P(y_i \mid x_i; w) = \prod_{i=1}^{N} z_i^{y_i} (1 - z_i)^{1-y_i}$$

**-**Likelihood cho biết mức độ phù hợp của mô hình, ta cần tìm w để để cực đại hóa nó (**Maximum Likelihood Estimation – MLE**).

-Để thuận tiện cho tính toán:

+Lấy log để biến tích thành tổng.

+Lấy dấu trừ để biến thành hàm mất mát cần tối thiểu hóa:

$$J(w) = -\log P(y \mid X; w) = -\sum_{i=1}^{N} \Big( y_i \log z_i + (1 - y_i) \log(1 - z_i) \Big)$$

-Đây chính là **Cross-Entropy Loss**.

### 2.2. Tôi ưu hàm mất mát:

-Với mỗi điểm dữ liệu, ta đạo hàm riêng theo w:

$$J(w; x_i, y_i) = -\big( y_i \log z_i + (1 - y_i) \log(1 - z_i) \big)$$

$$\frac{\partial J(w; x_i, y_i)}{\partial w} = (z_i - y_i) x_i$$

-Cập nhật tham số:

$$w := w - \eta \frac{\partial J}{\partial w} = w - \eta (z_i - y_i) x_i = w + \eta (y_i - z_i) x_i$$

Trong đó:

- $\eta$ = learning rate.
- $z_i = \sigma(w^T x_i)$ = xác suất dự đoán.
- $y_i \in \{0, 1\}$ = nhãn thực tế.

## 3. Softmax Regression

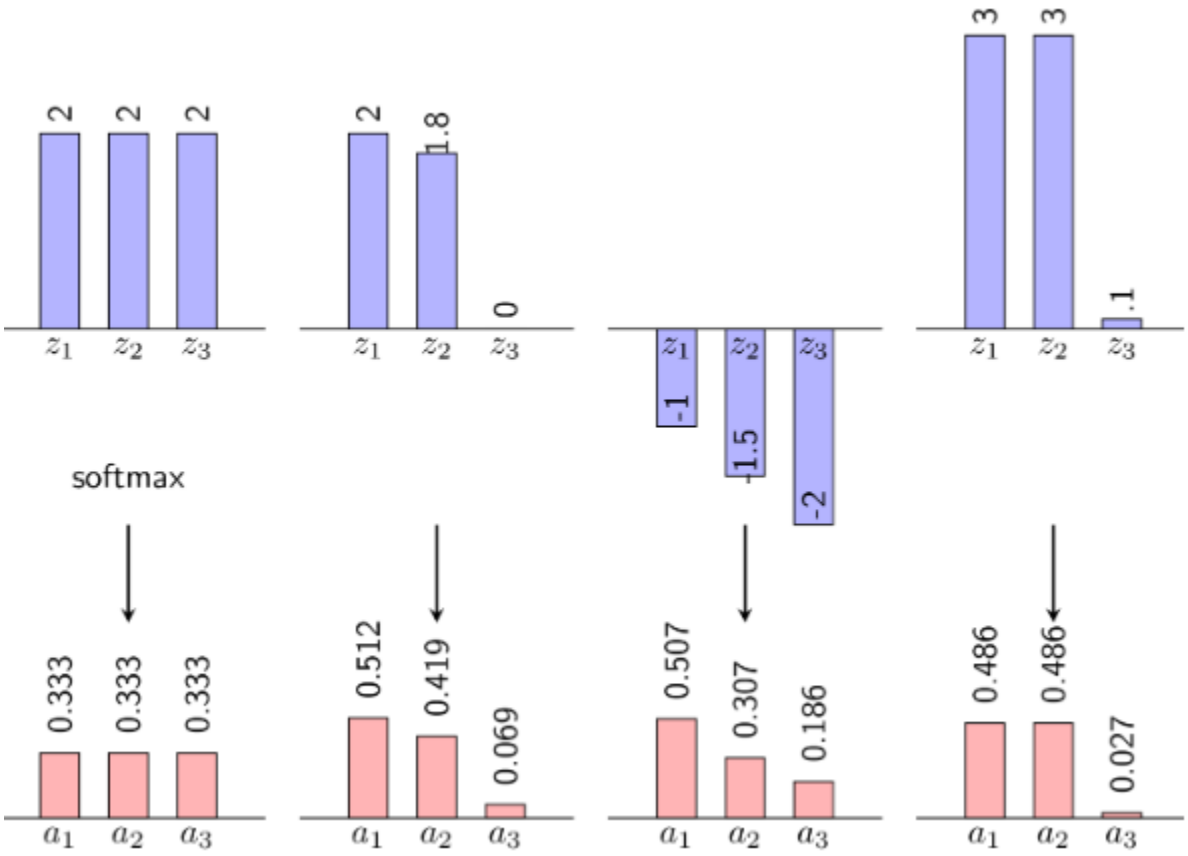
**-Softmax Regression** là phần mở rộng của **Logistic Regression** cho **bài toán phân loại nhiều lớp (multiclass classification)**.

-Nếu Logistic Regression chỉ xử lý **nhị phân (2 lớp)**, thì Softmax Regression có thể xử lý trường hợp **C lớp (C > 2)**.

### 1.Softmax Function

$$a_i = \frac{\exp(z_i)}{\sum_{j=1}^{C} \exp(z_j)}, \quad \forall i = 1, 2, \ldots, C$$

Ví dụ về đầu vào và đầu ra hàm softmax:




## 2.Hàm mất mát:

-Hàm **loss** trong Softmax Regression là **Cross-Entropy Loss**:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log \hat{y}_{i,c}$$

Trong đó:

- N: số lượng mẫu trong tập dữ liệu.
- C: số lớp.
- y: nhãn thật của mẫu thứ i, dạng one-hot (chỉ có 1 giá trị bằng 1, còn lại bằng 0).
- y_hat:  xác suất mô hình dự đoán mẫu thứ i thuộc lớp c.

## 3.Tối ưu hàm mất mát

-Làm tương tự như Logistic Regression:

+Tính đạo hàm riêng của loss theo W,b

+Cập nhật tham số:

$$W \leftarrow W - \eta \frac{\partial L}{\partial W}, \quad b \leftarrow b - \eta \frac{\partial L}{\partial b}$$

Với $\eta$ là learning rate

-Ngoài Gradient Descent cơ bản, có thể dùng:

**+Stochastic Gradient Descent (SGD)**: cập nhật theo từng mẫu.

**+Mini-batch Gradient Descent**: cập nhật theo nhóm mẫu nhỏ.

**+Các optimizer nâng cao**: Adam, RMSProp, Adagrad… (thường dùng trong deep learning).

# 4. Overfitting, Underfitting

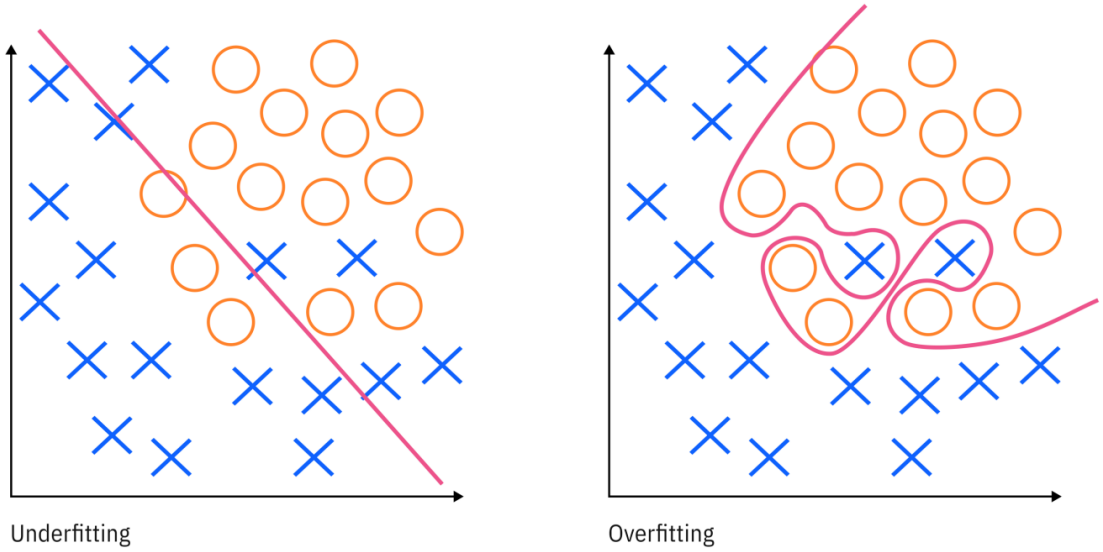
## 1. Định nghĩa:

**Underfitting (không vừa đủ học):** Mô hình quá đơn giản, không nắm bắt được xu hướng dữ liệu → hiệu suất kém cả trên tập huấn luyện và tập kiểm thử.

- Tỷ lệ lỗi **cao** trên cả hai tập.
- Nguyên nhân: mô hình đơn giản, thiếu đặc trưng, dữ liệu không đủ, hoặc dùng regularization quá mạnh.

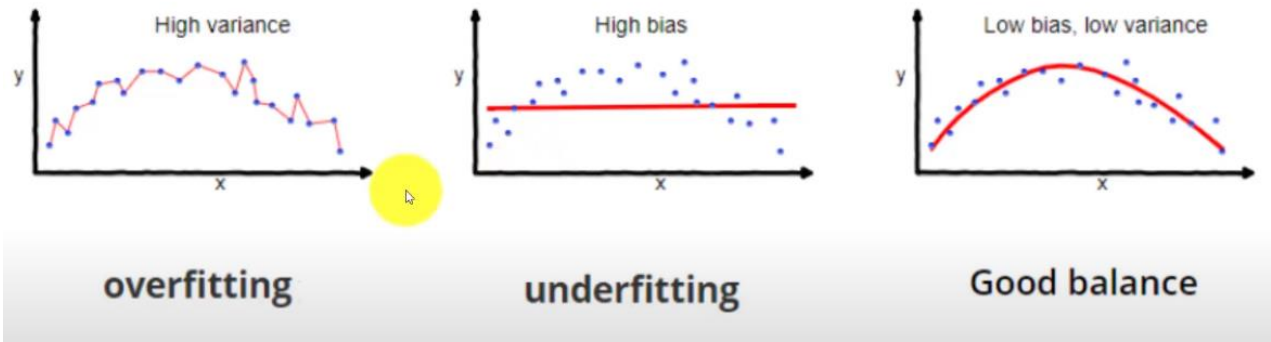
**Overfitting (học quá khớp):** Mô hình học quá kỹ dữ liệu huấn luyện, kể cả nhiễu → hiệu suất tốt trên huấn luyện nhưng thất bại khi gặp dữ liệu mới.

- Tỷ lệ lỗi **thấp** trên huấn luyện, nhưng **cao** trên kiểm thử.
- Nguyên nhân: mô hình quá phức tạp, dữ liệu ít hoặc nhiễu, quá nhiều tham số




## 2. Bias và Variance — Hai lỗi cốt lõi cần cân bằng

- **Bias cao, Variance thấp** → thường dẫn đến **underfitting** (mô hình không đủ linh hoạt).
- **Bias thấp, Variance cao** → thường dẫn đến **overfitting** (mô hình nhạy cảm quá mức với dữ liệu huấn luyện).



## 3. Cách khắc phục:

### 3.1. Underfitting:

**Tăng độ phức tạp của mô hình:** Thêm layers, thêm neurons, chọn thuật toán phức tạp hơn.

**Huấn luyện lâu hơn:** Tăng số epochs, giảm learning rate để mô hình học sâu hơn.

**Feature Engineering:** Tạo ra đặc trưng mới, chọn lọc đặc trưng hữu ích.

### 3.2. Overfitting:

- **Giảm số lượng biến giải thích**:

+ Lựa chọn thủ công bộ các biến số
+ Thuật toán lựa chọn biến số

-**Phương pháp Cross-validation:**

+ Thay vì tách dữ liệu cố định thành train/validation, ta chia dữ liệu thành *k* phần (fold).

+ **Cách làm (k-fold CV):**

1. Chia tập training thành *k* phần bằng nhau.

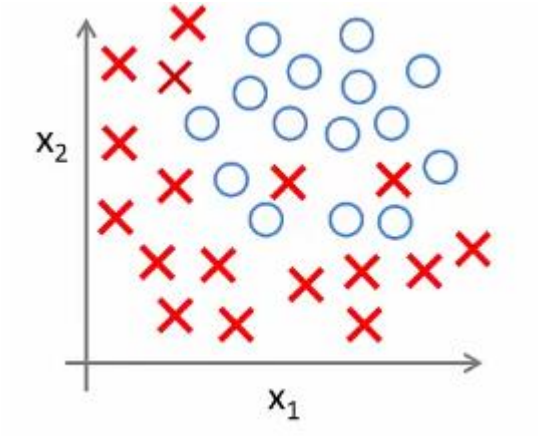
2. Lặp *k* lần: mỗi lần chọn 1 phần làm validation, phần còn lại để train.

3. Lấy trung bình kết quả của *k* lần để đánh giá mô hình.

- **Phương pháp Regularization**:

+ Giữ nguyên tất cả các biến số nhưng làm giảm đi độ lớn của các tham số $\theta j$.
+ Đưa ra kết quả tốt khi có rất nhiều biến số, và mỗi biến số đều có một phần ý nghĩa trong dự báo biến y.

Ví dụ: Hồi quy Logistic với Regularization:




Ta có:

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \dots)$$

Hàm loss của Hồi quy Logistic:

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

Hàm loss của Hồi quy Logistic với Regularization:

$$J(\theta) = \left[-\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

## 5. Đánh giá hệ thống phân lớp

### 1.Confusion Matrix :

Là một bảng 2x2 (cho binary classification) thể hiện số lượng dự đoán **đúng** và **sai** của mô hình:

| | **Dự đoán: Positive** | **Dự đoán: Negative** |
|---|---|---|
| **Thực tế: Positive** | TP (True Positive) | FN (False Negative) |
| **Thực tế: Negative** | FP (False Positive) | TN (True Negative) |

**-TP (dương thật)**: dự đoán đúng là positive.

**-TN (âm thật)**: dự đoán đúng là negative.

**-FP (dương giả)**: dự đoán positive nhưng thực tế negative.

**-FN (âm giả)**: dự đoán negative nhưng thực tế positive.

## 2.Accuracy Score :

Thể hiện tỉ lệ dự đoán đúng trên tổng số mẫu.

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

**Nhược điểm**: không tốt khi dữ liệu **mất cân bằng** (ví dụ 95% Negative, chỉ 5% Positive → mô hình đoán tất cả Negative thì Accuracy vẫn 95%).

## 3.Precision:

Trong tất cả mẫu **dự đoán dương tính**, có bao nhiêu mẫu thực sự dương tính.

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

**Ứng dụng**: khi chi phí của **dự đoán sai dương tính** (FP) cao.

## 4. Recall:

Trong tất cả mẫu **thực tế dương tính**, mô hình dự đoán đúng bao nhiêu.

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

**Ứng dụng**: khi chi phí của **dự đoán sai âm tính** (FN) cao.

## 5.F1-Score:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Trung hòa Precision và Recall

Dùng khi cần cân bằng giữa Precision và Recall, đặc biệt khi dữ liệu mất cân bằng.

## 6. ROC Curve:

Là đường cong biểu diễn **TPR (Recall)** theo **FPR** khi thay đổi ngưỡng phân loại:

-Trục X: **False Positive Rate** (FPR):

$$FPR = \frac{FP}{FP + TN}$$

-Trục Y: **True Positive Rate (Recall):**

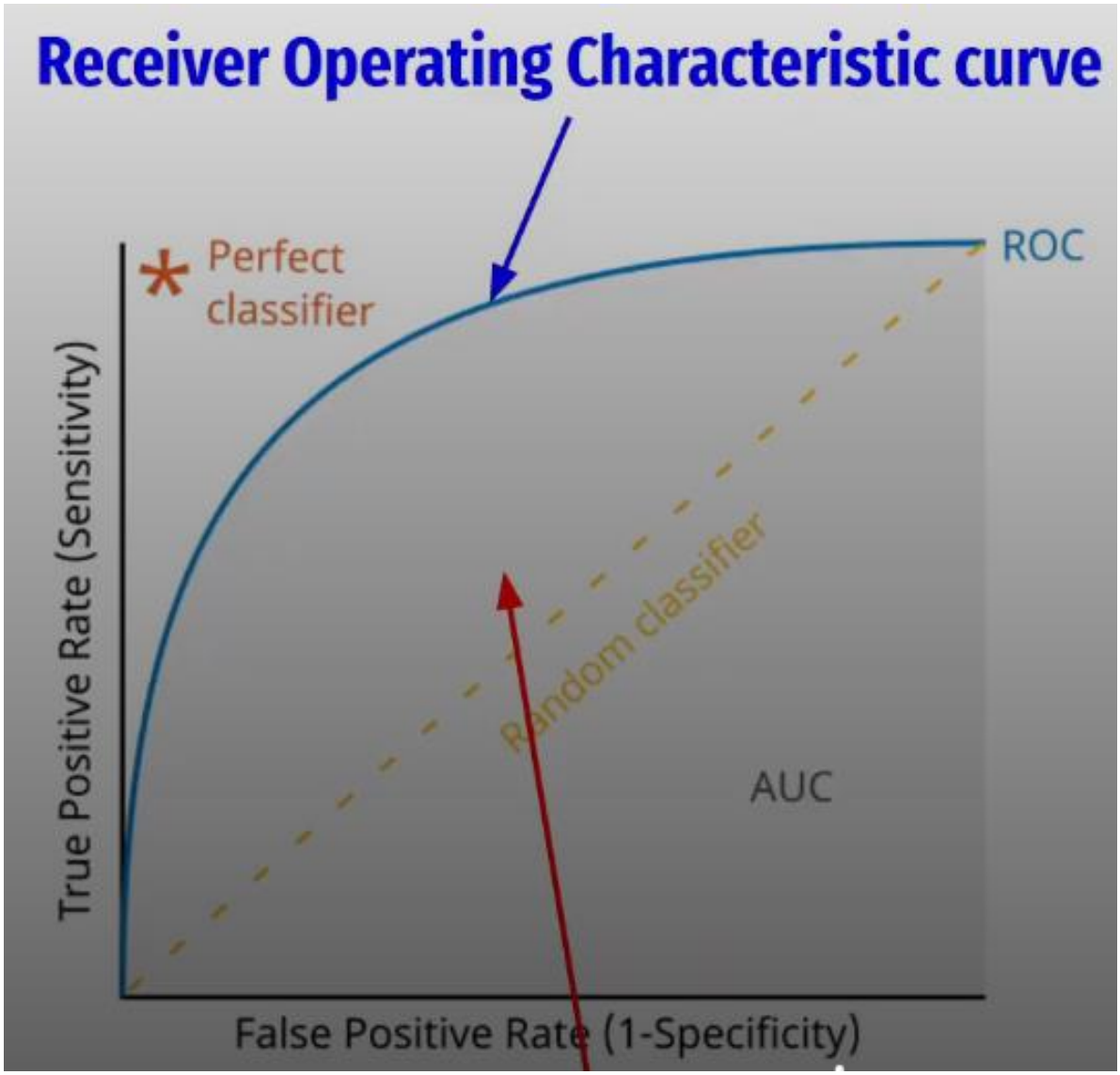
$$TPR = \frac{TP}{TP + FN}$$

-**AUC (Area Under Curve):** diện tích dưới ROC curve:

AUC= 1.0: Mô hình hoàn hảo

AUC=0.5: Mô hình tệ

-> Dùng ROC–AUC khi cần so sánh các mô hình và muốn một thước đo độc lập ngưỡng.


Receiver Operating Characteristic curve
Perfect classifier
ROC
Random classifier
AUC
True Positive Rate (Sensitivity)
False Positive Rate (1-Specificity)