Module 2 - Tuần 1 - Tổng hợp kiến thức Buổi học số 4

Time-Series Team

Ngày 4 tháng 7 năm 2025

Buổi học số 4 (Thứ 6, 04/07/2025) bao gồm bốn nội dung chính:

- ***Phần I: Vector & Matrix – Nền tảng đại số tuyến tính***
- ***Phần II: Background Subtraction – Trích xuất đối tượng từ ảnh***
- ***Phần III: Cosine Similarity – Đo độ tương đồng giữa các vector***
- ***Phần IV: Ứng dụng thực tiễn – Từ nhận diện giao thông đến KNN***

Phần I: Phần I: Vector Matrix – Nền tảng đại số tuyến tính

1 Phân tích từ dữ liệu thực tế

Trong thực tế, dữ liệu thường tồn tại dưới dạng bảng, trong đó mỗi hàng đại diện cho một đối tượng (sample), và mỗi cột là một đặc trưng (feature). Để đưa vào các mô hình học máy, dữ liệu này cần được biểu diễn dưới dạng **vector** và **ma trận**.

Ví dụ 1: Dữ liệu giá nhà

Diện tích (m^2)	Giá (nghìn \$)
4.6	5.9
6.7	9.1
⋮	⋮

Bảng 1: Dữ liệu đơn biến về giá nhà

Mô hình tuyến tính đơn giản:

$$\text{price} = a \cdot \text{area} + b$$

Biểu diễn bằng vector:

$$\vec{x} = \begin{bmatrix} 4.6 \\ 6.7 \\ \vdots \end{bmatrix}, \quad \vec{y} = \begin{bmatrix} 5.9 \\ 9.1 \\ \vdots \end{bmatrix}$$

Ví dụ 2: Dữ liệu quảng cáo

Mô hình hồi quy tuyến tính:

$$\text{Sales} = a_1 \cdot \text{TV} + a_2 \cdot \text{Radio} + a_3 \cdot \text{Newspaper} + b$$

TV	Radio	Newspaper	Sales
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12.0
⋮	⋮	⋮	⋮

Bảng 2: Dữ liệu đa biến về quảng cáo và doanh số

Biểu diễn ma trận - vector

Dữ liệu đặc trưng:

$$X = \begin{bmatrix} 230.1 & 37.8 & 69.2 \\ 44.5 & 39.3 & 45.1 \\ 17.2 & 45.9 & 69.3 \\ \vdots & \vdots & \vdots \end{bmatrix} \quad \text{(Ma trận } m \times 3\text{)}$$

Vector hệ số:

$$\vec{w} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

Vector kết quả (nhãn):

$$\vec{y} = \begin{bmatrix} 22.1 \\ 10.4 \\ 12.0 \\ \vdots \end{bmatrix}$$

Mô hình tổng quát:

$$X \cdot \vec{w} = \vec{y}$$

Tóm tắt

Thành phần thực tế	Biểu diễn toán học
Một dòng dữ liệu	Vector
Toàn bộ bảng dữ liệu	Ma trận X
Cột nhãn	Vector $\vec{y}$
Mô hình tuyến tính	$\vec{y} = X\vec{w} + b$

Bảng 3: Tương quan giữa dữ liệu và đại số tuyến tính

Tại sao cần dùng ma trận trong bài toán thực tế?

Trong chương trình phổ thông, ta hoàn toàn có thể giải các hệ phương trình tuyến tính nhỏ (ví dụ 2 hoặc 3 ẩn) bằng các phương pháp như thế, thế hoặc định thức Cramer. Tuy nhiên:

> Khi bài toán mở rộng với hàng chục, hàng trăm, thậm chí hàng nghìn biến (như trong dữ liệu hình ảnh, văn bản, hay tín hiệu âm thanh), các phương pháp giải tay trở nên không khả thi. Lúc này, **phép toán ma trận** trở thành công cụ bắt buộc để xử lý hiệu quả.

Cụ thể, hệ phương trình tuyến tính nhiều ẩn có thể được viết lại dưới dạng:

$$A \cdot \vec{x} = \vec{b}$$

Trong đó:

- A là ma trận hệ số (kích thước $m \times n$)
- $\vec{x}$ là vector ẩn cần tìm (kích thước $n \times 1$)
- $\vec{b}$ là vector kết quả (kích thước $m \times 1$)

Với biểu diễn này, ta có thể áp dụng các thuật toán tuyến tính hiệu quả và sử dụng thư viện tính toán như `NumPy`, `TensorFlow`, hoặc `PyTorch` để giải nhanh chóng – điều mà phương pháp thủ công không thể đáp ứng trong thực tế.

2 Vector và Ma Trận

2.1 Vector

Vector là một dãy số thực có thứ tự, biểu diễn cho các đặc trưng của một đối tượng.

- Ví dụ: Một căn nhà có thể được mô tả bởi vector đặc trưng:

$$\vec{x} = [\text{diện tích}, \text{số phòng}, \text{vị trí}] = [80, 3, 1]$$

- Một điểm ảnh (pixel) màu RGB có thể được biểu diễn bởi:

$$\vec{v} = [R, G, B] = [120, 200, 75]$$

2.2 Ma trận (matrix)

Ma trận là một tập hợp các vector được sắp xếp theo dạng bảng. Mỗi hàng (hoặc cột) là một vector.

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

Trong học máy:

- Tập đặc trưng đầu vào: ma trận $X \in \mathbb{R}^{m \times n}$
- Vector nhãn: $\vec{y} \in \mathbb{R}^{m \times 1}$

2.3 Các phép toán cơ bản

Cộng và trừ vector: Với hai vector cùng chiều $\vec{x}, \vec{y} \in \mathbb{R}^n$, phép cộng và trừ được thực hiện theo từng phần tử:

$$\vec{z} = \vec{x} \pm \vec{y} = [x_1 \pm y_1,\ x_2 \pm y_2,\ \ldots,\ x_n \pm y_n]$$

Ví dụ:

$$\vec{x} = [1, 2, 3], \quad \vec{y} = [4, 5, 6] \Rightarrow \vec{x} + \vec{y} = [5, 7, 9]$$

```python
x = [1, 2, 3]
y = [4, 5, 6]
z_add = [x[i] + y[i] for i in range(len(x))]
print(z_add)
# Output: [5, 7, 9]
```

```python
import numpy as np
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])
z_add = x + y
print(z_add)
# Output: [5 7 9]
```

Nhân với vô hướng: Với một vector $\vec{x} \in \mathbb{R}^n$ và một số thực $a \in \mathbb{R}$:

$$a \cdot \vec{x} = [a \cdot x_1,\ a \cdot x_2,\ \ldots,\ a \cdot x_n]$$

Ví dụ:

$$a = 2, \quad \vec{x} = [1, 3, 5] \Rightarrow a \cdot \vec{x} = [2, 6, 10]$$

```python
a = 2
x = [1, 3, 5]
scaled = [a * xi for xi in x]
print(scaled)
# Output: [2, 6, 10]
```

```python
x = np.array([1, 3, 5])
a = 2
scaled = a * x
print(scaled)
# Output: [ 2  6 10]
```

Tích Hadamard (theo từng phần tử): Với hai vector cùng chiều $\vec{x}, \vec{y} \in \mathbb{R}^n$:

$$\vec{x} \circ \vec{y} = [x_1 \cdot y_1,\ x_2 \cdot y_2,\ \ldots,\ x_n \cdot y_n]$$

Ví dụ:

$$\vec{x} = [1, 2, 3], \quad \vec{y} = [4, 5, 6] \Rightarrow \vec{x} \circ \vec{y} = [4, 10, 18]$$

```python
x = [1, 2, 3]
y = [4, 5, 6]
hadamard = [x[i] * y[i] for i in range(len(x))]
print(hadamard)
# Output: [4, 10, 18]
```

```
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])
hadamard = x * y
print("hadamard)
# Output: [ 4 10 18]
```

2.4 Tích vô hướng (dot product):

Với hai vector $\vec{x}, \vec{y} \in \mathbb{R}^n$:

$$\vec{x} \cdot \vec{y} = \sum_{i=1}^{n} x_i \cdot y_i$$

Ví dụ:

$$\vec{x} = [1, 2, 3], \quad \vec{y} = [4, 5, 6] \Rightarrow \vec{x} \cdot \vec{y} = 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 = 32$$

```
x = [1, 2, 3]
y = [4, 5, 6]
dot = sum([x[i] * y[i] for i in range(len(x))])
print("dot)
# Output: 32
```

```
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])
dot = np.dot(x, y)
print("dot)
# Output: 32
```

2.5 Norm – độ dài của vector:

Độ dài (norm) của vector $\vec{x} \in \mathbb{R}^n$ được định nghĩa:

$$\|\vec{x}\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

Ví dụ:

$$\vec{x} = [3, 4] \Rightarrow \|\vec{x}\| = \sqrt{3^2 + 4^2} = 5$$

```
import math
x = [3, 4]
norm = math.sqrt(sum([xi**2 for xi in x]))
print(norm)
# Output: 5.0
```

```
x = np.array([3, 4])
norm = np.linalg.norm(x)
print(norm)
# Output: 5.0
```

2.6 Các phép toán cơ bản với ma trận

Tổng quát: Ma trận cũng có thể thực hiện các phép toán tương tự như vector, bao gồm:

- **Cộng/trừ hai ma trận:** cùng kích thước, cộng hoặc trừ từng phần tử.
- **Nhân Hadamard:** phép nhân từng phần tử giữa hai ma trận cùng kích thước.
- **Chia Hadamard:** chia từng phần tử tương ứng giữa hai ma trận.

Cộng/trừ ma trận: **Tổng quát:** Với $A, B \in \mathbb{R}^{m \times n}$:

$$C = A \pm B \quad \text{với} \quad c_{ij} = a_{ij} \pm b_{ij}$$

Ví dụ:

$$A = \begin{bmatrix} 4 & 9 \\ 2 & 8 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \Rightarrow A + B = \begin{bmatrix} 5 & 12 \\ 4 & 12 \end{bmatrix}$$

$$A - B = \begin{bmatrix} 3 & 6 \\ 0 & 4 \end{bmatrix}$$

```python
A = [[4, 9], [2, 8]]
B = [[1, 3], [2, 4]]
C_add = [[A[i][j] + B[i][j] for j in range(2)] for i in range(2)]
C_sub = [[A[i][j] - B[i][j] for j in range(2)] for i in range(2)]
print(C_add)
# Output: [[5, 12], [4, 12]]
print(C_sub)
# Output: [[3, 6], [0, 4]]
```

```python
A = np.array([[4, 9], [2, 8]])
B = np.array([[1, 3], [2, 4]])
C_add = A + B
C_sub = A - B
print(C_add)
# Output:
# [[ 5 12]
#  [ 4 12]]
print(C_sub)
# Output:
# [[3 6]
#  [0 4]]
```

Nhân Hadamard (element-wise product): **Tổng quát:**

$$C = A \circ B \quad \text{với} \quad c_{ij} = a_{ij} \cdot b_{ij}$$

Ví dụ:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \Rightarrow A \circ B = \begin{bmatrix} 5 & 12 \\ 21 & 32 \end{bmatrix}$$

Chia Hadamard (element-wise division): Tổng quát:

$$C = A \oslash B \quad \text{với} \quad c_{ij} = \frac{a_{ij}}{b_{ij}}$$

Ví dụ:

$$A = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \Rightarrow A \oslash B = \begin{bmatrix} 5 & 2 \\ 3.5 & 2 \end{bmatrix}$$

```
C_mul = [[A[i][j] * B[i][j] for j in range(2)] for i in range(2)]
C_div = [[A[i][j] / B[i][j] for j in range(2)] for i in range(2)]
print(C_mul)
# Output: [[4, 27], [4, 32]]
print(C_div)
# Output: [[4.0, 3.0], [1.0, 2.0]]
```

```
C_mul = A * B
C_div = A / B
print(C_mul)
# Output:
# [[ 4 27]
#  [ 4 32]]
print(C_div)
# Output:
# [[4. 3.]
#  [1. 2.]]
```

2.7 Nhân ma trận với một số vô hướng (Scalar Multiplication)

Tổng quát: Cho một ma trận $A \in \mathbb{R}^{m \times n}$ và một số thực $c \in \mathbb{R}$, phép nhân ma trận với vô hướng được định nghĩa như sau:

$$c \cdot A = \begin{bmatrix} c \cdot a_{11} & \dots & c \cdot a_{1n} \\ \vdots & \ddots & \vdots \\ c \cdot a_{m1} & \dots & c \cdot a_{mn} \end{bmatrix}$$

Tức là, mỗi phần tử của ma trận đều được nhân với c.

```
c = 2
scaled = [[c * A[i][j] for j in range(2)] for i in range(2)]
print(scaled)
# Output: [[8, 18], [4, 16]]
```

```
scaled = c * A
print(scaled)
# Output:
# [[ 8 18]
#  [ 4 16]]
```

2.8 Phép nhân ma trận – vector

Tổng quát: Cho ma trận $A \in \mathbb{R}^{m \times n}$ và vector cột $\vec{v} \in \mathbb{R}^n$, phép nhân được định nghĩa như sau:

$$A \cdot \vec{v} = \begin{bmatrix} \text{dòng 1 của } A \cdot \vec{v} \\ \text{dòng 2 của } A \cdot \vec{v} \\ \vdots \end{bmatrix} = \vec{b} \in \mathbb{R}^m$$

Ví dụ:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \quad \vec{v} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \Rightarrow A \cdot \vec{v} = \begin{bmatrix} 1 \cdot 1 + 2 \cdot 2 \\ 3 \cdot 1 + 4 \cdot 2 \end{bmatrix} = \begin{bmatrix} 5 \\ 11 \end{bmatrix}$$

2.9 Phép nhân hai ma trận

Tổng quát: Cho hai ma trận $A \in \mathbb{R}^{m \times n}$ và $B \in \mathbb{R}^{n \times p}$, tích $C = A \cdot B$ là một ma trận $C \in \mathbb{R}^{m \times p}$, với mỗi phần tử c_{ij} được tính bằng:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

Ví dụ:

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 2 & 2 \\ 3 & 1 \end{bmatrix} \Rightarrow A \cdot B = \begin{bmatrix} 1 \cdot 2 + 3 \cdot 3 & 1 \cdot 2 + 3 \cdot 1 \\ 2 \cdot 2 + 4 \cdot 3 & 2 \cdot 2 + 4 \cdot 1 \end{bmatrix} = \begin{bmatrix} 11 & 5 \\ 16 & 8 \end{bmatrix}$$

Nhận xét: vector cũng là một ma trận với chỉ 1 hàng (hoặc 1 cột) nên phép nhân ma trận - vector cũng tương tự phép nhân hai ma trận.

```python
A = [[1, 3], [2, 4]]
B = [[2, 2], [3, 1]]
result = [[sum([A[i][k] * B[k][j] for k in range(2)]) for j in range(2)] for i in range(2)]
print(result)
# Output: [[11, 5], [16, 8]]
```

```python
A = np.array([[1, 3], [2, 4]])
B = np.array([[2, 2], [3, 1]])
result = A @ B
print(result)
# Output:
# [[11  5]
#  [16  8]]
```

2.10 Ma trận chuyển vị

Tổng quát: Cho ma trận $A \in \mathbb{R}^{m \times n}$, ma trận chuyển vị $A^\top$ là ma trận kích thước $n \times m$ với:

$$A^\top_{ij} = A_{ji}$$

Tức là hoán đổi hàng thành cột và ngược lại.

Ví dụ:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \Rightarrow A^\top = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

```python
A = [[1, 2], [3, 4]]
transpose = [[A[j][i] for j in range(2)] for i in range(2)]
print(transpose)
# Output: [[1, 3], [2, 4]]
```

```python
A = np.array([[1, 2], [3, 4]])
transpose = A.T
print(transpose)
# Output:
# [[1 3]
#  [2 4]]
```

Ví dụ: Chuyển ảnh màu sang ảnh xám bằng dot product

Ảnh màu (RGB) có thể được chuyển sang ảnh xám bằng cách sử dụng tích vô hướng giữa vector màu và vector trọng số độ sáng của từng kênh:

$$\text{Gray} = \begin{bmatrix} R & G & B \end{bmatrix} \cdot \begin{bmatrix} 0.213 \\ 0.715 \\ 0.072 \end{bmatrix}$$

Đây là công thức chuẩn theo khuyến nghị ITU-R BT.601 trong xử lý ảnh.

Ví dụ: Cho một pixel có giá trị RGB là $[120, 200, 75]$, ta tính được giá trị ảnh xám như sau:

$$\text{Gray} = 120 \cdot 0.213 + 200 \cdot 0.715 + 75 \cdot 0.072 = 172.535$$

```python

import cv2
import numpy as np
import matplotlib.pyplot as plt

image_path = 'sample.jpg'

bgr_img = cv2.imread(image_path)

rgb_img = cv2.cvtColor(bgr_img, cv2.COLOR_BGR2RGB)

#dot product
weights = np.array([0.213, 0.715, 0.072])
gray_dot = np.dot(rgb_img[...,:3], weights).astype(np.uint8)

#OpenCV
gray_cv2 = cv2.cvtColor(bgr_img, cv2.COLOR_BGR2GRAY)

# show
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.imshow(rgb_img)
plt.title('Ảnh màu ốgc (RGB)')
plt.axis('off')
```

```
plt.subplot(1, 3, 2)
plt.imshow(gray_dot, cmap='gray')
plt.title('Grayscale (Dot Product)')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(gray_cv2, cmap='gray')
plt.title('Grayscale (OpenCV)')
plt.axis('off')

plt.tight_layout()
plt.show()
```

Ví dụ: Lật và xoay ảnh bằng biến đổi ma trận

Các phép biến đổi hình học như lật và xoay ảnh đều có thể được biểu diễn bằng **ma trận biến đổi tuyến tính**:

- **Xoay ảnh** quanh gốc một góc θ sử dụng ma trận:

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

- **Lật ảnh** theo trục dọc/ngang:
  - Lật ngang: $\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$
  - Lật dọc: $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

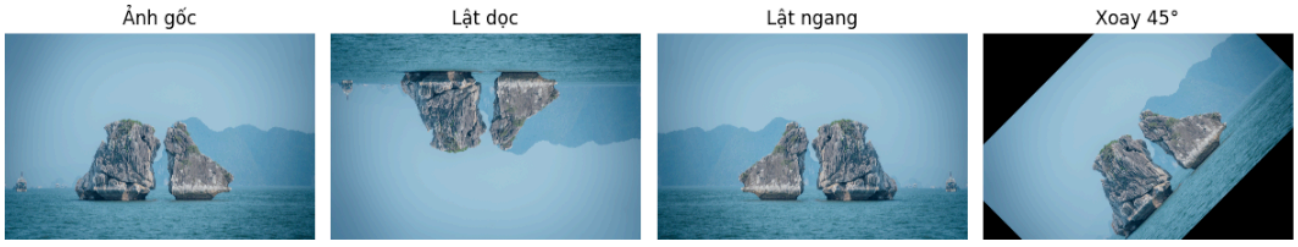
- **Việc *xoay ảnh*** thực chất là nhân từng điểm tọa độ (x, y) với ma trận xoay:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R(\theta) \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

- **Lật ảnh** là phép *co giãn có định hướng âm*, tức là nhân với ma trận có giá trị âm để đảo chiều trục.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

```

```
# Đọc ảnh và ểchuyn sang RGB
img = cv2.imread('sample.jpg')
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# ậLt ảnh
flip_x = cv2.flip(img_rgb, 0)  # ậLt ọdc
flip_y = cv2.flip(img_rgb, 1)  # ậLt ngang

# Xoay ảnh quanh tâm ảnh
(h, w) = img_rgb.shape[:2]
center = (w // 2, h // 2)
angle = 45
M = cv2.getRotationMatrix2D(center, angle, 1.0)
rotated = cv2.warpAffine(img_rgb, M, (w, h))

# ểHin ịth ảnh
plt.figure(figsize=(12, 6))
plt.subplot(1, 4, 1)
plt.imshow(img_rgb)
plt.title("Ảnh ốgc")
plt.axis('off')

plt.subplot(1, 4, 2)
plt.imshow(flip_x)
plt.title("ậLt ọdc")
plt.axis('off')

plt.subplot(1, 4, 3)
plt.imshow(flip_y)
plt.title("ậLt ngang")
plt.axis('off')

plt.subplot(1, 4, 4)
plt.imshow(rotated)
plt.title("Xoay 45°")
plt.axis('off')

plt.tight_layout()
plt.show()
```



Nhận xét: Đây là ứng dụng trực tiếp của kiến thức ma trận để biến đổi toạ độ không gian của từng điểm ảnh. Trong các mạng nơ-ron tích chập (CNN), việc hiểu các phép biến đổi hình học này là rất quan trọng cho *data augmentation* và *spatial reasoning*.

2.11. Hàm áp dụng từng phần tử (Hadamard operation)

Phép toán Hadamard là phép toán áp dụng **từng phần tử riêng lẻ** trên vector hoặc mảng. Ví dụ:

- $\sqrt{x_i}$ – căn bậc hai từng phần tử
- e^{x_i} – hàm mũ từng phần tử
- x_i^2, x_i^3 – luỹ thừa từng phần tử

```
import numpy as np

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

print(np.sqrt(data))
# Output: [1.    1.41  1.73  2.  ]

print(np.exp(data))
# Output: [2.71  7.39 20.08 54.60]

print(data ** 2)
# Output: [ 1  4  9 16]
```

Kết nối kiến thức: Các phép toán như $\sqrt{x}$, e^x, x^2,... là ví dụ điển hình của phép toán Hadamard – nghĩa là áp dụng độc lập lên từng phần tử của vector hoặc mảng. Trong học máy, đây là cơ sở cho việc áp dụng hàm kích hoạt (activation functions) như ReLU, sigmoid, tanh, v.v. trên mỗi neuron một cách song song.

2.12 `argmin`, `argmax` vs `min`, `max`

- `min()`, `max()` trả về **giá trị cực trị**
- `argmin()`, `argmax()` trả về **vị trí phần tử cực trị**

```
a = np.array([4, 7, 1, 5])

print(np.min(a))
# Output: 1

print(np.argmin(a))
# Output: 2

print(np.max(a))
# Output: 7

print(np.argmax(a))
# Output: 1
```

Kết nối kiến thức: Hàm `argmin` và `argmax` thường được sử dụng để chọn đầu ra có xác suất lớn nhất trong mô hình phân loại (ví dụ: chọn lớp dự đoán trong softmax). Chúng cũng rất quan trọng trong các bài toán tối ưu hoá như tìm vị trí của giá trị mất mát nhỏ nhất.

2.13 Tổng và trung bình

- Tổng:

$$\text{sum} = \sum_{i=1}^{n} x_i$$

- Trung bình:

$$\text{mean} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

```
data = np.array([6, 5, 7, 1, 9, 2])

print(np.sum(data))
# Output: 30

print(np.mean(data))
# Output: 5.0
```

Kết nối kiến thức: Tổng và trung bình là các phép toán cơ bản được sử dụng liên tục trong phân tích dữ liệu, thống kê mô tả, chuẩn hoá dữ liệu (mean normalization), và tính toán các loại loss function như MSE (mean squared error). Hiểu bản chất của chúng giúp hiểu sâu hơn về cách máy học đánh giá và điều chỉnh mô hình.

Phần II: Background Subtraction – Trích xuất đối tượng từ ảnh

Một ảnh màu là gì "về mặt ma trận"?

Ảnh số (digital image)

Một ảnh màu (RGB) có thể biểu diễn bằng **3 ma trận** kích thước $H \times W$:

$$\textbf{Ảnh màu} = \underbrace{\mathbf{R}}_{\text{Ma trận đỏ}} + \underbrace{\mathbf{G}}_{\text{Ma trận lục}} + \underbrace{\mathbf{B}}_{\text{Ma trận lam}}$$

Nói cách khác, ảnh I là một tensor 3 chiều:

$$I \in \mathbb{R}^{H \times W \times 3}$$

Mỗi pixel là một vector [R, G, B], ví dụ:

$$\text{Pixel}_{i,j} = [124, 200, 50]$$

Quy trình tách đối tượng (Background Subtraction)

1. Tính sai khác giữa ảnh gốc và ảnh nền

Cho hai ảnh màu B (background) và I (frame), cùng kích thước $H \times W \times 3$:

$$D(x, y, c) = |I(x, y, c) - B(x, y, c)|, \quad c \in \{R, G, B\}$$

Tức là phép trừ được thực hiện độc lập trên từng kênh màu.
Trong code, ta viết:

```
diff = cv2.absdiff(frame, background)
```

2. Ngưỡng hoá mask foreground

Sau khi tính sai khác trên từng kênh màu, ta áp dụng ngưỡng trực tiếp lên toàn bộ tensor D bằng cách threshold từng phần tử:

$$M(x, y, c) = \begin{cases} 0 & \text{nếu } D(x, y, c) \leq T \\ 255 & \text{nếu } D(x, y, c) > T \end{cases}$$

Trong code:

```
_, diff_binary = cv2.threshold(diff, 5, 255, cv2.THRESH_BINARY)
```

3. Ghép đối tượng vào nền mới

Sử dụng điều kiện lựa chọn từng phần tử trong toàn bộ tensor ảnh RGB để xác định pixel nào giữ lại từ ảnh gốc, pixel nào thay bằng nền mới:

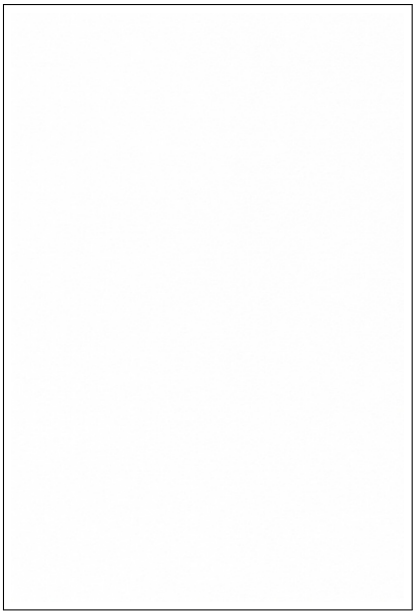
```
output = np.where(diff_binary == 0, new_bg, img)
```

Lệnh này sẽ lấy điểm ảnh từ new_bg nếu tương ứng là pixel nền (giá trị 0), hoặc từ img nếu có thay đổi (giá trị khác 0).

Thực thi

Ta cần tách đối tượng trong I ra khỏi nền B và ghép vào nền F mới.

(a) FakeBackground F

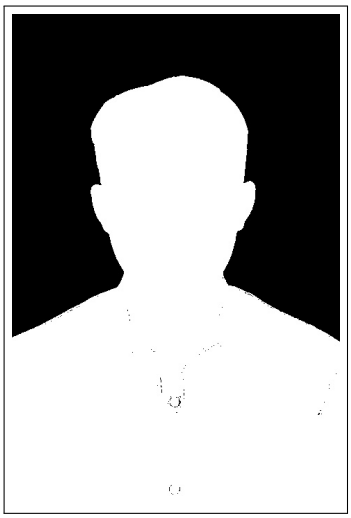


(b) Background B

(c) Coming image I

```python
import cv2
import numpy as np

bg = cv2.imread('background.jpg', 1)
img = cv2.imread('IMG_0053.JPG', 1)

diff = cv2.absdiff(bg, img)
diff_gray = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)
_, diff_binary = cv2.threshold(diff_gray, 15, 255, cv2.THRESH_BINARY)
```

Thu được:



Hình 2: Mask

Sau đó thực hiện

```
new_bg = cv2.imread('background_color.jpg', 1)

output = np.where(diff_binary[..., None] == 0, new_bg, img)
```

thu được output:

Phần III: Cosine Similarity – Đo độ tương đồng giữa các vector

1. Từ Tích Vô Hướng đến Hình Chiếu

Trong đại số tuyến tính, **tích vô hướng** (dot product) không chỉ là một phép tính số học — nó là một phép chiếu hình học:

$$\vec{a} \cdot \vec{b} = \|\vec{a}\| \cdot \|\vec{b}\| \cdot \cos(\alpha)$$

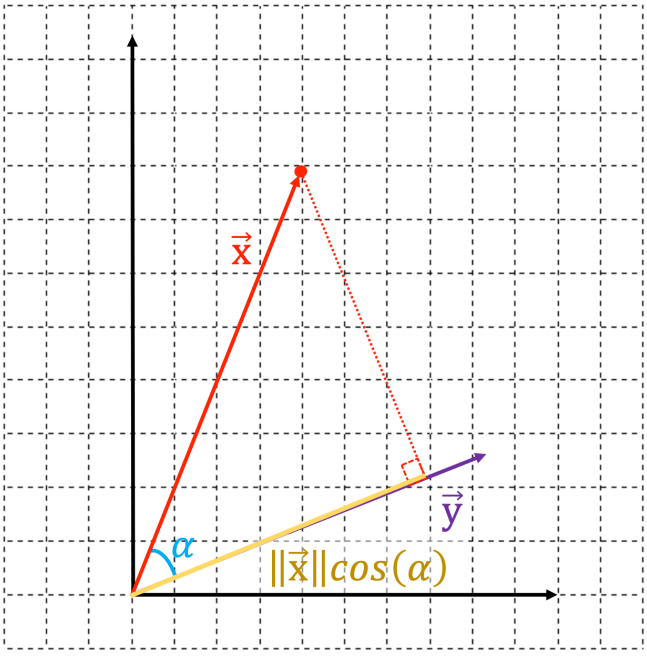
Trong đó:

- $\vec{a}, \vec{b}$ là hai vector trong không gian $\mathbb{R}^n$
- α là góc giữa chúng
- $\cos(\alpha)$ là tỷ lệ cosin của góc

Như vậy, tích vô hướng là phép *hạ hình chiếu của vector này lên vector kia*, sau đó nhân với độ dài của vector bị chiếu lên.

2. Hiểu Bằng Trực Quan Hình Học

Ta có 2 vector như sau:



→ Khi ta **chiếu vuông góc** vector $\vec{x}$ xuống $\vec{y}$, đoạn dài nhất nằm trên $\vec{y}$ chính là:

$$\|\vec{x}\| \cdot \cos(\theta)$$

Tức là phần **đóng góp** của $\vec{x}$ lên hướng của $\vec{y}$.

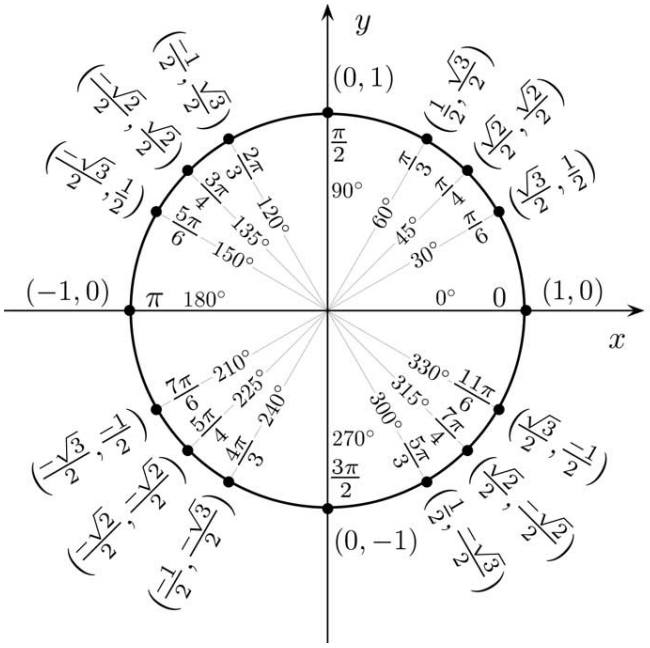
Cosine Là Hình Chiếu Trên Đường Tròn Lượng Giác

Đường tròn lượng giác là gì?

Nhắc lại kiến thức phổ thông:

- Đường tròn lượng giác là đường tròn đơn vị có bán kính $R = 1$, tâm là gốc toạ độ (0,0).
- Một điểm P chuyển động trên đường tròn sẽ tạo với trục Ox một **góc** α.
- Với mỗi góc α, toạ độ điểm P là:

$$P(cos\alpha, sin\alpha)$$



Hình 3: Đường tròn lượng giác

Cosine là hình chiếu trên trục Ox

Khi ta nối từ gốc tọa độ O đến điểm P, bạn tạo ra một vector $\vec{OP}$ có chiều dài 1 (do P nằm trên đường tròn đơn vị).

- Nếu bạn **chiếu điểm** P **vuông góc xuống trục Ox**, điểm chiếu đó chính là:

$$(\cos\alpha, 0)$$

Tức là: **Giá trị** $\cos(\theta)$ **chính là "độ dài hình chiếu của vector đơn vị lên trục Ox."**

Nói cách khác:
Cosine đo mức độ *"nằm theo hướng Ox"* của một vector đơn vị tạo góc θ với trục Ox.

Ánh xạ sang Cosine Similarity

Giả sử ta có hai vector chuẩn hóa $\hat{x}, \hat{y}$, với góc giữa chúng là θ.

Giống như điểm P ở trên, ta có thể nghĩ vector $\hat{x}$ đang được chiếu lên phương của $\hat{y}$.

Cosine Similarity lúc này chính là:

$$\cos(\theta) = \text{độ dài hình chiếu của } \hat{x} \text{ lên } \hat{y}$$

→ Ta quay lại hình ảnh quen thuộc: vector từ gốc O tới điểm P trên đường tròn đơn vị.

→ Dù vector xoay theo bất kỳ hướng nào, ta luôn có thể **chiếu nó về trục đơn vị cần so sánh.**

Ý nghĩa của Cosine

Hai công thức tính tích vô hướng

Theo đại số:

$$\vec{x} \cdot \vec{y} = \sum_{i=1}^{n} x_i y_i$$

- Đây là phép nhân từng phần tử rồi cộng lại.
- Dễ tính toán, thường dùng trong lập trình.

Theo hình học:

$$\vec{x} \cdot \vec{y} = \|\vec{x}\| \cdot \|\vec{y}\| \cdot \cos(\theta)$$

Phụ thuộc vào:

- Độ dài của từng vector
- Góc giữa hai vector

Suy ra công thức Cosine Similarity

Bằng cách kết hợp hai biểu thức trên:

$$\sum_{i=1}^{n} x_i y_i = \|\vec{x}\| \cdot \|\vec{y}\| \cdot \cos(\theta)$$

Ta suy ra:

$$\cos(\theta) = \frac{\sum_{i=1}^{n} x_i y_i}{\|\vec{x}\| \cdot \|\vec{y}\|}$$

Và đây chính là:

$$\text{Cosine Similarity}(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \cdot \|\vec{y}\|}$$

Vậy Cosine Similarity là gì?

Là một đại lượng đo lường **"độ hướng cùng chiều"** giữa hai vector.

Ý nghĩa:

- Nếu hai vector cùng hướng: $\theta = 0^\circ \Rightarrow \cos(\theta) = 1$
- Nếu vuông góc: $\theta = 90^\circ \Rightarrow \cos(\theta) = 0$
- Nếu ngược hướng: $\theta = 180^\circ \Rightarrow \cos(\theta) = -1$

$\Rightarrow$ **Cosine Similarity không đo độ lớn, mà đo hướng tương đối giữa hai vector.**
Giả sử:

$$\vec{x}, \vec{y} \in \mathbb{R}^n, \quad a, b > 0$$

Cosine Similarity:

$$\begin{aligned}
\cos(a\vec{x}, b\vec{y}) &= \frac{(a\vec{x}) \cdot (b\vec{y})}{\|a\vec{x}\| \cdot \|b\vec{y}\|} \\
&= \frac{ab(\vec{x} \cdot \vec{y})}{a\|\vec{x}\| \cdot b\|\vec{y}\|} \\
&= \frac{ab \sum x_i y_i}{ab\sqrt{\sum x_i^2} \cdot \sqrt{\sum y_i^2}} \\
\textbf{Rút gọn:} \quad &= \frac{\sum x_i y_i}{\sqrt{\sum x_i^2} \cdot \sqrt{\sum y_i^2}} = \cos(\vec{x}, \vec{y})
\end{aligned}$$

Cosine không phụ thuộc vào độ dài $\|\vec{x}\|, \|\vec{y}\|$, chỉ phụ thuộc vào góc giữa hai vector.

IV. Tại sao Cosine Similarity đo được "độ tương đồng"?

Vector càng "gần hướng" $\Rightarrow$ càng giống nhau

(góc nhỏ -> cos lớn)

- Nếu $\vec{x}$ và $\vec{y}$ chỉ khác độ dài nhưng cùng hướng $\Rightarrow$ Cosine vẫn bằng 1
- Ngược lại, nếu khác hướng $\Rightarrow$ Cosine giảm

Không bị ảnh hưởng bởi "mức độ"

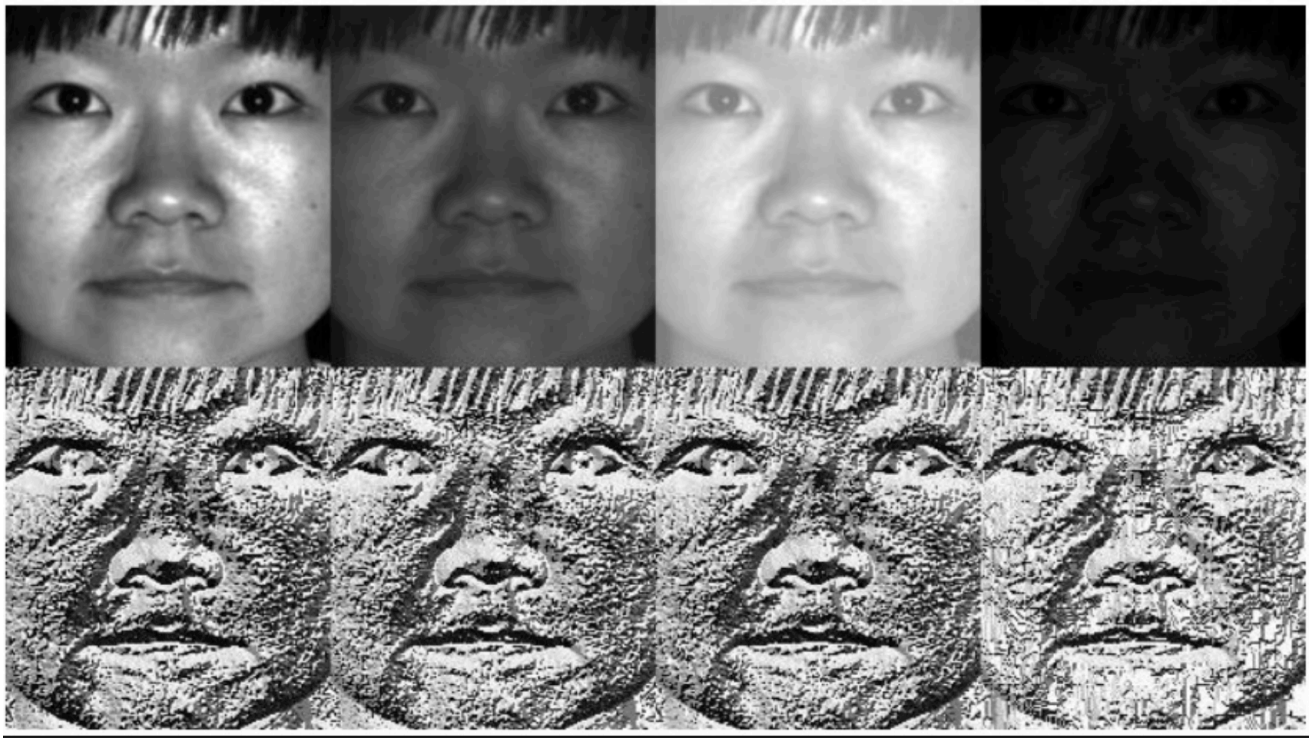
- Rất hữu ích khi dữ liệu có độ lớn khác nhau, nhưng bạn chỉ quan tâm đến "nội dung" hoặc "xu hướng đặc trưng".

Ánh xạ trực tiếp sang độ tương đồng

Vì Cosine là thước đo tuyến tính và nằm trong đoạn $[-1, 1]$, nên ta có thể:

- Dùng trực tiếp như chỉ số **tương đồng**
- Dùng $1 -$ Cosine Similarity làm khoảng cách Cosine cho thuật toán phân cụm hoặc KNN

Xét bộ 4 ảnh:



đều cùng là 1 người, nhưng với độ sáng khác nhau thì các vector tương ứng cũng khác nhau.

Nhưng với **Cosine Similarity** thì độ sáng của ảnh (độ lớn vector tướng ứng) không ảnh hưởng đến độ sai khác của ảnh.

Tính Cosine Similarity theo đại số (dùng numpy)

```python
import numpy as np

def cosine_similarity_numpy(vector1, vector2):
    dot_product = np.dot(vector1, vector2)
    norm1 = np.linalg.norm(vector1)
    norm2 = np.linalg.norm(vector2)
    return dot_product / (norm1 * norm2)
```

Tính Cosine Similarity dùng math

```python
import math

def cosine_similarity_manual(vector1, vector2):
    sumxy = sum(v1 * v2 for v1, v2 in zip(vector1, vector2))
    sumxx = sum(v1 * v1 for v1 in vector1)
    sumyy = sum(v2 * v2 for v2 in vector2)
    return sumxy / math.sqrt(sumxx * sumyy)
```

Test

```
vector1 = np.array([5, 3, 2, 7])
vector2 = np.array([2, 9, 4, 1])
print("Cosine similarity (NumPy):", cosine_similarity_numpy(vector1, vector2))

vector1 = [5, 3, 2, 7]
vector2 = [2, 9, 4, 1]
print("Cosine similarity (manual):", cosine_similarity_manual(vector1, vector2))
```

```
Cosine similarity (NumPy): 0.552005787925351
Cosine similarity (manual): 0.552005787925351
```

Phần IV: Ứng dụng thực tiễn

Distance Between Two Vectors — *Khoảng cách giữa hai vector*

1. Mục tiêu

Cho hai vector $\vec{x}, \vec{y} \in \mathbb{R}^n$, khoảng cách giữa chúng là:

$$\text{dist}(\vec{x}, \vec{y}) = \|\vec{x} - \vec{y}\|$$

⇒ Đây là độ dài của vector hiệu $\vec{x} - \vec{y}$, tức là khoảng cách "thực sự" từ điểm này đến điểm kia trong không gian.

2. Suy dẫn đến công thức Euclid

Bắt đầu với:

$$\text{dist}(\vec{x}, \vec{y}) = \|\vec{x} - \vec{y}\|$$

Vector hiệu:

$$\vec{x} - \vec{y} = [x_1 - y_1, x_2 - y_2, \ldots, x_n - y_n]$$

Độ dài của vector này là chuẩn L2 (norm bậc 2):

$$\|\vec{x} - \vec{y}\| = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}$$

Đây chính là **công thức khoảng cách Euclidean** trong $\mathbb{R}^n$.

3. Ví dụ 2D đơn giản

Cho:

$$\vec{x} = (1, 2), \quad \vec{y} = (4, 6)$$

Hiệu:

$$\vec{x} - \vec{y} = (-3, -4)$$

Khoảng cách:

$$\text{dist}(\vec{x}, \vec{y}) = \sqrt{(-3)^2 + (-4)^2} = \sqrt{9 + 16} = \sqrt{25} = 5$$

⇒ Đây là chiều dài cạnh của tam giác vuông ⇒ đúng trực giác hình học.

1 So sánh độ sai khác giữa các ảnh

Ta có 3 ảnh biển báo giao thông sau:

Sign 1

Sign 2

Sign 3

Sign 4

Chuyển mỗi ảnh thành vector

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

img1 = cv2.imread('sign1.png')
img1 = cv2.resize(img1, (300, 300))

img2 = cv2.imread('sign2.png')
img2 = cv2.resize(img2, (300, 300))

img3 = cv2.imread('sign3.png')
img3 = cv2.resize(img3, (300, 300))

img4 = cv2.imread('sign4.png')
img4 = cv2.resize(img4, (300, 300))
```

Dùng Distance (Đo giữa Sign2 và các ảnh còn lại)

```
abs21 = np.mean(np.abs(img1 - img2))
print(abs21)
abs23 = np.mean(np.abs(img1 - img3))
print(abs23)
abs24 = np.mean(np.abs(img1 - img4))
print(abs24)
```

Kết quả:

```
92.63648518518518
171.33612962962962
94.00288148148148
```

Dùng Cosine Similarity (Đo giữa Sign2 và các ảnh còn lại)

```
from numpy.linalg import norm

def cos_sim(a, b):
  a = a.flatten()
  b = b.flatten()

  a = a.astype(np.float64)
  b = b.astype(np.float64)
  return np.dot(a, b) / (norm(a) * norm(b))
```

Kết quả:

```
0.3123484650319436
0.9494836679384685
0.938305923592088
```

✉ Kết quả so sánh

So với sign2	Cosine Similarity (↑ càng giống)	L1 Distance (↓ càng giống)
sign3	0.312 (thấp nhất)	171.33 (lớn nhất)
sign4	0.949 (cao nhất)	94.00
sign1	0.938	92.63 (nhỏ nhất)

Nhận xét

- **sign3** là ảnh khác biệt lớn nhất với sign2:
  - Cosine thấp nhất (0.312) ⇒ khác hướng vector đặc trưng
  - Distance lớn nhất (171.33) ⇒ khác biệt mạnh về pixel
    ⇒ sign3 là ảnh khác loại rõ rệt
- **sign4** và **sign1** đều tương tự với sign2, nhưng:
  - Cosine: sign4 ∼ sign2 hơn (0.949 vs 0.938)
  - Distance: sign1 gần hơn (92.63 vs 94.00)
    ⇒ sign4 giống về hướng tổng thể
    ⇒ sign1 giống hơn về chi tiết pixel

Kết luận

Cả hai phương pháp đều đồng thuận rằng `sign3` là ảnh khác biệt nhất.
sign1 và sign4 đều gần sign2, nhưng theo tiêu chí khác nhau:

- Cosine đánh giá **sign4 giống hơn**
- Distance đánh giá **sign1 gần hơn**

⇒ Tuỳ vào mục tiêu (so sánh nội dung hay độ lệch pixel), bạn chọn Cosine hoặc L1.

2 Thuật toán K-Nearest Neighbors (KNN)

1. Định nghĩa

KNN (K-Nearest Neighbors) là một thuật toán học máy không tham số, được sử dụng để **phân loại** hoặc **hồi quy**. Ý tưởng chính là: *với một điểm dữ liệu mới, thuật toán sẽ tìm* ***k láng giềng gần nhất*** *trong tập huấn luyện và đưa ra dự đoán dựa trên các láng giềng đó.*

2. Các bước của KNN

1. **Chuẩn bị dữ liệu**: có tập huấn luyện và chọn số k.
2. **Tính khoảng cách**:

$$\text{dist}(\vec{x}_{\text{test}}, \vec{x}_i) = \|\vec{x}_{\text{test}} - \vec{x}_i\| = \sqrt{\sum_j (x_{\text{test}}^{(j)} - x_i^{(j)})^2}$$

3. **Chọn k điểm gần nhất**: sắp xếp theo khoảng cách tăng dần, chọn k điểm gần nhất.
4. **Dự đoán đầu ra**:
   - Với phân loại: chọn nhãn chiếm đa số.
   - Với hồi quy: lấy trung bình giá trị của các láng giềng.

3. Ví dụ

Giả sử dữ liệu test là $x_{\text{test}} = 2.4$.

- Nếu $k = 1$ thì láng giềng gần nhất có nhãn $1 \Rightarrow$ dự đoán là 1.
- Nếu $k = 3$ thì 3 láng giềng có nhãn đa số là $0 \Rightarrow$ dự đoán là 0.

4. Liên hệ với khoảng cách Euclid

Thuật toán KNN hoạt động dựa trên khái niệm "**gần**" trong không gian vector, vì vậy khoảng cách đóng vai trò rất quan trọng.

- **Khoảng cách Euclid** (chuẩn L2):

$$\|\vec{x} - \vec{y}\| = \sqrt{\sum (x_i - y_i)^2}$$

- Ngoài ra có thể dùng:
  - Khoảng cách Manhattan (L1)
  - Cosine Similarity (cho dữ liệu chuẩn hóa, sparse)

5. Ưu và nhược điểm

Ưu điểm	**Nhược điểm**
Đơn giản, dễ cài đặt	Chậm với dữ liệu lớn
Không cần huấn luyện mô hình	Nhạy với nhiễu, outliers
Linh hoạt với nhiều loại khoảng cách	Cần chọn k phù hợp

6. Kết luận

KNN là một thuật toán **dựa trên khoảng cách** – không học hàm số, mà sử dụng trực tiếp dữ liệu huấn luyện để dự đoán. Khoảng cách Euclid là lựa chọn phổ biến giúp đo độ gần, trong khi Cosine Similarity có thể thay thế nếu muốn đo hướng vector thay vì độ dài.