Singular Value Decomposition and Principal Component Analysis

Tổng quan

01 Introduction

02 Singular Value Decomposition (SVD)

03 Principal Component Analysis (PCA)

04 SVD & PCA

1. Introduction

21521943	-	Nguyễn Tiến Đạt
21520027	-	Đặng Quang Hải
23521355	-	Nguyễn Nhật Sơn
23521699	-	Nguyễn Đình Tú
24520047	-	Huỳnh Ngọc An

02 Singular Value Decomposition (SVD)

Giới thiệu

Singular Value Decomposition (SVD) là một trong những phương pháp thuộc nhóm matrix factorization. Mục đích ban đầu của phương pháp này là tìm ra một phép xoay không gian sao cho tích vô hướng của các vector không thay đổi.
Phương pháp SVD được phát triển dựa trên những tính chất của **ma trận trực giao và ma trận đường chéo** để tìm ra một ma trận xấp xỉ với ma trận gốc. Phương pháp này được ứng dụng rộng rãi trong các lĩnh vực xử lý hình ảnh, clustering, các thuật toán nén và giảm chiều dữ liệu và các bài toán recommendation.

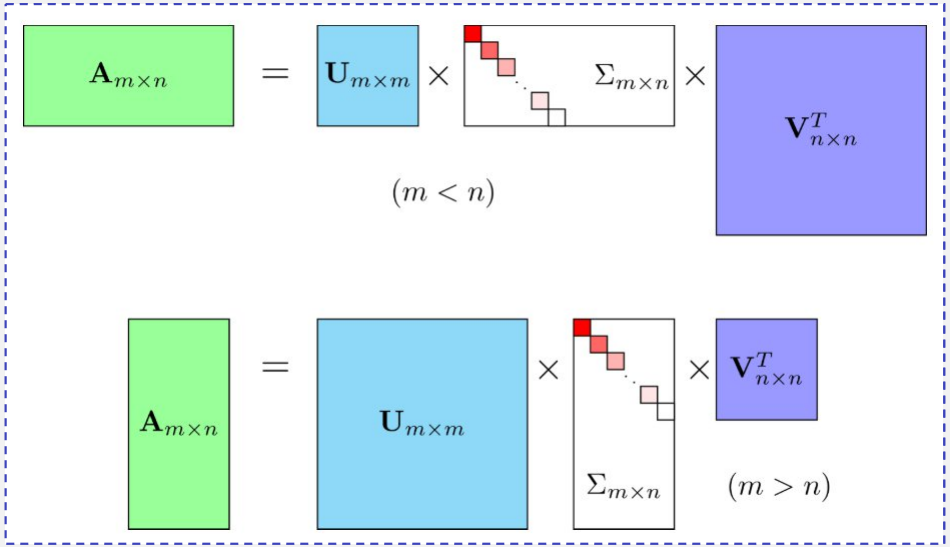
Khái niệm - SVD

Một ma trận $\mathbf{A_{m \times n}}$ bất kỳ đều có thể phân tích thành:

$$\mathbf{A_{m \times n} = U_{m \times m} S_{m \times n} (V_{n \times n})^T} \ (1)$$

- **U, V** là các ma trận trực giao
- **S** là ma trận đường chéo không vuông với các phần tử $\sigma_1 \geq \sigma_2 \geq \ \ldots \ \geq \sigma_r \geq 0$
  với r là rank của ma trận A (số các phần tử khác 0 trong S = rank của ma trận A)

Khái niệm - SVD



SVD của ma trận A trong 2 TH m > n và m < n

Nguồn gốc của tên gọi SVD

Bỏ qua chiều của mỗi ma trận, từ (1) ta có:

$$AA^T = USV^T(USV^T)^T$$
$$= USV^T VS^T U^T$$
$$= USS^T U^T \quad (V^TV = I \text{ do } V \text{ là ma trận trực giao})$$

ó $AA^T U = USS^T$ ($U^T U = I$ do U là ma trận trực giao) (nhân 2 vế của phương trình trên với U)

Nguồn gốc của tên gọi SVD

Ta thấy SS^T là 1 ma trận đường chéo với các phần tử trên đường chéo là σ_1^2, σ_2^2,..., σ_r^2

- Þ USS^T là **Eigen Decompostion** của AA^T với σ_i^2 , $1 \leq i \leq r$ là các giá trị riêng (Singular values) của AA^T. Cái tên **Singular Value Decomposition** bắt đầu từ đây.
- Þ Theo đó, mỗi cột của U là 1 vector riêng của AA^T. Ta gọi mỗi cột này là **left-singular vectors** của A (gọi là left vì trong phương trình (1) U nằm bên trái. Tương tự, ta có $A^T A = VSS^T$ và các cột của V là **right-singular vectors.**

Compact SVD

Từ (1), ta cũng có thể viết lại dưới dạng tổng của các ma trận rank 1 như sau:

$$A = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \ldots + \sigma_r u_r v_r^T$$

$u_1, u_2, \ldots, u_r$ là các left singular vector

$v_1, v_2, \ldots, v_r$ là các right singular vector

$\sigma_1, \sigma_2, \ldots, \sigma_r$ là các singular value

Compact SVD

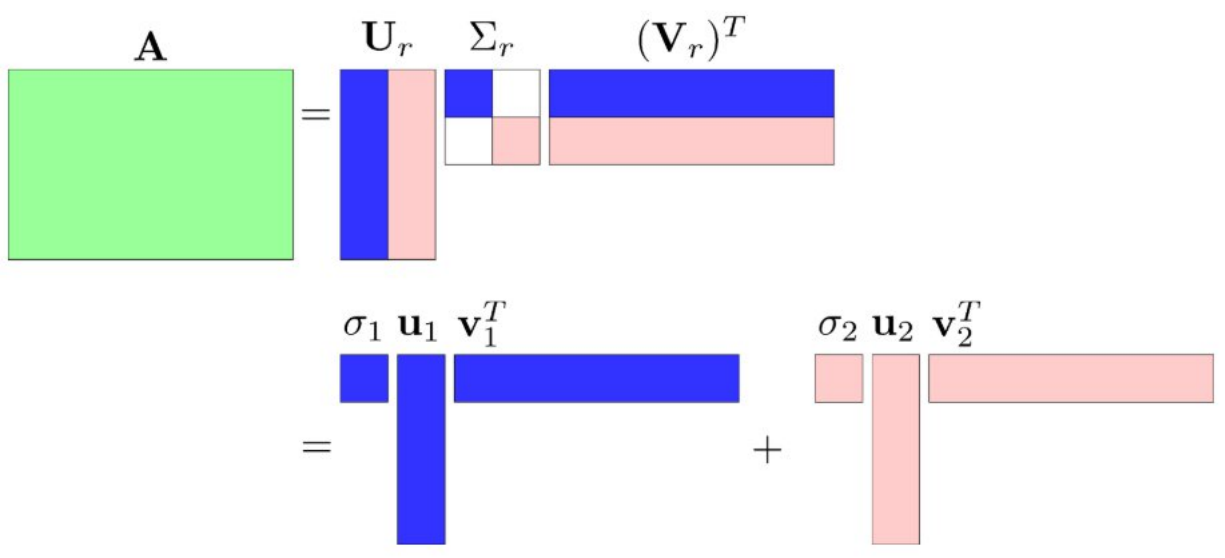
Từ (1), ta cũng có thể viết lại dưới dạng tổng của các ma trận rank 1 như sau:

$$A = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + .. + \sigma_r u_r v_r^T$$

Trong cách biểu diễn này, A chỉ phụ thuộc vào r hàng, cột đầu tiên của U,V và r giá trị khác 0 trên đường chéo của ma trân S. Nếu ma trận A có rank r rất nhỏ so với m,n thì chỉ cần giữ r singular values khác 0 và cột tương ứng của U, V để viết lại A.

⇒ Như vậy sẽ **tiết kiệm bộ nhớ và phép tính.**

Example (Compact SVD)



m=4, n=6, r=2

Truncated SVD (1 phương pháp low-rank approximation)

Trong ma trận S, có các giá trị $\sigma_1 \geq \sigma_2 \geq \ \ldots \ \geq \sigma_r \geq 0$, chỉ có 1 lượng nhỏ lớn còn lại thường nhỏ hoặc gần 0. Khi đó, ta có thể xấp xỉ A bằng tổng k < r các ma trận rank 1:

$$A \approx A_k = U_k S_k (V_k)^T = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \ \ldots \ + \sigma_r u_r v_r^T$$

Truncated SVD

Vậy làm thế nào để tìm k? Ta sẽ dựa trên công thức sau:

$$||A - A_k||_F^2 = \sum_{i=k+1}^{r} \sigma_i^2$$

Sai số do cách xấp xỉ trên chính là căn bậc hai của tổng bình phương của **Singular values** mà ta đã bỏ qua ở phần cuối của S. Sai số được định nghĩa là **Frobineous norm** của hiệu hai ma trận

Truncated SVD

k = 0, tức ma trận A gốc:

$$||A||_F^2 = \sum_{j=1}^{r} \sigma_i 2$$

$$\Rightarrow \frac{||A - Ak||_F 2}{||A||_F 2} = \frac{\sum_{i=k+1}^{r} \sigma_i 2}{\sum_{j=1}^{r} \sigma_i 2} \quad (2)$$

Nếu ta muốn giữ ít nhất **90%** dữ liệu, ta chỉ cần tìm k nhỏ nhất sao cho (2) ≥ **0.9**

Ứng dụng

Giảm chiều dữ liệu

Các ma trận A_k gần khít với A nên ta có thể dùng SVD để giảm chiều dữ liệu. Việc giảm chiều dữ liệu giúp ta có khả năng biểu diễn bộ dữ liệu đó một cách khá chính xác trên đồ thị. SVD là nền tảng toán học cho **Principal Component Analysis(PCA)**

Nén ảnh

Để lưu ảnh với **Truncated SVD**, ta lưu các ma trận $\mathbf{U_k \in \mathbb{R}^{m \times k}}$, $\mathbf{\Sigma_k \in \mathbb{R}^{k \times k}}$, $\mathbf{V_k \in \mathbb{R}^{n \times k}}$. Tổng số phần tử phải lưu là **k(m + n + 1)**. Giả sử mỗi phần tử được lưu bởi một số thực 4 byte, vậy số byte cần lưu trữ là **4k(m + n + 1).** Khi so giá trị này với ảnh gốc kích thước **mn**, mỗi giá trị là 1 số nguyên 1 byte thì ta có tỉ lệ nén là:

$$\frac{\mathbf{4k(m + n + 1)}}{\mathbf{mn}}$$

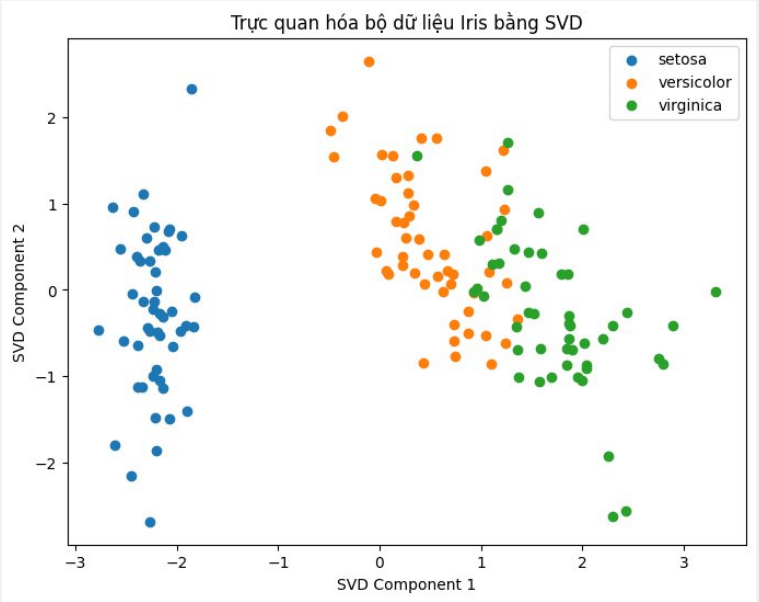
Ứng dụng

Recommendation System

Ý tưởng tương tự như nén ảnh, đó là căn cứ vào những cặp (user, item) đã được rating của **Utility matrix** để tìm ra một ma trận xấp xỉ tốt nhất và dự báo cho những cặp (user, item) chưa được **rating**.

Demo - Trực quan hoá bộ dữ liệu Iris bằng SVD

```python
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
import numpy as np

def custom_svd(X, k):

    # Truncated SVD
    U, S, VT = np.linalg.svd(X, full_matrices=False)

    # Giảm chiều dữ liệu
    X_result = U[:, :k] @ np.diag(S[:k])

    return X_result

iris = load_iris()
X = iris.data
y = iris.target

#Chuẩn hóa dữ liệu
scaler = StandardScaler()
X_std = scaler.fit_transform(X)

# Truncated SVD
X_svd = custom_svd(X_std, k=2)

# Vẽ bảng
plt.figure(figsize=(8,6))

for i, name in enumerate(iris.target_names):
    plt.scatter(
        X_svd[y == i, 0],
        X_svd[y == i, 1],
        label=name
    )

plt.xlabel("SVD Component 1")
plt.ylabel("SVD Component 2")
plt.title("Trực quan hóa bộ dữ liệu Iris bằng SVD")
plt.legend()
plt.show()
```



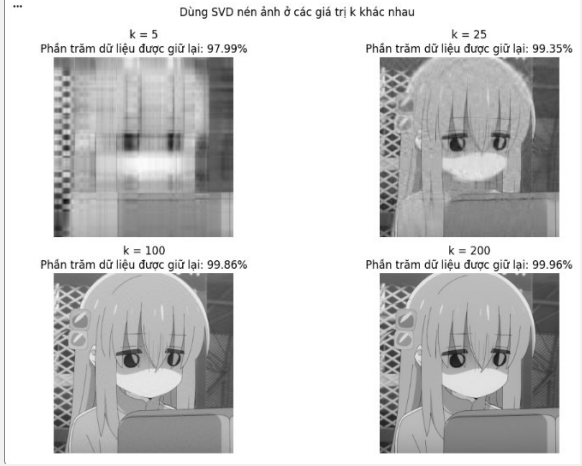
Demo - SVD nén ảnh

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import requests
from io import BytesIO

link_img = "https://i.pinimg.com/1200x/cc/80/f3/cc80f38579887963c2d71d7060081ea3.jpg"
response = requests.get(link_img)
img = Image.open(BytesIO(response.content))
img_arr = np.array(img)

plt.figure(figsize=(5,5))
plt.imshow(img)
plt.suptitle("Ảnh gốc")
plt.title("Image shape: %s"%str(img_arr.shape))
plt.axis("off")
plt.show()
```


```
# Chuyển sang ảnh xám
img_xam = np.array(img.convert("L"))

plt.figure(figsize=(5,5))
plt.imshow(img_xam, cmap="gray")
plt.title("Ảnh trước")
plt.axis("off")
plt.show()
```

Demo - SVD nén ảnh

```python
# SVD
U, S, VT = np.linalg.svd(img_xam, full_matrices=False)

ks = [5, 25, 100, 200]

plt.figure(figsize=(12,8))

for i, k in enumerate(ks):
    img_k = U[:, :k] @ np.diag(S[:k]) @ VT[:k, :]

    ratio_k = np.sum(S[:k]**2) / np.sum(S**2)

    plt.subplot(2, 2, i+1)
    plt.imshow(img_k, cmap="gray")
    plt.title(f"k = {k}\nPhần trăm dữ liệu được giữ lại: {ratio_k*100:.2f}%")
    plt.axis("off")

plt.suptitle("Dùng SVD nén ảnh ở các giá trị k khác nhau")
plt.show()
```

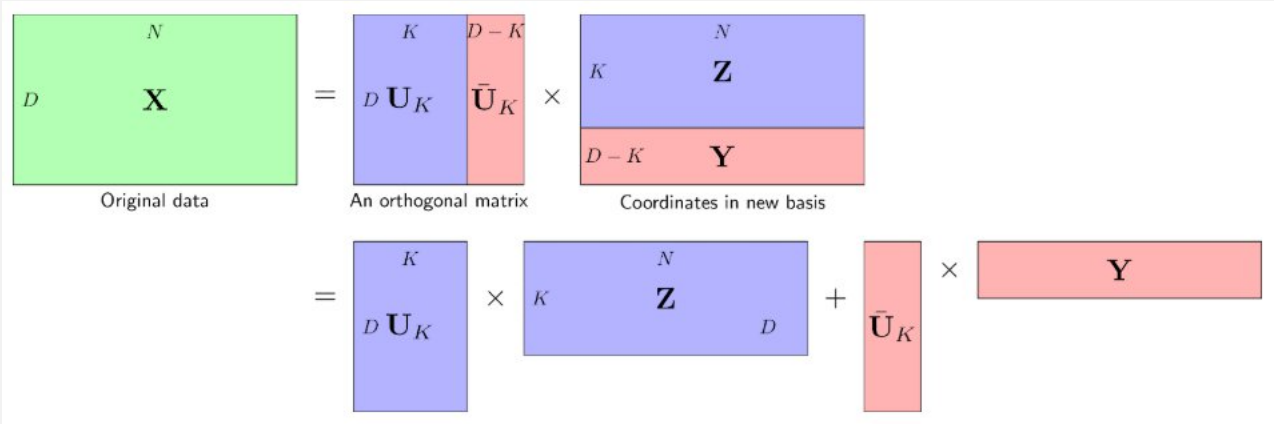


03 Principal Component Analysis (PCA)

Khái niệm - PCA

PCA (Phân tích thành phần chính) là một kỹ thuật giảm chiều dữ liệu và giúp chúng ta giảm số lượng thuộc tính trong một tập dữ liệu trong khi vẫn giữ được thông tin quan trọng nhất. Nó thay đổi các tập dữ liệu phức tạp bằng cách biến đổi các thuộc tính có quan hệ với nhau thành một tập hợp nhỏ các thành phần không tương quan.

Khái niệm - PCA

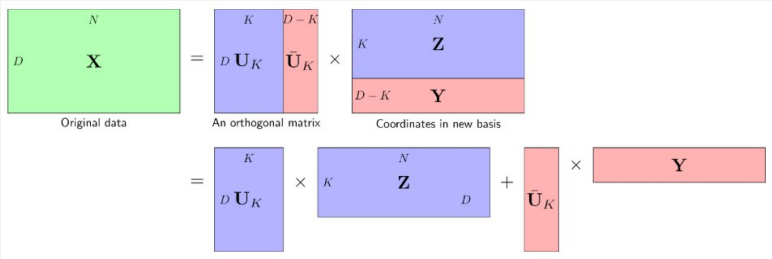


Ý tưởng chính của PCA: Tìm một hệ trực chuẩn mới sao cho trong hệ này, các thành phần quan trọng nhất nằm trong K thành phần đầu tiên.

Khái niệm - PCA

Mục đích của **PCA** là đi tìm ma trận trực giao U sao cho phần lớn thông tin được giữ lại ở phần màu xanh $\mathbf{U_K Z}$ và phần màu đỏ $\bar{\mathbf{U}}_\mathbf{K}\ \mathbf{Y}$ sẽ được lược bỏ và thay bằng một ma trận không phụ thuộc vào từng điểm dữ liệu. Nói cách khác, ta sẽ xấp xỉ Y bởi một ma trận có toàn bộ các cột là như nhau.

Chú ý rằng các cột này có thể phụ thuộc vào dữ liệu training nhưng không phụ thuộc vào dữ liệu test.



Các bước thực hiện PCA

Từ các suy luận phía trên, ta có thể tóm tắt lại các bước trong PCA như sau:

1 **Tính vector kỳ vọng của toàn bộ dữ liệu:**

$$\overline{X} = \frac{1}{N}\sum_{n=1}^{N} X_n$$

Các bước thực hiện PCA

Từ các suy luận phía trên, ta có thể tóm tắt lại các bước trong PCA như sau:

2 Trừ mỗi điểm dữ liệu đi vector kỳ vọng của toàn bộ dữ liệu:

$$\hat{X}_n = X_n - \bar{X}$$

3 Tính ma trận hiệp phương sai:

$$S = \frac{1}{N} \hat{X} \widehat{X^T}$$

Các bước thực hiện PCA

Từ các suy luận phía trên, ta có thể tóm tắt lại các bước trong PCA như sau:

4	Tính các trị riêng và vector riêng có norm bằng 1 của ma trận này, sắp xếp chúng theo thứ tự giảm dần của trị riêng.
5	Chọn K vector riêng ứng với K trị riêng lớn nhất để xây dựng ma trận U_K có các cột tạo thành một hệ trực giao. K vectors này, còn được gọi là các thành phần chính, tạo thành một không gian con gần với phân bố của dữ liệu ban đầu đã chuẩn hoá.

Các bước thực hiện PCA

Từ các suy luận phía trên, ta có thể tóm tắt lại các bước trong PCA như sau:

6	Chiếu dữ liệu ban đầu đã chuẩn hoá $\hat{X}$ xuống không gian con tìm được.
7	Dữ liệu mới chính là toạ độ của các điểm dữ liệu trên không gian mới

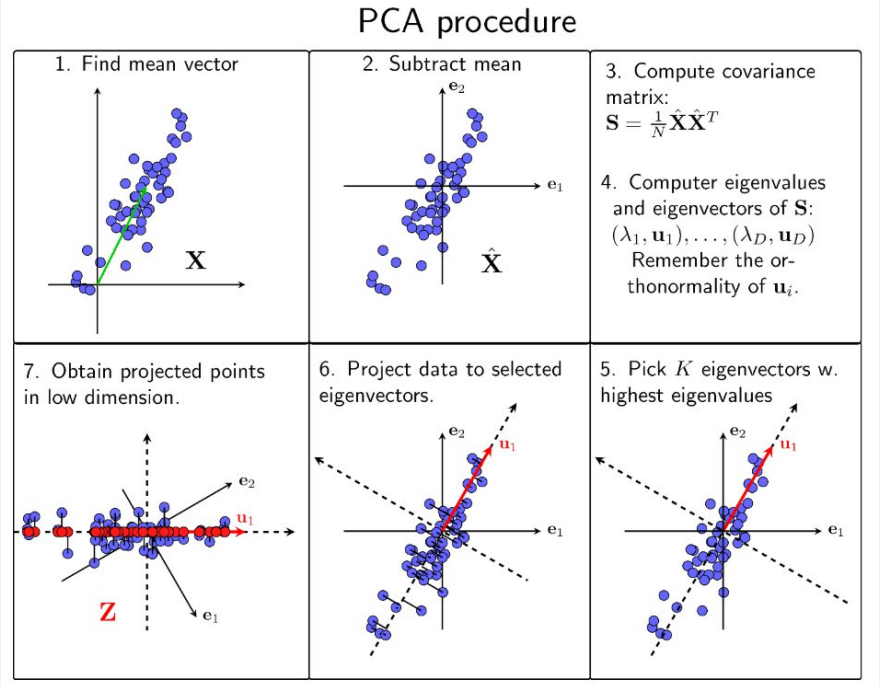
$$Z = U^T_K \hat{X}$$

Các bước thực hiện PCA

Dữ liệu ban đầu có thể tính được xấp xỉ theo dữ liệu mới như sau:

$$\mathbf{X} \approx U_k Z + \overline{X}$$

Các bước thực hiện PCA



Example (PCA)

Dataset 3 chiều đơn giản

4 điểm dữ liệu 3D:

$X_1 = \begin{bmatrix}2\\0\\1\end{bmatrix}$, $X_2 = \begin{bmatrix}3\\2\\1\end{bmatrix}$ $X3 = \begin{bmatrix}4\\1\\3\end{bmatrix}$ $X4 = \begin{bmatrix}5\\2\\4\end{bmatrix}$

Ma trận dữ liệu:

$$X = \begin{bmatrix}2 & 3 & 4 & 5\\0 & 1 & 1 & 2\\1 & 2 & 3 & 4\end{bmatrix}$$

Số mẫu N = 4

Example (PCA)

1 Tính vector kỳ vọng (mean)

$$\bar{X} = \frac{1}{4}\begin{bmatrix} 2+ & 3+ & 4+ & 5 \\ 0+ & 1+ & 1+ & 2 \\ 1+ & 2+ & 3+ & 4 \end{bmatrix} = \begin{bmatrix} 3.5 \\ 1 \\ 2.5 \end{bmatrix}$$

Example (PCA)

2 **Chuẩn hoá dữ liệu**

$$\widehat{X_n} = X_n - \bar{X}$$

$$\bar{X} = \frac{1}{4}\begin{bmatrix} -1.5 & -0.5 & 0.5 & 1.5 \\ -1 & 0 & 0 & 1 \\ -1.5 & -0.5 & 0.5 & 1.5 \end{bmatrix}$$

Example (PCA)

3 | **Tính ma trận hiệp phương sai**

$$S = \frac{1}{N}\hat{X}\hat{X}^T$$

$$S = \begin{bmatrix} 1.25 & 0.75 & 1.25 \\ 0.75 & 0.5 & 0.75 \\ 1.25 & 0.75 & 1.25 \end{bmatrix}$$

Example (PCA)

4 | **Trị riêng và vector riêng**

Tính eigenvalues của S:

$$|S-\lambda I| = 0$$
$$\lambda 1 = 3,\ \lambda 2 = 0,\ \lambda 3 = 0$$

Vector riêng tương ứng (đã chuẩn hoá):

$$u_1 = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\\1\end{bmatrix} \qquad u_2 = \begin{bmatrix}0\\1\\0\end{bmatrix} \qquad u_3 = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\\-1\end{bmatrix}$$

Example (PCA)

5 | **Chọn K = 2 thành phần chính**

Chọn 2 trị riêng lớn nhất:

$$U_2 = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 \\ 0 & 1 \\ \frac{1}{\sqrt{2}} & 0 \end{bmatrix}$$

Example (PCA)

6 **Chiếu dữ liệu 3D xuống không gian 2D**

$$Z = U^T{}_2 \hat{X}$$

Kết quả dữ liệu 2D:

$$Z = \begin{bmatrix} -2.12 & -0.71 & 0.71 & 2.12 \\ -1 & 0 & 0 & 1 \end{bmatrix}$$

Mỗi cột là **toạ độ mới (PC1, PC2)**

Example (PCA)

7 **Khôi phục xấp xỉ dữ liệu ban đầu**

$$\mathbf{X} \approx \mathbf{U_k Z} + \overline{\mathbf{X}}$$

Sau khi khôi phục giống dữ liệu ban đầu

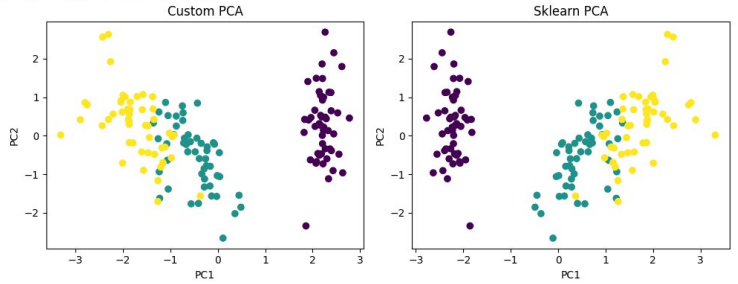
Ứng dụng

- **Nhận dạng**: **Eigenface** là một trong các phương pháp phổ biến nhất trong bài toán nhận dạng khuôn mặt. Eigenface thực ra chính là PCA. Các Eigenfaces chính là các eigenvectors ứng với các trị riêng lớn nhất của ma trận hiệp phương sai.
- **PCA cũng được ứng dụng trong lĩnh vực y tế**, nơi có nhiều nguồn dữ liệu có tương quan với nhau như tuổi thọ, căn bệnh...
- **Xác định các sự kiện không bình thường**
- **Nén dữ liệu**

Demo – So sánh giữa Custom PCA với Sklearn PCA

```
print("So sánh phương sai")
print("Custom PCA:", var_explained[:2])
print("Sklearn PCA:", var_sk)

plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.title("Custom PCA")
plt.scatter(Z[:,0], Z[:,1], c=y)
plt.xlabel("PC1")
plt.ylabel("PC2")


plt.subplot(1,2,2)
plt.title("Sklearn PCA")
plt.scatter(Z_sk[:,0], Z_sk[:,1], c=y)
plt.xlabel("PC1")
plt.ylabel("PC2")

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

```
So sánh phương sai
Custom PCA: [0.72962445 0.22850762]
Sklearn PCA: [0.72962445 0.22850762]
```



Demo – PCA nén ảnh

```
plt.figure(figsize=(20, 10))

#Hiển thị ảnh gốc
plt.subplot(1, len(k_list) + 1, 1)
plt.imshow(img_np)
plt.title("GỐC (100% data)")
plt.axis('off')

#Hiển thị các ảnh đã nén và giải nén qua PCA
for i, (k, res_img, ratio) in enumerate(results):
    plt.subplot(1, len(k_list) + 1, i + 2)
    plt.imshow(res_img)
    plt.title(f"K = {k}\n(~{ratio:.1f}% data)")
    plt.axis('off')

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

04 SVD & PCA

Mối liên hệ giữa SVD & PCA

Giả sử rằng dữ liệu đã được tiền xử lý và có kỳ vọng bằng 0.
Nói một cách đơn giản, PCA yêu cầu tính trị riêng và vector riêng của ma trận hiệp phương sai, tức là tính

$$\frac{1}{n-1}XX^T$$

trong đó X là ma trận dữ liệu.
Bởi vì ma trận hiệp phương sai là ma trận đối xứng, cụ thể là ma trận đường chéo, các vector riêng có thể được normalize để chúng trực giao

$$\frac{1}{n-1}XX^T = \frac{1}{n-1}WDW^T$$

Mặt khác, khai triển SVD đối với X, nghĩa là $\mathbf{X = U\Sigma V^T}$.
Chúng ta xây dựng ma trận hiệp phương sai từ phép phân tích này.

Mối liên hệ giữa SVD & PCA

$$\frac{1}{n-1}XX^T = \frac{1}{n-1}(U\Sigma V^T)(U\Sigma^2 V^T)^T = \frac{1}{n-1}(U\Sigma VT)(V\Sigma V^T)^T$$

Bởi vì V trực giao cho nên

$$\frac{1}{n-1}XX^T = \frac{1}{n-1}U\Sigma 2U^T$$

Điều này ngụ ý rằng bình phương các trị riêng của XX^T là các **singular value** của X.

Nói tóm lại, việc sử dụng SVD để hiện thực PCA sẽ tốt hơn về mặt số lượng tính toán so với khi bắt đầu từ việc tạo ra một ma trận hiệp phương sai, bởi vì tạo XX^T có thể không bảo toàn được độ chính xác.

Source Code & Demo

Bài thuyết trình cùng toàn bộ Demo được lưu trữ tại *GitHub*

Thank you for your attention!