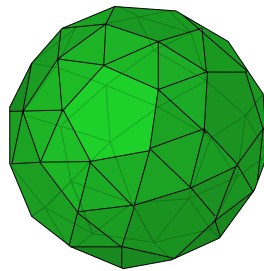


Projects in Mathematics and Applications

# Principal Component Analysis

Ngày 17 tháng 8 năm 2021

Hoàng Huyền Trang <sup>\*</sup>  
Dương Quang Thành <sup>†</sup>  
Trần Phan Anh Danh <sup>‡</sup>



---

<sup>\*</sup>Trường Đại học Khoa học Tự nhiên - ĐHQGHN

<sup>†</sup>Gettysburg College

<sup>‡</sup>Trường Phổ thông Năng Khiếu - ĐHQG TP HCM

# Mục lục

<b>1</b>	<b>Thuật toán Principal Component Analysis</b>	<b>1</b>
1.1	Bài toán tối ưu và công thức nghiệm của PCA . . . . .	1
1.2	Các bước của thuật toán PCA . . . . .	2
<b>2</b>	<b>Thuật toán Kernel Principal Component Analysis</b>	<b>3</b>
2.1	Nhược điểm của PCA và hướng cải thiện . . . . .	3
2.2	Thành lập công thức cho Kernel PCA. Kernel trick và vai trò . . . . .	4
2.3	So sánh giữa PCA và Kernel PCA. Nhược điểm của Kernel PCA. . . . .	6
<b>3</b>	<b>Cài đặt thuật toán PCA</b>	<b>6</b>
3.1	Tự xây dựng thuật toán PCA thủ công . . . . .	6
3.2	Sử dụng thuật toán PCA của thư viện <i>sklearn</i> . . . . .	7
<b>4</b>	<b>Áp dụng thuật toán PCA bằng Python trên các tập dữ liệu</b>	<b>7</b>
4.1	Áp dụng trên tập dữ liệu Swiss Roll của thư viện <i>sklearn</i> . . . . .	7
4.2	Áp dụng trên tập dữ liệu Make Moons của thư viện <i>sklearn</i> . . . . .	9
4.3	Áp dụng thuật toán PCA để nén ảnh . . . . .	10

## Lời cảm ơn

Hai tuần được học tập ở trại hè PiMA 2021 là khoảng thời gian thật sự ý nghĩa. Chúng tôi đã học hỏi được nhiều điều, đó giúp chúng tôi đạt được thành quả như ngày hôm nay. Chúng tôi xin gửi lời cảm ơn đến BTC trại hè đã xây dựng một môi trường ý nghĩa, để chúng tôi có cơ hội học tập và hoàn thành dự án. Cảm ơn các anh chị mentors đã giảng dạy những kiến thức Toán thú vị, cho chúng tôi cái nhìn rõ hơn về Toán Ứng dụng nói chung và Khoa học Dữ liệu nói riêng. Đặc biệt, chúng tôi xin cảm ơn các mentors như anh Linh, chị Thư, và chị Phương đã đồng hành cùng nhóm để xây dựng và hoàn thành đề tài. Cuối cùng, chúng tôi muốn cảm ơn các bạn trại sinh đã cùng nhau học tập và vui chơi với chúng tôi trong suốt thời gian qua.

## Giới thiệu đề tài

Thuật toán giảm chiều dữ liệu có thể được hình thức hóa như sau. Cho tập dữ liệu

$$T = (x_1, x_2, \dots, x_m)$$

và một ánh xạ đặc trưng

$$\varphi : T \mapsto \mathbb{R}^N$$

ma trận dữ liệu  $X \in \mathbb{R}^{N \times m}$  được định nghĩa bởi  $(\varphi(x_1), \varphi(x_2), \dots, \varphi(x_m))$ .

Các kĩ thuật trong giảm chiều dữ liệu tập trung vào việc tìm một ma trận biểu diễn  $K$  - chiều là  $Y \in \mathbb{R}^{K \times m}$  ( $K < N$ ), sao cho biểu diễn mới này giữ được đặc tính của ma trận dữ liệu  $X$ .

Trong báo cáo này, chúng tôi sẽ trình bày một phương pháp cơ bản dùng để giảm số chiều của dữ liệu là phương pháp phân tích thành phần chính - PCA.

Nội dung của bản báo cáo bao gồm 3 phần chính:

**Phần 1. Thuật toán Principal Component Analysis**

**Phần 2. Thuật toán Kernel Principal Component Analysis**

**Phần 3. Thực hành thuật toán PCA bằng Python**

# 1 Thuật toán Principal Component Analysis

## 1.1 Bài toán tối ưu và công thức nghiệm của PCA

Xét tập dữ liệu  $\{x_n\}$ , trong đó mỗi  $x_n$  là một vector thuộc không gian vector Euclid  $N$ -chiều,  $n = 1, \dots, m$ .

**Input:**  $\{x_n\}$ ,  $n = 1, \dots, m$  thuộc  $\mathbb{R}^N$ .

**Output:**  $\{y_n\}$ ,  $n = 1, \dots, m$  thuộc  $\mathbb{R}^K$  với  $K < N$ .

Phương pháp PCA được thực hiện bởi một phép chiếu trực giao, chiếu tập dữ liệu ban đầu ( $N$ -chiều) vào một không gian tuyến tính có số chiều thấp hơn ( $K$ -chiều), sao cho phương sai của dữ liệu đầu ra là lớn nhất.

Một cách tương đương, PCA là thuật toán mà ta cần tối thiểu hóa bình phương khoảng cách giữa mỗi điểm dữ liệu và hình chiếu tương ứng của chúng trên không gian mới  $K$ -chiều. **(2)**

Cho  $\bar{X} \in \mathbb{R}^{m \times N}$  là một ma trận dữ liệu có trung bình tại tâm, tức là  $\sum_{i=1}^m x_i = 0$ .

Kí hiệu  $P_K$  là tập hợp các ma trận của phép chiếu trực giao  $N$ -chiều có hạng bằng  $K$ .

Theo định nghĩa **(2)**, thuật toán PCA hoàn toàn được xác định/ định nghĩa bởi nghiệm ma trận  $P$  của phép chiếu trực giao của bài toán tối ưu sau

$$\min_{P \in P_K} \|P\bar{X} - \bar{X}\|_F^2$$

**(3).**

Chú ý:  $\|A\|_F$  là chuẩn Frobenius của ma trận  $A$ .

Kí hiệu  $C = \frac{1}{m} \bar{X} \bar{X}^T$  là ma trận hiệp phương sai của  $\bar{X}$ .

Định lý sau đây nói về nghiệm của **(3)**, tức là ma trận  $P^* \in P_K$  thỏa mãn điều kiện vừa thiết lập trên.

**Định lý.** Gọi  $P^*$  là nghiệm của PCA, hay cũng là nghiệm ma trận chiếu vuông góc của **(3)**. Khi đó,  $P^* = U_K U_K^T$ ,  $U_K \in \mathbb{R}^{N \times K}$  trong đó  $U_K$  là ma trận tạo bởi  $K$  vector riêng ứng với  $K$  giá trị riêng lớn nhất của  $C = \frac{1}{m} \bar{X} \bar{X}^T$ . Hơn nữa, biểu diễn trong không gian  $K$ -chiều tương ứng của  $X$  là  $Y = U_K^T X$ .

ii

**Chứng minh.** Vì  $P$  là ma trận của một phép chiếu trực giao nên  $P = P^T$ . Mặt khác, toán tử  $\text{Tr}$  là tuyến tính và  $P$  là một ma trận lũy đẳng (nghĩa là  $P^2 = P$ ) nên theo tính chất của chuẩn Frobenius, ta có

$$\begin{aligned} \|P\bar{X} - \bar{X}\|_F^2 &= \text{Tr}[(P\bar{X} - \bar{X})^T (P\bar{X} - \bar{X})] = \text{Tr}[\bar{X}^T P^2 \bar{X} - 2\bar{X}^T P \bar{X} + \bar{X}^T \bar{X}] \\ &= \text{Tr}[\bar{X}^T \bar{X}] - \text{Tr}[\bar{X}^T P \bar{X}] \end{aligned}$$

Rõ ràng  $\text{Tr}[\bar{X}^T \bar{X}]$  là một hằng số nên bài toán đã cho đưa về việc tìm giá trị lớn nhất của  $\text{Tr}[\bar{X}^T P \bar{X}]$ . Do  $P$  là ma trận của một phép chiếu trực giao nên tồn tại ma trận  $U \in \mathbb{R}^{N \times K}$  với các cột trực giao sao cho  $P = U U^T$ . Ta được

$$\text{Tr}[\bar{X}^T P \bar{X}] = \text{Tr}[U^T \bar{X} \bar{X}^T U] = \sum_{i=1}^K u_i^T \bar{X} \bar{X}^T u_i$$

(trong đó  $u_i$  là cột thứ  $i$  của ma trận  $U$ ,  $i = 1, \dots, K$ ).

Ta thấy rằng tổng trên đạt giá trị lớn nhất khi và chỉ khi  $u_i$  là các vector riêng có 2-norm bằng 1 ứng với  $K$  giá trị riêng lớn nhất của ma trận hiệp phương sai  $C$ . Hơn nữa, ma trận  $\bar{X} \bar{X}^T$  có cùng vector riêng với ma trận hiệp phương sai  $C$ .

Ta kết luận rằng:  $P\bar{X} = U_K U_K^T \bar{X}$  là Xấp xỉ của dữ liệu ban đầu trong không gian  $K$ -chiều và  $Y = U_K^T \bar{X}$  là tọa độ của dữ liệu cũ trong cơ sở trực giao mới.

**Chú ý:**  $\varphi: \mathbb{R}^N \rightarrow \mathbb{R}^K$  - được gọi là hàm nhúng trong PCA.

$\varphi^{-1}: \mathbb{R}^K \rightarrow \mathbb{R}^N$  - được gọi là hàm xấp xỉ ngược của PCA.

## 1.2 Các bước của thuật toán PCA

Sau khi chứng minh được công thức nghiệm của phương pháp PCA, ta có thể tóm tắt lại các bước của thuật toán này như sau:

**Bước 1:** Tính vector trung bình của toàn bộ dữ liệu

$$\bar{x} = \frac{1}{m} \sum_{n=1}^m x_n$$

**Bước 2:** Trừ mỗi điểm dữ liệu cho vector trung bình của toàn bộ dữ liệu. Ta xếp các vector được chuẩn hóa thành các cột của ma trận  $\bar{X}$  - ma trận dữ liệu có trung bình tại tâm.

$$\hat{x}_n = x_n - \bar{x}$$

**Bước 3:** Tính ma trận hiệp phương sai  $C$  của dữ liệu đã được chuẩn hóa

$$C = \frac{1}{m} \sum_{n=1}^m (x_n - \bar{x})(x_n - \bar{x})^T$$

hoặc

$$C = \frac{1}{m} \bar{X} \bar{X}^T$$

**Bước 4:**

+ Tính các giá trị riêng và vector riêng có chuẩn bằng 1 của ma trận  $C$ , sắp xếp chúng theo thứ tự giảm dần của trị riêng.

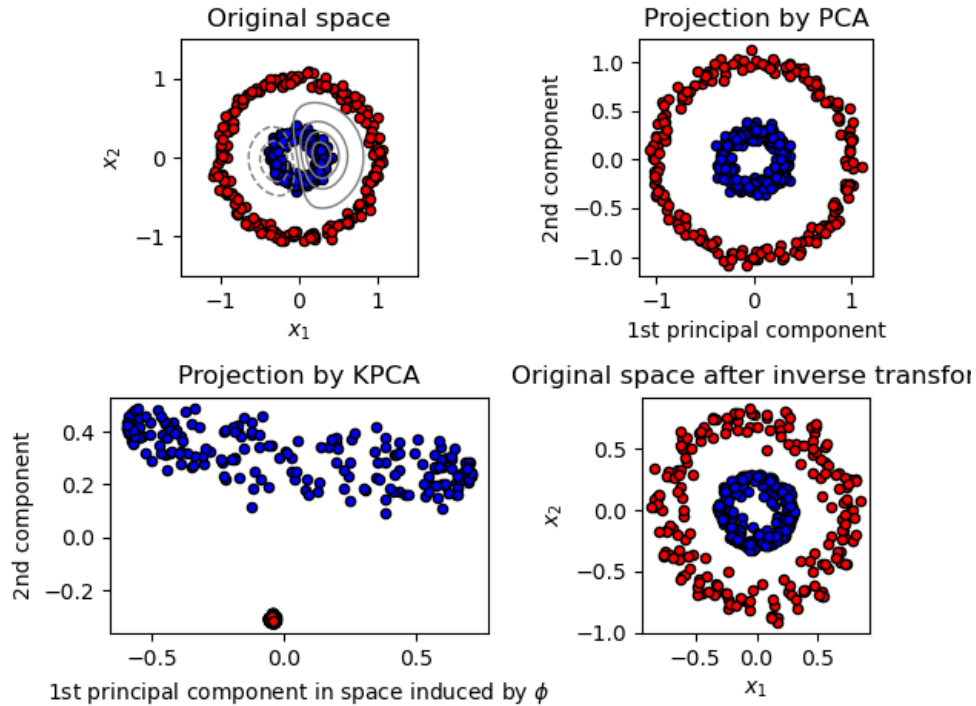
+ Chọn  $K$  vector riêng ứng với  $K$  giá trị riêng lớn nhất để tạo thành ma trận trực giao  $U$ .

**Bước 5:** Chiếu dữ liệu ban đầu đã chuẩn hoá xuống không gian con tìm được, dữ liệu mới chính là tọa độ của các điểm dữ liệu trên không gian mới.

## 2 Thuật toán Kernel Principal Component Analysis

### 2.1 Nhược điểm của PCA và hướng cải thiện

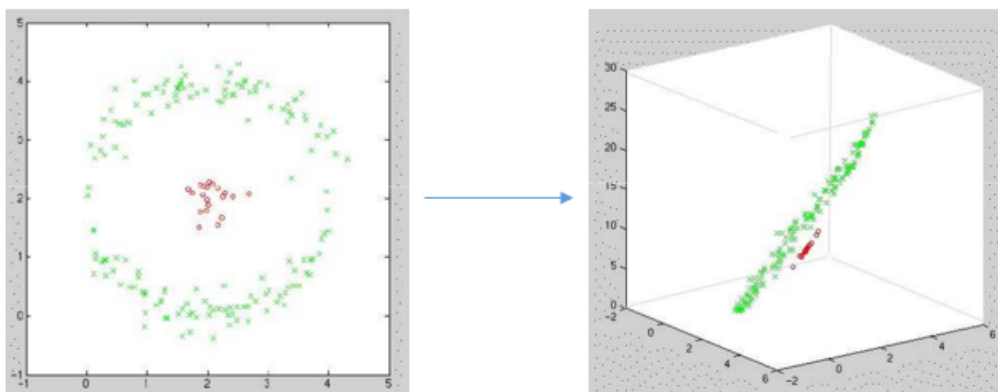
Thuật toán PCA giả sử rằng tập dữ liệu đầu vào  $\{\mathbf{x}_n\}$  không ngẫu nhiên và phân bố gần một siêu phẳng<sup>1</sup> nào đó. Do đó đối với các tập dữ liệu không tuyến tính, thuật toán PCA thất bại trong việc chiếu tập dữ liệu đầu vào thành hai vùng rời nhau (xanh, đỏ). Một ví dụ đối với tập dữ liệu này đó là phân bố đồng tâm như hình sau đây [1].



Một hướng để cải thiện kết quả thuật toán PCA này đó là ta phải tìm cách sao cho tập dữ liệu đầu vào  $\{\mathbf{x}_n\}$  của chúng ta xuất hiện mối quan hệ tuyến tính. Đối với mẫu phân bố đồng tâm, ta có thể làm xuất hiện quan hệ tuyến tính này thông qua việc chiếu tập dữ liệu đầu vào  $\{\mathbf{x}_n\}$  lên không gian ba chiều  $\mathbb{R}^3$  bằng ánh xạ

$$\Phi: \mathbb{R}^2 \mapsto \mathbb{R}^3, \quad (x_1, x_2) \mapsto (x_1, x_2, x_1^2 + x_2^2).$$

Hình ảnh minh họa ánh xạ này như bên dưới [3].



Ở trong không gian ba chiều này, các điểm dữ liệu phân bố xung quanh các mặt phẳng (các điểm xanh tạo thành một mặt phẳng, các điểm đỏ tạo thành một mặt phẳng), từ đó xuất hiện mối quan hệ tuyến tính và do đó ta có thể áp dụng thuật toán PCA một cách hiệu quả. Tiếp theo chúng ta sẽ áp dụng trực giác này để thiết lập thuật toán Kernel PCA.

<sup>1</sup>Trong không gian Euclide  $D$  chiều, một siêu phẳng là một không gian con có  $D - 1$  chiều. Ví dụ, một mặt phẳng (2 chiều) là siêu phẳng trong không gian 3 chiều.

## 2.2 Thành lập công thức cho Kernel PCA. Kernel trick và vai trò

Cho một tập dữ liệu đầu vào  $\{\mathbf{x}_n\}$  gồm có  $N$  phần tử, mỗi phần tử có số chiều  $D$ . Không mất tính tổng quát, giả sử trung bình cộng của các dữ liệu này bằng 0, tức là  $\sum_{i=1}^N \mathbf{x}_i = 0$ .

Ma trận hiệp phương sai của tập dữ liệu này là  $\mathbf{S} = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^T$ .

Bây giờ ta chiếu mỗi điểm dữ liệu  $\mathbf{x}_n$  vào không gian đặc trưng  $F$  (có số chiều lớn hơn  $D$ ) thông qua ánh xạ  $\Phi$  (như ý tưởng ở mục 2.1), tức là

$$\Phi: \mathbb{R}^D \mapsto F, \quad \mathbf{x} \mapsto \Phi(\mathbf{x}).$$

Ta tạm thời giả sử trung bình cộng của các  $\Phi(\mathbf{x}_n)$  cũng bằng 0, tức là  $\sum_{i=1}^N \Phi(\mathbf{x}_i) = 0$ . Ma trận hiệp phương sai  $\mathbf{S}$  trở thành

$$\hat{\mathbf{S}} = \frac{1}{N} \sum_{i=1}^N \Phi(\mathbf{x}_i) \Phi(\mathbf{x}_i)^T. \quad (1)$$

Chúng ta áp dụng thuật toán PCA vào không gian đặc trưng  $F$  này, tức là ta đi tìm các trị riêng  $\lambda_i$  và vector riêng tương ứng  $\mathbf{v}_i$  sao cho

$$\hat{\mathbf{S}} \mathbf{v}_i = \lambda_i \mathbf{v}_i, \quad (2)$$

và các  $\lambda_i$  đạt giá trị lớn nhất, tương ứng là các thành phần chính. Ở đây ta áp dụng thêm điều kiện chuẩn hóa các vector  $\mathbf{v}_i$ , tức  $\mathbf{v}_i^T \mathbf{v}_i = 1$ .

Thay (1) vào (2), ta được

$$\frac{1}{N} \sum_{n=1}^N \Phi(\mathbf{x}_n) [\Phi(\mathbf{x}_n)^T \mathbf{v}_i] = \lambda_i \mathbf{v}_i, \quad (3)$$

Từ (3), ta nhận thấy mọi nghiệm  $\mathbf{v}_i$  của (2) đều thuộc  $\text{span}\{\Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2), \dots, \Phi(\mathbf{x}_N)\}$ . Do đó ta có thể viết vector  $\mathbf{v}_i$  dưới dạng

$$\mathbf{v}_i = \sum_{n=1}^N a_{in} \Phi(\mathbf{x}_n), \quad (4)$$

trong đó  $a_{in}$  là các số thực. Thay (4) vào (3) và nhân bên trái hai vế cho  $\Phi(\mathbf{x}_l)^T$ , ta được

$$\frac{1}{N} \sum_{n=1}^N \Phi(\mathbf{x}_l)^T \Phi(\mathbf{x}_n) \sum_{m=1}^N a_{im} \Phi(\mathbf{x}_n)^T \Phi(\mathbf{x}_m) = \lambda_i \sum_{n=1}^N a_{in} \Phi(\mathbf{x}_l)^T \Phi(\mathbf{x}_n). \quad (5)$$

Từ phương trình (5), ta nhận thấy rằng các hạng tử  $\Phi(\mathbf{x}_n)$  chỉ xuất hiện dưới các tích vô hướng như  $\Phi(\mathbf{x}_l)^T \Phi(\mathbf{x}_n)$ . Do đó để giải tìm các thành phần chính  $\lambda_i$  từ (5), ta chỉ cần tính các tích vô hướng hiệu quả mà không cần thực hiện ánh xạ  $\Phi$ .

Bây giờ ta thực hiện kernel trick, đó là đặt hàm kernel  $k(\mathbf{x}_m, \mathbf{x}_n) = \Phi(\mathbf{x}_m)^T \Phi(\mathbf{x}_n)$  và ma trận kernel (còn được gọi là ma trận Gram)  $\mathbf{K}$  có dạng

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \cdots & k(\mathbf{x}_1, \mathbf{x}_N) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \cdots & k(\mathbf{x}_2, \mathbf{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & k(\mathbf{x}_N, \mathbf{x}_2) & \cdots & k(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix}. \quad (6)$$

Phương trình (5) lúc này trở thành

$$\mathbf{K} \mathbf{a}_i = N \lambda_i \mathbf{a}_i,$$

trong đó  $\mathbf{a}_i$  là vector cột có các phần tử là  $a_{ni}$  với  $n = 1, 2, \dots, N$ . Chúng ta giải tìm các trị riêng ( $N\lambda_i$ ) lớn nhất của ma trận  $\mathbf{K}$  và từ đó tìm được các thành phần chính  $\lambda_i$ .

Các hàm kernel  $k(\mathbf{x}_m, \mathbf{x}_n)$  thường sử dụng (do đã ứng dụng thành công trong [6]) là các hàm:

- Kernel đa thức:  $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^n$ .
- Kernel cơ sở xuyên tâm (RBF):  $k(\mathbf{x}, \mathbf{y}) = e^{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}}$ .
- Kernel sigmoid:  $k(\mathbf{x}, \mathbf{y}) = \tanh(\kappa(\mathbf{x} \cdot \mathbf{y}) + \Theta)$ .

Quay trở lại, ban đầu ta đã giả sử  $\sum_{i=1}^N \Phi(\mathbf{x}_i) = 0$ , tuy nhiên thông thường không phải lúc nào điều này cũng đúng. Do đó ta cần tịnh tiến tập dữ liệu ảnh  $\{\Phi(\mathbf{x}_n)\}$  sao cho trung bình cộng của chúng bằng 0. Khi đó, mỗi điểm dữ liệu ảnh của ta sẽ trở thành

$$\hat{\Phi}(\mathbf{x}_n) = \Phi(\mathbf{x}_n) - \frac{1}{N} \sum_{i=1}^N \Phi(\mathbf{x}_i).$$

Mỗi phần tử  $K_{nm}$  của ma trận Gram ở (6) trở thành

$$\begin{aligned} \hat{K}_{nm} &= \hat{\Phi}(\mathbf{x}_n)^T \hat{\Phi}(\mathbf{x}_m) \\ &= \Phi(\mathbf{x}_n)^T \Phi(\mathbf{x}_m) - \frac{1}{N} \sum_{l=1}^N \Phi(\mathbf{x}_n)^T \Phi(\mathbf{x}_l) - \frac{1}{N} \sum_{l=1}^N \Phi(\mathbf{x}_l)^T \Phi(\mathbf{x}_m) + \frac{1}{N^2} \sum_{j=1}^N \sum_{l=1}^N \Phi(\mathbf{x}_j)^T \Phi(\mathbf{x}_l) \\ &= k(\mathbf{x}_n, \mathbf{x}_m) - \frac{1}{N} \sum_{l=1}^N k(\mathbf{x}_l, \mathbf{x}_m) - \frac{1}{N} \sum_{l=1}^N k(\mathbf{x}_n, \mathbf{x}_l) + \frac{1}{N^2} \sum_{j=1}^N \sum_{l=1}^N k(\mathbf{x}_j, \mathbf{x}_l). \end{aligned}$$

Viết dưới dạng ma trận, ta có

$$\hat{\mathbf{K}} = \mathbf{K} - \mathbf{1}_N \mathbf{K} - \mathbf{K} \mathbf{1}_N + \mathbf{1}_N \mathbf{K} \mathbf{1}_N,$$

trong đó  $\mathbf{1}_N$  đại diện cho ma trận cỡ  $N \times N$  và có mọi phần tử bằng  $\frac{1}{N}$ . Ta áp dụng thuật toán PCA vào ma trận Gram  $\hat{\mathbf{K}}$  này bình thường.

### Các bước thực hiện Kernel PCA

Đối với một tập dữ liệu  $\{\mathbf{x}_n\}$ , ta thực hiện các bước sau đây [3]:

1. Chọn một hàm kernel  $k(\mathbf{x}, \mathbf{y})$ .
2. Tính các phần tử của ma trận kernel  $\mathbf{K}$ .
3. Trọng tâm hóa lại ma trận  $\mathbf{K}$  thành

$$\hat{\mathbf{K}} = \mathbf{K} - \mathbf{1}_N \mathbf{K} - \mathbf{K} \mathbf{1}_N + \mathbf{1}_N \mathbf{K} \mathbf{1}_N.$$

4. Giải bài toán trị riêng  $\hat{\mathbf{K}} \mathbf{a}_i = \lambda_i \mathbf{a}_i$ .
5. Ứng với mỗi thành phần chính  $\lambda_i$ , hình chiếu của điểm dữ liệu  $\mathbf{x}$  lên vector  $\mathbf{v}_i$  là

$$y_i(\mathbf{x}) = \Phi(\mathbf{x})^T \mathbf{v}_i = \sum_{n=1}^N a_{in} k(\mathbf{x}, \mathbf{x}_n).$$

## Giải thích về ý nghĩa về việc đặt hàm kernel

Ở trên ta đã chọn hàm kernel  $k(\mathbf{x}_m, \mathbf{x}_n) = \Phi(\mathbf{x}_m)^T \Phi(\mathbf{x}_n)$  với một công thức tính tích vô hướng (đa thức, cơ sở xuyên tâm, sigmoid, ...). Trong bài báo [6], các tác giả đã chứng minh rằng thông qua giải tích hàm, với mọi hàm kernel  $k(\mathbf{x}_m, \mathbf{x}_n)$  mà ta chọn, luôn tồn tại ánh xạ  $\Phi$  thỏa mãn định nghĩa tích vô hướng đưa ra trong không gian đặc trưng  $F$ .

Thông thường, người ta sẽ sử dụng kernel cơ sở xuyên tâm (Gaussian) và sigmoid thay cho kernel đa thức, vì các kernel này có thể cho ra các kết quả đúng đắn hơn nếu càng cho nhiều dữ liệu đầu vào<sup>2</sup>.

## 2.3 So sánh giữa PCA và Kernel PCA. Nhược điểm của Kernel PCA.

Cả hai thuật toán PCA và Kernel PCA đều có cốt lõi là PCA, tuy nhiên đối tượng áp dụng là khác nhau. Cụ thể:

- PCA dùng ngay trên tập thông tin ban đầu, Kernel PCA áp dụng trên không gian đặc trưng  $F$ .
- PCA tìm các trị riêng của ma trận hiệp phương sai  $\mathbf{S}$  có kích cỡ  $D \times D$ , còn Kernel PCA tìm các trị riêng của ma trận Gram  $\hat{\mathbf{K}}$  có kích cỡ  $N \times N$ .

Nhược điểm lớn của Kernel PCA đó là quá trình tính trị riêng  $\lambda_i$  của ma trận Gram  $\hat{\mathbf{K}}$ , do kích thước của  $\hat{\mathbf{K}}$  phụ thuộc vào số điểm dữ liệu đầu vào  $N$ . Với  $N$  rất lớn, các thuật toán tìm trị riêng thường dùng như phương pháp Power sẽ tính toán lâu, do đó người ta thường xấp xỉ các trị riêng này.

## 3 Cài đặt thuật toán PCA

### 3.1 Tự xây dựng thuật toán PCA thủ công

Sử dụng các hàm có sẵn của thư viện *numpy*, ta có thể áp dụng các bước xây dựng PCA được mô tả ở trên:

```
def PCA(A, k):  
    M = mean(A, axis=0)  
    # trung bình các cột  
    C = A - M  
    # căn giữa  
    V = cov(C.T)  
    # ma trận Hiệp phương sai  
    eigen_values, eigen_vectors = eigh(V)  
    # tính trị riêng, vector riêng  
    eigen_vectors=eigen_vectors.T[::-1]  
    # sắp xếp theo thứ tự giảm dần của eigen_values  
    S = eigen_vectors[:k]  
    # chọn n eigen_vectors có eigen_values lớn nhất  
    return S.dot(C.T).T
```

Lưu ý: số chiều dữ liệu sau khi sử dụng PCA ký hiệu là biến  $k$ , và tập dữ liệu được ký hiệu là  $A$ .

---

<sup>2</sup>Tham khảo thêm tại <https://qr.ae/pG0o4q>.

### 3.2 Sử dụng thuật toán PCA của thư viện sklearn

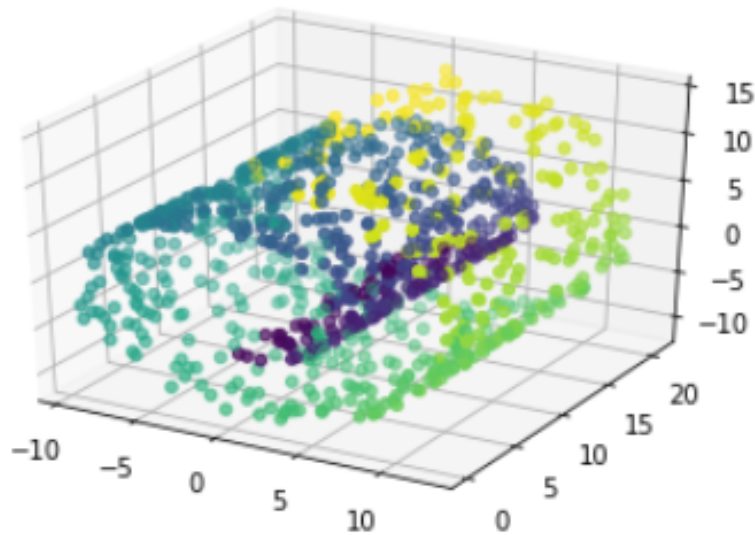
Thư viện *sklearn* có cung cấp sẵn thuật toán PCA như sau:

```
pca = PCA(k)
pca.fit(A)
P = pca.transform(A)
P
```

Lưu ý: số chiều dữ liệu sau khi sử dụng PCA ký hiệu là biến  $k$ , và tập dữ liệu được ký hiệu là  $A$ .

## 4 Áp dụng thuật toán PCA bằng Python trên các tập dữ liệu

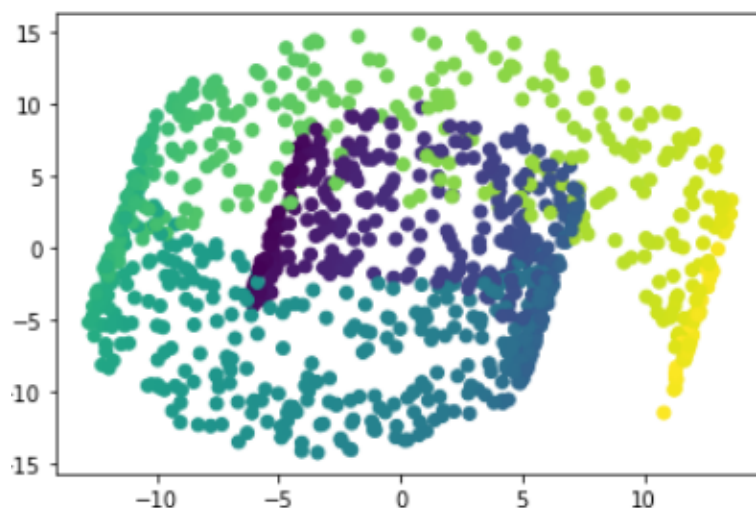
### 4.1 Áp dụng trên tập dữ liệu Swiss Roll của thư viện sklearn



Hình ảnh dữ liệu Swiss Roll trong không gian 3 chiều.

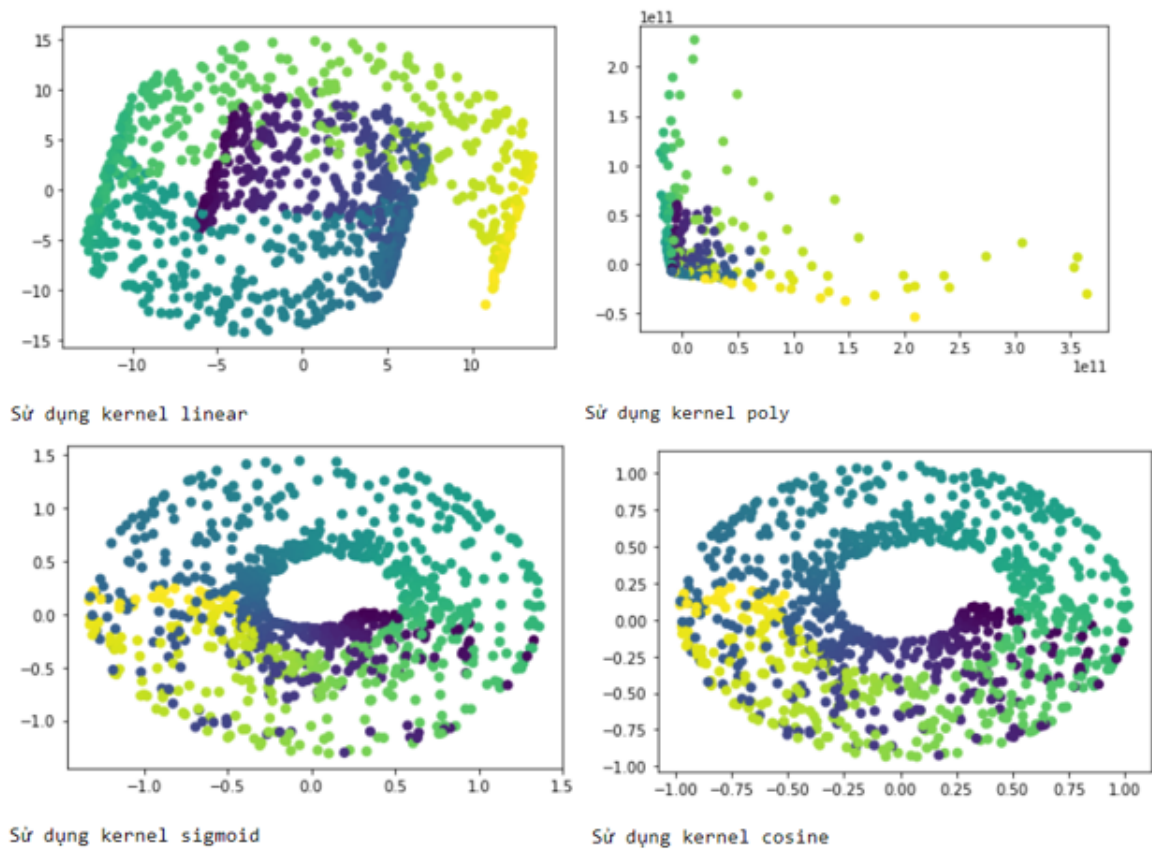
Có thể thấy, các điểm trong tập dữ liệu được sắp xếp theo hình một tấm thảm được cuộn lại. Vì thế, khi giảm chiều dữ liệu xuống còn 2 chiều, chúng ta mong muốn dữ liệu được “trải ra” như tấm thảm phẳng.

Đây là kết quả khi sử dụng thuật toán PCA khi giảm chiều dữ liệu:



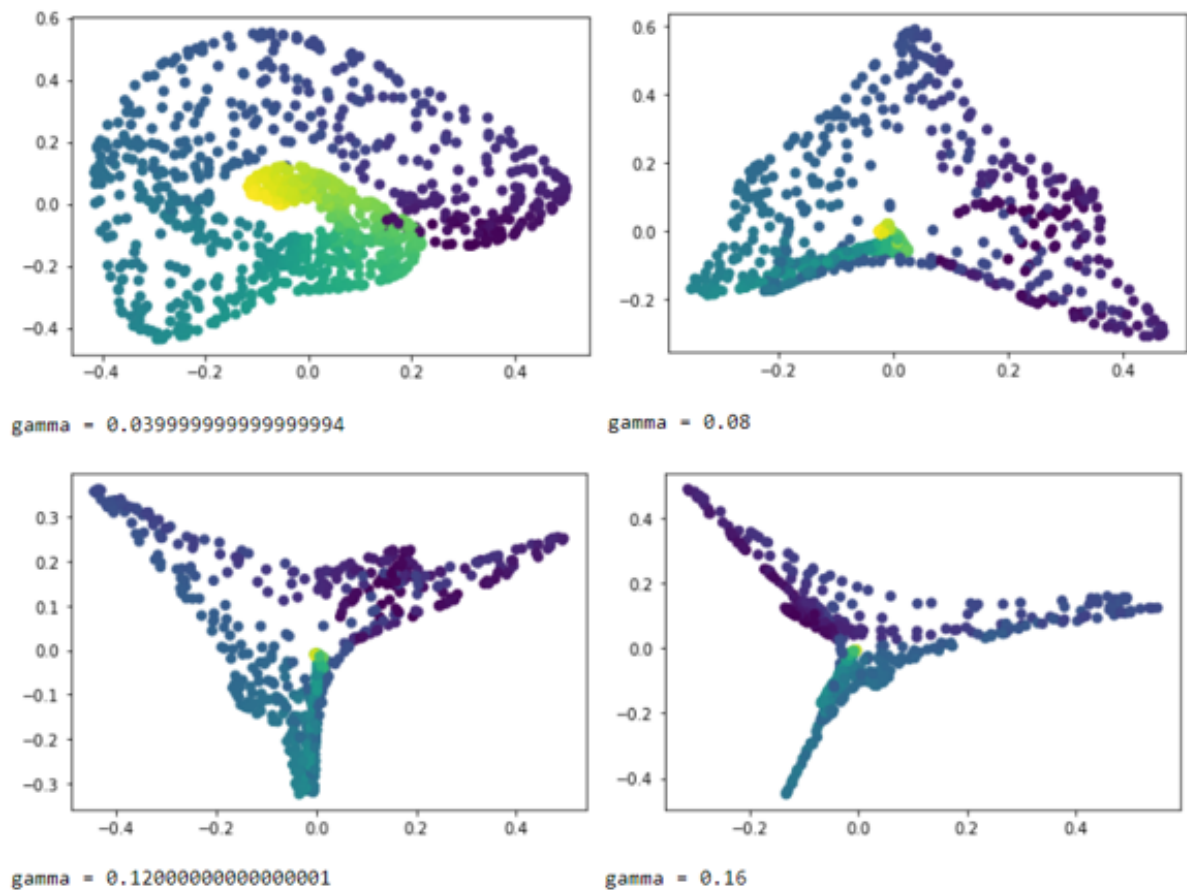
Hình ảnh dữ liệu Swiss Roll sau khi áp dụng PCA thông thường giảm còn 2 chiều.

Vì kết quả không được như ý muốn, chúng ta có thể thử sử dụng Kernel PCA để giảm chiều với một số kernel phổ biến. Đây là một số hình ảnh:



Hình ảnh dữ liệu Swiss Roll sau khi áp dụng Kernel PCA giảm còn 2 chiều với một số kernel thông dụng.

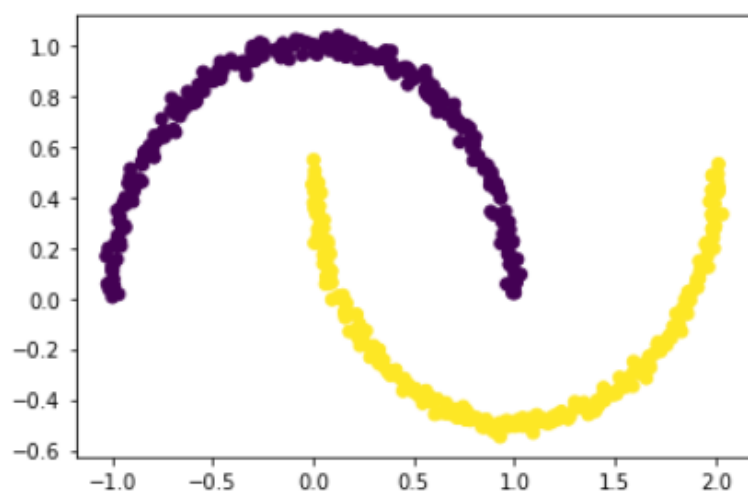
Chúng ta cũng có thể sử dụng kernel *rbf*, một kernel khá phổ biến với thông số  $\gamma$  có các giá trị khác nhau:



Hình ảnh dữ liệu Swiss Roll sau khi áp dụng Kernel PCA giảm còn 2 chiều với kernel *rbf* và một số giá trị  $\gamma$  khác nhau..

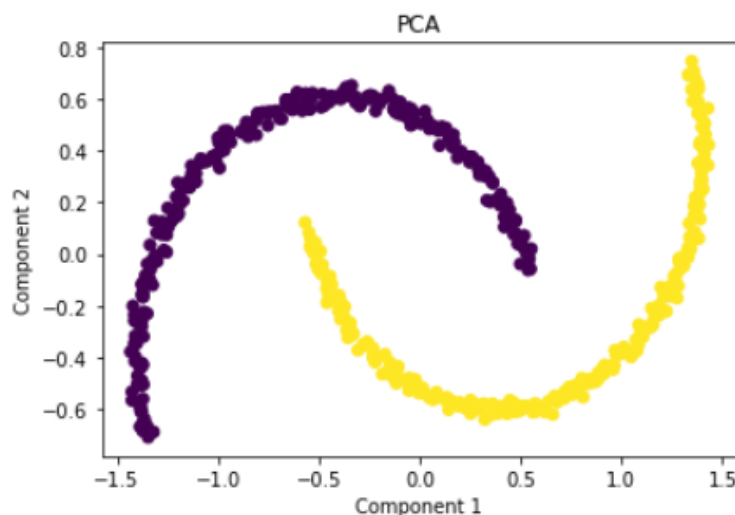
Đây là một số hình ảnh tốt nhất mà chúng tôi có thể xây dựng được khi giảm chiều dữ liệu Swiss Roll bằng phương pháp PCA. Vì thế, có thể kết luận rằng thuật toán PCA không thật sự phù hợp cho dữ liệu này.

## 4.2 Áp dụng trên tập dữ liệu Make Moons của thư viện sklearn



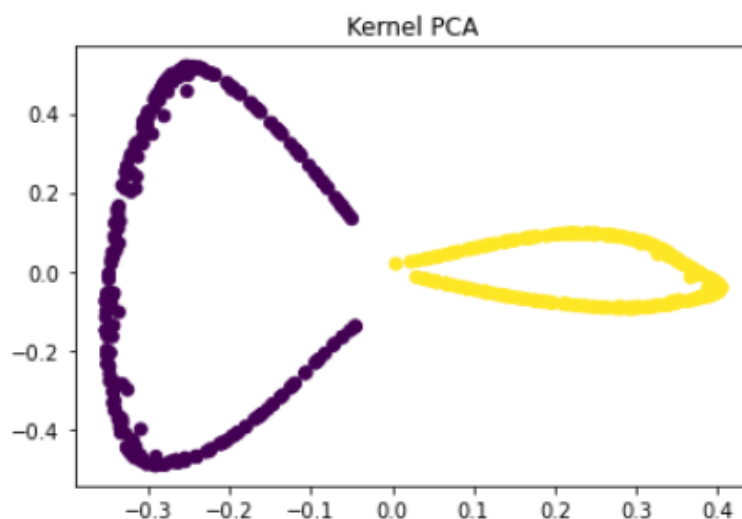
Hình ảnh dữ liệu Make Moons ban đầu trên không gian 2 chiều.

Ta sẽ sử dụng thuật toán PCA để chuyển tập dữ liệu đang có thành một tập dữ liệu hai chiều. Sử dụng PCA thông thường:



Hình ảnh dữ liệu Make Moons sau khi dùng PCA thông thường chuyển thành dữ liệu 2 chiều.

Sử dụng Kernel PCA với kernel là rbf,  $\gamma = 15$ :



Hình ảnh dữ liệu Make Moons sau khi dùng kernel PCA chuyển thành dữ liệu 2 chiều với kernel *rbf* và  $\gamma = 15$ .

Mặc dù PCA thông thường không tác động đáng kể đến dữ liệu, kernel PCA lại cho một kết quả khá khác biệt. Vì chúng tôi sử dụng thuật toán PCA để chuyển một tập dữ liệu 2 chiều thành một tập 2 chiều tương tự, thuật toán PCA không mang ý nghĩa về giảm chiều dữ liệu. Tuy nhiên, vẫn có một số lý do khiến nhóm đã quyết định chọn tập dữ liệu này:

- Thứ nhất, tập dữ liệu này cho thấy được cách hoạt động của Kernel PCA. Kernel PCA có khả năng hoạt động tốt trong những tập dữ liệu phi tuyến tính. Bằng cách chọn tập dữ liệu 2 chiều để dễ trực quan hóa, chúng ta có thể thấy được kernel PCA hoạt động như thế nào.
- Thứ hai, một trong những mục tiêu của PCA nói chung là tối đa hóa phương sai. Hiểu nôm na, tối đa hóa phương sai chính là làm nổi bật những khác nhau của các điểm dữ liệu trong tập dữ liệu cho trước. So sánh hình ảnh trước và sau khi sử dụng kernel PCA, ta có thể thấy dữ liệu sau khi giảm chiều sẽ phù hợp hơn cho một số thuật toán, chẳng hạn như thuật toán phân cụm dữ liệu K-means, một thuật toán phân cụm dựa vào khoảng cách giữa các điểm và các tâm.

### 4.3 Áp dụng thuật toán PCA để nén ảnh

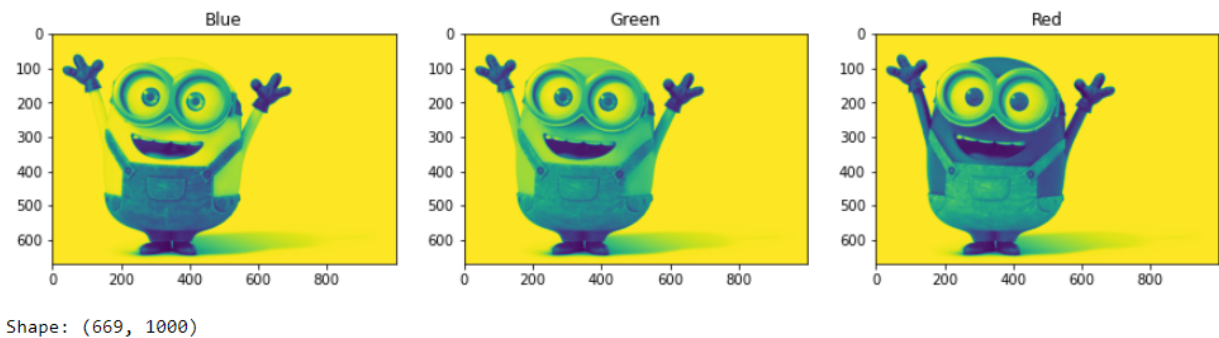
PCA là một công cụ khá tốt trong việc nén ảnh do có thể giữ được các thành phần chính để không mất quá nhiều thông tin.



Hình ảnh ban đầu cần giảm dữ liệu. Ảnh bởi Ryan Nakamura.

Áp dụng PCA, chúng ta sẽ nén bức ảnh trên. Ở đây, nhóm mình sẽ cung cấp 2 phương pháp nén ảnh.

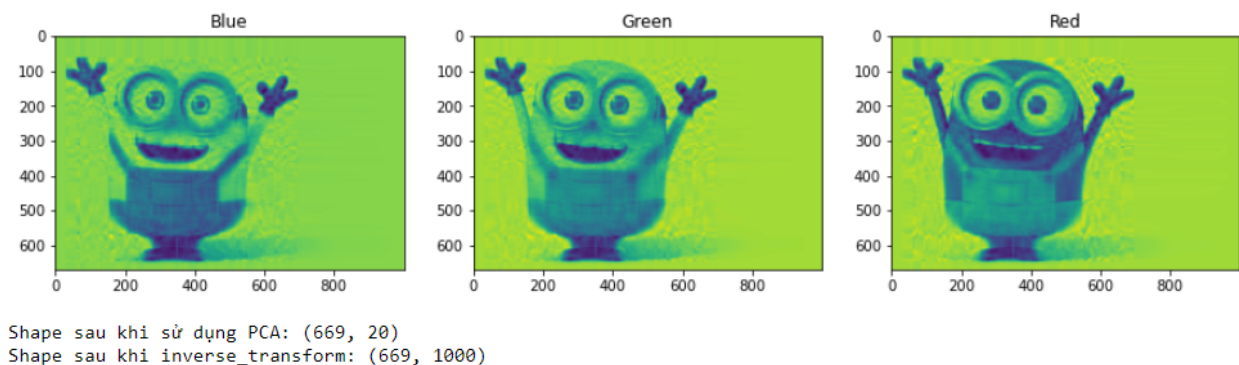
Ở cả hai phương pháp, do ảnh sử dụng là ảnh màu, ta đều phải chia ảnh thành 3 kênh màu. Ở đây, tụi mình dùng thư viện cv2 để chia ảnh làm 3 kênh xanh dương, xanh lá, và đỏ.



Ảnh ban đầu được chia thành 3 kênh màu.

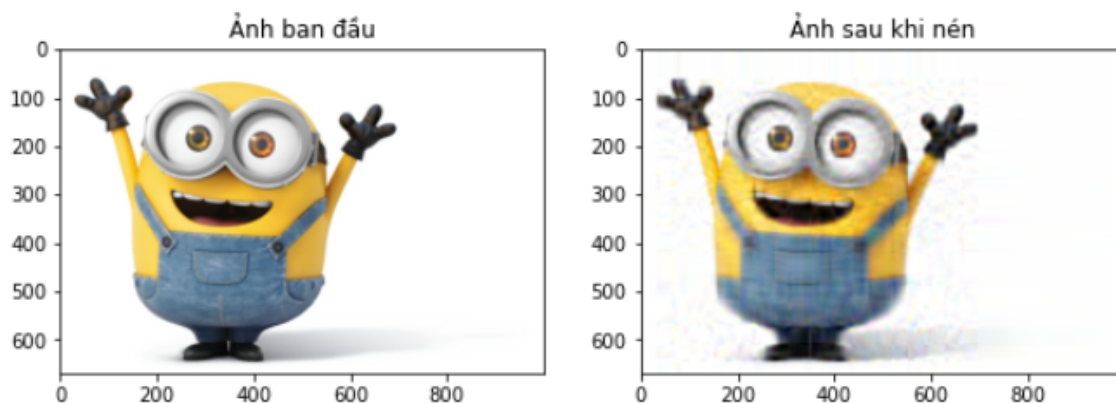
#### 4.3.1 Cách 1

Sau khi chia thành 3 kênh màu, các kênh được biểu thị bằng một ma trận. Với phương pháp 1, ta có thể áp dụng PCA giảm chiều dữ liệu lên trực tiếp ma trận này. Ma trận ban đầu có 669 dòng X 1000 cột, sau khi giảm chiều ma trận còn 669 dòng X 20 cột. Sau đó, để dựng lại ảnh, ta có thể dùng hàm `inversetransform`, tương đương hàm xấp xỉ ngược, để khôi phục lại ảnh sau khi nén.



Ảnh 3 kênh màu sau khi được giảm chiều.

Gộp 3 kênh màu lại, ta có thể dựng lại ảnh sau khi nén:



Mean squared error: 142.67377188374587

Gộp 3 kênh lại và so sánh với ảnh ban đầu.

Với mỗi điểm dữ liệu được miêu tả bởi 3 kênh màu, trong kênh giá trị dao động từ 1 đến 255, Mean Squared Error gần bằng 142 là một giá trị khá tốt. Tỷ số nén ở đây là  $\frac{20}{1000} = 2\%$ .

#### 4.3.2 Cách 2

Sau khi chia ảnh thành 3 kênh màu, theo phương pháp 2, ta có thể chia từng kênh - là các ma trận - thành các patch. Có thể hiểu rằng ta cắt nhỏ ảnh thành các phần nhỏ hơn và bằng nhau. Sau đó, ta áp dụng PCA trên từng patch này. Ở đây mỗi patch có kích cỡ 3 dòng X 10 cột, và chúng ta sẽ giảm còn 5 thành phần chính.

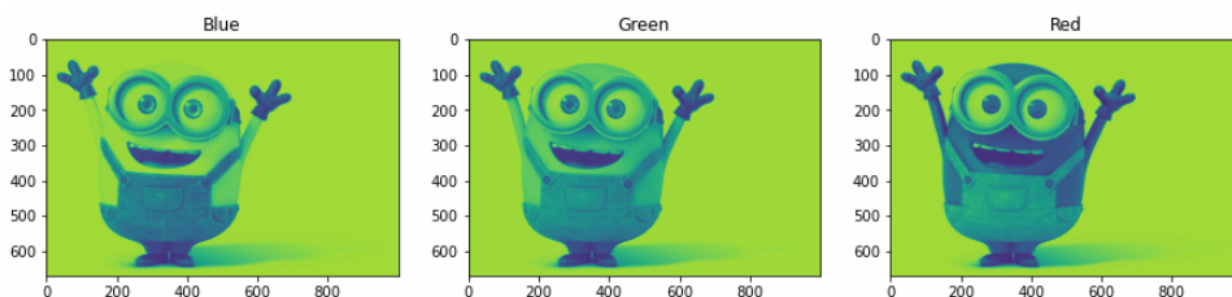
Sau khi giảm chiều, chúng ta có thể sắp xếp lại các patch theo vị trí ban đầu. Khi sử dụng phương pháp này, khi vẽ heatmap của các thành phần chính, có thể thấy các heatmap mang ít nhiều hình ảnh của hình ảnh ban đầu.



Ảnh heatmaps của các thành phần chính từ 3 kênh màu.

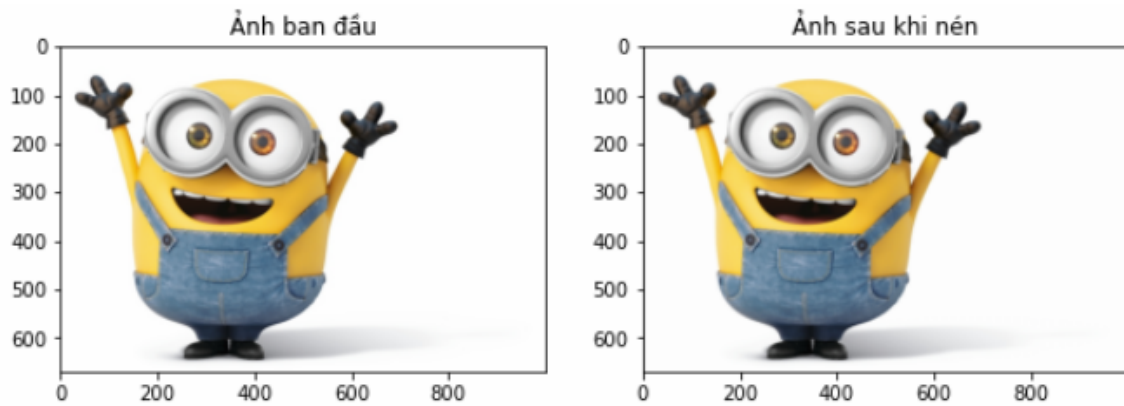
Từ trái sang phải là các heatmap của 5 thành phần chính, lần lượt từ trên xuống dưới với các kênh xanh dương, xanh lá, và đỏ.

Sau đó, ta có thể dùng hàm xấp xỉ ngược để dựng lại ảnh theo từng kênh.



Ảnh 3 kênh màu được dựng lại từ các thành phần chính của các patch.

Cuối cùng, ta gộp khác kênh lại để dựng lại ảnh màu sau khi nén.



Mean squared error: 25.66137772365858

Gộp 3 kênh lại và so sánh với ảnh ban đầu.

Tỉ số nén ở đây chính là  $\frac{5}{30} = 16.7\%$ .

## Tài liệu

- [1] Machine Learning in Python. <https://scikit-learn.org/stable/>.
- [2] *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, New York, NY, USA, 1992. Association for Computing Machinery.
- [3] Rita Osadchy's lecture on Kernel PCA, 2011.
- [4] Schölkopf B., Smola A., and Müller KR. Kernel principal component analysis. [8], pages 583–588.
- [5] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.
- [6] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. [2], pages 144–152.
- [7] Afshin Rostamizadeh By Mehryar Mohri and Ameet Talwalkar. *Foundations of Machine Learning*.
- [8] Wulfram Gerstner, Alain Germond, Martin Hasler, and Jean-Daniel Nicoud, editors. *Proceedings of the 7th International Conference on Artificial Neural Networks*. Springer, 1997.