

# Principal component analysis

PiMA 2021: The Mathematics of Data Science

---

**Hoàng Huyền Trang   Dương Quang Thành   Trần Phan Anh Danh**

Ngày 7 tháng 8 năm 2021



1. Principal component analysis
2. Kernel principal component analysis
3. Thực hành bằng Python

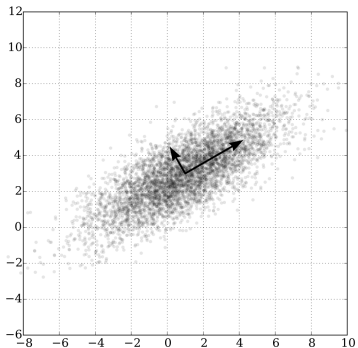
# Principal component analysis

---



Phương pháp PCA dựa trên một quan sát:

- + Dữ liệu thường không phân bố ngẫu nhiên trong không gian.
- + Thay vào đó chúng phân bố gần các siêu phẳng nào đó.





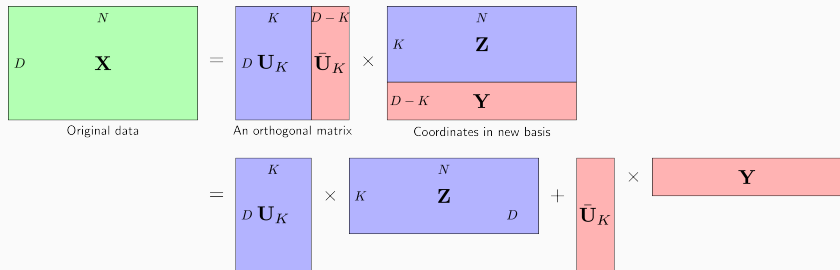
Xét tập dữ liệu  $\{x_n\}$  và  $n = 1, \dots, N$ . Các  $x_n$  là các biến thuộc không gian vector Euclid với số chiều  $D$ .

**Input:** Các vector  $\{x_n\}$   $n = 1, \dots, N$  thuộc không gian vector Euclid -  $D$  chiều

**Output:** Các vector  $\{z_n\}$   $n = 1, \dots, K$  thuộc không gian vector  $K$  chiều.

Hệ cơ sở trực chuẩn  $U$  sao cho:  $Z = U_K^T X$

( $U_K$  tạo bởi  $K$  cột đầu tiên của Ma trận trực giao  $U$ )



Nguồn ảnh: Blog Machine Learning cơ bản

Theo hình vẽ, ta có:  $X = U_K Z + \bar{U}_K Y$

$$Z = U_K^T X$$

$$Y = \bar{U}_K^T X$$



$$Y \approx b1^T$$

( $1^T \in \mathbb{R}^{1 \times N}$  là vector hàng có toàn bộ các phần tử bằng 1)

Giả sử đã tìm được  $U$ , ta cần tìm  $b$  thoả mãn:

$$b = \operatorname{argmin}_b \|Y - b1^T\|_F^2 \implies b = \overline{U}_K^T \bar{x}$$

( $\bar{x}$  - là vector trung bình của toàn bộ dữ liệu)

**Nhận xét:** Sẽ thuận tiện hơn nếu vector trung bình  $\bar{x} = 0$ . Việc này đạt được nếu ngay từ đầu khi chúng ta trừ mỗi vector dữ liệu đi vector trung bình của toàn bộ dữ liệu.



Bài toán trở thành:

$$X \approx \bar{X} = U_K Z + \bar{U}_K \bar{U}_K^T \bar{X} \mathbf{1}^T$$

**Hàm mất mát:**

$$J = \frac{1}{N} \|X - \bar{X}\|_F^2 = \sum_{K+1}^D u_i^T S u_i$$

với  $S$  là ma trận hiệp phương sai của dữ liệu.

**Định nghĩa:**

$$S = \frac{1}{N} \sum_1^N (x_n - \bar{x})(x_n - \bar{x})^T$$



Xét  $L = \sum_{i=1}^D u_i^T S u_i$

Ta có:  $L = \sum_{i=1}^D u_i^T S u_i = \text{Trace}(S) = \sum_{i=1}^D \lambda_i$

Với  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_D \geq 0$  là các giá trị riêng của ma trận nửa xác định dương  $S$ .

Việc tối thiểu hàm mất mát  $J = \sum_{K+1}^D u_i^T S u_i$  tương đương với việc tối đa hàm  $F := L - J = \sum_{i=1}^K u_i^T S u_i$



## Bổ Đề:

$F = \sum_{i=1}^K u_i S u_i^T$  đạt giá trị lớn nhất khi và chỉ khi  $u_i$  là các vector riêng ứng với  $K$  trị riêng  $\lambda_i$   $i = \overline{1, K}$

$$\max F = \sum_{i=1}^K \lambda_i$$



## Các bước thực hiện PCA:

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

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

**Bước 2:** Trừ mỗi điểm dữ liệu cho vector trung bình của toàn bộ dữ liệu.

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

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

$$S = \frac{1}{N} \sum_1^N (x_n - \bar{x})(x_n - \bar{x})^T$$



## Bước 4:

- + Tính các trị riêng và vector riêng có norm bằng 1 của ma trận  $S$ , 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 lập ma trận trực giao  $U_K$ .

**Bước 5:** Chiều dữ liệu ban đầu đã chuẩn hoá xuống không gian con tìm được.

(Dữ liệu mới chính là toạ độ của các điểm dữ liệu trên không gian mới)

# Kernel principal component analysis

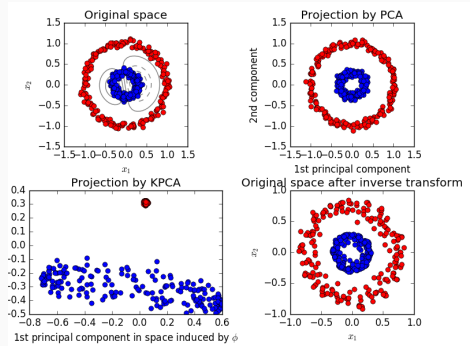
---



## Ví dụ (phân bố đồng tâm).

- Sử dụng không gian con tuyến tính (PCA) cho đầu ra không có ích (mẫu vẫn giữ nguyên).

→ Kernel PCA ở đây chiếu tập dữ liệu ban đầu thành hai vùng (màu đỏ và màu xanh) rời nhau.

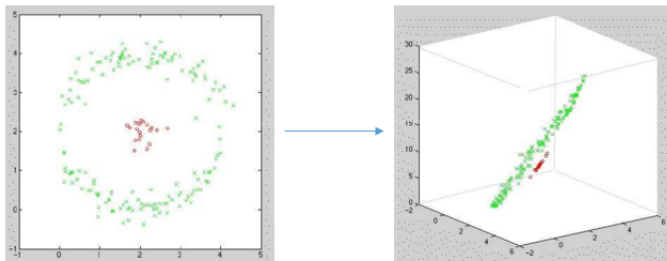


[lijiancheng0614.github.io](https://github.com/lijiancheng0614)



Ở không gian hai chiều, mẫu phân bố đồng tâm là không thể phân tách tuyến tính được (không tồn tại một đường thẳng chia tập dữ liệu thành hai phần mà mỗi phần chỉ chứa một màu).

Bằng ánh xạ  $\Phi: \mathbb{R}^2 \mapsto \mathbb{R}^3$ ,  $(x_1, x_2) \mapsto (x_1, x_2, x_1^2 + x_2^2)$  thì tập dữ liệu đã có thể phân tách tuyến tính được.



Rita Osadchy



Cho một tập dữ liệu  $\{\mathbf{x}_n\}$  với  $n = 1, 2, \dots, N$  có số chiều  $D$ , và

$$\sum_{i=1}^N \mathbf{x}_i = \mathbf{0}.$$

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 các điểm dữ liệu  $\mathbf{x}_n$  vào một không gian đặc trưng  $F$  thông qua ánh xạ  $\Phi$ :

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



Giả sử rằng trung bình cộng của các dữ liệu qua hàm  $\Phi$  bằng 0, tức

$$\text{là } \sum_{i=1}^N \Phi(\mathbf{x}_i) = 0.$$

Khi đó ma trận hiệp phương sai trở thành

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

Ta áp dụng thuật toán PCA lên không gian đặc trưng này, tức là ta đi tìm các trị riêng  $\lambda_i$  và vector riêng  $\mathbf{v}_i$  của  $\hat{\mathbf{S}}$ :

$$\hat{\mathbf{S}} \mathbf{v}_i = \lambda_i \mathbf{v}_i$$

sao cho  $\lambda_i$  lớn nhất.



Thay định nghĩa của ma trận  $\hat{\mathbf{S}}$  vào, ta được

$$\frac{1}{N} \sum_{i=1}^N \Phi(\mathbf{x}_i) [\Phi(\mathbf{x}_i)^T \mathbf{v}_i] = \lambda_i \mathbf{v}_i.$$

Ta có  $\mathbf{v}_i$  thuộc  $\text{span} \{\Phi(\mathbf{x}_1), \Phi(\mathbf{x}_2), \dots, \Phi(\mathbf{x}_N)\}$  nên có thể viết

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

Thay  $\mathbf{v}_i$  vào và nhân bên trái hai vế cho  $\Phi(\mathbf{x}_l)^T$ , ta được

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



Nhắc lại phương trình cần giải là

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

Trong phương trình này, các vector  $\Phi(\mathbf{x}_i)$  chỉ xuất hiện dưới dạng tích vô hướng. Do đó ta không cần phải thực hiện ánh xạ  $\Phi$  một cách tường minh, mà chỉ cần tính các tích vô hướng như  $\Phi(\mathbf{x}_l)^T \Phi(\mathbf{x}_n)$  một cách hiệu quả.



Sử dụng phép đặt hàm kernel  $k(\mathbf{x}_m, \mathbf{x}_n) = \Phi(\mathbf{x}_m)^T \Phi(\mathbf{x}_n)$  và ma trận kernel cỡ  $N \times N$ :

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

ta cần giải phương trình trị riêng

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

trong đó  $\mathbf{a}_i$  là vector cột có các phần tử là  $a_{ni}$  với  $n = 1, 2, \dots, N$ .



Đối với mỗi hàm kernel  $k(\mathbf{x}, \mathbf{y})$ , luôn tồn tại ánh xạ  $\Phi$  sao cho tích vô hướng của hai phần tử bất kỳ ở không gian đặc trưng  $F$  đều được tính theo công thức của hàm kernel ta chọn.

Các hàm kernel thông thường được sử dụng:

- 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:  $k(\mathbf{x}, \mathbf{y}) = \tanh(\kappa(\mathbf{x} \cdot \mathbf{y}) + \Theta)$ .



Thực tế,  $\sum_{i=1}^N \Phi(\mathbf{x}_i) \neq 0$  trong một số trường hợp. Do đó ta cần "tịnh tiến" mỗi điểm dữ liệu sao cho trung bình cộng của chúng bằng 0, tức là

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

Ma trận  $K$  ban đầu ta dùng để giải phương trình trị riêng trở thành

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

trong đó  $\mathbf{1}_N$  là ma trận cỡ  $N \times N$  và có mọi phần tử bằng  $\frac{1}{N}$ .

Phương trình trị riêng lúc này cần giải được rút gọn thành

$$\hat{\mathbf{K}} \mathbf{a}_i = \lambda_i \mathbf{a}_i.$$



**Bước 1.** Chọn một hàm kernel  $k(\mathbf{x}, \mathbf{y})$ .

**Bước 2.** Tính các phần tử của ma trận kernel  $\mathbf{K}$ .

**Bước 3.** Trọng tâm hóa lại ma trận  $\mathbf{K}$  thành

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

**Bước 4.** Giải bài toán trị riêng  $\hat{\mathbf{K}} \mathbf{a}_i = \lambda_i \mathbf{a}_i$ .

**Bước 5.** Ứng với mỗi thành phần chính  $\lambda_i$ , hình chiếu của điểm dữ liệu  $\mathbf{x}$  lên vector  $\mathbf{v}_i$  là

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



- PCA áp dụng để tìm trị riêng của ma trận hiệp phương sai  $\mathbf{S}$  có kích cỡ  $D \times D$ , còn Kernel PCA tìm trị riêng của ma trận kernel  $\hat{\mathbf{K}}$  có kích cỡ  $N \times N$ .
- Ta cần tìm trị riêng của ma trận  $\hat{\mathbf{K}}$  có kích cỡ phụ thuộc vào số điểm dữ liệu ban đầu  $N$ . Do đó đối với các tập dữ liệu lớn người ta thường xấp xỉ.

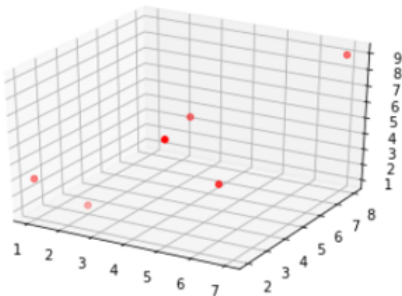
# Thực hành bằng Python

---



```
A = array([[1,2,3],  
A
```

```
array([[1, 2, 3],  
       [4, 5, 6],  
       [7, 8, 9],  
       [2, 3, 1],  
       [6, 3, 4],  
       [5, 2, 7]])
```



Tạo dữ liệu  $A$  như trên. Áp dụng PCA giảm còn 2 chiều.



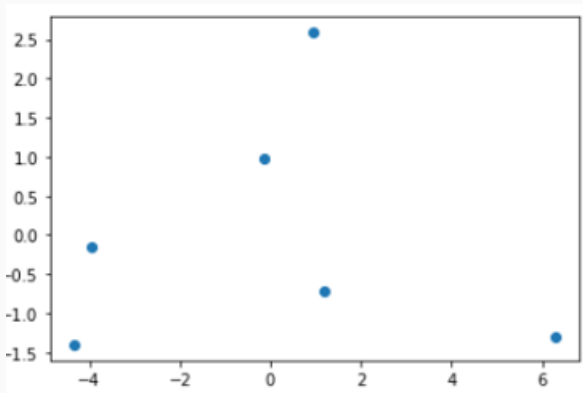
```
n = 2
M = mean(A, axis=0)
C = A - M
V = cov(C.T)
eigen_values, eigen_vectors = eigh(V)
eigen_vectors = eigen_vectors.T[:, -1]
S = eigen_vectors[:n]
P_1 = S.dot(C.T).T
P_1
```

```
array([[ 3.94781377, -0.14848076],
       [-1.17675134, -0.72218272],
       [-6.30131644, -1.29588469],
       [ 4.34621662, -1.40117416],
       [ 0.14231865,  0.98060456],
       [-0.95828127,  2.58711777]])
```

```
pca = PCA(2)
pca.fit(A)
P_2 = pca.transform(A)
P_2
```

```
array([[ -3.94781377, -0.14848076],
       [ 1.17675134, -0.72218272],
       [ 6.30131644, -1.29588469],
       [-4.34621662, -1.40117416],
       [-0.14231865,  0.98060456],
       [ 0.95828127,  2.58711777]])
```

Thuật toán PCA: tự xây dựng và sử dụng của sklearn.

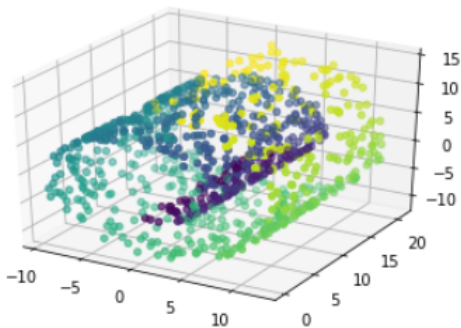


Trực quan hóa dữ liệu sau khi thực hiện PCA.



```
Shape: (1000, 3)
```

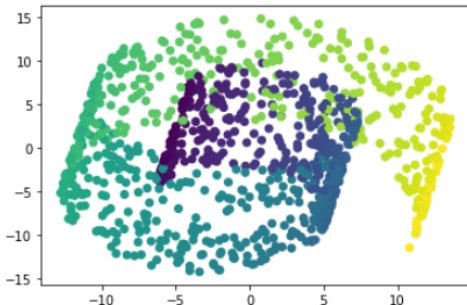
```
<matplotlib.pyplot.art3d.Path3DCollection at 0x2481426a3c8>
```



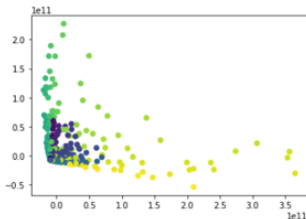
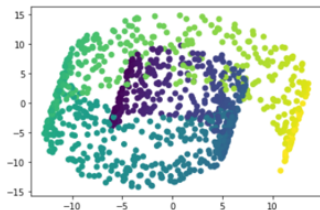
Hình ảnh dữ liệu Swiss Roll ban đầu.



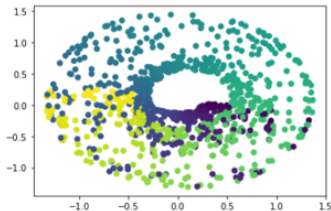
```
pca = PCA(2)
pca.fit(X)
P_3 = pca.transform(X)
data2 = pd.DataFrame(P_3)
data2.head()
plt.scatter(data2[0], data2[1], c = color)
plt.show()
```



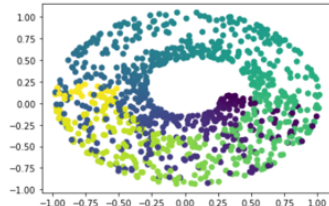
Hình ảnh dữ liệu Swiss Roll sau khi sử dụng PCA.



Sử dụng kernel linear



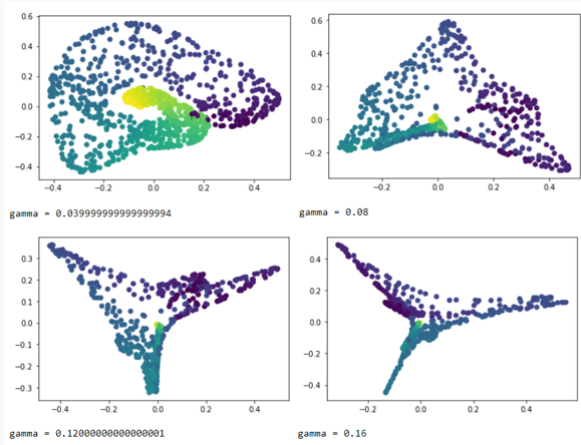
Sử dụng kernel poly



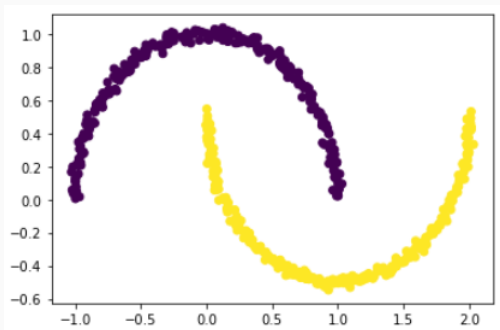
Sử dụng kernel sigmoid

Sử dụng kernel cosine

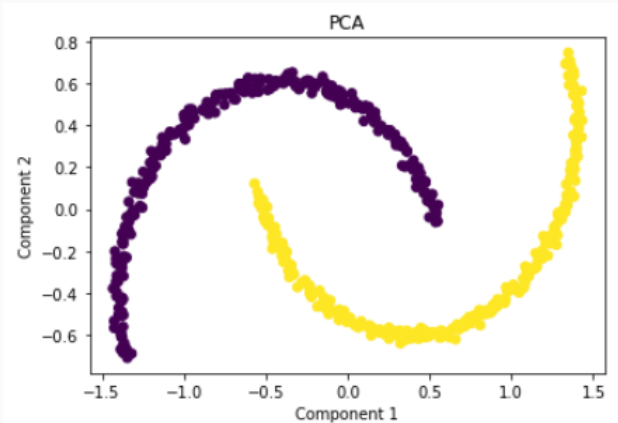
Áp dụng Kernel PCA cho Swiss Roll.



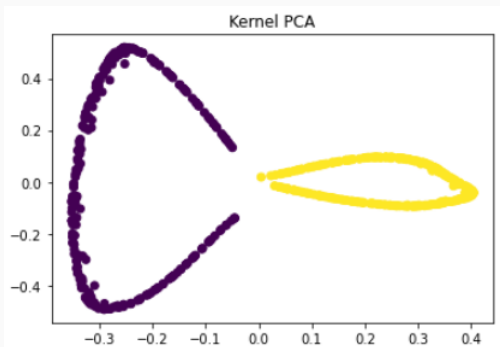
Áp dụng Kernel PCA cho Swiss Roll (tt). Sử dụng kernel = 'rbf' với các thông số  $\gamma$  khác nhau.



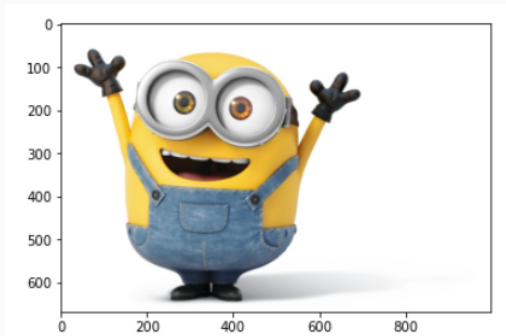
Dữ liệu ban đầu.



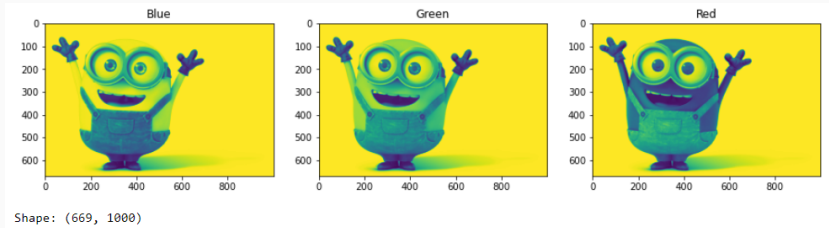
Áp dụng PCA thông thường lên dữ liệu.



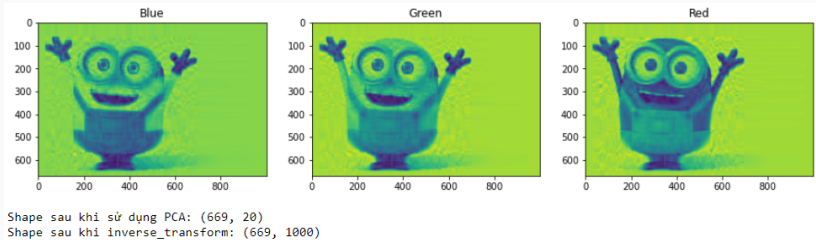
Áp dụng Kernel PCA lên dữ liệu:  $\text{kernel} = \text{rbf}$ ,  $\gamma = 15$ .



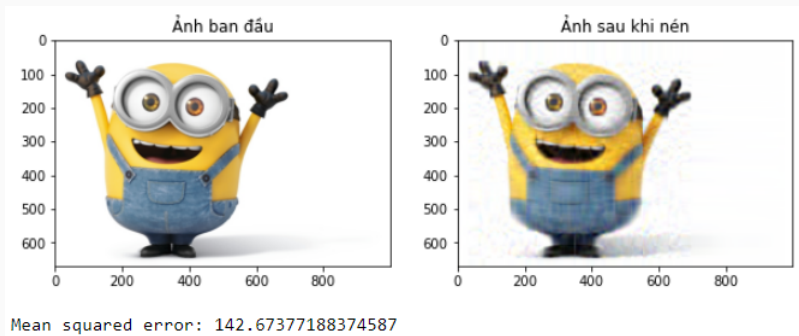
Ảnh bởi Ryan Nakamura.



Sử dụng cv2 để chia thành 3 kênh màu.

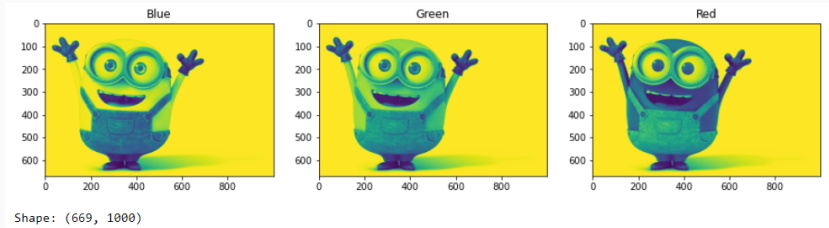


Áp dụng PCA giảm chiều ma trận của từng kênh.



Gộp lại các kênh sau khi sử dụng PCA.

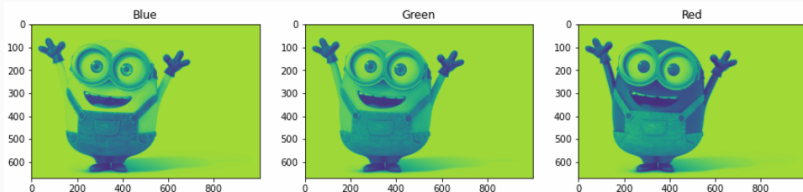
Compression rate:  $\frac{20}{1000} = 2\%$ .



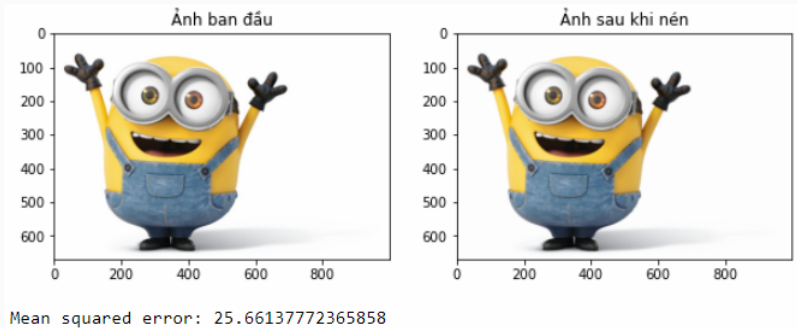
Sử dụng cv2 để chia thành 3 kênh màu.



Patch-based.



Áp dụng PCA giảm chiều ma trận của từng patch.



Merge các kênh lại tạo hình ảnh đã nén.

Compression rate:  $\frac{5 \cdot 22300}{669 \cdot 1000} = 16.7\%$ .



Christopher M. Bishop.

**Pattern Recognition and Machine Learning.**

Springer, 2006.



Tiep Vu.

**Principal Component Analysis.**

<https://machinelearningcoban.com/2017/06/15/pca/>,  
2017.