ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

Đồ án cuối kỳ:

KERNEL PRINCIPAL COMPONENT ANALYSIS

Môn học: Lập trình Python cho Máy học

Mã môn: CS116.M12.KHCL

Sinh viên thực hiện:

- Thái Trần Khánh Nguyên - 19520188
- Nguyễn Khánh Như - 19520209
- Lê Văn Trí - 19521043

Tp. Hồ Chí Minh, tháng 12 năm 2021

Mục lục

1. Giới thiệu chung:

Khi giải quyết những vấn đề về Máy học, việc biểu diễn dữ liệu phù hợp có tác động rất lớn đến hiệu quả của các mô hình. Trong thực tế, các features vector có thể có số chiều rất lớn, từ vài trăm đến vài nghìn. Không những vậy, số lượng của các điểm dữ liệu cũng thường không nhỏ. Nếu chúng ta không có một cách biểu diễn dữ liệu hợp lý sẽ có những tác động xấu đến mô hình của mình, gây những khó khăn trong việc lưu trữ hay tốc độ tính toán. Chính vì thế, Giảm chiều Dữ liệu (Dimensionality Reduction) là một trong những kỹ thuật rất quan trọng trong Máy học.

Một trong những phương pháp giảm số chiều dữ liệu hiện nay khá phổ biến chính là Principal Component Analysis (PCA – Phân tích thành phần chính). Phương pháp này dựa trên việc quan sát và phân tích dữ liệu để tìm ra các thành phần quan trọng của dữ liệu và giữ lại cá thành phần đó. Việc làm này sẽ giúp cho dữ liệu ban đầu của chúng ta từ N features thành K features (K < N).

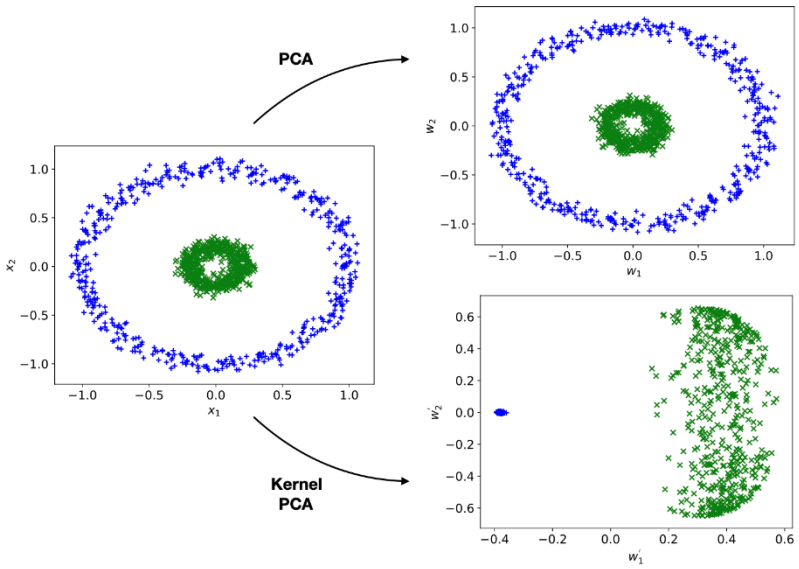

Figure-1: So sánh giữa PCA và Kernel PCA

Tuy nhiên phương pháp này còn vài hạn chế đó là PCA cố gắng tìm ra một không gian con tuyến tính mà dữ liệu được giới hạn. Vậy trong trường hợp dữ liệu đó phân bố theo dạng phi tuyến như hình Figure-1 bên trên thì sao? Bằng mắt thường, chúng ta có thể tách dữ liệu này thành hai vùng rõ rệt được nhưng rõ ràng là sẽ không thể sử dụng một hàm tuyến tính nào để có thể tính toán được nó. Chính vì thế, một phương pháp khác được ra đời để có thể giải quyết những vấn đề này được biết với tên gọi **Kernel PCA** (non-linear extension of PCA).

Kernel PCA sử dụng một hàm kernel để chiếu tập dữ liệu vào một không gian con đặc trưng có số chiều cao hơn, nơi nó có thể phân tách về dạng tuyến tính. Nó tương tự với ý tưởng Support Vector Machine. Ở những phần sau, nhóm em sẽ đi vào giải thích chi tiết cũng như áp dụng vào những bài toán thực tế để hiểu được bản chất cũng như những ưu điểm của nó.

2. Kernel PCA:

2.1 Ý tưởng Kernel PCA:

Trước tiên, để có thể nắm được phương pháp Kernel PCA, ta cần hiểu được bản chất của PCA trước bởi Kernel PCA được xây dựng dựa trên phương pháp PCA những có những cải tiến để giải quyết những vấn đề mà PCA không làm được.

PCA là một phương pháp giảm kích thước tuyến tính và trích xuất đặc trưng cho dữ liệu High-Dimensional (nhiều chiều hay nhiều features). Nó ánh xạ dữ liệu ban đầu từ không gian N chiều sang không gian con K chiều (K < N), trích xuất vector đặc trưng chính của dữ liệu đầu vào và đạt được mục đích phân tích dữ liệu gốc với thành phần chính. Hay một cách hiểu đơn giản thì PCA chính là một phương pháp đi tìm một hệ cơ sở mới có số chiều ít hơn so với ban đầu sao cho thông tin vẫn giữ được những thông tin quan trọng của dữ liệu.

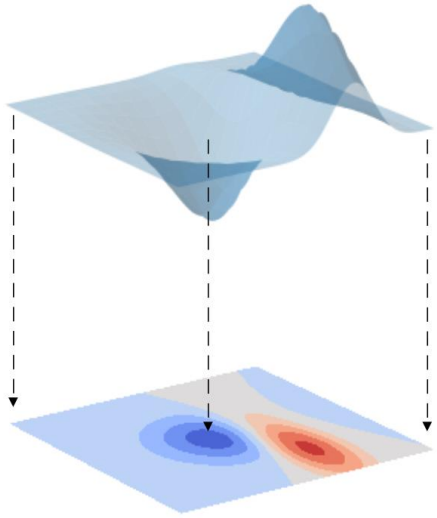

Figure-2: Minh họa phương pháp PCA

Phương pháp PCA được sử dụng khá phổ biến và tương đối hiệu quả để biến đổi từ dữ liệu có số lượng thuộc tính lớn và nhiễu nhưng có độ tương quan với nhau thành một bộ dữ liệu có số chiều nhỏ hơn dựa trên các phép biến đổi tuyến tính. Tuy nhiên trong nhiều ứng dụng thực tế, hiệu quả của phương pháp này rất hạn chế vì nền tảng xây dựng thuật toán dựa trên dữ liệu tuyến tính bởi những bộ dữ liệu không chỉ gói gọn trong những hàm tuyến tính mà còn có cả những hàm phi tuyến.

Để có thể áp dụng thuật toán này vào dữ liệu phi tuyến, đã có nhiều nghiên cứu ứng dụng các kỹ thuật khác nhau để có thể biến đổi dữ liệu đã cho thành dữ liệu được cho là tuyến tính. Nghiên cứu của Kramer vào năm 1991 đã tìm cách phát triển thuật toán PCA phi tuyến dựa trên mạng nơ ron. Tuy nhiên mạng này tương đối phức tạp và rất khó tìm được giá trị tối ưu do có 5 lớp. Nghiên cứu của Dong và McAvoy cũng sử dụng mạng nơ ron với giả thiết rằng sự phi tuyến của dữ liệu đầu vào có thể tương ứng với tổ hợp tuyến tính của một số đại lượng ngẫu nhiên và vì vậy có thể tách thành tổng các hàm của các đại lượng đó. Cách thức chuyển đổi đó chỉ có thể thực hiện được với một số rất hạn chế các bài toán phi tuyến. Trong khoảng những năm cuối của thế kỷ trước, một

phương pháp PCA phi tuyến mới đã được xây dựng và phát triển, có tên là Kernel PCA (PCA dựa trên hàm nhân) bởi Scholkopf và đồng nghiệp của ông.

Trong Kernel PCA, thay vì chỉ thực hiện các phép biến đổi tuyến tính trong các chiều ban đầu thì phương pháp này sẽ tìm cách dùng các hàm phi tuyến để ánh xạ vào các chiều dữ liệu rộng hơn nơi mà có thể thực hiện PCA một cách hiệu quả hơn.

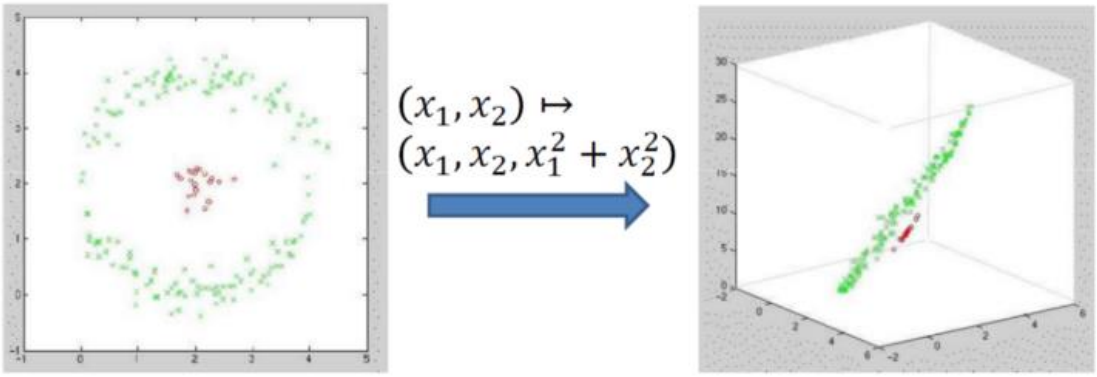

Figure 3: Minh họa phương pháp Kernel PCA

Như trong ví dụ này, có thể thấy nếu chỉ quan sát trên 2 chiều không gian thì rất khó cho phương pháp PCA hoạt động một cách hiệu quả. Nhưng nếu mở rộng chiều không gian lên thì điều đó lại hoàn toàn khác. Đối với hình bên phải gần như không có một đường thẳng nào có thể phân tách tốt hai loại dữ liệu (điểm màu đỏ và điểm màu xanh). Còn hình bên phải, không khó để có thể nhận ra có một mặt phẳng có thể phân tách tốt hai loại dữ liệu này và đó chính là ý tưởng cốt lõi của Kernel PCA.

2.2 Phân tích toán học:

Trong phần này, chúng ta sẽ đi sâu vào tìm hiểu về phương pháp Kernel PCA cũng như cách hoạt động của nó. Kernel PCA tìm cách giải quyết những vấn đề hạn chế của thuật toán PCA cổ điển bằng cách tổng quát hóa thành giảm kích thước chiều không gian phi tuyến. Bằng cách biến đổi không gian đặc trưng thành một không gian có chiều cao hơn, nơi có thể nắm bắt một cách hiệu quả một không gian con làm giảm kích thước của không gian đặc trưng ban đầu.

Trước khi đi vào những công thức toán phức tạp để hiểu rõ thuật toán này, nhóm em sẽ định nghĩa lại bài toán cũng như những ký hiệu toán học sẽ sử dụng trong bài báo cáo này. Giả sử ta có bộ dữ liệu $x_i \in \mathbb{R}^D, \forall\, i = 1, 2, \ldots, n$. Mục tiêu của chúng ta sẽ ánh xạ dữ liệu này vào không gian con có k chiều ($k \ll D$). Và phép chiếu này sẽ được biểu diễn là $\hat{x} = \mathbf{Ax}$, trong đó ma trận $\mathbf{A} = [\mathbf{u}_i, \ldots, \mathbf{u}_k]^T$ $\boldsymbol{v}$à $\mathbf{u}_i^T\mathbf{u}_i = \mathbf{1}\ \forall \boldsymbol{i} = \mathbf{1}, \ldots, \boldsymbol{k}$.

Để xây dựng phương pháp Kernel PCA, nhóm tác giả cũng tiếp cận giống với phương pháp cổ điển, tuy nhiên có điểm khác biệt là họ giả sử không gian đặc trưng mới có giá trị trung bình bằng không:

$$\frac{1}{N}\sum_{i=1}^{N}\phi(x_i) = 0 \qquad (1)$$

Sau đó, tính ma trận hiệp phương sai (covariance matrix):

$$\sum = \frac{1}{N}\sum_{i=1}^{N}\phi(x_i)\phi(x_i)^T \qquad (2)$$

Giá trị riêng và tương quan vector riêng:

$$\sum \boldsymbol{u}_i = \lambda_k \boldsymbol{u}_i \qquad \forall i = 1,..,k \qquad (3)$$

Thế phương trình (2) vào (3) ta được:

$$\frac{1}{N}\sum_{i=1}^{N}\phi(x_i)\{\phi(x_i)^T \boldsymbol{u}_k\} = \lambda_k \mathbf{u}_k \qquad (4)$$

Đặt: $a_{ki} = \frac{1}{\lambda_k N}\phi(x_i)^T \mathbf{u_k}$ suy ra:

$$\mathbf{u_k} = \sum_{i=1}^{N} a_{ki}\phi(x_i) \qquad (5)$$

Dựa theo lý thuyết về kernel của Mercer, ta biết rằng kernel k có dạng:

$$\kappa(\mathbf{x}_\mathrm{i}, \mathbf{x}_\mathrm{j}) = \phi(x_i)^T\phi(x_j) \qquad (6)$$

Chuyển phương trình (5) vào phương trình và nhân cả hai vế với $\varphi(x_l)$:

$$\frac{1}{N}\sum_{i=1}^{N}\phi(x_l)\phi(x_i)\sum_{j=1}^{N}a_{kj}\phi(x_i)\phi(x_j) = \lambda_k a_{ki}\phi(x_l)\phi(x_i) \qquad (7)$$

Chúng ta có thể viết lại phương trình (7) thành:

$$\frac{1}{N}\kappa(x_l, x_i)\sum_{i=1}^{N}a_{kj}\kappa(x_i, x_j) = \lambda_k \sum_{i=1}^{N}\kappa(x_l, x_i) \qquad (8)$$

$$\mathbf{K}^2\boldsymbol{a_k} = \lambda_k N \boldsymbol{a_k} \qquad (9)$$

với $\mathbf{K} = \kappa(x_i, x_j)$.

Kết quả của chuyển đổi Kernel PCA là:

$$\hat{x} = \phi(\mathrm{x})^T\mathbf{u}_k = \sum_{i=1}^{N}a_{ki}\kappa(x, x_i) \qquad (9)$$

Ta có:

$$\widehat{\mathbf{K}} = \left\| \phi(x_i) - \frac{1}{N}\sum_{j=1}^{N}\phi(x_j) \right\| = \left(\phi(x_i) - \frac{1}{N}\sum_{j=1}^{N}\phi(x_j) \right)^T \left(\phi(x_i) - \frac{1}{N}\sum_{j=1}^{N}\phi(x_j) \right)$$

Sau khi mở rộng chúng ta có được phương trình:

$$= K_{ij} - \sum_{i=1}^{N}\phi(x_i)^T\phi(x_j)\frac{1}{N} - \frac{1}{N}\sum_{i=1}^{N}\phi(x_j)^T\phi(x_i) - \frac{1}{N}\sum_{i=1}^{N}\phi(x_j)\phi(x_j)$$

Phương trình này thường được viết gọn thành dạng:

$$\widehat{\mathbf{K}} = \mathbf{K} - 1_{1/N}\mathbf{K} - \mathbf{K}1_{1/N} + 1_{1/N}\mathbf{K}1_{1/N}$$

với $\mathbf{K} = \mathbf{K}_{ij}$. $\widehat{\mathbf{K}}$ được gọi là Gram matrix (normalized kernel matrix).

Thuật toán Kernel PCA

Input: $\phi(x) \in \kappa(x_i, x_j)$ với $\phi(x) \in \mathbb{R}^D$. $\mathbf{K} = \kappa(x_i, x_j)$.

Output: $\hat{x} \in \mathbb{R}^k$ với $k \ll D$.

begin

1. Chọn kernel κ;
2. Xây dựng ma trận Gram: $\widehat{\mathbf{K}} = \mathbf{K} - 1_{1/N}\mathbf{K} - \mathbf{K}1_{1/N} + 1_{1/N}\mathbf{K}1_{1/N}$
3. Giải vấn đề eigen: $\mathbf{K}\mathbf{u_k} = \lambda\mathbf{u}_k$
4. Ánh xạ sang chiều không gian con mới: $\hat{x} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{u_k}\kappa$

end

2.3 Một số dạng Kernel thường sử dụng:

Một số dạng Kernel thường được sử dụng:

- **Linear Kernel**: dạng này chính là PCA cổ điển mà chúng ta hay sử dụng và là kernel đơn giản dễ hiểu nhất trong số các loại kernel. Công thức của nó sẽ có dạng:

$$K(x_i, x_j) = \mathbf{x}_i \mathbf{x}_j^T$$

- **Polynomial Kernel:** là kernel dạng đa thức.

$$\mathrm{K}(\mathrm{x_i}, \mathrm{x_j}) = (\mathbf{x}_i \mathbf{x}_j^{\mathrm{T}} + \mathrm{c})^{\mathrm{d}}$$

  Với $c \geq 0$ và d là một số dương để chỉ bậc của đa thức. d có thể không là số tự nhiên vì mục đích chính của ta không phải là bậc của đa thức mà là cách tính kernel. Polynomial kernel có thể dùng để mô tả hầu hết các đa thức có bậc không vượt quá d nếu d là một số tự nhiên.

- **RBF (Radial Basis Function) Kernel:** Hay còn được gọi là Guassian Kernel. Đây là dạng được sử dụng phổ biến trong thực tế và thường là chế độ mặc định của nhiều framework hiện nay.

$$\mathrm{K}(\mathrm{x_i}, \mathrm{x_j}) = \exp(-\gamma \left\| \mathbf{x}_i - \mathbf{x}_j \right\|^2)$$

  Với $\gamma = \frac{1}{2\sigma^2}$

- **Sigmoid Kernel:** Dựa trên hàm số sigmoid thể hiện cho sự biến đổi giá trị giữa phạm vi 0 và 1.

$$\mathrm{K}(\mathrm{x_i}, \mathrm{x_j}) = \tanh(\gamma \mathbf{x}_i \mathbf{x}_j^{\mathrm{T}} + \mathrm{c}).$$

  Với $c \geq 0, \gamma = \frac{1}{2\sigma^2}$

- **Cosine Kernel:**

$$\mathrm{K}(\mathrm{x_i}, \mathrm{x_j}) = \frac{\mathbf{x}_i \mathbf{x}_j^{\mathrm{T}}}{\|\mathbf{x}_i\| \|\mathbf{x}_j\|}$$

Phía trên là một số hàm kernel thường hay được sử dụng để giải quyết các vấn đề về giảm chiều dữ liệu. Tùy bài toán và tính huống thực tế mà mỗi hàm sẽ cho những kết quả khác nhau. Ở đây nhóm em còn nhiều hạn chế là chưa thể rút ra được xem những trường hợp nào nên sử dụng hàm nào. Những thực nghiệm trong đồ án này tụi em chủ yếu sẽ thực hiện theo hướng thử và sai xem hàm nào cho kết quả tốt nhất.

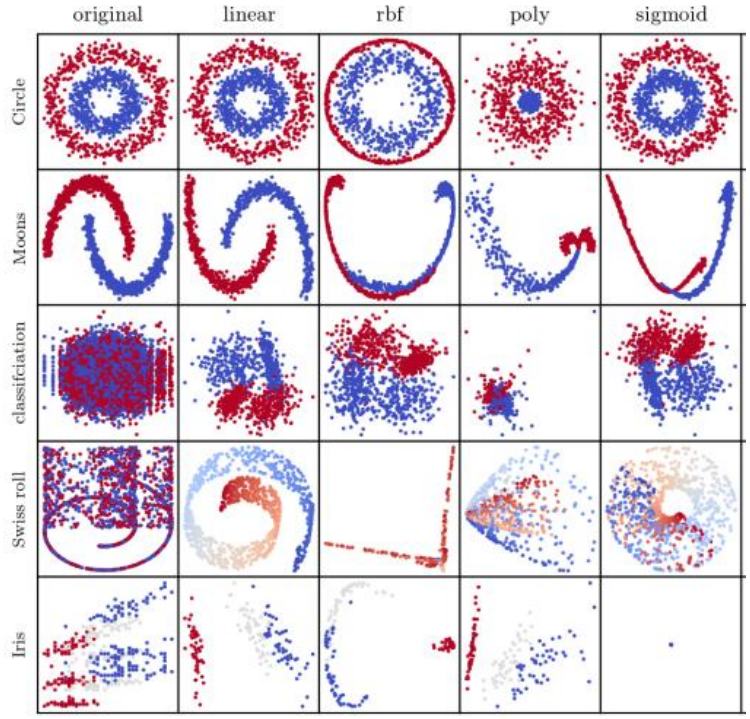

Figure-4: So sánh giữa các bộ Kernel

Figure-4 bên trên minh họa khi áp dụng phương pháp Kernel PCA với những bộ dữ liệu cổ điển với những hàm kernel khác nhau. Có thể thấy đối với từng bộ dữ liệu việc áp dụng các kernel khác nhau sẽ cho ra những kết quả rất khác biệt.

3. Ưu và nhược điểm so với các phương pháp khác:

3.1 Kernel PCA và PCA:

Từ đầu bài báo cáo đến giờ, nhóm em đã có những so sánh giữa Kernel PCA và tiền thân của nó là PCA. Trong phần này nhóm em sẽ tổng hợp lại những điểm khác nhau và ưu điểm của từng phương pháp.

Về bản chất Kernel PCA đã bao gồm luôn cả PCA (Khi sử dụng hàm Linear). Tuy nhiên, sẽ có một vài điểm khác biệt giữa hai phương pháp này:

- Cấu trúc Linear và Nonlinear: Kernel PCA có thể nắm bắt được cấu trúc phi tuyến trong dữ liệu, trong khi PCA không thể làm được điều này.
- Khả năng diễn giải (Interpretability): PCA cung cấp một tập hợp các trọng số xác định ánh xạ tuyến tính từ không gian ban đầu đến không gian con có chiều thấp hơn. Các trọng số này có thể cung cấp thông tin hữu ích về cấu trúc của dữ liệu. Trong khi đó, không tồn tại trọng số như vậy đối với Kernel PCA vì ánh xạ của nó là phi tuyến.
- Ánh xạ nghịch đảo (Inverse mapping): Đối với mặt này thì, PCA lại chiếm lợi thế khi cung cấp một ánh xạ nghịch đảo từ không gian chiều thấp trở lại không gian ban đầu. Vì vậy, các điểm dữ liệu đầu vào có thể được tái tạo gần giống với bản gốc của dữ liệu. Kernel PCA vốn dĩ không cung cấp một ánh xạ nghịch đảo, mặc dù có thể ước tính một ánh xạ bằng cách sử dụng các phương pháp bổ sung. Tuy nhiên, để làm được điều đó sẽ cần chi phí phức tạp hơn và nhiều tài nguyên tính toán hơn.
- Chi phí tính toán: PCA thường có yêu cầu về bộ nhớ và thời gian chạy thấp hơn kPCA và có thể được mở rộng thành các bộ dữ liệu lớn. Có nhiều chiến lược khác nhau để mở rộng Kernel PCA, nhưng điều này đòi hỏi phải tính toán gần đúng

Với những so sánh trên thì nhóm em nhận thấy khi gặp những bài toán phức tạp thì sử dụng Kernel PCA sẽ mang lại hiệu quả tối ưu hơn. Còn khi gặp những bài toán đơn giản, tuyến tính thì việc sử dụng Kernel PCA sẽ không thực sự mang lại hiệu quả bằng PCA.

3.2 Ưu và nhược điểm của Kernel PCA:

Sau quá trình tìm hiểu về Kernel PCA làm các thực nghiệm, và tìm hiểu thêm về một số phương pháp giảm chiều dữ liệu khác nhau để có góc nhìn khách quan để đánh giá ưu và nhược điểm của phương pháp này. Chúng em xin đưa ra những nhận xét như sau:

Ưu điểm:

- Trong nhiều bài toán phi tuyến phức tạp, Kernel PCA sử dụng một hàm số tạo ra một không gian đặc trưng nhiều chiều và trên không gian đặc trưng có thể phân lớp dữ liệu bằng một hàm tuyến tính và khả năng biểu diễn dữ liệu tốt tương đương với không gian cũ, tức vẫn đảm bảo độ biến thiên của dữ liệu trên mỗi chiều mới.
- Giảm thời gian tính toán cũng như giảm độ nhiễu của input đầu vào mang lại độ chính xác và hiệu quả cao trong việc xử lý các bài toán phi tuyến.

- Các trục tọa độ trong không gian mới là tổ hợp tuyến tính và phi tuyến của không gian cũ, do đó về mặt ngữ nghĩa, Kernel PCA vẫn biểu diễn rất tốt dữ liệu ban đầu.
- Trong không gian đặc trưng, các liên kết tiềm ẩn của dữ liệu có thể được phát hiện và thể hiện, nếu đặt trong không gian ban đầu thì khó phát hiện, hoặc không thể hiện rõ.

Nhược điểm:

- Cần xây dựng, lựa chọn hàm kernel phù hợp với từng bài toán cụ thể.

Tuy nhiên, không nên quá lạm dụng việc giảm chiều dữ liệu khi thật sự không cần thiết hoặc giảm chiều dữ liệu với cường độ quá cao. Cần phân tích dữ và đánh giá dữ liệu hợp lý xem nên chọn những phương pháp nào để xử lý hay giảm chiều dữ liệu. Việc lạm dụng quá mức sẽ không những không cải thiện được hiệu quả mà gây ra những tác động tiêu cực như giảm độ chính xác, tăng thời gian huấn luyện, …

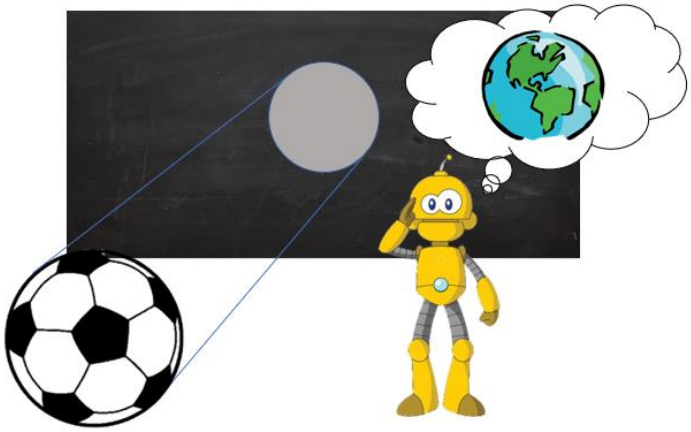

Figure-5

4. Các siêu tham số và cách hiệu chỉnh trong Kernel PCA:

4.1 Các siêu tham số trong Kernel PCA:

Đối với phần tham số trong phương pháp Kernel PCA, nhóm em sử dụng framework của sklearn. Tuy nhiên để có thể ứng dụng vào thực tế cần hiểu thêm về các siêu tham số do framework này cung cấp để có thể hiệu chỉnh theo từng bài toán. Sau đây nhóm em sẽ trình bày về những siêu tham số như tên gọi, cách sử dụng, …

```
from sklearn.decomposition import KernelPCA
transformer = KernelPCA(n_components=7, kernel='linear')
```

Parameters:

- **n_components: int, default=None**

  Số lượng các thành phần. Nếu không có, tất cả các thành phần khác 0 được giữ lại.

- **kernel: {'linear', 'poly', 'rbf', 'sigmz`oid', 'cosine', 'precomputed'}, default='linear'**

  Kernel được sử dụng cho PCA.

- **gamma: float, default=None**

  Kernel coefficient của các kernel rbf, poly và sigmoid. Bị bỏ qua bởi các kernel khác. Nếu gamma là None, thì nó được đặt thành 1/n_features.

- **degree: int, default=3**

  Bậc của poly kernels. Bị bỏ qua bởi các kernel khác.

- **coef0: float, default=1**

  Thuật ngữ độc lập trong kernel poly và sigmoid. Bị bỏ qua bởi các kernel khác.

- **kernel_params: dict, default=None**

  Các tham số và giá trị của kernel được truyền dưới dạng đối tượng có thể gọi. Bị bỏ qua bởi các kernel khác.

- **alpha: float, default=1.0**

  Siêu tham số học phép biến đổi nghịch đảo (khi fit_inverse_transform = True).

- **fit_inverse_transform: bool, default=False**

  Tìm hiểu phép biến đổi nghịch đảo cho các kernel không được tính toán trước (tức là học cách tìm pre-image của một điểm).

- **eigen_solver: {'auto', 'dense', 'arpack', 'randomized'}, default='auto'**

  Chọn eigensolver để sử dụng. Nếu n_components ít hơn nhiều so với số lượng mẫu huấn luyện, thì randomized (hoặc arpack với khoảng mở rộng nhỏ hơn) có thể hiệu quả hơn so với eigensolver dày đặc.

- **tol: float, default=0**

  Sự cho phép hội tụ cho arpack. Nếu 0, giá trị tối ưu sẽ được chọn bởi arpack.

- **max_iter: int, default=None**

  Số lần lặp lại tối đa cho arpack. Nếu không có, giá trị tối ưu sẽ được chọn bởi arpack.

- **iterated_power: int >= 0, or 'auto', default='auto'**

  Số lần lặp lại cho phương thức lũy thừa được tính toán bởi svd_solver == 'randomized'. Khi 'auto', nó được đặt thành 7 khi n_components < 0.1* min(X.shape), các trường hợp khác nó được đặt thành 4.

- **remove_zero_eig: bool, default=False**

  Nếu remove_zero_eig = True, thì tất cả các thành phần có giá trị riêng = 0 sẽ bị loại bỏ, do đó số lượng thành phần trong đầu ra có thể < n_components (và đôi khi thậm chí bằng 0 do numerical instability). Khi n_components = 0, tham số này bị bỏ qua và các thành phần không có giá trị riêng sẽ bị loại bỏ.

- **random_state: int, RandomState instance or None, default=None**

  Được sử dụng khi eigen_solver = 'arpack' hoặc 'randomized'. Chuyển một số nguyên cho các kết quả có thể lặp lại qua nhiều lệnh gọi hàm.

- **copy_X: bool, default=True**

  Nếu copy_X = True, đầu vào X được mô hình sao chép và lưu trữ trong thuộc tính X_fit_. Nếu không có thay đổi nào được thực hiện đối với X, việc cài đặt copy_X=False sẽ tiết kiệm bộ nhớ bằng cách lưu trữ một tham chiếu.

- **n_jobs: int, default=None**

  Số lượng công việc sẽ chạy song song. None nghĩa là 1 trừ khi trong joblib.parallel_backend context. -1 nghĩa là sử dụng tất cả các bộ xử lý.

Attributes:

- **eigenvalues_: ndarray of shape (n_components,)**

  Các giá trị riêng của kernel matrix ở giữa theo thứ tự giảm dần. Nếu n_components và remove_zero_eig không được đặt, thì tất cả các giá trị sẽ được lưu trữ.

- **lambdas_: ndarray of shape (n_components,)**
- **eigenvectors_: ndarray of shape (n_samples, n_components)**

  Vector riêng của kernel matrix ở giữa. Nếu n_components và remove_zero_eig không được đặt, thì tất cả các thành phần sẽ được lưu trữ.

- **alphas_: ndarray of shape (n_samples, n_components)**
- **dual_coef_: ndarray of shape (n_samples, n_features)**

  Ma trận biến đổi nghịch đảo. Chỉ khả dụng khi fit_inverse_transform = True.

- **X_transformed_fit_: ndarray of shape (n_samples, n_components)**

  Phép chiếu dữ liệu được điều chỉnh trên các thành phần chính của hạt nhân. Chỉ khả dụng khi fit_inverse_transform = True.

- **X_fit_: ndarray of shape (n_samples, n_features)**

  Dữ liệu được sử dụng để phù hợp với mô hình. Nếu copy_X=False, thì X_fit_là một tham chiếu. Thuộc tính này được sử dụng cho các lệnh gọi để chuyển đổi.

- **n_features_in_: int**

  Số lượng features được nhìn thấy trong quá trình điều chỉnh.

- **feature_names_in_: ndarray of shape (n_features_in_,)**

  Tên của các features được nhìn thấy trong quá trình điều chỉnh. Chỉ được xác định khi X có tên đối tượng là tất cả các chuỗi.

4.2 Cách hiệu chỉnh các siêu tham số:

Vì Kernel PCA là một thuật toán học tập không giám sát, không có thước đo hiệu suất rõ ràng nào để có thể giúp chúng ta trong việc chọn các giá trị kernel và siêu tham số tốt nhất. Việc giảm kích thước thường là một bước chuẩn cho cho nhiệm vụ huấn luyện một mô hình nào đó. Chính vì thế có một phương pháp có thể giúp ta tìm được bộ siêu tham số tối ưu cho từng bài toán bằng phương pháp grid search.

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

clf = Pipeline([ ("kpca", KernelPCA(n_components=2)),
                 ("log_reg", LogisticRegression())])

param_grid = [{"kpca__gamma": np.linspace(0.03, 0.05, 10),
               "kpca__kernel": ["rbf", "sigmoid"]}]

grid_search = GridSearchCV(clf, param_grid, cv=3)
grid_search.fit(X, y)
```

Một cách tiếp cận khác mà nhóm em tìm thấy đó chính là chọn ra kernel và các siêu tham số sao cho độ lỗi thấp nhất khi xây dựng lại bộ dữ liệu. Tuy nhiên việc này sẽ khó thực hiện hơn so với phương pháp Grid Search. Figure-5 cho thấy tập dữ liệu ban đầu (original space) và tập dữ liệu 2D kết quả sau khi Kernel PCA được áp dụng bằng cách sử dụng RBF kernel (Reduced space). Phép biến đổi này về mặt toán học tương đương với việc sử dụng feature map φ (bản đồ đặc trưng) để ánh xạ tập huấn luyện thành không gian đặc trưng vô hạn chiều (Feature space) sau đó chiếu tập huấn luyện đã biến đổi xuống 2D bằng PCA tuyến tính.

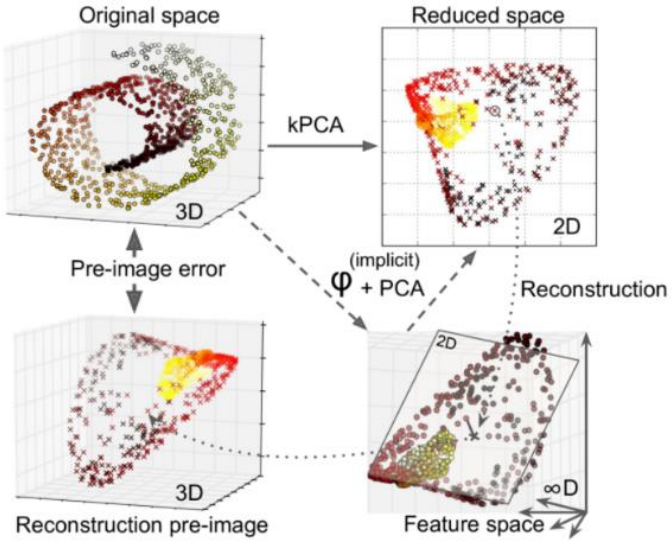

Figure-6: Reconstruction

Lưu ý rằng nếu chúng ta có thể đảo ngược bước PCA tuyến tính cho một trường hợp nhất định trong không gian thu nhỏ, thì điểm được tái tạo sẽ nằm trong không gian đặc trưng, không phải trong không gian ban đầu (ví dụ: như điểm được biểu thị bằng dấu X trong biểu đồ). Vì không gian đặc trưng là vô hạn chiều, chúng ta không thể tính điểm tái tạo và do đó chúng ta không thể tính sai số tái tạo thực sự. May mắn thay, có thể tìm thấy một điểm trong không gian ban đầu có thể lập bản đồ gần với điểm được tái tạo. Điểm này được gọi là hình ảnh tiền tái tạo. Khi có hình ảnh trước này, có thể đo khoảng cách bình phương của nó đến phiên bản gốc. Sau đó, có thể chọn hạt nhân và siêu tham số để giảm thiểu lỗi hình ảnh trước khi tái tạo này.

5. Thực nghiệm:

5.1 Sign Language Digits:

Trong phần thực nghiệm đối với đồ án này, nhóm em đã thực hiện trên bộ dữ liệu Sign Language Digits (Chữ số trong ngôn ngữ ký hiệu tay). Đây là một bộ dữ liệu khá kinh điển khi làm về các bài toán classification.

a. Mô tả chi tiết:

Sign Language (Ngôn ngữ ký hiệu) là ngôn ngữ sử dụng giao tiếp thủ công để truyền đạt ý nghĩa. Điều này có thể bao gồm sử dụng đồng thời cử chỉ tay, chuyển động, định hướng của ngón tay, cánh tay hoặc cơ thể và nét mặt để truyền đạt ý tưởng của người nói. Tuy nhiên, đối với bộ dữ liệu này chỉ xoay quanh việc nhận dạng các con số được diễn tả bằng ngôn ngữ ký hiệu.

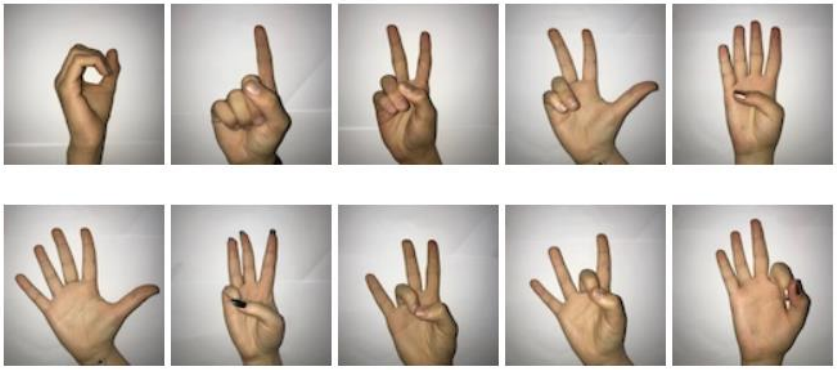

Figure-7

Các thông số liên quan:

- Nguồn bộ dữ liệu: [Sign Language Digits Dataset | Kaggle]
- Kích thước ảnh: 64 x 64
- Color space: Grayscale
- Định dạng file: npy
- Số lượng classes: 10 (Các số từ 0-9)
- Số lượng hình ảnh: 2062

b. Thực nghiệm:

Để có thể đưa vào các mô hình máy học thì nhóm em sẽ flatten dữ liệu ảnh thành các vector đặc trưng. Sau đó chia dữ liệu thành các tập train và tập val.

```
1  X_train_flatten = X_train.reshape(number_of_train,X_train.shape[1]*X_train.shape[2])
2  X_test_flatten = X_test .reshape(number_of_test,X_test.shape[1]*X_test.shape[2])
3  print("X train flatten",X_train_flatten.shape)
4  print("X test flatten",X_test_flatten.shape)
```

```
X train flatten (1649, 4096)
X test flatten (413, 4096)
```

Ngoài ra, sau khi quan sát dữ liệu nhóm em nhận thấy labels của các hình ảnh được gán nhãn theo dạng one-hot vector. Để thuận tiện trong quá trình huấn luyện thì nhóm em sẽ chuyển nhãn về theo đúng số thay vì để như ban đầu.

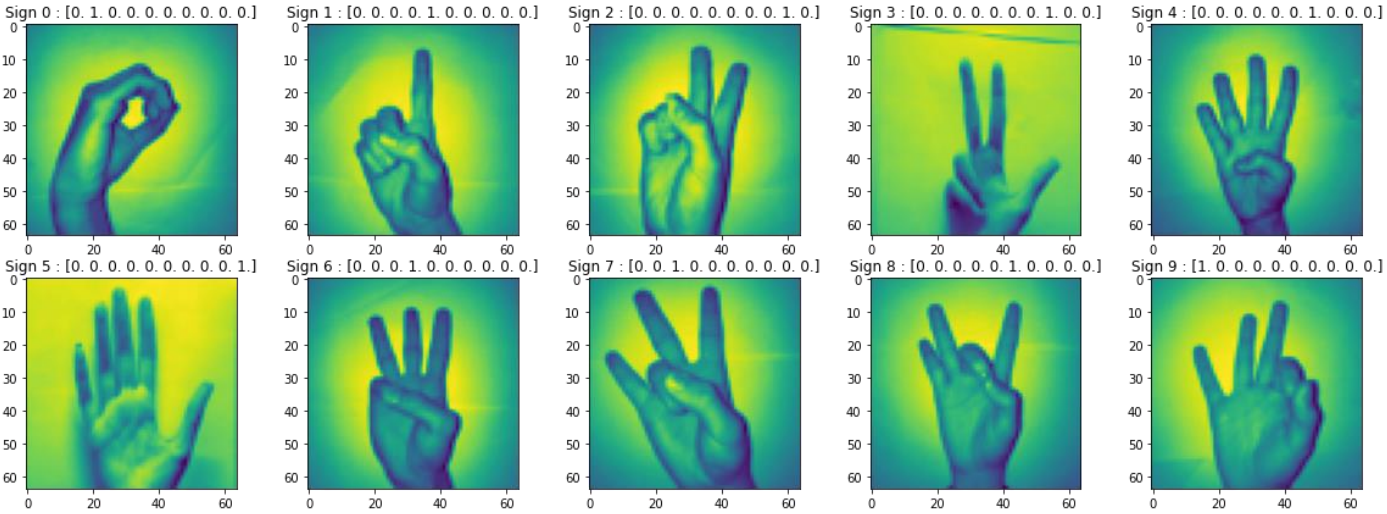

Figure-8

Lý do nhóm chúng em suy nghĩ đến hướng sử dụng Kernel PCA cho bài toán này bởi vì như trong bức hình minh họa không thật sự sát với bàn tay, có những khoảng trống thừa. Ngoài ra Thứ thật sự cần quan tâm là các ngón tay nên nhưng những chi tiết thừa như bàn tay hay cổ tay có thể giảm bớt được.

Grid Search: Để điều chỉnh các tham số trong quá trình thực nghiệm, nhóm em sử dụng grid search để lưu lại các kết quả điều chỉnh khác nhau và tìm xem với bộ siêu tham số nào sẽ cho ra kết quả tối ưu nhất. Trong phần 4 mà nhóm em đã trình bày trước đó, có rất nhiều các siêu tham số khác nhau để chúng ta có thể tinh chỉnh. Nên việc tinh chỉnh hết toàn bộ các siêu tham số trên sẽ mất rất nhiều thời gian và có một số tham số không quan trọng hoặc để cho phù hợp với người sử dụng chứ không ảnh hưởng nhiều đến kết quả. Chính vì thể trong phần này nhóm em sẽ tập trung điều chỉnh các tham số quan trọng đối với Kernel PCA. Cụ thể là: kernel, n_components, gamma, degree (đối với kernel = poly). Ngoài ra, để có thể đánh giá khách quan thì nhóm em sử dụng hai model là Logistic Regression và Support Vector Machine để huấn luyện.

```
clf_LR = Pipeline([
                ("kpca", KernelPCA()),
                ("log_reg", LogisticRegression(solver='lbfgs', max_iter=5000))
                ])
n_components = [100, 300, 500, 700, 1000, 1500, 2000]
param_grid_LR = [{"kpca__kernel": ["linear"], "kpca__n_components": n_components },

              {"kpca__kernel": ["rbf"], "kpca__gamma": np.linspace(0.03, 0.05, 10, 15), "kpca__n_components": n_components },

              {"kpca__kernel": ["sigmoid"], "kpca__gamma": np.linspace(0.03, 0.05, 10, 15), "kpca__n_components": n_components },

              { 'kpca__kernel': ['poly'], 'kpca__degree': [1,2,3,5],
               "kpca__gamma": np.linspace(0.03, 0.05, 10, 15), "kpca__n_components": n_components }]
```

c. Kết quả thực nghiệm và nhận xét:

Kết quả thực nghiệm nhóm em sẽ tổng hợp lại các số liệu trong quá trình thực nghiệm, tuy nhiên dữ liệu khá nhiều nên nhóm em sẽ liệt kê các thông tin trọng tâm. Ở đây nhóm em chủ yếu sẽ đưa ra những kết quả tốt nhất đối với từng kernel khác nhau.

Model	Kernel	Components	Gamma	Degree	Accuracy	Fit time
Logistic Regression	None	None	None	None	0.7627	173.6045
Logistic Regression	linear	2000	None	None	0.7459	3.3774
Logistic Regression	rbf	1500	0.03	None	0.7337	1.9063
Logistic Regression	sigmoid	100	0.03	None	0.1055	0.2730
Logistic Regression	poly	1500	0.04778	1	0.7695	1.6749
SVM	None	None	None	None	0.8474	5.1934
SVM	linear	100	None	None	0.8574	0.7326
SVM	rbf	2000	0.03	None	0.8351	4.2805
SVM	sigmoid	100	0.03	None	0.1008	0.6792
SVM	poly	100	0.04111	1	0.8574	0.7282

Dựa vào bảng kết quả bên trên, nhóm em có một số kết luận đối với thực nghiệm trên bộ dữ liệu này. Mặc dù khi áp dụng Kernel PCA độ chính xác tăng khá ít so với việc không sử dụng (từ 0.5% đến 1%). Tuy nhiên về thời gian huấn luyện thì có sự chệnh lệch khá rõ rệt khi tổng thời gian áp dụng KPCA và thời gian huấn luyện mô hình giảm rất nhiều so với việc không áp dụng (Logistic Regression: giảm từ 173s đến 1.674s, SVM: 5.19s còn 0.7282s). Và về mặc bộ nhớ, đối với kết quả tốt nhất thì số features của vector đặc trưng là 100 -ít hơn gần 40 lần. Điều này cho thấy mặc dù chưa cải thiện nhiều về độ chính xác nhưng KPCA lại cho hiệu suất tốt hơn hẳn.

Kết luận:

- Độ chính xác cao nhất: 0.8574
- Thời gian: 0.7282s
- Áp dụng Kernel PCA: có áp dụng
- Bộ siêu tham số:
  - kernel: poly
  - n_components: 100
  - gramma: 0.04111
  - degree: 1

5.2 Modified National Institute of Standards and Technology (MNIST):

Trong phần này, nhóm chúng em sẽ khai thác trên một nhánh của bộ dữ liệu MNIST. Một bộ dữ liệu kinh điển trong lĩnh vực Computer Vision. Bộ dữ liệu mà chúng em làm thực nghiệm là Nhận dạng chữ số viết tay (Handwritten digit Recognition). Một bộ dữ liệu quen thuộc cho những thực nghiệm các thuật toán máy học.

a. Mô tả chi tiết:

Bộ dữ liệu này gồm các hình ảnh chứa các con số được viết tay. Và nhiệm vụ đặt ra trong bài toán này là phân loại xem chữ số trong bức hình đó là chữ số mấy.

Figure-9

Thông tin chi tiết:

- Nguồn dữ liệu: Digit Recognizer | Kaggle
- Số lượng hình ảnh: 60000 ảnh cho mục đích train và 10000 ảnh cho mục đích test
- Kích thước hình ảnh: 28x28
- Số lượng classes: 10 classes (0-9)
- Color space: Grayscale

b. Thực nghiệm:

Trong phần thực nghiệm của bộ dữ liệu này, để cho thuận tiện nhóm em sử dụng thư viện của sklearn là fetch_openml để load dữ liệu đã được trải phẳng thành vector đặc trưng. Do hạn chế về tài nguyên phần cứng nên nhóm em không thể chạy hết 60000 hình ảnh trong quá trình huấn luyện. Chính vì thế, sau khi đánh về tài nguyên bộ nhớ và thử xem có thể huấn luyện tối đa bao nhiêu hình thì nhóm em nhận thấy có sử dụng được 15000 hình, còn về tập test thì vẫn sẽ giữ nguyên.

```
[ ]  1  from sklearn.preprocessing import StandardScaler
     2  scaler = StandardScaler()
     3  # Fit on training set only.
     4  scaler.fit(train_img)
     5  # Apply transform to both the training set and the test set.
     6  train_img = scaler.transform(train_img)
     7  test_img = scaler.transform(test_img)
```

Trước khi đi vào việc tìm ra các bộ siêu tham số, nhóm em chuẩn hóa dữ liệu đầu vào bằng thư viện StandardScaler do sklearn cung cấp.

Tương tự với phần thực nghiệm trước nhóm em sẽ sử dụng grid search để tìm ra bộ siêu tham số tốt nhất khi sử dụng Kernel PCA. Còn về mô hình dùng để huấn luyện nhóm em sẽ sử dụng Logistic Regression cho thực nghiệm đối này.

```
clf = Pipeline([
 ("kpca", KernelPCA()),
 ("log_reg", LogisticRegression())
 ])

param_grid = [
 {"kpca__kernel": ["linear"], "kpca__n_components": [300, 500, 700]},

 {"kpca__kernel": ["rbf"], "kpca__gamma": np.linspace(0.03, 0.05, 10), "kpca__n_components": [300, 500, 700]},

 {"kpca__kernel": ["sigmoid"], "kpca__gamma": np.linspace(0.03, 0.05, 10), "kpca__n_components": [300, 500, 700]},

 { "kpca__kernel": ["poly"], "kpca__degree": [1, 2, 3, 4, 5, 10],
  "kpca__gamma": np.linspace(0.03, 0.05, 10), "kpca__n_components": [300, 500, 700]}
 ]
```

c. Kết quả thực nghiệm và nhận xét:

Kết quả thực nghiệm, nhóm em lưu các kết quả thực nghiệm vào bảng để thuận tiện trong việc đánh giá phương pháp thực hiện. Bảng bên dưới là top 20 kết quả có độ chính xác tốt nhất mà nhóm em đã sắp xếp lại theo tiêu chí về độ chính xác.

kpca__kernel	kpca__n_components	kpca__gamma	kpca__degree	Accuracy	Fit_time
poly	700	0.032222222	2	0.9482	53.68501
poly	700	0.03	2	0.948133	52.41203
poly	700	0.034444444	2	0.948133	53.54691
poly	700	0.036666667	2	0.9476	47.55243
poly	700	0.043333333	2	0.9476	46.01911
poly	700	0.038888889	2	0.947533	45.51847
poly	700	0.041111111	2	0.9474	45.4452
poly	700	0.045555556	2	0.947067	45.69093
poly	700	0.047777778	2	0.947067	46.01312
poly	700	0.05	2	0.947	46.16082
poly	500	0.03	2	0.942467	46.09978
poly	500	0.032222222	2	0.942133	47.96661
poly	500	0.034444444	2	0.9418	49.81483
poly	500	0.036666667	2	0.941133	43.09482
poly	500	0.038888889	2	0.940867	43.00708
poly	500	0.045555556	2	0.940667	43.20221
poly	500	0.043333333	2	0.940533	44.27225
poly	500	0.041111111	2	0.940333	43.01343
poly	500	0.05	2	0.940067	43.37427

Với bảng số liệu này, chúng ta có thể thấy được bộ siêu tham số tốt nhất mà mô hình đạt được có kernel là poly với độ chính xác là 0.9482 cao hơn hẳn so với lúc chưa dùng Kernel PCA (0.8892). Logistic là một mô hình đơn giản, thông qua tiền xử lí sử dụng kPCA chúng ta đã giúp mô hình đạt được độ chính xác cao so với khi chỉ sử dụng PCA hay các tiền xử lí thông thường như Scale. Tuy nhiên kPCA khá tốn bộ nhớ và thời gian tìm, tinh chỉnh tham số. Đối với bộ dữ liệu nêu trên, kPCA mất trung bình khoảng 45s cho mỗi lần chạy.

Kết luận: Sử dụng Kernel PCA mang lại cải thiện rõ rệt về độ chính xác (tăng 6%) về độ chính xác đồng với về tốc độ huấn luyện.

- Độ chính xác tốt nhất: 0.9482
- Áp dụng Kernel PCA: có áp dụng
- Thời gian huấn luyện: 53.685
- Bộ siêu tham số:
  - kernel: poly
  - n_components: 700
  - gamma: 0.3222
  - degree: 2

5.3 Heart Disease:

Trong phần cuối cùng của phần thực nghiệm, nhóm chúng em tiếp tục thử nghiệm phương pháp Kernel PCA trên một bộ dữ liệu hoàn toàn khác hai bộ dữ liệu vừa rồi.

a. Mô tả chi tiết:

Bộ dữ liệu về Hear Disease này chứa 14 đặc trưng. Các đặc trrưng này mô tả về các chỉ số về mặt y học được dùng để dự đoán xem bệnh nhân có vấn đề về tim mạch hay không. Và các đặc trưng này được đánh số từ V1-V14 trong file .csv cung cấp.

Nguồn cung cấp: [UCI Machine Learning Repository: Heart Disease Data Set]

```
1  data.head()
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14
0	63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
1	67	1	4	160	286	0	2	108	1	1.5	2	3	3	1
2	67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
3	37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0	3	0

Các đặc trưng đó bao gồm:

1. Age: Tuổi của bệnh nhân.
2. Sex: Giới tính của bệnh nhân.
3. Cp: Cơn đau thắt ngực đã trải qua (1: đau thắt ngực điển hình, 2: đau thắt ngực không điển hình, 3: không đau thắt ngực, 4: không có triệu chứng).
4. Trestbps: Chỉ số huyết áp.
5. Chol: Chỉ số cholesterol.
6. Fbs: Chỉ số đường huyết.
7. Restecg: Điện tâm đồ (0 = bình thường, 1 = có bất thường sóng ST-T, 2 = hiển thị phì đại thất trái có thể xảy ra hoặc xác định theo tiêu chí của Estes).
8. Thalach: Nhịp tim tối đa.
9. Exang: Tập thể dục gây ra cơn đau thắt ngực (0: không có, 1: có).
10. Oldpeak: ST depression induced by exercise relative to rest ('ST' relates to positions on the ECG plot. See more here).
11. Slop: the slope of the peak exercise ST segment (Value 1: upsloping, Value 2: flat, Value 3: downsloping).
12. Ca: Số lượng major vessels (0-3).
13. Thal: Một rối loạn về máu được gọi là thalassemia (3 = bình thường; 6 = khiếm khuyết cố định; 7 = khiếm khuyết có thể hồi phục).
14. Target: Vấn đề về tim mạch (0: không có, 1: có).

b. Thực nghiệm:

Trước khi đi vào giải quyết vấn đề được đặt ra, nhóm em đã quan sát một số điểm của bộ dữ liệu này để tìm ra hướng đi tốt nhất đối với bài toán này.

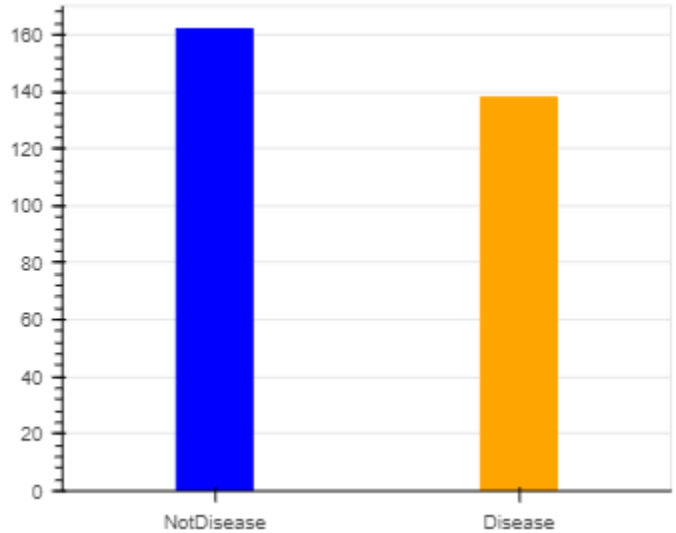

Figure 10: Phân phối dữ liệu giữa các classes

Trong Figure-9, có thể thấy được dữ liệu không bị mất cân bằng khi phân phối giữa các classes khá đồng đều.

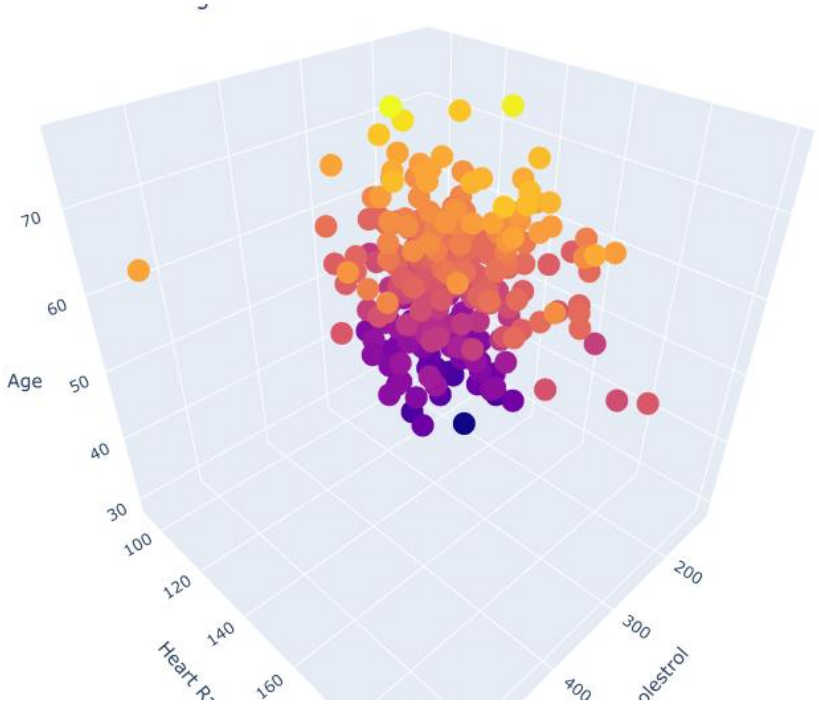

Figure-10: Sự tương quan giữa Age-Heart Rate-Cholestrol

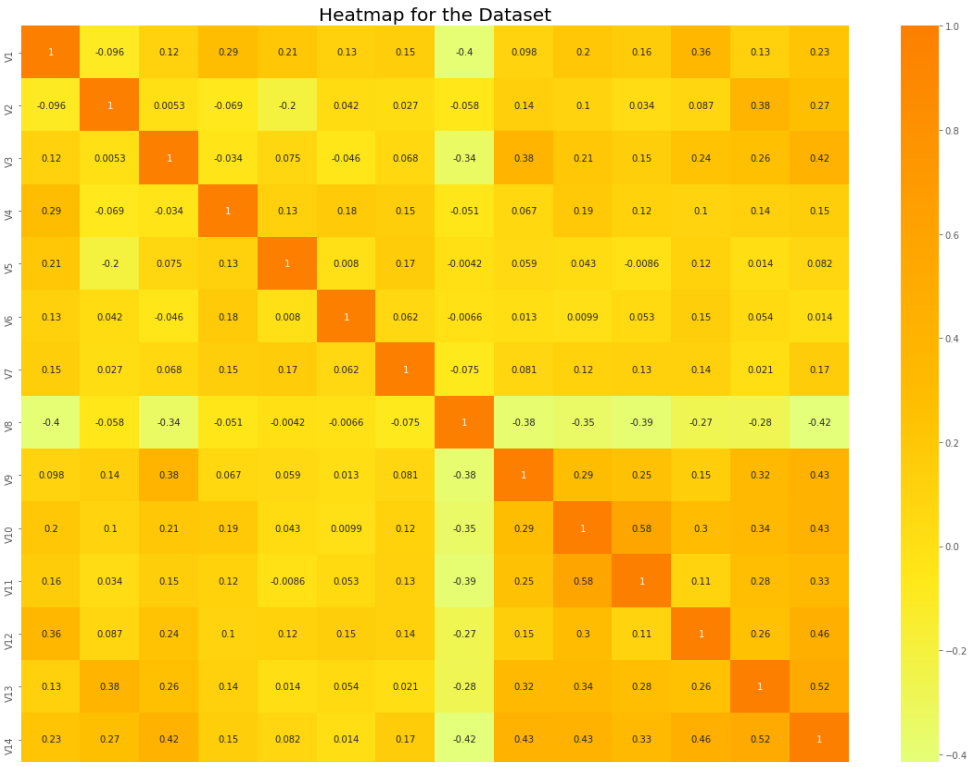

Figure-11: Heatmap

Heatmap ở trên là để hiển thị mối tương quan giữa các thuộc tính khác nhau của tập dữ liệu đã cho. Heatmap cho thấy rằng hầu như tất cả các tính năng / thuộc tính được đưa ra trong tập dữ liệu rất ít tương quan với nhau.

Cách thực nghiệm nhóm em cũng sẽ làm tương tự hai thực nghiệm trước đó, cố gắng tối ưu độ chính xác bằng cách tìm ra bộ siêu tham số phù hợp nhất đối với bài toán.

c. Kết quả thực nghiệm và nhận xét:

Dựa trên những gì phân tích dữ liệu ở trên, nhóm em có một vài kết luận sau đối với việc sử dụng Kernel PCA cho bộ dữ liệu này. Nhóm em nhận thấy sẽ không phù hợp khi áp dụng phương pháp Kernel PCA đối với bài toán này. Đầu tiên về số features của bộ dữ liệu này khá ít. Và mức độ tương quan rất ít hầu như là các biến độc lập với nhau chính vì thế rất khó có thể áp dụng phương pháp giảm số chiều dữ liệu. Ngoài ra tập huấn luyện cũng rất ít khoảng 300 mẫu. Kết quả thực nghiệm cho thấy những gì nhóm phân tích khá chính xác khi độ chính xác của mô hình không những không tăng mà còn giảm so với không thực hiện giảm chiều dữ liệu.

Model	Kernel	Components	Gamma	Degree	Accuracy	Fit time
SVM	None	None	None	None	0.8918	0.0299
SVM	linear	8	None	None	0.8366	0.0083
SVM	rbf	8	0.03	None	0.8433	0.0148
SVM	sigmoid	8	0.03	None	0.8466	0.0165
SVM	poly	8	0.0366	2	0.8433	0.0307

Qua thực nghiệm này, nhóm nhận thấy không phải lúc nào giảm số chiều dữ liệu cũng sẽ mang lại hiểu quả. Cần có những bước phân tích dữ liệu để xem dữ liệu ấy có phù hợp để giảm hay không. Ví dụ như nhiều features gây nhiễu, số lượng features rất lớn, …

6. Kết luận:

Trong lĩnh vực nghiên cứu về khai phá dữ liệu nói chung cũng như nghiên cứu về thuật toán phân lớp nói riêng, vấn đề xử lý dữ liệu lớn ngày càng trở thành vấn đề cấp thiết và đóng vai trò chủ đạo trong giải quyết các bài toán thực tế. Hiện nay, phương pháp hàm nhân đã được sử dụng để tăng khả năng áp dụng PCA khi giải quyết các bài toán phi tuyến.

Điểm cốt lõi của phương pháp này việc lựa chọn hàm nhân. Thông thường ta hay sử dụng lại những hàm nhân đã có sẵn như: Hàm Gauss, Hàm Đa Thức, Hàm Euclidean. Tuy nhiên cũng có thể xây dựng hàm nhân mới dựa vào những hàm nhân đã có sẵn. Việc xây dựng hàm nhân phù hợp với những mô hình thực nghiệm trong lĩnh vực Tin-Sinh học sẽ tiếp tục được nghiên cứu và phát triển trong thời gian tới.

7. Tài liệu tham khảo:

[1] B. Scholkopf, A.J. Smola, K. Muller | Nonlinear component analysis as a kernel eigenvalue problem, Neural Computation 10 (5) | 1998.

[2] B. Scholkopf, A.J. Smola | Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond (Adaptive Computation and Machine Learning) | MIT press | 2002.

[3] Kenneth Ezukwoke, Samaneh Zareian | KERNEL METHODS FOR PRINCIPAL COMPONENT ANALYSIS (PCA) A comparative study of classical and kernel PCA | 2019 | DOI:10.13140/RG.2.2.17763.09760

[4] Nguyễn Hà Nam | Tối ưu hóa KPCA bằng GA để chọn các thuộc tính đặc trưng nhằm tăng hiệu quả phân lớp của thuật toán Random Forest | 2009

[5] Hands-On Machine Learning with Sckit-Learn, Keras, and TensorFlow, 2nd Edition by Aurélien Géron.

_____ HẾT _____