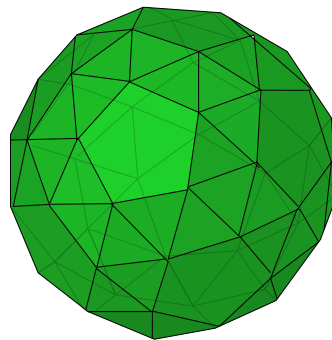


Projects in Mathematics and Applications

LOCALLY LINEAR EMBEDDING

Ngày 7 tháng 8 năm 2021

Phan Hoàng Quân ^{*} † Mai Kiên Quốc
Phạm Duy Tùng [‡] § Đặng Đức Trường



^{*}Trường Phổ thông Năng Khiếu - Đại học Quốc gia TP.HCM

[†]Trường THPT Chuyên Lê Hồng Phong, TP.HCM

[‡]Trường Đại học Bách Khoa Hà Nội

[§]Trường Đại học Bách Khoa Hồ Chí Minh

Lời cảm ơn

Đầu tiên, chúng em xin chân thành cảm ơn Ban tổ chức PiMA vì, mặc cho những trở ngại và thách thức gây ra do đại dịch COVID-19, vẫn đã tổ chức thành công ra một kì trại Toán học vô cùng bổ ích. Cảm ơn các anh chị mentors và các diễn giả đã gieo những hạt giống đam mê Toán học và Khoa học dữ liệu, cũng như truyền tải những kiến thức rất thú vị mà những học sinh cấp 3 và năm đầu đại học như chúng em ít có cơ hội được tiếp xúc.

Đặc biệt, chúng em xin cảm ơn anh Linh Trần, cũng như chị Phương Đình, anh Nguyễn Nguyễn, chị Thư Nguyễn, đã truyền cảm hứng và theo sát chúng em trong quá trình thực hiện đề tài này. Trong quá trình biên soạn, sai sót là khó tránh khỏi. Nhóm tác giả mong nhận được sự góp ý từ phía bạn đọc. Xin chân thành cảm ơn.

Tóm tắt nội dung

Bài báo cáo này giới thiệu thuật toán Locally Linear Embedding (LLE), một phương pháp giảm chiều dữ liệu thuộc lớp Manifold Learning. Chúng tôi sẽ trình bày nền tảng và các bước thực hiện thuật toán, cũng như cách áp dụng LLE vào một số tập dữ liệu lý tưởng và để nén ảnh. Cuối cùng, bài báo cáo này đề cập một số cải biến của LLE; nêu lên các ưu điểm và một số lưu ý khi áp dụng phương pháp này.

Mục lục

1	Dimensionality Reduction và sơ lược về Locally Linear Embedding	1
1.1	Bài toán Dimensionality Reduction	1
1.2	Manifold Learning và Manifold	1
1.3	Tổng quan về Locally Linear Embedding	2
2	Locally Linear Embedding	3
2.1	Xây dựng neighbor graph	3
2.2	Xây dựng ma trận trọng số	3
2.3	Xác định tập dữ liệu mới từ ma trận trọng số	6
3	Áp dụng mô hình	8
3.1	Thuật toán LLE cơ bản	8
3.2	Thuật toán LLE sử dụng ϵ -neighborhoods	11
3.3	Ứng dụng LLE vào bài toán nén ảnh	12
4	Một số cải biến của Locally Linear Embedding	15
4.1	Hessian Locally Linear Embedding	15
4.2	Local Tangent Space Alignment	15
5	Kết luận đánh giá	17
6	Phụ lục	18
6.1	Đại số tuyến tính	18
6.2	Giải tích ma trận	20
6.3	Lựa chọn siêu tham số neighbor - k	20

1 Dimensionality Reduction và sơ lược về Locally Linear Embedding

1.1 Bài toán Dimensionality Reduction

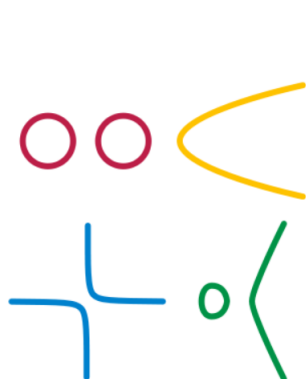
Các tập dữ liệu lớn có thể gây nhiều trở ngại trong việc giải các bài toán Machine Learning. Kích thước của dữ liệu càng lớn, tốc độ của thuật toán càng giảm, và việc trích xuất các thông tin ý nghĩa từ dữ liệu càng khó khăn. Trong một số trường hợp, các tập dữ liệu lớn có thể chứa các features gây nhiễu, làm giảm độ chính xác của thuật toán.

Những bất lợi trên dẫn đến nhu cầu giảm kích thước của các tập dữ liệu quá lớn, và từ đó bài toán **Dimensionality Reduction (Giảm chiều dữ liệu)** ra đời. Bài toán được phát biểu như sau: Cho các điểm dữ liệu x_1, x_2, \dots, x_n thuộc không gian D chiều. Với mỗi điểm x_i ($1 \leq i \leq n$), tìm một điểm y_i thuộc không gian d chiều (với $D > d$), sao cho tính chất tối ưu được thỏa mãn. Khi đó y_i được gọi là representation (biểu diễn) của x_i .

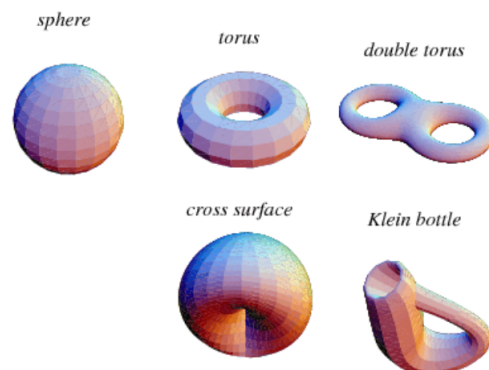
Bài toán Dimensionality Reduction có nhiều ý nghĩa thực tế. Giải được bài toán này giúp tăng tốc độ học, tránh overfit (sự quá khớp), giảm dung lượng lưu trữ, hỗ trợ trực quan hóa dữ liệu trên không gian 2 (hoặc 3) chiều, và tìm những features ẩn, quy luật giúp giải thích dữ liệu.

1.2 Manifold Learning và Manifold

Một cách tiếp cận bài toán Dimensionality Reduction là **Manifold Learning (Học đa tạp)** - một nhóm các thuật toán giảm chiều dữ liệu không giám sát. Định nghĩa của manifold vượt quá phạm vi của bài báo cáo này, tuy nhiên, ta quan tâm tính chất sau đây của manifold: một manifold d chiều là một cấu trúc hình học mà cục bộ tại mọi điểm của nó trông giống như một không gian Euclide d chiều. Trái Đất (hay, tổng quát hơn, một mặt cầu 3 chiều) là một manifold 2-chiều: nhìn một cách cục bộ, bề mặt Trái Đất trông giống một không gian 2 chiều, nhưng thực chất Trái Đất là một cấu trúc 3 chiều. Bên dưới là một số ví dụ về manifold:



Hình 1: Các manifold 1-chiều



Hình 2: Các manifold 2-chiều

Manifold Learning bắt đầu với giả thiết các điểm dữ liệu trong không gian D chiều đều nằm

trên một manifold d chiều với $D > d$, và sử dụng các tính chất cục bộ của manifold để tìm ra representation d chiều của các điểm dữ liệu ban đầu.¹ Một hệ quả của giả thiết đa tạp là mỗi điểm dữ liệu và các neighbors² của nó cùng nằm trên một không gian (gần như) tuyến tính cục bộ. Điều này có nghĩa là mỗi điểm dữ liệu có thể được biểu diễn bằng một tổ hợp tuyến tính (có ràng buộc) của các neighbors của nó. Hệ quả này là nền tảng quan trọng cho một trong các bước thực hiện của Locally Linear Embedding.

1.3 Tổng quan về Locally Linear Embedding

Locally Linear Embedding (LLE) là thuật toán giảm chiều dữ liệu thuộc nhóm Manifold Learning. Tổng quan về thuật toán như sau:

- Input:
 - Tập dữ liệu cần giảm chiều X , có kích thước $m \times D$, với m điểm dữ liệu (mỗi hàng của X là một điểm dữ liệu) và D features,
 - Tham số cho bài toán xây dựng neighbor graph,
 - Số chiều d của tập dữ liệu đầu ra ($d < D$).
- Output:
 - Tập dữ liệu đã được giảm chiều Y , có kích thước $m \times d$, với m điểm dữ liệu (mỗi hàng của Y là một điểm dữ liệu) và D features.
- Các bước thực hiện:
 1. Xây dựng neighbor graph, tức xác định neighbors cho từng điểm dữ liệu,
 2. Xấp xỉ mỗi điểm dữ liệu bằng một tổ hợp tuyến tính (có ràng buộc) của các neighbors của nó,³
 3. Sử dụng các trọng số của tổ hợp tuyến tính tìm được trong bước 2, suy ra representation (trong không gian ít chiều hơn) của điểm dữ liệu ban đầu.

Giống như các thuật toán Manifold Learning khác, LLE giả định rằng các điểm dữ liệu trong không gian D chiều nằm trên một manifold d chiều (với $D > d$). Ta có thể gán cho manifold này một hệ trục tọa độ manifold. LLE giảm chiều dữ liệu bằng cách biến tọa độ so với hệ trục tọa độ toàn cục của một điểm dữ liệu bất kỳ thành tọa độ so với hệ trục tọa độ manifold bằng các phép biến đổi tuyến tính. Với những điều kiện ràng buộc, các trọng số của tổ hợp tuyến tính tìm được ở bước 2 là bất biến với các phép biến đổi tuyến tính (sẽ được đề cập rõ hơn ở 2.2). Điều này có nghĩa là cho dù xét trên bất cứ hệ trục tọa độ nào, các trọng số tìm được ở bước 2 đều phản ánh được mối tương quan hình học giữa một điểm dữ liệu và các neighbors của nó. Vì thế ở bước 3, ta giữ nguyên trọng số để đi tìm representation của các điểm dữ liệu ban đầu, cũng là đi tìm representation của mỗi điểm dữ liệu dựa vào tương quan hình học với các neighbors.

¹Đây gọi là giả thiết đa tạp. Tất nhiên, không phải lúc nào giả thiết đa tạp cũng đúng. Vì vậy không thể áp dụng Manifold Learning một cách tùy tiện, mà phải xét đến tính phù hợp của tập dữ liệu.

²Neighbors là một khái niệm trong topology. Trong phạm vi này, ta có thể hiểu một điểm q là neighbor của p nếu q đủ "gần" p . Phương thức đo độ "gần" sẽ được đề cập ở phần 2.1.

³Sự khả thi của bước này được dựa trên giả thiết đa tạp và hệ quả của nó, đã được đề cập ở phần 1.2.

2 Locally Linear Embedding

2.1 Xây dựng neighbor graph

Trong LLE, một neighbor graph là một đồ thị vô hướng xác định cho một tập dữ liệu, mà trong đó mỗi điểm dữ liệu là một đỉnh. Đồng thời, điểm p và q sẽ được nối với nhau bằng một cạnh vô hướng nếu q là neighbor của p .

Để xác định được hai điểm p và q có là neighbors hay không thì trước hết phải xác định một phương pháp đo lường, qua đó tính toán mức độ tương quan giữa các điểm dữ liệu. Độ tương quan giữa 2 điểm dữ liệu càng lớn thì khoảng cách càng nhỏ. Các khoảng cách thường được áp dụng trong các thuật toán xây dựng neighbor graph là khoảng cách Euclid:

$EUD(p, q) = \sqrt{\sum_{i=1}^n (q_i - p_i)^2}$ và khoảng cách Cosine: $COD(p, q) = 1 - \cos(p, q)$. Ngoài ra còn một vài phương pháp đo lường nữa như Mahalanobis, Hamming...

Các thuật toán xây dựng neighbor graph phổ biến được dùng trong bài toán LLE bao gồm **ϵ -neighborhoods**, **K-nearest neighbors** và **Locality-sensitive hashing**.

- **ϵ -neighborhoods**: Hai điểm p và q được coi là neighbor nếu khoảng cách giữa hai điểm nhỏ hơn giá trị ϵ xác định. Trong thuật toán này, ϵ là hyperparameter cần điều chỉnh. Nếu ϵ quá lớn thì mọi điểm sẽ là neighbors của nhau, còn nếu quá nhỏ thì sẽ xuất hiện tình trạng những điểm không có neighbors.
- **k-nearest neighbors**: Với mỗi điểm p , thuật toán chọn k ⁴ điểm gần nhất với p làm neighbors. Điều này đảm bảo rằng mỗi điểm dữ liệu đều có đủ k neighbors. Tuy nhiên cần phải điều chỉnh hyperparameter k phù hợp để tránh tình trạng tại điểm có mật độ thưa thì neighbor của nó sẽ nằm xa hơn so với các điểm ở vị trí có mật độ cao.
- **Locality-sensitive hashing**: Thuật toán LSH là một phương pháp xấp xỉ để xây dựng neighbor graph. LSH hoạt động dựa trên một thuật toán hash sao cho giá trị hash của các điểm có độ tương quan lớn thì có xác suất bằng nhau lớn. Nói cách khác, hàm hash ở đây tối đa hoá xác suất để các điểm gần nhau rơi vào cùng một giá trị hash (ngược lại với tiêu chí tối thiểu hoá của hàm hash thông thường). Một số hàm hash thường được dùng trong bài toán tìm neighborhood có thể kể đến như: Bit sampling, MinHash, SimHash,... Tuy độ chính xác không cao bằng 2 phương pháp trên nhưng ưu điểm của phương pháp xấp xỉ này nằm ở độ phức tạp tính toán.

Có thể thấy rằng, mỗi thuật toán xây dựng neighbors graph đều có ưu, nhược điểm riêng. Ta cần lựa chọn cũng như điều chỉnh những tham số thích hợp với mỗi bộ dữ liệu để đạt được hiệu quả tối ưu. Sau khi đã có được neighbor graph, tức xác định được các neighbors của mỗi điểm dữ liệu, ta xây dựng ma trận trọng số.

2.2 Xây dựng ma trận trọng số

Mục tiêu của bước này là xấp xỉ mỗi điểm dữ liệu bằng một tổ hợp tuyến tính của các neighbors của nó. Tức với mỗi điểm dữ liệu x_i , ta xấp xỉ:

⁴Xem Lựa chọn siêu tham số neighbor-k ở Phụ lục

$$x^{(i)} \approx \sum_{j=1}^m w_{i,j} \cdot x^{(j)}$$

với mọi $i, j \in \{1, 2, \dots, m\}$, và ràng buộc $w_{i,j} = 0$ nếu x_j không phải là neighbor của x_i .

Tất nhiên, ta muốn mất mát xảy ra do việc xấp xỉ này là nhỏ nhất. Bài toán này tương đương tìm một ma trận trọng số W để tối ưu hàm mất mát sau:

$$L(W) = \sum_{i=1}^m \left\| x^{(i)} - \sum_{j=1}^m w_{i,j} \cdot x^{(j)} \right\|^2 \quad (1)$$

Như đã đề cập ở phần 1.3, ta cần đề ra những điều kiện ràng buộc để nghiệm tối ưu của (1) là bất biến với các phép biến đổi tuyến tính, cụ thể là phép vị tự, phép quay và phép tịnh tiến. Dựa vào dạng của (1), dễ thấy nghiệm tối ưu bất biến với phép quay và phép vị tự. Đồng thời để nghiệm tối ưu của (1) bất biến với phép tịnh tiến, ta đặt thêm ràng buộc $\sum_{j=1}^m w_{i,j} = 1$. Thật vậy, với điều kiện này, tịnh tiến toàn bộ các điểm dữ liệu theo một vector t bất kỳ, ta có:

$$\begin{aligned} (x^{(i)} + t) - \sum_{j=1}^m w_{i,j} \cdot (x^{(j)} + t) &= x^{(i)} + t - \left(\sum_{j=1}^m w_{i,j} \cdot x^{(j)} + \sum_{j=1}^m w_{i,j} \cdot t \right) \\ &= x^{(i)} - \sum_{j=1}^m w_{i,j} \cdot x^{(j)} + \left(t - t \cdot \sum_{j=1}^m w_{i,j} \right) \\ &= x^{(i)} - \sum_{j=1}^m w_{i,j} \cdot x^{(j)} \end{aligned}$$

Như vậy, bài toán xây dựng ma trận trọng số được phát biểu như sau:

Tìm ma trận trọng số W sao cho:

$$W = \underset{W}{\operatorname{argmin}} L(W)$$

Thỏa mãn $w_{i,j} = 0$ nếu x_j không phải là neighbor của x_i , và $\sum_{j=1}^m w_{i,j} = 1$.

Đặt:

$$\epsilon_i = \left\| x^{(i)} - \sum_{j=1}^m w_{i,j} \cdot x^{(j)} \right\|^2$$

với mọi $i, j \in \{1, 2, \dots, m\}$. $L(W)$ đạt cực tiểu tương đương với mọi ϵ_i được tối ưu hóa.

Vì tính tiền dữ liệu không làm thay đổi hàm mất mát, tiến hành tính tiền mỗi điểm dữ liệu theo vector $-x^{(i)}$:

$$\epsilon_i = \left\| \sum_{j=1}^m w_{i,j} \cdot (x^{(j)} - x^{(i)}) \right\|^2$$

Giả sử $x^{(i)}$ có k neighbors. Gọi các neighbors của $x_1^{(i)}$ lần lượt là $x_1^{(i)}, x_2^{(i)}, \dots, x_k^{(i)}$. Gọi thêm:

- Z_i là ma trận $k \times D$ có các vector hàng lần lượt là $x_1^{(i)} - x^{(i)}, x_2^{(i)} - x^{(i)}, \dots, x_k^{(i)} - x^{(i)}$.
- W_i là ma trận $k \times 1$ chứa trọng số của các neighbors của $x^{(i)}$ ⁵.

Khi đó: $\epsilon_i = W_i^T Z_i Z_i^T W_i = W_i^T G_i W_i$, với $G_i = Z_i Z_i^T$ là một ma trận Gram ⁶.

Đến đây, sử dụng phương pháp nhân tử Lagrange để tối ưu hóa ϵ_i với điều kiện $\sum_j w_{i,j} = 1$.

Điều kiện này tương đương với $\mathbf{1}^T W_i - 1 = 0$, với $\mathbf{1}$ là vector cột có các phần tử đều bằng 1. Đặt:

$$\mathcal{L}(W_i, \lambda) = W_i^T G_i W_i - \lambda (\mathbf{1}^T W_i - 1)$$

Lấy đạo hàm lần lượt theo hai biến, ta có:

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial W_i} = 0 \\ \frac{\partial \mathcal{L}}{\partial \lambda} = 0 \end{cases} \iff \begin{cases} 2G_i W_i - \lambda \mathbf{1} = 0 \\ \mathbf{1}^T W_i - 1 = 0 \end{cases} \iff \begin{cases} G_i W_i = \frac{\lambda}{2} \mathbf{1} \\ \mathbf{1}^T W_i - 1 = 0 \end{cases}$$

Nếu G_i khả nghịch, hệ phương trình trên tương đương:

$$\begin{cases} W_i = \frac{\lambda}{2} G_i^{-1} \mathbf{1} \\ \mathbf{1}^T W_i - 1 = 0 \end{cases}$$

Một cách giải hệ phương trình trên là cho $\lambda = 1$, giải ra W_i bằng phương trình thứ nhất, rồi sau đó chuẩn hóa W_i để thỏa phương trình thứ hai.

Nếu G_i không khả nghịch ⁷ thì tồn tại vô số nghiệm cho bài toán tìm trọng số. Tất nhiên, ta có thể chọn một nghiệm bất kỳ để làm tham số cho bài toán tìm representation. Tuy nhiên, trong thực tế, việc lập trình cho thuật toán chọn một nghiệm bất kỳ sẽ tạo sự bất ổn định giữa việc biểu diễn các điểm dữ liệu khác nhau, và giữa các lần chạy của thuật toán. Để tránh trường hợp này, ta sử dụng kỹ thuật chính quy hóa. ⁸ Xét hàm mất mát đã được chính quy hóa:

$$\epsilon_i = \left\| \sum_{j=1}^m w_{i,j} \cdot (x^{(j)} - x^{(i)}) \right\|^2 + \alpha \|w_i\|^2 = W_i^T G_i W_i + \alpha W_i^T W_i$$

⁵Ở đây chúng ta chỉ lưu tâm đến các trọng số của các neighbors của $x^{(i)}$ vì các trọng số khác bằng 0

⁶Xem Định nghĩa ma trận Gram ở Phụ lục Đại số tuyến tính

⁷Điều này xảy ra khi $k > D$, tức số neighbors của điểm dữ liệu lớn hơn số features ban đầu

⁸Xem phần l_2 regularization ở Phụ lục Đại số tuyến tính

Giải một cách tương tự như trên, ta có nghiệm tối ưu của bài toán là:

$$W_i = \frac{\lambda}{2}(G_i + \alpha \mathbf{I})^{-1} \mathbf{1}$$

với \mathbf{I} là ma trận đơn vị.

2.3 Xác định tập dữ liệu mới từ ma trận trọng số

Mục tiêu của bước này là từ ma trận trọng số trong bước 2, ta tìm biểu diễn trong không gian \mathbb{R}^d của tập dữ liệu. Giả sử dữ liệu đầu ra là $Y \in \mathbb{R}^{n \times d}$, ta cần tối thiểu hóa hàm mất mát

$$\Phi(Y) = \sum_i \left\| Y_i - \sum_j W_{ij} Y_j \right\|^2. \quad (2)$$

hay có thể viết dưới dạng ma trận:

$$\Phi(Y) = \text{tr}(Y^T M Y) \quad (3)$$

trong đó $M = (I - W)^T(I - W)$.

Hàm mất mát (2) có ý nghĩa giữ lại quan hệ tuyến tính địa phương của X khi xây dựng dữ liệu mới Y .

Ta cần thêm một số điều kiện để chuẩn hóa dữ liệu đầu ra:

$$\sum_i Y_i = 0 \iff Y^T \mathbf{1} = 0 \quad (4)$$

và

$$\frac{1}{n} Y^T Y = I_d \quad (5)$$

ở đó $\mathbf{1}$ là vector cột có các thành phần đều là 1.

Hai điều kiện này đảm bảo dữ liệu đầu ra sẽ được chuẩn hóa, nghĩa là có trung bình là vector 0 và có ma trận hiệp phương sai là ma trận đơn vị. Ta sẽ giải bài toán tối thiểu hóa $\Phi(Y)$ với điều kiện đẳng thức trên đường chéo của (5), nghiệm tối ưu của bài toán này sẽ thỏa mãn điều kiện (4) và (5).

Sử dụng phương pháp nhân tử Lagrange, ta có Lagrangian:

$$L = \text{tr}(Y^T M Y) - \text{tr} \left[\Lambda^T \left(\frac{1}{n} Y^T Y - I_d \right) \right] \quad (6)$$

trong đó Λ là ma trận đường chéo có các phần tử là nhân tử Lagrange.

Để đạo hàm của Lagrangian theo ma trận Y ⁹ chỉ có các thành phần bằng 0 thì:

$$M Y - \frac{1}{n} Y \Lambda = 0 \quad (7)$$

Vì vậy, các cột của Y là các vector riêng của M . Để ý rằng M có dạng $A^T A$ nên chỉ có giá trị riêng là số thực không âm. Mặt khác, Y có d cột, rất nhỏ so với n (là số chiều và cũng là số giá trị riêng tối đa của M), nên luôn tìm được các vector riêng v_1, v_2, \dots, v_d để tạo thành Y .

⁹Xem phần phụ lục giải tích ma trận

Chú ý rằng, từ mục 2.2 có $W\mathbf{1} = \mathbf{1}$, vì thế $(I - W)\mathbf{1} = 0$ và do đó $\mathbf{1}$ là vector riêng tương ứng với giá trị riêng 0 của M . Vậy nếu $v_i \neq \mathbf{1}$, $\forall i = 1, 2, \dots, d$ thì $Y^T \mathbf{1} = 0$, và từ đó có được (4). Để (5) được thỏa mãn thì ta sẽ lựa chọn các vector riêng có chuẩn là \sqrt{n} và đôi một trực giao nhau. Vì các vector riêng ứng với giá trị riêng khác nhau thì trực giao, do đó chỉ cần chọn các vector riêng ứng với cùng một giá trị riêng trực giao nhau.

Khi các cột của Y bao gồm các vector riêng của M (lần lượt tương ứng với giá trị riêng $\lambda_1, \lambda_2, \dots, \lambda_d$), giá trị của hàm mục tiêu sẽ là:

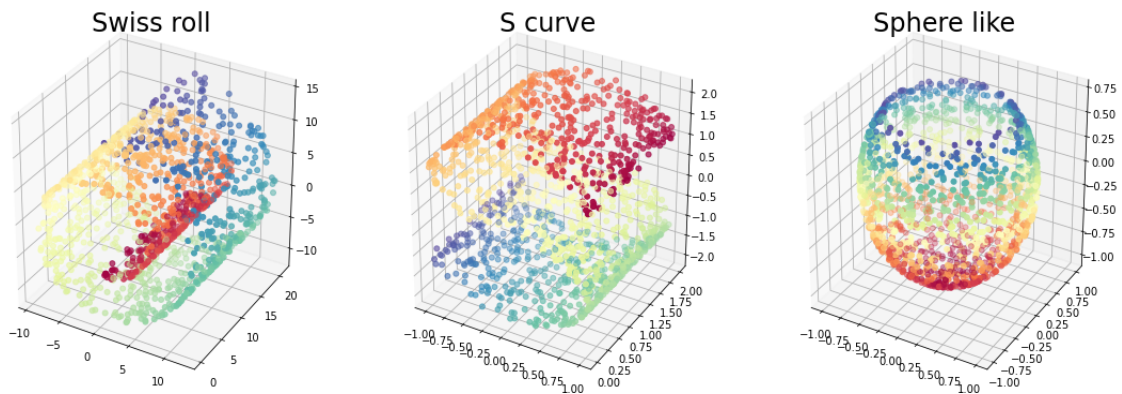
$$\Phi(Y) = \sum_i Y_i^T M Y_i = \sum_i \lambda_i Y_i^T Y_i = n \sum_i \lambda_i \quad (8)$$

Ta cần tối thiểu hóa $\Phi(Y)$, do đó sẽ chọn d vector riêng đôi một trực giao ứng với d trị riêng nhỏ nhất của M (không tính $\mathbf{1}$ ứng với trị riêng 0) để tạo thành Y . Chú ý rằng luôn có thể chọn ra $d+1$ giá trị riêng nhỏ nhất của M do M là ma trận Gram nên M nửa xác định dương và vì vậy các trị riêng luôn không âm.

3 Áp dụng mô hình

3.1 Thuật toán LLE cơ bản

Để có một góc nhìn trực quan hơn về thuật toán LLE đã được trình bày ở các phần trước, ta tiến hành áp dụng thuật toán LLE lên một vài bộ dữ liệu dưới đây.



Hình 3: Các tập datasets

Mỗi tập dữ liệu là tập hợp các điểm dữ liệu 3 chiều được biểu diễn dưới dạng ma trận kích thước $n \times 3$ với $n \approx 1500$. Mục tiêu ở đây là áp dụng thuật toán LLE để giảm chiều của dữ liệu xuống còn 2 chiều, tức là một ma trận kích thước $n \times 2$. Dưới đây là đoạn code sử dụng thư viện sklearn.

```
import matplotlib.pyplot as plt
from sklearn.manifold import LocallyLinearEmbedding

# Ma tran input duoc luu trong bien X
fig = plt.figure()
fig.set_size_inches(13, 8)

# Cac gia tri k can thu
k_set = [5, 6, 7, 10, 11, 12, 15, 16, 17, 20,
         21, 22, 25, 26, 27, 30, 40, 60, 80, 120]

# Voi moi gia tri k ta chay thuat toan LLE va bieu dien output tuong ung
for id, k in enumerate(k_set):
    # Giam chieu bang LLE
    embedding = LocallyLinearEmbedding(n_components=2, n_neighbors=k,
                                       random_state=117)
    X_transformed = embedding.fit_transform(X)

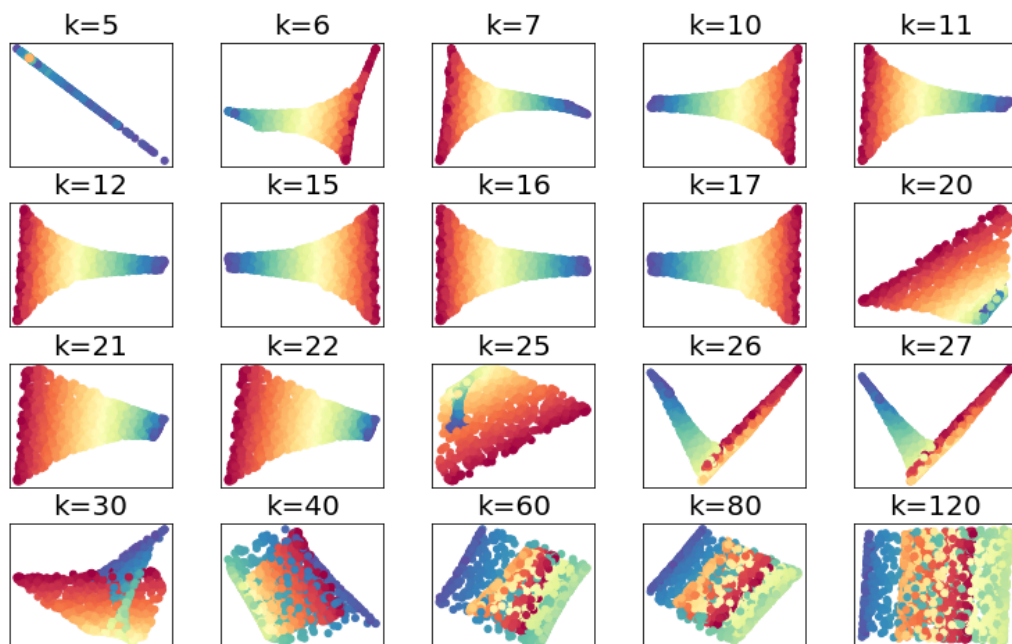
    # Bieu dien output bang matplotlib
    ax = fig.add_subplot(4, 5, id + 1)
    ax.scatter(X_transformed[:, 0], X_transformed[:, 1],
               c=color, cmap=plt.cm.Spectral, s=25)
```

```
# Xóa các trục và đặt tên cho từng output
plt.xticks([]), plt.yticks([])
plt.title(f'k={k}', fontsize=20)

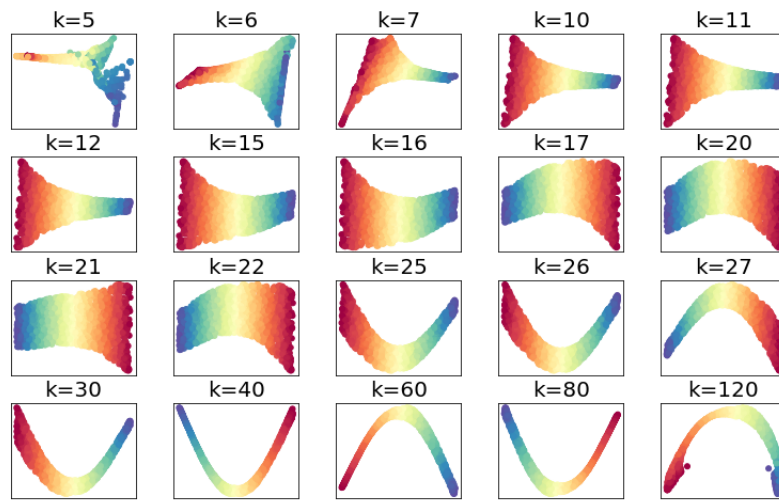
plt.subplots_adjust(wspace=0.3, hspace=0.3)
plt.show()
```

Dưới đây là kết quả thu được từ mỗi bộ dữ liệu. Dựa vào các kết quả thực nghiệm, ta có thể rút ra các nhận xét sau:

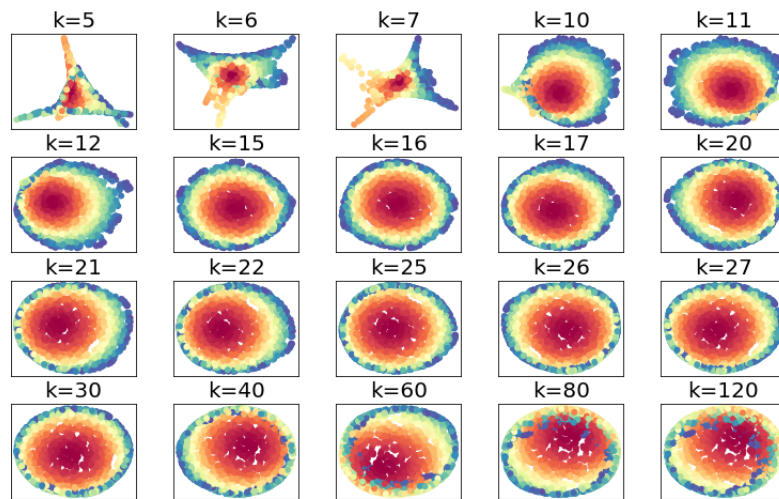
- Kết quả của thuật toán LLE phụ thuộc rất lớn vào việc xây dựng neighbor graph, cụ thể ở đây là hyperparameter k trong thuật toán K-nearest neighbors.
- Việc lựa chọn k quá nhỏ sẽ khiến cho manifold bị chia thành nhiều mảnh nhỏ. Trong khi đó, lựa chọn k quá lớn sẽ làm mất đi các cấu trúc hình học cục bộ.
- Việc lựa chọn hyperparameter k phụ thuộc vào tính chất của bộ dữ liệu. Qua minh họa của output với các k khác nhau, ta có thể ước lượng tương đối được giá trị k tối ưu cho từng bộ dữ liệu. Cụ thể: $k_{swissroll} \in [17, 40]$, $k_{curve} \in [20, 40]$ và $k_{sphere} \in [15, 30]$.



Hình 4: Kết quả trên tập swiss roll



Hình 5: Kết quả trên tập s curve



Hình 6: Kết quả trên tập sphere like

3.2 Thuật toán LLE sử dụng ϵ -neighborhoods

Như đã được trình bày ở các phần trên, việc xây dựng neighbor graph là một phần quan trọng trong thuật toán LLE. Ngoài thuật toán k-nearest neighbors - thuật toán được dùng trong cài đặt LLE của sklearn, thì một thuật toán khác khá phổ biến chính là **ϵ -neighborhoods**. Dưới đây là một đề xuất cài đặt thuật toán LLE sử dụng **ϵ -neighborhoods** và kết quả khi chạy trên tập dữ liệu swiss roll.

```
import numpy as np
from sklearn.neighbors import radius_neighbors_graph

def locally_linear_embedding(X, eps, out_dim=2):
    # Xây dựng neighbor graph bằng  $\epsilon$ -neighborhood
    G = radius_neighbors_graph(X, radius=eps, mode='distance')

    # Tìm ma trận trọng số W
    W = np.zeros((X.shape[0], X.shape[0]))

    for i, x in enumerate(X):
        Z = np.zeros((0, X.shape[1]))
        neighbor = []

        for id in range(G.indptr[i], G.indptr[i+1]):
            neighbor.append(G.indices[id])
            Z = np.vstack((Z, [X[G.indices[id]] - x]))

        # Bảo lưu nếu eps quá nhỏ dẫn tới tình trạng
        # 1 điểm không có bất cứ neighbor nào
        if not len(neighbor):
            raise Exception('eps is too small')

        C = Z @ Z.T
        _eps = 0.001 * np.trace(C)
        C = C + _eps * np.eye(C.shape[0])

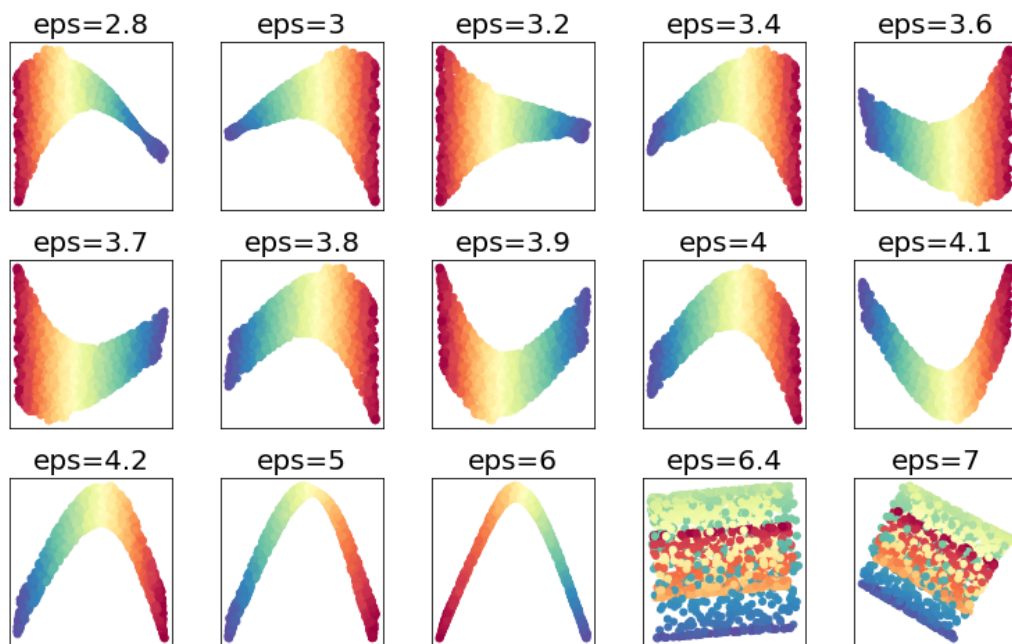
        one = np.full((C.shape[0], 1), 1)
        w = np.linalg.solve(C, one)

        w /= np.sum(w)
        for id, wj in enumerate(w):
            W[i, neighbor[id]] = wj

    # Tìm các điểm biểu diễn Y
    A = np.eye(W.shape[0]) - W
    A = A.T @ A

    eigen_values, eigen_vectors = np.linalg.eig(A)
    id = eigen_values.argsort()
    eigen_values = eigen_values[id]
    eigen_vectors = eigen_vectors[:, id]

    return eigen_vectors.T[1:out_dim + 1].T
```



Hình 7: Kết quả trên tập s curve sử dụng thuật toán e-neighborhood

Qua minh hoạ kết quả, có thể thấy sử dụng thuật toán **k-nearest neighbors**, **ϵ -neighborhoods** trong LLE cho kết quả tương tự nhau (**ϵ -neighborhoods** có phần nhỉnh hơn với $\epsilon \in [3.6; 3.8]$). Tuy nhiên, việc lựa chọn ϵ phù hợp có phần phức tạp hơn so với việc lựa chọn k , và phụ thuộc khá nhiều vào tính chất hình học cũng như phân bố mật độ dữ liệu của bộ dữ liệu.

3.3 Ứng dụng LLE vào bài toán nén ảnh

Nén ảnh là một phương pháp có tính ứng dụng cao trong khoa học dữ liệu. Nó giúp giảm kích thước của một bức ảnh - tức giảm số lượng dữ liệu cần thiết để mô tả một bức ảnh. Việc nén ảnh sẽ góp phần bóc tách các tính chất đặc trưng của ảnh cũng như tăng tốc các thuật toán thị giác máy tính. Ở đây, công dụng giảm chiều dữ liệu của LLE sẽ được tận dụng để nén một bức ảnh đen trắng như sau.



Hình 8: Hình đen trắng cần nén

Mỗi bức ảnh được coi là một ma trận kích thước $H \times W$ với mỗi giá trị tương ứng với một điểm ảnh. Các bước thực hiện nén ảnh như sau:

- Phân mảng:
 - Tiến hành phân nhỏ bức ảnh thành n mảng nhỏ x_i kích thước $h \times w$ ($h < H, w < W$) (các mảng có thể chồng một phần lên nhau).
 - Trải phẳng từng mảng x_i thành ma trận dòng, ta được một ma trận X kích thước $n \times (h \times w)$ có dạng như sau $\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$.
- Áp dụng LLE: Coi ma trận X là một ma trận biểu diễn n điểm dữ liệu với số chiều là $h \times w$. Ta áp dụng thuật toán LLE để giảm số về d chiều, thu được một ma trận có kích thước $n \times d$ là $Y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$.
- Biểu diễn ảnh nén dưới dạng heatmap: Với mỗi ma trận cột của ma trận Y ta có thể biểu diễn lại nó sao cho vị trí của mỗi phần tử ứng với vị trí của các mảng ban đầu. Qua đó, ta sẽ được k heatmap.
- Tính tỉ lệ nén bằng công thức $\text{compression rate} = \frac{w \times h \times d}{W \times H}$

Dưới đây là một đề xuất cài đặt sử dụng LLE để nén bức ảnh trên và các kết quả được biểu diễn dưới heatmap.

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

def compress(image, height, width, slice_h, slice_w, feature):
    # Them cac vung dem
    h, w = image.shape
    data = np.zeros((h + slice_h, w + slice_h))
    data[:h, :w] = image

    # Chia anh thanh cac mang
    patches = np.zeros((0, height * width))
    cur_h, cur_w = 0, 0
    cnt = 0
    while cur_h < h:
        while cur_w < w:
            patches = np.append(patches, [data[cur_h:cur_h + height, cur_w:cur_w + width].flatten().tolist()], axis=0)

            cur_w += slice_w
        cnt += 1
        cur_h += slice_h
        cur_w = 0

    # Nen cac mang bang LLE
    embedding = LocallyLinearEmbedding(n_components=feature, n_neighbors=15,
                                       random_state=117)

    Y = embedding.fit_transform(patches).T

    # Bieu dien anh nen duoi dang heatmap
    fig = plt.figure()
```

```

fig.set_size_inches(15, 4)

for i in range(feature):
    ax = fig.add_subplot(2, feature//2, i+1)
    ax.imshow(Y[i].reshape((cnt, Y[i].shape[0] // cnt)))
    plt.xticks([]), plt.yticks([])

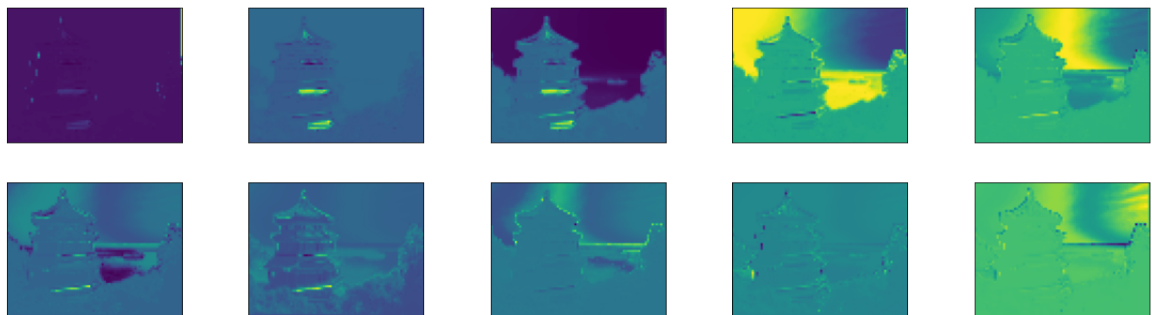
plt.subplots_adjust(left=0,
                    bottom=0,
                    right=1,
                    top=1,
                    wspace=0.3,
                    hspace=0.3)

plt.show()

# Tính tỉ lệ nén
rate = Y.shape[0] * Y.shape[1] / (image.shape[0] * image.shape[1])
print(f'Compression rate: {rate}')
# Tỉ lệ nén thu được là 0.17857142857142858

# Ảnh được lưu trong biến grayscale
compress(grayscale, 10, 10, 7, 8, 10)

```



Hình 9: Các heatmap thu được sau khi nén

Mỗi heatmap thu được biểu diễn một tính chất đặc trưng nhất định của ảnh gốc. Ví dụ: heatmap 3 và 4 thể hiện các mảng sáng tối, heatmap 7 và 8 thể hiện đường nét của các vật thể.

4 Một số cải biến của Locally Linear Embedding

4.1 Hessian Locally Linear Embedding

Hessian Locally Linear Embedding (HLLE) được nghiên cứu và phát biểu bởi D.L. Donoho và C. Grimes. Cải biến của HLLE so với LLE là HLLE thực hiện giảm chiều dữ liệu bằng cách tối thiểu hoá các yếu tố phụ thuộc vào đa tạp chứa tập dữ liệu. Quá trình vận hành HLLE có 5 bước:

1. **ϵ -neighborhoods** và **k-nearest neighbors** đều được sử dụng ở bước đầu tiên (nhưng **k-nearest neighbors** thường được sử dụng nhiều hơn). Chọn hyperparameter k sao cho $k \geq \frac{(d+2)(d+1)}{2}$.
2. Xác định toạ độ của tangent space tại mỗi điểm dữ liệu x_i bằng phương pháp PCA trên vùng lân cận của điểm x_i đó. Gọi tập hợp các neighbors của x_i là $X_i = \{x_{i1}, x_{i2}, \dots, x_{ik}\}$. Áp dụng Principal Component Analysis để tìm ra hệ trục chuẩn của X_i là một ma trận $k \times d$ là $V_i = [v_1 \ v_2 \ \dots \ v_d]$. Suy ra, các vector cột của V_i là hàm toạ độ tiếp tuyến của X_i .
3. Đặt $\mathbf{1} = [1, \dots, 1]^T \in \mathbb{R}^k$ và $Q_i = [v_j \boxtimes v_j]_{1 \leq j \leq d}$. Định nghĩa ma trận

$$V_a = [\mathbf{1}, V_i, Q_i]$$

Áp dụng phương pháp Gram-Schmidt để tìm ra ma trận chuẩn hoá $[\mathbf{1}, V_i, \tilde{Q}_i]$ của V_a . Từ đó, hàm Hessian lân cận là $W_i = \tilde{Q}_i(\tilde{Q}_i)^T$.

4. Cho ma trận Kernel ban đầu là ma trận 0 kích cỡ $n \times n$. Công thức của K là

$$K(N(i), N(i)) := K(N(i), N(i)) + W_i$$

với $K(N(i), N(i))$

- 5.

4.2 Local Tangent Space Alignment

Tương tự như LLE, **Local Tangent Space Alignment (LTSA)** tận dụng các tính chất tuyến tính cục bộ của manifold để tìm ra mối quan hệ cục bộ của một neighborhood.¹⁰ Để xác định mối quan hệ cục bộ này, LLE xấp xỉ mỗi điểm dữ liệu bằng một tổ hợp tuyến tính có ràng buộc của các neighbors; LTSA sử dụng các tangent space.¹¹ Cụ thể, với input và output giống của LLE, các bước của LTSA như sau:

1. Xây dựng neighbor graph,

¹⁰Neighborhood là một khái niệm trong topology. Neighbourhood của một điểm là một tập hợp các neighbors của điểm đó.

¹¹Một cách dễ hiểu, tangent space của manifold \mathcal{M} tại điểm p là một không gian vector chứa toàn bộ các vector tiếp tuyến với \mathcal{M} tại p .

2. Xét một điểm dữ liệu x_i và neighborhood $N(i)$ tương ứng. Xác định tangent space tại điểm x_i . Tìm tọa độ cục bộ của các điểm dữ liệu thuộc $N(i)$ trên tangent space,
3. Căn chỉnh các tangent space để đồng nhất hóa các hệ trục tọa độ cục bộ trên các tangent space khác nhau thành một hệ trục tọa độ toàn cục trong không gian d chiều.

LTSA nhanh, nhưng lại bất ổn định khi tập dữ liệu có noise (nhiều). So với LLE, LTSA nhạy cảm với kích thước của neighborhood, vì vậy khi xây dựng neighbor graph cần phải xác định thuật toán và kích thước neighborhood phù hợp.

5 Kết luận đánh giá

Trong bài báo cáo này, chúng tôi đã trình bày thuật toán Locally Linear Embedding, một thuật toán Manifold Learning thường được sử dụng để giảm chiều dữ liệu. Với nền tảng là giả thiết đa tạp, LLE bắt đầu với việc xây dựng neighbor graph, biểu diễn mỗi điểm dữ liệu bằng một tổ hợp tuyến tính có ràng buộc của các neighbors, và dùng ma trận trọng số tìm được để xác định tập dữ liệu mới ít chiều hơn.

So với các thuật toán giảm chiều dữ liệu phi tuyến tính khác, LLE có những ưu điểm quan trọng, bao gồm đảm bảo tìm được nghiệm tối ưu toàn cục và yêu cầu ít siêu tham số. Nhu cầu lưu trữ của LLE chỉ giới hạn bằng với kích cỡ của ma trận trọng số, tức $n \times k$, với n là số điểm dữ liệu và k là số neighbors. Hơn nữa, các ma trận được hình thành qua các bước của LLE thường là các ma trận thưa.¹² Sử dụng các kỹ thuật lập trình, ta có thể tận dụng các ma trận thưa để giảm không gian lưu trữ và thời gian chạy thuật toán.

Khi áp dụng thuật toán này cũng cần một số lưu ý. Như đã đề cập từ đầu, tập dữ liệu phải nằm (xấp xỉ gần) trên một đa tạp - đây là giả định căn bản nhất của LLE. Đồng thời phải chọn phương pháp và siêu tham số xây dựng neighbor graph phù hợp. Thường trong thực tế sẽ chọn phương pháp k-nearest neighbors vì phương pháp này không phụ thuộc vào kích cỡ và chiều của tập dữ liệu. Tuy nhiên, việc chọn k cho một tập dữ liệu là không dễ. k quá nhỏ hay quá lớn đều sẽ ảnh hưởng xấu đến thuật toán.¹³ Với phương pháp ϵ -neighborhood, việc chọn ϵ phù hợp thật sự rất khó, bởi vì giá trị phù hợp phụ thuộc rất nhiều vào kích cỡ và hình dạng cục bộ của manifold của dữ liệu.

¹²Ma trận thưa là ma trận mà đa số các phần tử là 0. Không có định nghĩa chặt chẽ bao nhiêu phần tử cần bằng 0 để ma trận được coi là thưa nhưng với một tiêu chí chung là số phần tử khác 0 phải xấp xỉ bằng số hàng hoặc các cột.

¹³May mắn thay, LLE vẫn cho kết quả khá tốt trên một khoảng k khá rộng.

6 Phụ lục

6.1 Đại số tuyến tính

6.1.1 Định thức

(Điều kiện khả nghịch) Cho ma trận $A = [a_{ij}] \in M_{n \times n}(\mathbb{R})$. Ta có: $\det A \neq 0 \Leftrightarrow A$ khả nghịch.

Chứng minh:

(\Rightarrow) Xét ma trận $B = [A_{\neq ij}]$, trong đó $A_{\neq ij}$ bằng $(-1)^{i+j}$ nhân với định thức ma trận A sau khi bỏ đi hàng i và cột j . Gọi d_{ik} là định thức của A khi thay hàng i bởi hàng k . Đặt $C = B^T A$, ta có:

$$C_{ij} = \sum_{k=1}^n B_{ik} a_{kj} = \sum_{k=1}^n A_{\neq ki} a_{kj} = d_{ij} = \begin{cases} \det(A) & \text{khi } i = j \\ 0 & \text{khi } i \neq j \end{cases} \quad (9)$$

do đó $C = \det(A)I$. Vì $\det(A) \neq 0$ nên $\frac{1}{\det(A)} B^T A = I$, từ đó suy ra ma trận A khả nghịch.

(\Leftarrow) Ma trận A khả nghịch nên tồn tại ma trận B thỏa mãn $AB = I$. Vì thế:

$$1 = \det(I) = \det(AB) = \det(A)\det(B) \quad (10)$$

do đó $\det(A) \neq 0$

(Định thức và giá trị riêng) Cho ma trận $A \in M_{n \times n}(\mathbb{R})$. Định thức của A bằng tích các giá trị riêng (kể cả bội) của A và $(-1)^n$.

Chứng minh:

Giả sử $\lambda_1, \lambda_2, \dots, \lambda_n$ là các giá trị riêng của A , khi đó chúng là nghiệm của phương trình ẩn λ là $\det(A - \lambda I) = 0$. Sử dụng khai triển phần phụ để tính định thức cho $\det(A - \lambda I)$, ta sẽ nhận được đa thức bậc n biến λ , gọi là $P(\lambda)$. Đa thức này có hệ số cao nhất là $(-1)^n$ và hệ số tự do là $P(0) = \det(A)$. Theo định lý Viète:

$$\lambda_1 \lambda_2 \dots \lambda_n = \frac{\det(A)}{(-1)^n} = (-1)^n \det(A). \quad (11)$$

6.1.2 Ma trận Gram

(Định nghĩa ma trận Gram) G được gọi là ma trận Gram nếu G có thể biểu diễn dưới dạng $A^T A$.

(Điều kiện xác định dương của ma trận Gram) Ma trận Gram nửa xác định dương. Ma trận Gram xác định dương khi và chỉ khi các vector tạo thành ma trận Gram độc lập tuyến tính.

Chứng minh:

Với mọi vector $v \in \mathbb{R}^n$, ta có:

$$v^T G v = v^T A^T A v = (Av)^T (Av) = \|Av\|^2 \geq 0 \quad (12)$$

Như vậy G là ma trận nửa xác định dương.

G là nửa xác định dương, khi đó G xác định dương khi và chỉ khi $\det(G) > 0$, điều này tương đương với các vector tạo thành ma trận Gram độc lập tuyến tính.

Mặt khác, G xác định dương nếu với vector v , $v^T G v = 0$ thì $v = 0$, tương đương với mệnh đề $\|Av\| = 0$ kéo theo $v = 0$, hay là $Av = 0$ kéo theo $v = 0$, có nghĩa là các vector tạo thành A sẽ độc lập tuyến tính.

6.1.3 Ma trận đối xứng

Tính chất: các giá trị riêng của ma trận đối xứng thực luôn là số thực.

Chứng minh:

Gọi λ, v lần lượt là giá trị riêng và vector riêng (có thể phức) của ma trận thực đối xứng A . Ta có $A = A^T = \bar{A} = \bar{A}^T$ và:

$$\lambda \bar{v}^T v = \bar{v}^T (\lambda v) = \bar{v}^T (Av) = \bar{v}^T \bar{A}^T v = (\bar{A}v)^T v = (\bar{\lambda} \bar{v})^T v = \bar{\lambda} \bar{v}^T v \quad (13)$$

trong đó $\bar{\lambda}$ là liên hợp của λ trong tập số phức \mathbb{C} , \bar{v} là vector phức có các thành phần tương ứng liên hợp với v , \bar{A} là ma trận có các phần tử tương ứng liên hợp với A .

Vì $\bar{v}^T v$ là chuẩn của v và ta có $v \neq 0$ nên $\bar{v}^T v \neq 0$, từ đó có được $\bar{\lambda} = \lambda$, do vậy $\lambda \in \mathbb{R}$

6.1.4 Chính quy hóa

Cho biết ma trận X và vector y . Tìm vector w sao cho $\epsilon(w) = \|Xw - y\|^2$ đạt min.

Bài toán trên là một bài toán Linear Square Error - đi tìm trọng số chuẩn mực. Tuy nhiên trong những trường hợp bài toán trên có vô số nghiệm, để xác định nghiệm duy nhất, ta sử dụng phương pháp chính quy hóa, tức thêm thông tin về nghiệm. Chính quy hóa cho phép ta lợi dụng việc bài toán có vô số nghiệm để tìm ra nghiệm duy nhất có tính chất ta muốn. Ý tưởng của chính quy hóa là di chuyển nghiệm tối ưu đến một điểm gần nó để xác định nghiệm duy nhất (tránh trường hợp tồn tại vô số nghiệm tối ưu) và giảm độ phức tạp của thuật toán (để tránh overfit).

Đối với các bài toán tìm trọng số như trong LLE, ta có giả định rằng giữa vô số nghiệm cùng làm cho hàm mất mát tối ưu, việc ưu tiên chọn nghiệm có giá trị các trọng số gần bằng

nhau sẽ cho kết quả thuật toán tốt hơn. Đặt $X = \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix}$ (với x_1, \dots, x_n là các vector cột) và

$w = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$ (với $w_1, \dots, w_n \in \mathbb{R}$). Khi $w_1 \approx w_2 \approx \dots \approx w_n$ thì tầm quan trọng, hay mức độ

đóng góp, trong việc xấp xỉ y của x_1, \dots, x_n là gần như nhau, và ta giả định rằng điều này là tốt cho thuật toán.

Nhận thấy $w_1 \approx w_2 \approx \dots \approx w_n$ khi norm của w nhỏ (hệ quả của bất đẳng thức Cauchy), ta thêm đại lượng chính quy hóa $\alpha \|w\|^2$ vào hàm mất mát và ta đi tối ưu hóa hàm mất mát đã được chính quy hóa này:

$$\epsilon^*(w) = \|Xw - y\|^2 + \alpha \|w\|^2$$

Với α là một số thực dương, gọi là bậc chính quy hóa, thể hiện tầm quan trọng của tính chất norm nhỏ so với việc tìm nghiệm tối ưu hàm mất mát gốc. Nếu α lớn, ta coi trọng tính chất norm nhỏ hơn, đồng nghĩa với việc nghiệm tối ưu mới sẽ càng xa so với nghiệm tối ưu ban đầu. Ngược lại, ta chỉ coi trọng việc tối ưu hàm mất mát gốc, và tính chất norm nhỏ không còn quan trọng nữa. Vì vậy ta phải chọn α sao cho phù hợp; thường ta chọn α trong khoảng $[10^{-3}, 10^{-1}]$.

6.2 Giải tích ma trận

(Định nghĩa đạo hàm của hàm vô hướng theo ma trận)

Xét hàm số $f(\mathbf{X}): \mathbb{R}^{m \times n} \rightarrow \mathbb{R}$. Ta kí hiệu:

$$\frac{\partial f}{\partial \mathbf{X}} = \left[\frac{\partial f}{\partial X_{ij}} \right]_{m \times n}$$

là đạo hàm của hàm f theo ma trận \mathbf{X}

Một số biểu thức đạo hàm dùng trong báo cáo:

1. $\frac{\partial}{\partial \mathbf{X}} \text{Tr}(\mathbf{X}^T \mathbf{B} \mathbf{X}) = \mathbf{B} \mathbf{X} + \mathbf{B}^T \mathbf{X}$
2. $\frac{\partial}{\partial \mathbf{X}} \text{Tr}(\mathbf{B} \mathbf{X}^T \mathbf{X}) = \mathbf{X} \mathbf{B}^T + \mathbf{X} \mathbf{B}$

6.3 Lựa chọn siêu tham số neighbor - k

Ở các bước trong thuật toán LLE nêu trên có nhắc đến hyper-parameter k (số neighbors của một điểm dữ liệu), thế làm cách nào để xác định được siêu tham số k phù hợp cho từng tập dữ liệu? Đây là một phương pháp được sử dụng phổ biến: **Phương Sai Thặng Dư**.

Phương sai thặng dư dùng để tính độ sai lệch giữa kết quả thực tế và kết quả dự kiến; trong bài toán LLE, khoảng cách giữa các điểm dữ liệu ban đầu tượng trưng cho “dự kiến” còn khoảng cách giữa các điểm dữ liệu sau khi giảm chiều tượng trưng cho “thực tế”. Do đó, giá trị k phù hợp là giá trị k cho giá trị phương sai thặng dư bé nhất. Giả sử $k \in \{1, 2, \dots, k_{max}\}$. Với mỗi giá trị k , dùng phương pháp LLE để tìm tập nén Y của tập dữ liệu X ban đầu. Gọi D_X và D_Y lần lượt là ma trận khoảng cách các phần tử trong X và Y , ρ_{D_X, D_Y}^2 là độ tương giao tuyến tính giữa D_X và D_Y . Ta có $\rho_{D_X, D_Y}^2 := \frac{S_{D_X, D_Y}}{S_{D_X} S_{D_Y}}$ (S là độ lệch chuẩn). Phương sai thặng dư theo biến k được xác định bởi công thức:

$$\sigma_k^2(D_X, D_Y) := 1 - \rho_{D_X, D_Y}^2$$

Giá trị k phù hợp nhất cho tập dữ liệu X là giá trị cho kết quả phương sai thặng dư bé nhất. Nghĩa là:

$$k := \underset{x}{\operatorname{argmin}} \sigma_k^2(D_X, D_Y)$$

Tuy nhiên, ta có thể tránh việc thử với tất cả giá trị $k \in \{1, 2, \dots, k_{max}\}$ bằng cách phân nhóm theo thứ bậc. Ta tính biểu thức $L(W)$ 1 với từng $k \in \{1, 2, \dots, k_{max}\}$ và rút ra các biên k cho giá trị $L(W)$ bé hơn so với khi biên là $k - 1$ hoặc $k + 1$ (tức điểm cực tiểu). Lúc này, thực hiện quy trình như trên mới các điểm cực tiểu thu được suy ra giá trị k phù hợp.

Tài liệu

- [1] Benyamin Ghojogh, Fakhri Karray, and Mark Crowley. March 25, 2019. *Eigenvalue and Generalized Eigenvalue Problems: Tutorial*. <https://arxiv.org/pdf/1903.11240.pdf><https://arxiv.org/pdf/1903.11240.pdf>.
- [2] Lê Quang Tiến. August 25, 2018. *Locally Linear Embedding (LLE)*. <https://thetalog.com/machine-learning/locally-linear-embedding/><https://thetalog.com/machine-learning/locally-linear-embedding/>.
- [3] Jianzhong Wang. 2012. *Geometric Structure of High-Dimensional Data and Dimensionality Reduction*. <https://bit.ly/JianzhongWang2012><https://bit.ly/JianzhongWang2012>.
- [4] Benyamin Ghojogh, Ali Ghodsi, Fakhri Karray, and Mark Crowley. November 22, 2020. *Locally Linear Embedding and its Variants: Tutorial and Survey*. <https://bit.ly/LLEtutupdated><https://bit.ly/LLEtutupdated>.
- [5] Sam T. Roweis and Lawrence K. Saul <https://cs.nyu.edu/roweis/lle/algorithm.html><https://cs.nyu.edu/roweis/lle/algorithm.html>.
- [6] Ashwini Kumar Pal. 2018. <https://blog.paperspace.com/dimension-reduction-with-lle/><https://blog.paperspace.com/dimension-reduction-with-lle/>.