

THUẬT TOÁN

Linear Discriminant Analisis

TITLE

ĐỖ VĂN TRƯỜNG

USERNAME

CLASS

CNTT 14 - 05

Slide thuật toán

CONTENT

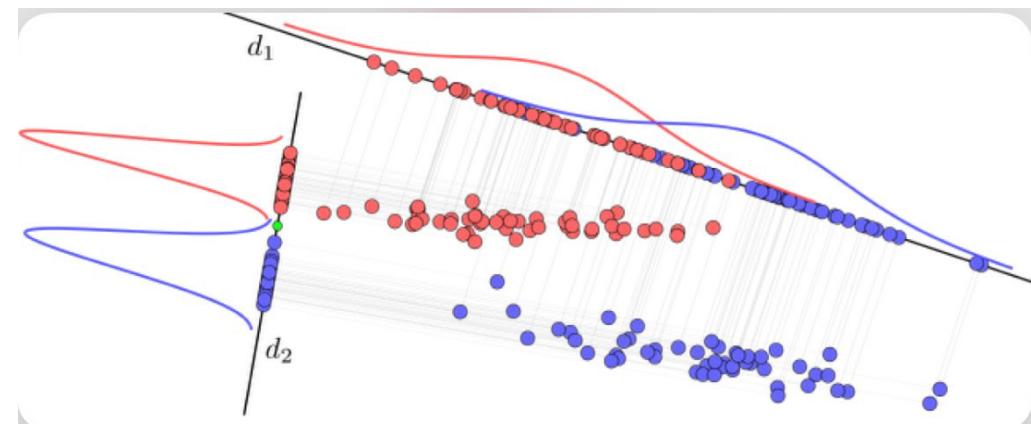
Research ->

I. GIỚI THIỆU VỀ THUẬT TOÁN

Hôm nay tôi sẽ trình bày về thuật toán Linear Discriminant Analisis

Linear Discriminant Analysis (LDA) được ra đời nhằm giải quyết vấn đề này. LDA là một phương pháp giảm chiều dữ liệu cho bài toán classification. LDA có thể được coi là một phương pháp giảm chiều dữ liệu (dimensionality reduction), và cũng có thể được coi là một phương pháp phân lớp (classification), và cũng có thể được áp dụng đồng thời cho cả hai, tức giảm chiều dữ liệu sao cho việc phân lớp hiệu quả nhất. Số chiều của dữ liệu mới là nhỏ hơn hoặc bằng $C-1$ trong đó C là số lượng classes. Từ 'Discriminant' được hiểu là những thông tin đặc trưng cho mỗi class, khiến nó không bị lẫn với các classes khác. Từ 'Linear' được dùng vì cách giảm chiều dữ liệu được thực hiện bởi một ma trận chiếu (projection matrix), là một phép biến đổi tuyến tính (linear transform).

Trong Mục 2 dưới đây, tôi sẽ trình bày về trường hợp binary classification, tức có 2 classes. Mục 3 sẽ tổng quát lên cho trường hợp với nhiều classes hơn 2. Mục 4 sẽ có các ví dụ và code Python cho LDA.



II. Linear Discriminant Analysis cho bài toán với 2 classes

2.1. Ý tưởng cơ bản

Mọi phương pháp classification đều được bắt đầu với bài toán binary classification, và LDA cũng không phải ngoại lệ.

Quay lại với Hình 1, các đường hình chuông thể hiện đồ thị của các hàm mật độ xác suất (probability density function - pdf) của dữ liệu được chiếu xuống theo từng class. Phân phối chuẩn ở đây được sử dụng như là một đại diện, dữ liệu không nhất thiết luôn phải tuân theo phân phối chuẩn.

Độ rộng của mỗi đường hình chuông thể hiện độ lệch chuẩn của dữ liệu. Dữ liệu càng tập trung thì độ lệch chuẩn càng nhỏ, càng phân tán thì độ lệch chuẩn càng cao. Khi được chiếu lên

d_1 , dữ liệu của hai classes bị phân tán quá nhiều, khiến cho chúng bị trộn lẫn vào nhau. Khi được chiếu lên

d_2 , mỗi classes đều có độ lệch chuẩn nhỏ, khiến cho dữ liệu trong từng class tập trung hơn, dẫn đến kết quả tốt hơn.

Tuy nhiên, việc độ lệch chuẩn nhỏ trong mỗi class chưa đủ để đảm bảo độ Discriminant của dữ liệu. Xét các ví dụ trong Hình 2.

Hình 2: Khoảng cách giữa các kỳ vọng và tổng các phương sai ảnh hưởng tới độ discriminant của dữ liệu. a) Khoảng cách giữa hai kỳ vọng là lớn nhưng phương sai trong mỗi class cũng lớn, khiến cho hai phân phối chồng lấn lên nhau (phản màu xám). b) Phương sai cho mỗi class là rất nhỏ nhưng hai kỳ vọng quá gần nhau, khiến khó phân biệt 2 class. c) Khi phương sai đủ nhỏ và khoảng cách giữa hai kỳ vọng đủ lớn, ta thấy rằng dữ liệu discriminant hơn.

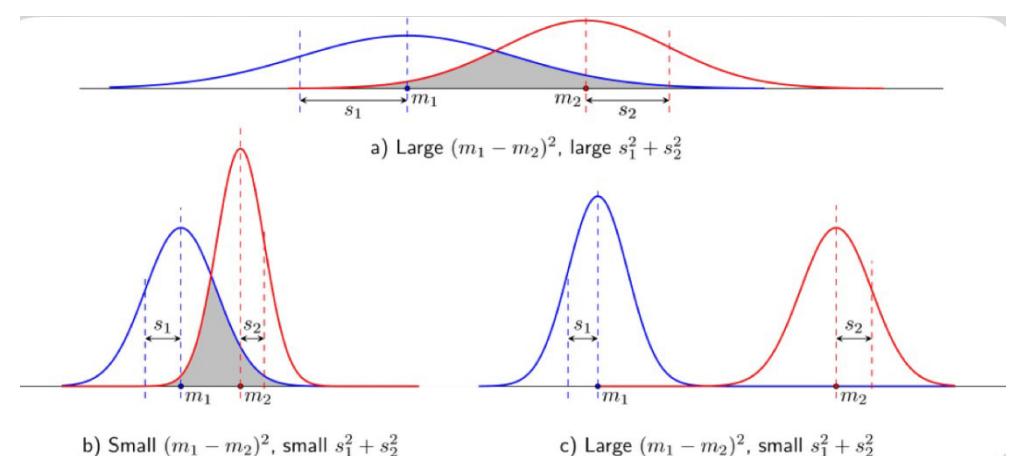
Hình 2a) giống với dữ liệu khi chiếu lên

d_1

Hình 2b) là trường hợp khi độ lệch chuẩn của hai class đều nhỏ, tức dữ liệu tập trung hơn. Tuy nhiên, vấn đề với trường hợp này là khoảng cách giữa hai class, được đo bằng khoảng cách giữa hai kỳ vọng m_1 và m_2 , là quá nhỏ, khiến cho phần chồng lấn cũng chiếm một tỉ lệ lớn, và tất nhiên, cũng không tốt cho classification.

Hình 2c) là trường hợp khi hai độ lệch chuẩn là nhỏ và khoảng cách giữa hai kỳ vọng là lớn, phần chồng lấn nhỏ không đáng kể.

Có thể bạn đang tự hỏi, độ lệch chuẩn và khoảng cách giữa hai kỳ vọng đại diện cho các tiêu chí gì:



Có thể bạn đang tự hỏi, độ lệch chuẩn và khoảng cách giữa hai kỳ vọng đại diện cho các tiêu chí gì:

Như đã nói, độ lệch chuẩn nhỏ thể hiện việc dữ liệu ít phân tán. Điều này có nghĩa là dữ liệu trong mỗi class có xu hướng giống nhau. Hai phương sai s_1^2, s_2^2 còn được gọi là các **within-class variances**.

Khoảng cách giữa các kỳ vọng là lớn chứng tỏ rằng hai classes nằm xa nhau, tức dữ liệu giữa các classes là khác nhau nhiều. Bình phương khoảng cách giữa hai kỳ vọng ($m_1 - m_2$)² còn được gọi là between-class variance

1.1.2.1

2.2. Xây dựng hàm mục tiêu

Giả sử rằng có N điểm dữ liệu $x^{(1)}, x^{(2)}, \dots, x^{(N)}$ trong đó $N < N$ điểm đầu tiên thuộc class thứ nhất, $N = N - N$ điểm cuối cùng thuộc class thứ hai. Ký hiệu $C1 = \{n | 1 \leq n \leq N\}$ là tập hợp các chỉ số của các điểm thuộc class 1 và $C2 = \{m | N+1 \leq m \leq N\}$ là tập hợp các chỉ số của các điểm thuộc class 2. Phép chiếu dữ liệu xuống 1 đường thẳng có thể được mô tả bằng một vector hệ số w , giá trị tương ứng của mỗi điểm dữ liệu mới được cho bởi:

Chú ý rằng các within-class variances ở đây không được lấy trung bình như variance thông thường. Điều này được lý giải là tầm quan trọng của mỗi within-class variance nên tỉ lệ thuận với số lượng điểm dữ liệu trong class đó, tức within-class variance bằng variance nhân với số điểm trong class đó. Thế nên ta không chia trung bình nữa.

LDA là thuật toán đi tìm giá trị lớn nhất của hàm mục tiêu:

$$J(w) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2} \quad (4)$$

Tiếp theo, chúng ta sẽ đi tìm biểu thức phụ thuộc giữa tử số và mẫu số trong vế phải của (4) vào w .

Với tử số:

$$(m_1 - m_2)^2 = w^T \underbrace{(m_1 - m_2)(m_1 - m_2)^T w}_{S_B} = w^T S_B w \quad (5)$$

S_B còn được gọi là **between-class covariance matrix**. Đây là một ma trận đối xứng nửa xác định dương.

Với mẫu số:

$$s_1^2 + s_2^2 = \sum_{k=1}^2 \sum_{n \in C_k} (w^T (x_n - \bar{m}_k))^2 = w^T \underbrace{\sum_{k=1}^2 \sum_{n \in C_k} (x_n - \bar{m}_k)(x_n - \bar{m}_k)^T}_{S_W} w = w^T S_W w \quad (6)$$

S_w còn được gọi là **within-class covariance matrix**. Đây cũng là một ma trận đối xứng nửa xác định dương vì nó là tổng của hai ma trận đối xứng nửa xác định dương.

2.3. Nghiệm của bài toán tối ưu

Nghiệm w của (7) sẽ là nghiệm của phương trình đạo hàm hàm mục tiêu bằng 0. Sử dụng chain rule cho đạo hàm hàm nhiều biến và công thức $\nabla_w w^T A w = 2A w$ nếu A là một ma trận đối xứng, ta có:

$$\nabla_w J(w) = \frac{1}{(w^T S_w w)^2} (2S_B w (w^T S_w w) - 2w^T S_B w^T S_w w) = 0 \quad (8) \Leftrightarrow S_B w = \frac{w^T S_B w}{w^T S_w w} S_w w \quad (9) \quad S_w^{-1} S_B w = J(w) w \quad (10)$$

Lưu ý: Trong (10), ta đã giả sử rằng ma trận S_w là khả nghịch. Điều này không luôn luôn đúng, nhưng có một trick nhỏ là ta có thể xấp xỉ S_w bởi $\tilde{S}_w \approx S_w + \lambda I$ với λ là một số thực dương nhỏ. Ma trận mới này là khả nghịch vì trị riêng nhỏ nhất của nó bằng với trị riêng nhỏ nhất của S_w cộng với λ tức không nhỏ hơn $\lambda > 0$. Điều này được suy ra từ việc S_w là một ma trận nửa xác định dương. Từ đó suy ra \tilde{S}_w là một ma trận xác định dương vì mọi trị riêng của nó là thực dương, và vì thế, nó khả nghịch. Khi tính toán, ta có thể sử dụng nghịch đảo của \tilde{S}_w .

Kỹ thuật này được sử dụng rất nhiều khi ta cần sử dụng nghịch đảo của một ma trận nửa xác định dương và chưa biết nó có thực sự là xác định dương hay không.

Quay trở lại với (10), vì $J(w)$ là một số vô hướng, ta suy ra w phải là một vector riêng của $S_w^{-1} S_B$ ứng với một trị riêng nào đó. Hơn nữa, giá trị của trị riêng này bằng với $J(w)$. Vậy, để hàm mục tiêu là lớn nhất thì $J(w)$ chính là trị riêng lớn nhất của $S_w^{-1} S_B$. Dấu bằng xảy ra khi w là vector riêng ứng với trị riêng lớn nhất đó. Bạn đọc có thể hiểu phần này hơn khi xem cách lập trình trên Python ở Mục 4.

Từ có thể thấy ngay rằng nếu w là nghiệm của (7) thì $k w$ cũng là nghiệm với k là số thực khác không bất kỳ. Vậy ta có thể chọn w sao cho $(m_1 - m_2)^T w = J(w) = L =$ trị riêng lớn nhất của $S_w^{-1} S_B$. Khi đó, thay định nghĩa của S_B ở (5) vào (10) ta có:

$$Lw = S_w^{-1} (m_1 - m_2) (m_1 - m_2)^T w = L S_w^{-1} (m_1 - m_2)$$

3. Linear Discriminant Analysis cho multi-class classification problems

3.1. Xây dựng hàm mất mát

Trong mục này, chúng ta sẽ xem xét trường hợp tổng quát khi có nhiều hơn 2 classes. Giả sử rằng chiều của dữ liệu D lớn hơn số lượng classes C.

Giả sử rằng chiều mà chúng ta muốn giảm về là D' và dữ liệu mới ứng với mỗi điểm dữ liệu x là:

$$y = \mathbf{W}^T \mathbf{x}$$

với $\mathbf{W} \in \mathbb{R}^{D \times D'}$.

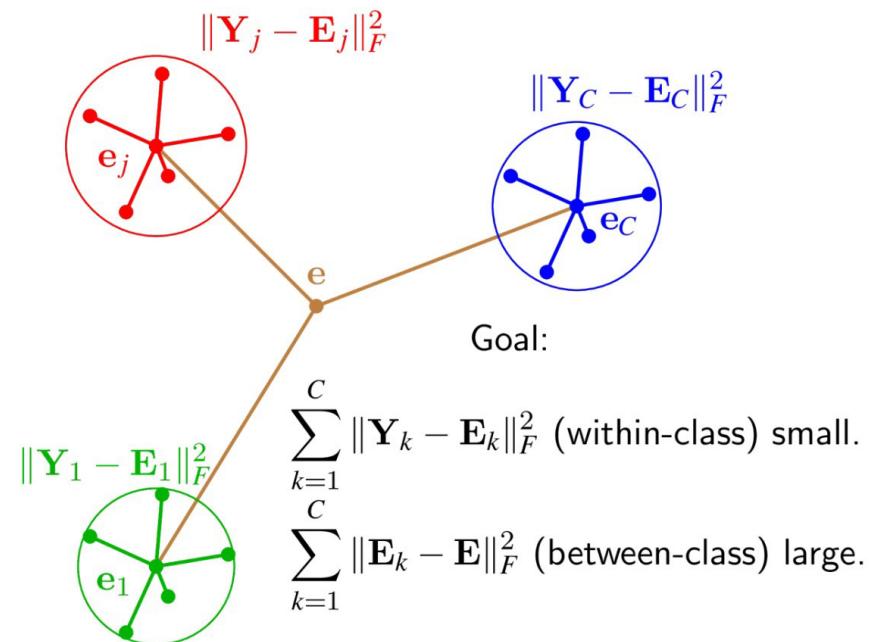
Chú ý rằng LDA ở đây không sử dụng bias.

Một vài ký hiệu:

- $\mathbf{X}_k, \mathbf{Y}_k = \mathbf{W}^T \mathbf{X}_k$ lần lượt là ma trận dữ liệu của class k trong không gian ban đầu và không gian mới với số chiều nhỏ hơn.
- $\mathbf{m}_k = \frac{1}{N_k} \sum_{n \in \mathcal{C}_k} \mathbf{x}_n \in \mathbb{R}^{D'}$ là vector kỳ vọng của class k trong không gian ban đầu.
- $\mathbf{e}_k = \frac{1}{N_k} \sum_{n \in \mathcal{C}_k} \mathbf{y}_n = \mathbf{W}^T \mathbf{m}_k \in \mathbb{R}^{D'}$ là vector kỳ vọng của class k trong không gian mới.
- \mathbf{m} là vector kỳ vọng của toàn bộ dữ liệu trong không gian ban đầu và \mathbf{e} là vector kỳ vọng trong không gian mới.

Một trong những cách xây dựng hàm mục tiêu cho multi-class LDA được minh họa trong Hình 3.

Hình 3: LDA cho multi-class classification problem. Mục đích cũng là sự khác nhau giữa các thành phần trong 1 class (within-class) là nhỏ và sự khác nhau giữa các classes là lớn. Các điểm dữ liệu có màu khác nhau thể hiện các class khác nhau.



Độ phân tán của một tập hợp dữ liệu có thể được coi như tổng bình phương khoảng cách từ mỗi điểm tới vector kỳ vọng của chúng. Nếu tất cả các điểm đều gần vector kỳ vọng của chúng thì độ phân tán của tập dữ liệu đó được coi là nhỏ. Ngược lại, nếu tổng này là lớn, tức trung bình các điểm đều xa trung tâm, tập hợp này có thể được coi là có độ phân tán cao.

Dựa vào nhận xét này, ta có thể xây dựng các đại lượng:

3.1.1. Within-class nhỏ

Within-class variance của class k có thể được tính như sau:

$$\sigma_k^2 = \sum_{n \in \mathcal{C}_k} \|\mathbf{y}_n - \mathbf{e}_k\|_F^2 = \|\mathbf{Y}_k - \mathbf{E}_k\|_F^2 \quad (15) \quad = \|\mathbf{W}^T(\mathbf{X}_k - \mathbf{M}_k)\|_F^2 \quad (16) \quad = \text{trace}(\mathbf{W}^T(\mathbf{X}_k - \mathbf{M}_k)(\mathbf{X}_k - \mathbf{M}_k)^T \mathbf{W})$$

Với E_k một ma trận có các cột giống hệt nhau và bằng với vector kỳ vọng e_k . Có thể nhận thấy $E_k = WTM_k$ với M_k là ma trận có các cột giống hệt nhau và bằng với vector kỳ vọng m_k trong không gian ban đầu.

Vậy đại lượng đo within-class trong multi-class LDA có thể được đo bằng:

$$s_W = \sum_{k=1}^C \sigma_k^2 = \sum_{k=1}^C \text{trace} (\mathbf{W}^T (\mathbf{X}_k - \mathbf{M}_k) (\mathbf{X}_k - \mathbf{M}_k)^T \mathbf{W}) \quad (17) = \text{trace} (\mathbf{W}^T \mathbf{S}_W \mathbf{W}) \quad (18)$$

với:

$$\mathbf{S}_W = \sum_{k=1}^C \|\mathbf{X}_k - \mathbf{M}_k\|_F^2 = \sum_{k=1}^C \sum_{n \in \mathcal{C}_k} (\mathbf{x}_n - \mathbf{m}_k) (\mathbf{x}_n - \mathbf{m}_k)^T \quad (19)$$

và nó có thể được coi là within-class covariance matrix của multi-class LDA. Ma trận S_B này là một ma trận nửa xác định dương theo định nghĩa.

3.1.2. Between-class lớn

Việc betwwen-class lớn, như đã đề cập, có thể đạt được nếu tất cả các điểm trong không gian mới đều xa vector kỳ vọng chung e . Việc này cũng có thể đạt được nếu các vector kỳ vọng của mỗi class xa các vector kỳ vọng chung (trong không gian mới). Vậy ta có thể định nghĩa đại lượng between-class như sau:

$$s_B = \sum_{k=1}^C N_k \|\mathbf{e}_k - \mathbf{e}\|_F^2 = \sum_{k=1}^C \|\mathbf{E}_k - \mathbf{E}\|_F^2 \quad (20)$$

Ta lấy N_k làm trọng số vì có thể có những class có nhiều phần tử so với các classes còn lại.

Chú ý rằng ma trận E có thể có số cột linh động, phụ thuộc vào số cột của ma trận E_k mà nó đi cùng (và bằng N_k).

Lập luận tương tự như (17),(18), bạn đọc có thể chứng minh được:

$$s_B = \text{trace} (\mathbf{W}^T \mathbf{S}_B \mathbf{W}) \quad (21)$$

$$\mathbf{S}_B = \sum_{k=1}^C (\mathbf{M}_k - \mathbf{M})(\mathbf{M}_k - \mathbf{M})^T = \sum_{k=1}^C N_k (\mathbf{m}_k - \mathbf{m})(\mathbf{m}_k - \mathbf{m})^T \quad (22)$$

và số cột của ma trận M cũng linh động theo số cột của M_k . Ma trận này là tổng của các ma trận đối xứng nửa xác định dương, nên nó là một ma trận đối xứng nửa xác định dương.

3.2. Hàm mất mát cho multi-class LDA

Với cách định nghĩa và ý tưởng về within-class nhỏ và between-class lớn như trên, ta có thể xây dựng bài toán tối ưu:

$$\mathbf{W} = \arg \max_{\mathbf{W}} J(\mathbf{W}) = \arg \max_{\mathbf{W}} \frac{\text{trace} (\mathbf{W}^T \mathbf{S}_B \mathbf{W})}{\text{trace} (\mathbf{W}^T \mathbf{S}_W \mathbf{W})}$$

Nghiệm cũng được tìm bằng cách giải phương trình đạo hàm hàm mục tiêu bằng 0. Nhắc lại về đạo hàm của hàm trace theo ma trận:

$$\nabla_{\mathbf{W}} \text{trace} (\mathbf{W}^T \mathbf{A} \mathbf{W}) = 2\mathbf{A}\mathbf{W}$$

với $\mathbf{A} \in \mathbb{R}^{D \times D}$ là một ma trận đối xứng. (Xem trang 597 của tài liệu này).

Với cách tính tương tự như (8)–(10), ta có:

$$\nabla_{\mathbf{W}} J(\mathbf{W}) = \frac{2 \left(\mathbf{S}_B \mathbf{W} \text{trace} (\mathbf{W}^T \mathbf{S}_W \mathbf{W}) - \text{trace} (\mathbf{W}^T \mathbf{S}_B \mathbf{W}) \mathbf{S}_W \mathbf{W} \right)}{\left(\text{trace} (\mathbf{W}^T \mathbf{S}_W \mathbf{W}) \right)^2} = \mathbf{0} \quad (23) \Leftrightarrow \mathbf{S}_W^{-1} \mathbf{S}_B \mathbf{W} = J\mathbf{W} \quad (24)$$

Từ đó suy ra mỗi cột của W là một vector riêng của $S_B^T S_B$ ứng với trị riêng lớn nhất của ma trận này.

Nhận thấy rằng các cột của W cần phải độc lập tuyến tính. Vì nếu không, dữ liệu trong không gian mới $\tilde{y} = Wx$ sẽ thuộc tuyến tính và có thể tiếp tục được giảm số chiều mà không ảnh hưởng gì.

Vậy các cột của W là các vector độc lập tuyến tính ứng với trị riêng cao nhất của $S_B^T S_B$. Câu hỏi đặt ra là: Có nhiều nhất bao nhiêu vector riêng độc lập tuyến tính ứng với trị riêng lớn nhất của $S_B^T S_B$?

Số lượng lớn nhất các vector riêng độc lập tuyến tính ứng với 1 trị riêng chính là rank của không gian riêng ứng với trị riêng đó, và không thể lớn hơn rank của ma trận.

Ta có một bổ đề quan trọng:

Bổ đề:

$$\text{rank}(S_B) \leq C - 1 \quad (23)$$

Chứng minh: (Tuy nhiên, việc chứng minh này không thực sự quan trọng, chỉ phù hợp với những bạn muốn hiểu sâu)

Viết lại (22) dưới dạng:

$$S_B = P P^T$$

với $P \in R^{D \times C}$ mà cột thứ k của nó là:

$$p_k = \sqrt{N_k}(\mathbf{m}_k - \mathbf{m})$$

Thêm nữa, cột cuối cùng là một tổ hợp tuyến tính của các cột còn lại. Lý do là:

$$\mathbf{m}_C - \mathbf{m} = \mathbf{m}_C - \frac{\sum_{k=1}^C N_k \mathbf{m}_k}{N} = \sum_{k=1}^{C-1} \frac{N_k}{N} (\mathbf{m}_k - \mathbf{m})$$

Như vậy ma trận P có nhiều nhất $C-1$ cột độc lập tuyến tính, vậy nên rank của nó không vượt quá $C-1$.

Cuối cùng, S_B là tích của hai ma trận với rank không quá $C-1$, nên rank (S_B) không vượt quá $C-1$.

Nếu bạn cần ôn lại vài kiến thức về rank:

- Hạng (rank) của một ma trận, không nhất thiết vuông, là số lượng lớn nhất các cột độc lập tuyến tính của ma trận đó. Vậy nên rank của một ma trận không thể lớn hơn số cột của ma trận đó.
- $\text{rank}(A) = \text{rank}(A^T)$. Vậy nên số lượng lớn nhất các cột độc lập tuyến tính cũng chính bằng số lượng lớn nhất các hàng độc lập tuyến tính.
- $\text{rank}(AB) \leq \min\{\text{rank}(A), \text{rank}(B)\}$ với A, B là hai ma trận bất kỳ có thể nhân với nhau được.
- $\text{rank}(A+B) \leq \text{rank}(A) + \text{rank}(B)$ với A, B là hai ma trận cùng chiều bất kỳ.

Từ đó ra có $\text{rank}(S_B^{-1} S_B) \leq \text{rank} S_B \leq C - 1$.

Vậy số chiều của không gian mới là một số không lớn hơn $C-1$. Xem ra số chiều theo multi-class LDA đã được giảm đi rất nhiều. Nhưng chất lượng của dữ liệu mới như thế nào, chúng ta cần làm một vài thí nghiệm.

Tóm lại, nghiệm của bài toán multi-class LDA là các vector riêng độc lập tuyến tính ứng với trị riêng cao nhất của $S_B^{-1} S_B$.

Lưu ý: Có nhiều cách khác nhau để xây dựng hàm mục tiêu cho multi-class LDA dựa trên việc định nghĩa within-class variance nhỏ và between-class variance lớn. Chúng ta đang sử dụng hàm trace để đong đếm hai đại lượng này. Có nhiều cách khác nữa, ví dụ như sử dụng định thức. Tuy nhiên, có một điểm chung giữa các cách tiếp cận này là chiều của không gian mới sẽ không vượt quá $C-1$.

4. Ví dụ trên Python

4.1. LDA với 2 classes

Tạo dữ liệu giả:

```
# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals
# list of points
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.backends.backend_pdf import PdfPages
np.random.seed(22)

means = [[0, 5], [5, 0]]
cov0 = [[4, 3], [3, 4]]
cov1 = [[3, 1], [1, 1]]
N0 = 50
N1 = 40
N = N0 + N1
X0 = np.random.multivariate_normal(means[0], cov0, N0) # each row is a data point
X1 = np.random.multivariate_normal(means[1], cov1, N1)
```

Các điểm cho 2 classes này được minh họa bởi các điểm màu lam và đỏ trên Hình 4.

Tiếp theo, chúng ta đi tính các within-class và between-class covariance matrices:

```
# Build S_B
m0 = np.mean(X0.T, axis = 1, keepdims = True)
m1 = np.mean(X1.T, axis = 1, keepdims = True)

a = (m0 - m1)
S_B = a.dot(a.T)

# Build S_W
A = X0.T - np.tile(m0, (1, N0))
B = X1.T - np.tile(m1, (1, N1))

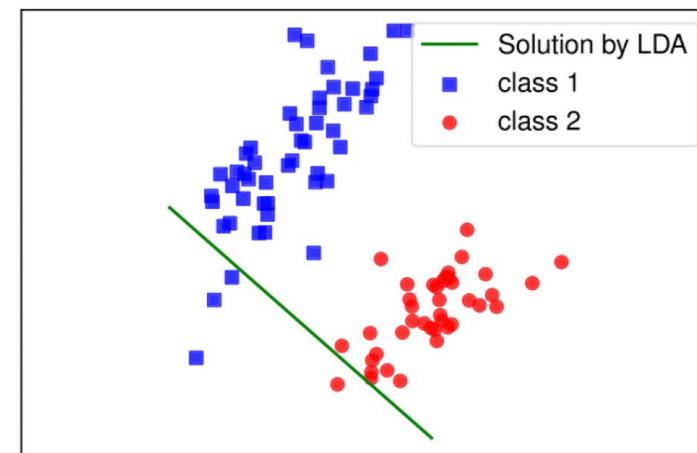
S_W = A.dot(A.T) + B.dot(B.T)
```

Nghiệm của bài toán là vector riêng ứng với trị riêng lớn nhất của $\text{np.linalg.inv}(S_W) \cdot S_B$

```
_> W = np.linalg.eig(np.linalg.inv(S_W).dot(S_B))
w = W[:,0]
print(w)

[ 0.75091074 -0.66040371]
```

Đường thẳng có phương w được minh họa bởi đường màu lục trên Hình 4. Ta thấy rằng nghiệm này hợp lý với dữ liệu có được.



Hình 4: Ví dụ minh họa về LDA trong không gian 2 chiều. Đường thẳng màu lục là đường thẳng mà dữ liệu sẽ được chiếu lên. Ta có thể thấy rằng, nếu chiếu lên được thẳng này, dữ liệu của hai classes sẽ nằm về hai phía của một điểm trên đường thẳng đó.
Để kiểm chứng độ chính xác của nghiệm tìm được, ta cùng so sánh nó với nghiệm tìm được bởi thư viện sklearn.

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis  
X = np.concatenate((X0, X1))  
y = np.array([0]*N0 + [1]*N1)  
clf = LinearDiscriminantAnalysis()  
clf.fit(X, y)  
  
print(clf.coef_ / np.linalg.norm(clf.coef_)) # normalize  
[[ 0.75091074 -0.66040371]]
```

Ta thấy rằng nghiệm tìm theo công thức và nghiệm tìm theo thư viện là như nhau. Như vậy việc phân tích ở Mục 2 là hoàn toàn chính xác.

Một ví dụ khác so sánh PCA và LDA có thể được tìm thấy tại đây: Comparison of LDA and PCA 2D projection of Iris dataset.

5. Tài liệu tham khảo

[1] Linear discriminant analysis.

[2] Bishop, Christopher M. "Pattern recognition and Machine Learning.", Springer (2006). Chapter 4. (book)

[3] Comparison of LDA and PCA 2D projection of Iris dataset

