

Bộ Giáo dục và Đào tạo
Trường Đại học Công nghệ - ĐHQGHN



Báo cáo về chủ đề **Classification Of Electrical Brain Signals**

Học phần: Tính toán khoa học thần kinh và ứng dụng (AIT3012)

Giảng Viên Hướng Dẫn: TS. Lê Vũ Hà

Ngành: Trí tuệ nhân tạo - QH-2022-I/CQ-A-AI

Người thực hiện: Nguyễn Bảo Sơn

Mã sinh viên: 22022613

Phần I. Loading Data	3
1.1. Giới thiệu Dữ liệu	3
1.2. Bandpass Filtering	7
1.3. Spatial Filtering	9
Phần II. Data Visualization	11
2.1. Các Hàm Vẽ Tín Hiệu	11
2.2. Các Hàm Áp Dụng t-SNE	12
2.3. Vẽ Tín Hiệu	12
Phần III. Feature Extraction	20
3.1. Data preparation	20
3.2. Using Common Spatial Patterns (CSP)	22
3.3. Principal Component Analysis (PCA)	31
3.4. Linear Discriminant Analysis (LDA)	36
3.5. Visual Comparisons	39
Phần IV. Classification	40
4.1. Apply 8 Classifiers	40
4.2. Một số bộ phân loại (với dữ liệu PCA) khác	44
Phần V. Clustering	49
5.1. Silhouette Score and Plot	49
5.2. K-Means:	52
5.3. DBSCAN	55

Phần I. Loading Data

1.1. Giới thiệu Dữ liệu

- Dữ liệu đầu vào
 - Dữ liệu A là “BCICIV_calib_ds1a.mat”
 - Dữ liệu B là “BCICIV_calib_ds1b.mat”
- Dữ liệu này chứa các tín hiệu EEG liên tục với tần số lấy mẫu là 100 Hz. Đối với dữ liệu hiệu chuẩn, có các dấu hiệu chỉ ra thời điểm trình bày kích thích và các lớp mục tiêu tương ứng. Dữ liệu được cung cấp dưới định dạng file BCICIV_calib_ds1a.mat và BCICIV_calib_ds1d.mat với các biến sau:
 1. **cnt**: Chứa tín hiệu EEG. Đây là ma trận có kích thước [thời gian x kênh] được lưu trữ dưới kiểu dữ liệu INT16. Để chuyển đổi các giá trị này sang microvolts (uV), ta sử dụng công thức sau:
cnt = 0.1 * double(cnt)
 2. **mrk**: Chứa thông tin về các dấu hiệu mục tiêu với các trường sau (dữ liệu đánh giá không bao gồm biến này):
 - **v pos**: Vị trí của các dấu hiệu trong tín hiệu EEG, tính bằng đơn vị mẫu.
 - **v y**: Các lớp mục tiêu (-1 cho lớp một hoặc 1 cho lớp hai).
 3. **nfo**: Chứa thông tin bổ sung với các trường sau:
 - **v fs**: Tần số lấy mẫu.
 - **v clab**: Mảng cell chứa nhãn các kênh.
 - **v classes**: Mảng cell chứa tên các lớp cho hình ảnh chuyển động.
 - **v xpos**: Tọa độ X của các vị trí điện cực trong một phép chiếu 2D.
 - **v ypos**: Tọa độ Y của các vị trí điện cực trong một phép chiếu 2D
- Đối với mỗi file được chọn, ta sử dụng vector **pos** để lấy điểm bắt đầu của mỗi cửa sổ thời gian. Chúng ta đã tải dữ liệu bằng cách sử dụng **scipy.io.loadmat**. Sau đó, ta trích xuất tín hiệu EEG từ cột ‘cnt’ và chuyển đổi chúng sang giá trị uV. Tiếp theo, ta trích xuất cấu trúc marker (vị trí và lớp), nếu có thể, và trích xuất cấu trúc thông tin bổ sung, bao gồm **nfo**, tần số lấy mẫu (fs), nhãn kênh (channel_name), lớp hình ảnh chuyển động (classes), tọa độ X của các điện cực (xpos) và tọa độ Y của các điện cực (ypos).
- Thông tin dữ liệu:

Dữ liệu A:

```
Continuous EEG signals (cnt): (190594, 59)
Positions of cues (pos): [ 2091  2891  3691  4491  5291  6091  6891  7692  8492  9292
10092 10892 11692 12492 13292 16294 17094 17894 18694 19494
20294 21094 21894 22694 23494 24294 25094 25894 26694 27494
30495 31295 32095 32895 33695 34495 35295 36095 36895 37695
38495 39295 40095 40895 41695 44696 45496 46296 47096 47896
48696 49496 50296 51096 51896 52696 53496 54296 55096 55896
58895 59695 60495 61295 62095 62895 63695 64495 65295 66095
66895 67695 68495 69295 70095 73094 73894 74694 75495 76295
77095 77895 78695 79495 80295 81095 81895 82695 83495 84295
87294 88094 88894 89694 90494 91294 92094 92894 93694 94494
97385 98185 98985 99785 100585 101385 102185 102985 103785 104585
105385 106185 106985 107785 108585 111584 112384 113184 113984 114784
115584 116384 117184 117984 118785 119585 120385 121185 121985 122785
125787 126587 127387 128187 128987 129787 130587 131387 132187 132987
133787 134587 135387 136187 136987 139986 140786 141586 142386 143186
143986 144786 145586 146386 147186 147986 148786 149586 150386 151186
154185 154985 155785 156585 157385 158185 158985 159785 160586 161386
162186 162986 163786 164586 165386 168385 169185 169985 170785 171585
172385 173185 173985 174785 175585 176385 177185 177985 178785 179585
182584 183384 184184 184984 185784 186584 187384 188184 188984 189784]
Target classes (y): [ 1 1 -1 1 1 1 1 -1 -1 1 -1 1 -1 -1 -1 1 -1 -1 1 -1 -1
-1 -1 -1 1 -1 -1 -1 -1 -1 -1 -1 -1 1 1 1 -1 -1 1 -1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 -1 -1 -1 -1 1 -1 1 -1 -1 -1 -1
1 -1 -1 -1 1 -1 1 1 -1 -1 1 1 1 -1 1 1 1 -1 -1 1 1 1 -1
1 1 -1 1 -1 1 1 1 -1 1 1 -1 1 -1 -1 -1 1 1 -1 1 1
1 1 -1 1 -1 -1 1 1 -1 -1 1 1 -1 1 -1 -1 -1 1 1 -1 1 1
1 1 -1 1 -1 -1 1 1 -1 -1 1 1 1 1 1 -1 -1 -1 1 -1 -1
1 -1 -1 -1 1 1 -1 1 1 1 1 1 1 -1 1 -1 1 -1 1 -1 -1 -1
-1 -1 1 1 1 1 -1 -1 -1 -1 1 1 -1 1 -1 -1 -1 1 -1 1 -1
-1 1 -1 -1 -1 1 1 1]
```

```
Sampling rate (fs): 100
Channel labels (clab): ['AF3', 'AF4', 'F5', 'F3', 'F1', 'Fz', 'F2', 'F4', 'F6', 'FC5', 'FC3', 'FC1', 'FCz', 'FC2', 'FC4', 'FC6', 'CFC7',
Motor imagery classes: ['left', 'foot']
X-positions of electrodes (xpos): [-0.20109028  0.20109028 -0.48547489 -0.32894737 -0.16535231  0.
0.16535231  0.32894737  0.48547489 -0.60591541 -0.39919579 -0.19765935
0.
0.19765935  0.39919579  0.60591541 -0.74834683 -0.52472976
-0.30963911 -0.10226303  0.10226303  0.30963911  0.52472976  0.74834683
-0.87719298 -0.64569058 -0.421549 -0.20773757  0.
0.20773757
0.421549  0.64569058  0.87719298 -0.74834683 -0.52472976 -0.30963911
-0.10226303  0.10226303  0.30963911  0.52472976  0.74834683 -0.60591541
-0.39919579 -0.19765935  0.
0.19765935  0.39919579  0.60591541
-0.48547489 -0.32894737 -0.16535231  0.
0.16535231  0.32894737
0.48547489 -0.10395865  0.10395865 -0.17113186  0.17113186]
Y-positions of electrodes (ypos): [ 0.68656518  0.68656518  0.52547424  0.46520183  0.43208641  0.421549
0.43208641  0.46520183  0.52547424  0.27165704  0.23384348  0.21394494
0.20773757  0.21394494  0.23384348  0.27165704  0.15177169  0.12553103
0.11086096  0.10426648  0.10426648  0.11086096  0.12553103  0.15177169
0.
0.
0.
0.
0.
0.
0.
0.
-0.15177169 -0.12553103 -0.11086096
-0.10426648 -0.10426648 -0.11086096 -0.12553103 -0.15177169 -0.27165704
-0.23384348 -0.21394494 -0.20773757 -0.21394494 -0.23384348 -0.27165704
-0.52547424 -0.46520183 -0.43208641 -0.421549 -0.43208641 -0.46520183
-0.52547424 -0.65583812 -0.65583812 -0.86033797 -0.86033797]
```

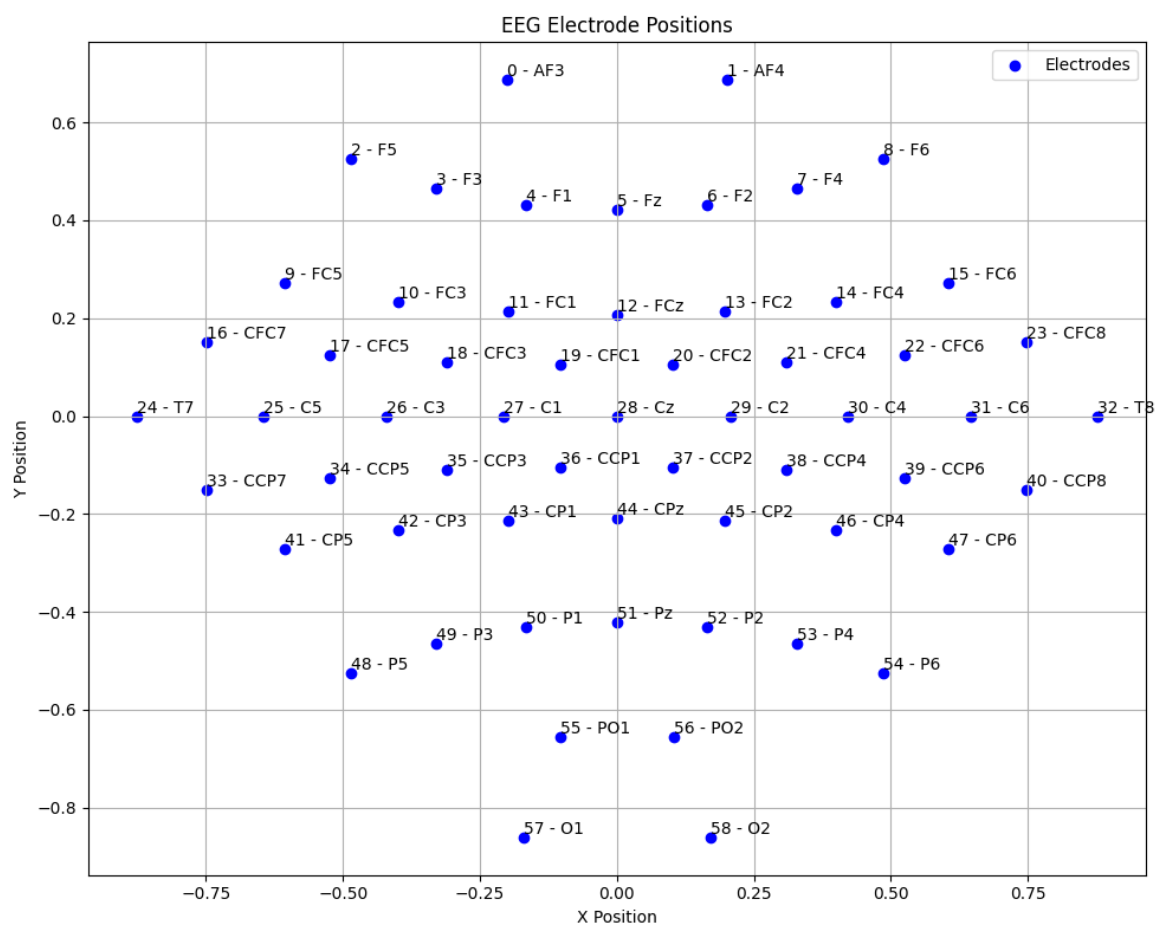
Dữ liệu B:

```
Continuous EEG signals (cnt): (190594, 59)
Positions of cues (pos): [ 2085 2885 3685 4485 5285 6085 6885 7685 8485 9285
10085 10885 11685 12485 13285 16286 17086 17886 18687 19486
20286 21086 21886 22686 23486 24286 25086 25886 26686 27486
30487 31287 32087 32887 33687 34487 35287 36087 36887 37687
38487 39287 40087 40887 41687 44688 45488 46288 47088 47888
48688 49488 50288 51088 51888 52688 53488 54288 55088 55888
58887 59687 60487 61287 62087 62887 63687 64487 65287 66087
66887 67687 68487 69287 70087 73088 73888 74688 75488 76288
77088 77888 78688 79488 80288 81088 81888 82688 83488 84288
87287 88087 88887 89687 90487 91287 92087 92887 93687 94487
97390 98190 98990 99790 100590 101390 102190 102990 103790 104590
105390 106190 106990 107790 108590 111590 112390 113190 113990 114790
115590 116390 117190 117990 118790 119590 120390 121190 121990 122790
125789 126589 127389 128189 128989 129789 130589 131389 132189 132989
133789 134589 135389 136189 136989 139988 140788 141588 142388 143188
143988 144788 145588 146388 147188 147988 148788 149588 150388 151188
154188 154988 155788 156588 157388 158188 158988 159788 160588 161388
162188 162988 163788 164588 165388 168387 169187 169987 170787 171587
172387 173187 173987 174787 175587 176387 177187 177987 178787 179587
182586 183386 184186 184986 185786 186586 187386 188186 188986 189786]
Target classes (y): [ 1 -1 1 -1 1 1 1 1 1 -1 1 -1 -1 -1 1 -1 1 1 1 -1 -1 -1 -1
1 -1 -1 -1 -1 -1 -1 -1 1 -1 1 1 -1 1 1 -1 -1 1 -1 -1 1 -1 -1
1 -1 1 -1 1 1 -1 1 -1 1 1 -1 1 1 -1 1 1 -1 1 -1 -1 1 -1 -1
1 -1 -1 1 -1 1 -1 -1 1 -1 1 1 1 1 1 -1 1 -1 -1 1 -1 -1
-1 1 1 -1 -1 1 1 -1 -1 -1 1 1 1 -1 1 -1 -1 -1 1 -1 -1
-1 1 1 1 1 -1 -1 -1 -1 1 -1 -1 -1 1 1 1 1 1 1 1 -1 1
1 -1 -1 1 1 1 1 -1 -1 -1 1 1 -1 -1 1 -1 1 -1 -1 -1 -1
-1 -1 -1 1 1 1 -1 1 -1 1 1 1 -1 -1 1 1 1 -1 -1 1 1
-1 -1 1 1 -1 1 1 1 1 -1 1 1 1 -1 -1 1 1 1 -1 1 1
-1 -1 1 1 -1 1 1 1]
```

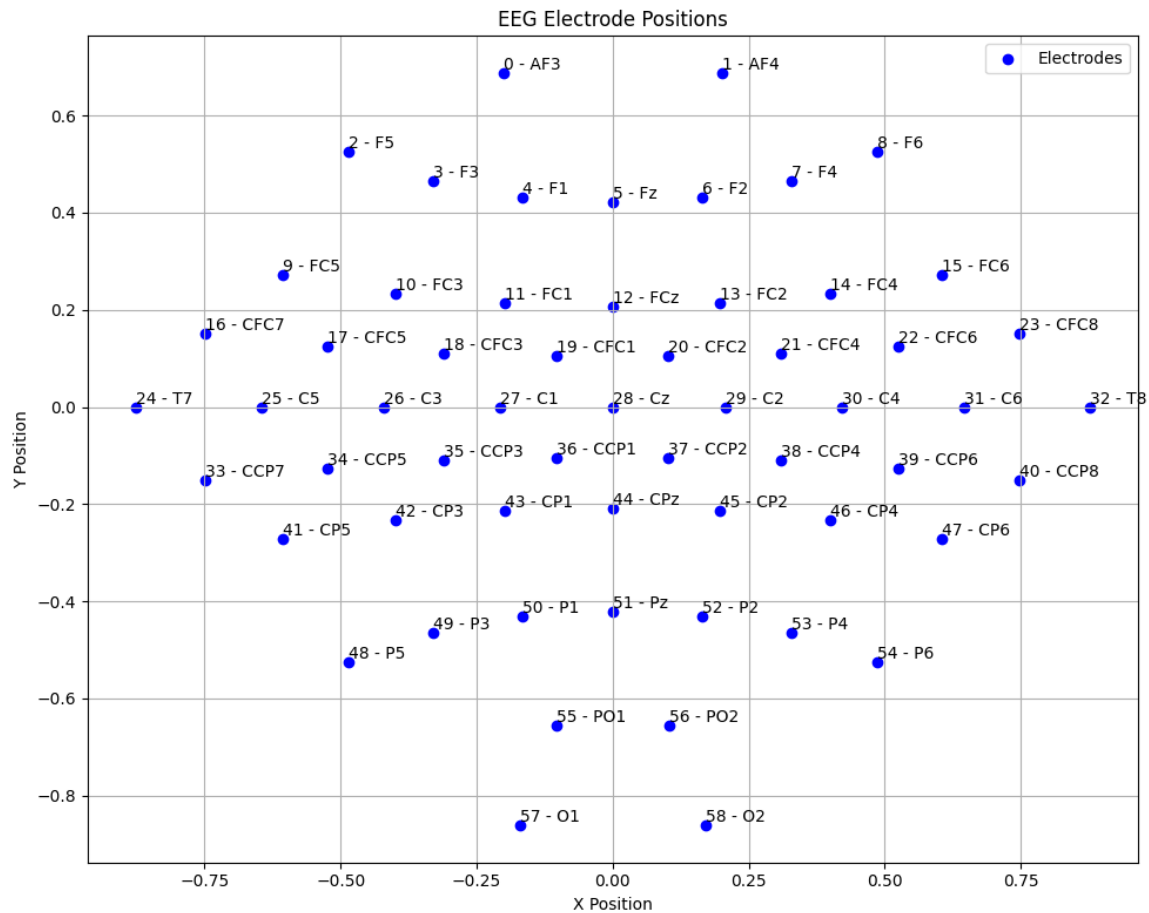
```
Sampling rate (fs): 100
Channel labels (clab): ['AF3', 'AF4', 'F5', 'F3', 'F1', 'Fz', 'F2', 'F4', 'F6', 'FC5', 'FC3', 'FC1', 'FCz', 'FC2', 'FC4', 'FC6', 'CFC7',
Motor imagery classes: ['left', 'right']
X-positions of electrodes (xpos): [-0.20109028 0.20109028 -0.48547489 -0.32894737 -0.16535231 0.
0.16535231 0.32894737 0.48547489 -0.60591541 -0.39919579 -0.19765935
0.
0.19765935 0.39919579 0.60591541 -0.74834683 -0.52472976
-0.30963911 -0.10226303 0.10226303 0.30963911 0.52472976 0.74834683
-0.87719298 -0.64569058 -0.421549 -0.20773757 0.
0.20773757
0.421549 0.64569058 0.87719298 -0.74834683 -0.52472976 -0.30963911
-0.10226303 0.10226303 0.30963911 0.52472976 0.74834683 -0.60591541
-0.39919579 -0.19765935 0.
0.19765935 0.39919579 0.60591541
-0.48547489 -0.32894737 -0.16535231 0.
0.16535231 0.32894737
0.48547489 -0.10395865 0.10395865 -0.17113186 0.17113186]
Y-positions of electrodes (ypos): [ 0.68656518 0.68656518 0.52547424 0.46520183 0.43208641 0.421549
0.43208641 0.46520183 0.52547424 0.27165704 0.23384348 0.21394494
0.20773757 0.21394494 0.23384348 0.27165704 0.15177169 0.12553103
0.11086096 0.10426648 0.10426648 0.11086096 0.12553103 0.15177169
0.
0.
0.
0.
0.
0.
0.
0.
-0.15177169 -0.12553103 -0.11086096
-0.10426648 -0.10426648 -0.11086096 -0.12553103 -0.15177169 -0.27165704
-0.23384348 -0.21394494 -0.20773757 -0.21394494 -0.23384348 -0.27165704
-0.52547424 -0.46520183 -0.43208641 -0.421549 -0.43208641 -0.46520183
-0.52547424 -0.65583812 -0.65583812 -0.86033797 -0.86033797]
```

- Biểu đồ các vị trí điện cực, kèm theo nhãn kênh tương ứng:

Dữ liệu A:



Dữ liệu B:



1.2. Bandpass Filtering

- Sử dụng các hàm **butter** và **filtfilt** từ thư viện `scipy.signal` để lọc tín hiệu.
 - Hàm **butter** được sử dụng để thiết kế bộ lọc Butterworth. Nó trả về các hệ số bộ lọc.
 - Hàm **filtfilt** áp dụng bộ lọc vào một chuỗi dữ liệu bằng cách lọc theo chiều xuôi và chiều ngược để đảm bảo không có biến dạng pha.
- **Butter_bandpass:**

```
def butter_bandpass(lowcut, highcut, fs, order=3):
    nyquist = 0.5 * fs
    low = lowcut / nyquist
    high = highcut / nyquist
    b, a = butter(order, [low, high], btype='band')
    return b, a
```

- **lowcut:** Tần số cắt thấp của bộ lọc bandpass.
- **highcut:** Tần số cắt cao của bộ lọc bandpass.
- **fs:** Tần số lấy mẫu của tín hiệu.
- **order:** Bậc của bộ lọc (mặc định là 3). Bậc cao hơn có nghĩa là độ dốc cắt mạnh hơn.

- **nyquist**: Tần số Nyquist, là một nửa tần số lấy mẫu. Đây là tần số cao nhất có thể được đại diện chính xác ở một tần số lấy mẫu nhất định.
- **low và high**: Đây là các tần số cắt chuẩn hóa. Việc chuẩn hóa được thực hiện bằng cách chia các tần số cắt thực tế cho tần số Nyquist.
- **butter(order, [low, high], btype='band')**: Hàm này thiết kế một bộ lọc bandpass Butterworth với bậc và tần số cắt được chỉ định. Nó trả về các hệ số bộ lọc b và a.

- **Bandpass_filter:**

```
def bandpass_filter(data, lowcut, highcut, fs, order=3):
    b, a = butter_bandpass(lowcut, highcut, fs, order=order)
    y = filtfilt(b, a, data, axis=0)
    return y
```

- **data**: Tín hiệu đầu vào cần lọc.
 - **lowcut**: Tần số cắt thấp của bộ lọc bandpass.
 - **highcut**: Tần số cắt cao của bộ lọc bandpass.
 - **fs**: Tần số lấy mẫu của tín hiệu.
 - **order**: Bậc của bộ lọc (mặc định là 3).
 - **butter_bandpass(lowcut, highcut, fs, order)**: Gọi hàm đã được định nghĩa trước đó để lấy các hệ số bộ lọc b và a.
 - **filtfilt(b, a, data, axis=0)**: Áp dụng bộ lọc Butterworth vào dữ liệu. Hàm filtfilt thực hiện lọc theo cả hai hướng (tiến và lùi), giúp loại bỏ sự méo pha. Tham số axis=0 chỉ ra rằng việc lọc sẽ được thực hiện dọc theo trục đầu tiên (thường dùng cho dữ liệu chuỗi thời gian). Hàm trả về tín hiệu đã lọc y.
- Biết rằng dải tần số mu là (8, 12) và dải tần số beta là (13, 30), chúng ta lọc tín hiệu EEG bằng hàm bộ lọc bandpass, kết quả thu được là EEG_mu và EEG_beta. Không cần phải phân tách riêng biệt mu và beta, do đó chúng ta tạo ra tín hiệu EEG_mu_beta bao gồm cả hai dải tần số này, tức là dải (8, 30). Kết quả là:

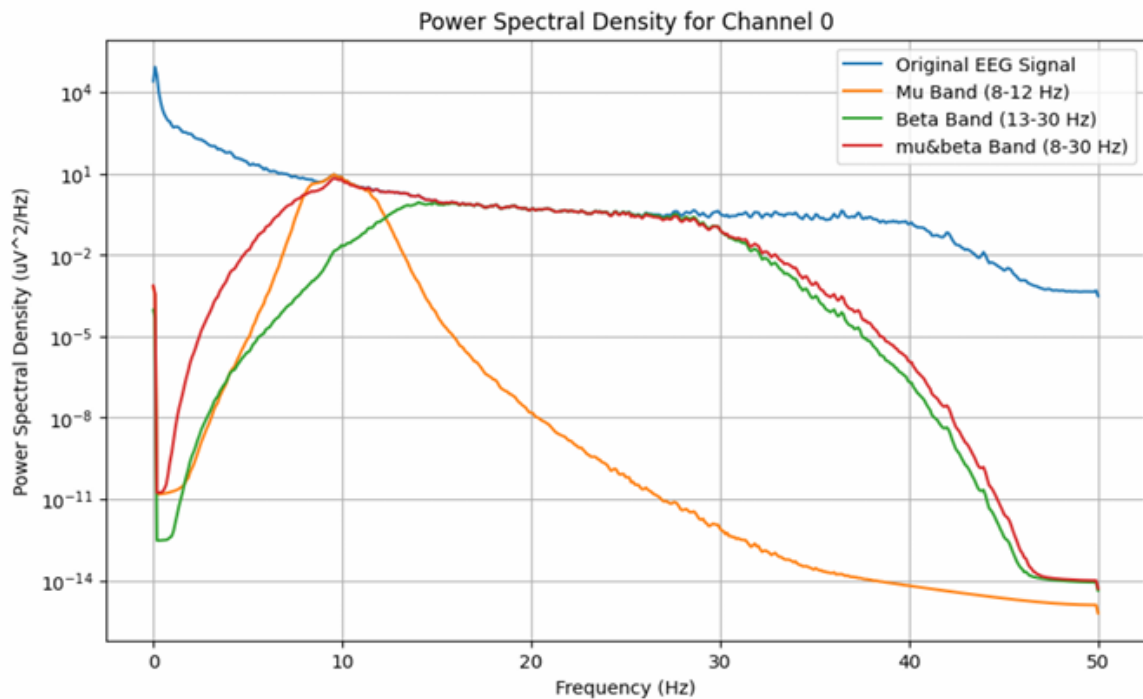
• **Dữ liệu A:**

```
Original EEG signals (EEG): (190594, 59)
Mu band EEG signals (EEG_mu): (190594, 59)
Beta band EEG signals (EEG_beta): (190594, 59)
mu+beta EEG signals (EEG_mu_beta): (190594, 59)
```

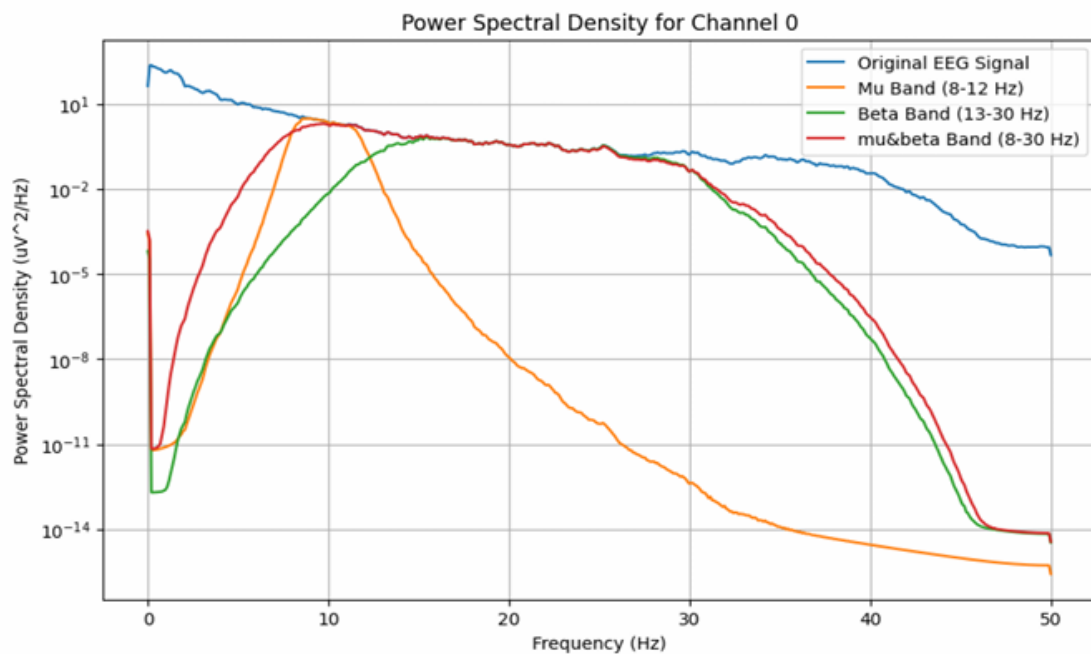
• **Dữ liệu B:**

```
Original EEG signals (EEG): (190594, 59)
Mu band EEG signals (EEG_mu): (190594, 59)
Beta band EEG signals (EEG_beta): (190594, 59)
mu+beta EEG signals (EEG_mu_beta): (190594, 59)
```

- Các vị trí điện cực cùng với nhãn kênh tương ứng:
- **Dữ liệu A:**



• Dữ liệu B



- Trong đồ thị này, dải mu là (8, 12), dải beta là (13, 30), và sự kết hợp của cả hai có dải tần (8, 30), điều này chứng tỏ rằng bộ lọc bandpass của chúng ta đã hoạt động như mong đợi. Dải này là nơi có mật độ (cao) nhất.

1.3. Spatial Filtering

Lọc không gian là một kỹ thuật trong xử lý tín hiệu được sử dụng để nâng cao các thành phần tín hiệu mong muốn và giảm nhiễu hoặc các hiện tượng bất thường bằng

cách xử lý dữ liệu trên các vị trí không gian khác nhau, chẳng hạn như các điện cực trong ghi âm EEG. Mục đích chính của lọc không gian bao gồm giảm nhiễu, tăng cường tín hiệu và xác định nguồn gốc. Kỹ thuật này hoạt động bằng cách áp dụng các phép biến đổi toán học lên các tín hiệu được ghi lại từ nhiều cảm biến hoặc kênh khác nhau.

1. Car Filtering

- Lọc Tham chiếu Trung bình Chung (CAR) là một kỹ thuật được sử dụng trong xử lý tín hiệu EEG để giảm nhiễu và dị vật bằng cách tham chiếu lại mỗi kênh với trung bình của tất cả các kênh. Phương pháp này giúp cải thiện tỷ lệ tín hiệu trên nhiễu bằng cách loại bỏ các thành phần nhiễu chung có mặt ở tất cả các kênh.
 - **mean_signal = np.mean(data, axis=1, keepdims=True)**: Tính toán tín hiệu trung bình qua tất cả các kênh cho mỗi điểm thời gian. Tham số **keepdims=True** đảm bảo rằng kết quả có cùng số chiều với dữ liệu đầu vào, điều này là cần thiết để thực hiện phép toán đúng trong bước tiếp theo.
 - **car_filtered = data - mean_signal**: Trừ tín hiệu trung bình từ tín hiệu của mỗi kênh tại mỗi điểm thời gian. Điều này thực sự là việc tham chiếu lại mỗi kênh với trung bình chung, loại bỏ nhiễu chung.

```
def apply_car_filter(data):  
    mean_signal = np.mean(data, axis=1, keepdims=True)  
    car_filtered = data - mean_signal  
    return car_filtered
```

- Chúng tôi đã áp dụng bộ lọc CAR cho tín hiệu mu, tín hiệu beta và sự kết hợp của chúng (mu_beta). Kết quả có các kích thước như sau:

- **Dữ liệu A:**

```
→ CAR-filtered Mu band EEG signals (EEG_mu_car): (190594, 59)  
CAR-filtered Beta band EEG signals (EEG_beta_car): (190594, 59)  
CAR-filtered mu+Beta band EEG signals (EEG_mu_beta_car): (190594, 59)
```

- **Dữ liệu B:**

```
CAR-filtered Mu band EEG signals (EEG_mu_car): (190473, 59)  
CAR-filtered Beta band EEG signals (EEG_beta_car): (190473, 59)  
CAR-filtered mu+Beta band EEG signals (EEG_mu_beta_car): ([190473, 59])
```

2. Laplacian Filtering

- Lọc Laplacian là một kỹ thuật lọc không gian chủ yếu được sử dụng trong xử lý tín hiệu EEG để làm nổi bật hoạt động cục bộ bằng cách trừ đi trung bình tín hiệu của các điện cực láng giềng từ tín hiệu của mỗi điện cực. Phương pháp này giúp làm nổi bật hoạt động não cục bộ và hỗ trợ trong việc nhận diện các sự kiện thần kinh tại chỗ.
 - **electrode_locations**: Mảng 2D chứa tọa độ (x, y) của mỗi điện cực.
 - **radius**: Bán kính trong đó các điện cực láng giềng được xem xét để tính toán trung bình. Đây có thể coi là một siêu tham số. Chúng ta cần thử nghiệm và tìm bán kính tốt nhất.

- **filtered_data = np.zeros_like(data)**: Tạo một mảng có cùng kích thước với dữ liệu để lưu trữ tín hiệu đã lọc.
 - **n_channels = data.shape[0]**: Số lượng kênh (điện cực).
 - **n_samples = data.shape[1]**: Số lượng mẫu thời gian.
 - **channel_data = data[i]**: Trích xuất tín hiệu của kênh hiện tại.
 - **channel_location = electrode_locations[i]**: Lấy tọa độ của kênh hiện tại.
 - **distance = np.linalg.norm(electrode_locations - channel_location, axis=1)**: Tính khoảng cách Euclid từ kênh hiện tại đến tất cả các kênh khác.
 - **neighbors = np.nonzero(distance <= radius)**: Xác định các kênh láng giềng trong bán kính chỉ định.
 - **filtered_data[i] = channel_data - np.mean(data[neighbors], axis=0)**: Trừ đi tín hiệu trung bình của các kênh láng giềng từ tín hiệu của kênh hiện tại để có tín hiệu đã lọc Laplacian.
- Chúng tôi đã tạo ra một mảng 2D gọi là **electrode_locations** kết hợp các tọa độ x và y của các điện cực.
 - Sau khi sử dụng bộ lọc, chúng tôi đã chuyển đổi lại tín hiệu đã lọc Laplacian về kích thước ban đầu.
 - Chúng tôi đã áp dụng bộ lọc Laplacian cho tín hiệu mu, tín hiệu beta và sự kết hợp của chúng (mu_beta). Kết quả có các kích thước như sau:

- **Dữ liệu A:**

```
→ Laplacian-filtered Mu band EEG signals (EEG_mu_laplacian): (190594, 59)
Laplacian-filtered Beta band EEG signals (EEG_beta_laplacian): (190594, 59)
Laplacian-filtered mu&Beta band EEG signals (EEG_mu_beta_laplacian): (190594, 59)
```

- **Dữ liệu B:**

```
aplacian-filtered Mu band EEG signals (EEG_mu_laplacian): (190473, 59) |
Laplacian-filtered Beta band EEG signals (EEG_beta_laplacian): (190473, 59)
Laplacian-filtered mu&Beta band EEG signals (EEG_mu_beta_laplacian): (190473, 59)
```

Phần II. Data Visualization

2.1. Các Hàm Vẽ Tín Hiệu

- Vẽ tín hiệu và so sánh các đặc tính của tín hiệu trước và sau khi lọc.
 - **plot_signals**: Hàm này vẽ tín hiệu gốc và tín hiệu đã lọc cho một kênh xác định. Nó hữu ích để so sánh tác động của các bộ lọc.
 - **amplitude_plot**: Hàm này vẽ tín hiệu từ nhiều kênh cho dữ liệu EEG gốc, đã lọc CAR và đã lọc Laplacian. Nó giúp trực quan hóa sự khác biệt giữa các kênh.
 - **amplitude_scatter_plot**: Hàm này tạo biểu đồ phân tán của biên độ tín hiệu tại một điểm thời gian cụ thể, cho thấy phân phối không gian của tín hiệu trước và sau khi lọc.

2.2. Các Hàm Áp Dụng t-SNE

- **t-SNE (t-distributed Stochastic Neighbor Embedding)** là một kỹ thuật giảm chiều dữ liệu giúp trực quan hóa dữ liệu có chiều cao.
 - **apply_tsne**: Hàm này áp dụng t-SNE vào dữ liệu EEG, giảm chiều của dữ liệu để trực quan hóa.
 - **plot_tsne**: Hàm này vẽ kết quả t-SNE cho dữ liệu EEG gốc, đã lọc CAR và đã lọc Laplacian, cho phép so sánh trực quan.
 - **plot_tsne_seperated**: Hàm này vẽ kết quả t-SNE cho một bộ dữ liệu duy nhất, cung cấp cái nhìn rõ ràng về sự phân cụm và cấu trúc trong dữ liệu.

2.3. Vẽ Tín Hiệu

Kênh được chọn cho các biểu đồ này là kênh 0, và dải tần được chọn là dải mu và beta.

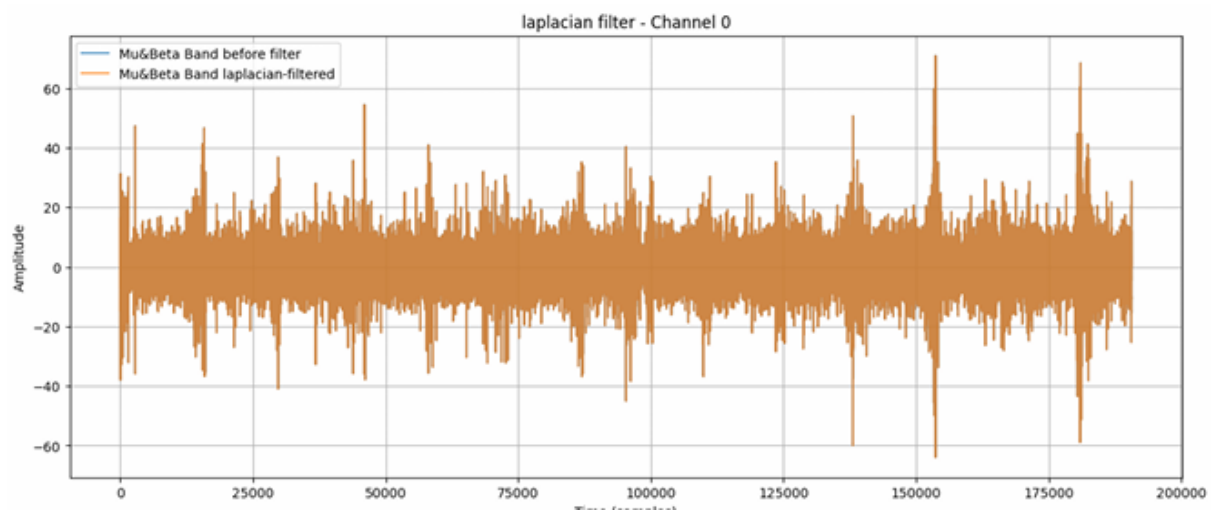
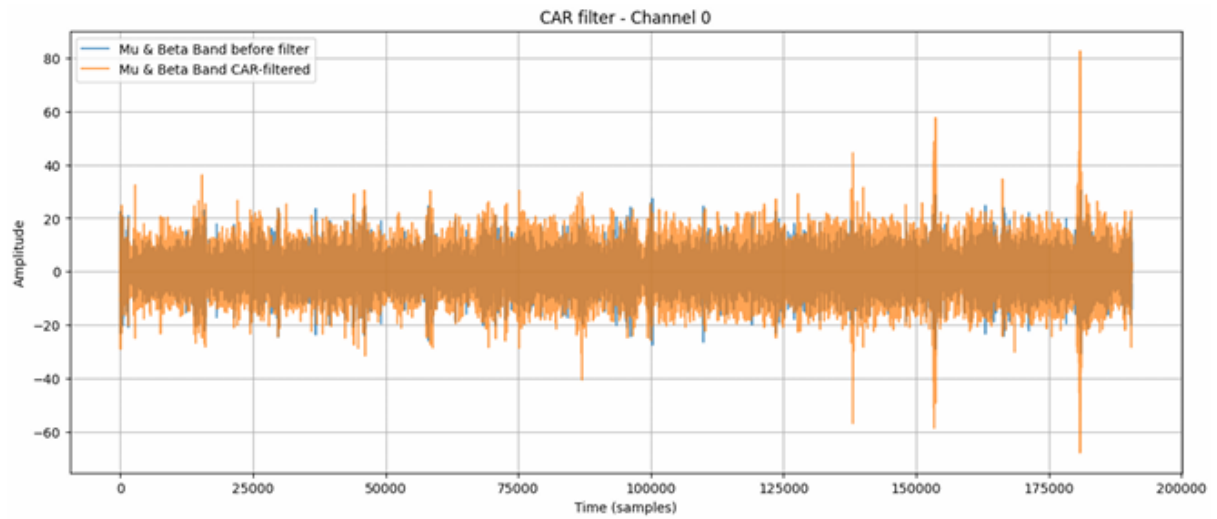
1. Vẽ Tác Động Lọc

Các biểu đồ này so sánh tín hiệu gốc và tín hiệu đã lọc cho dải Mu+Beta, làm nổi bật những thay đổi do bộ lọc CAR và Laplacian.

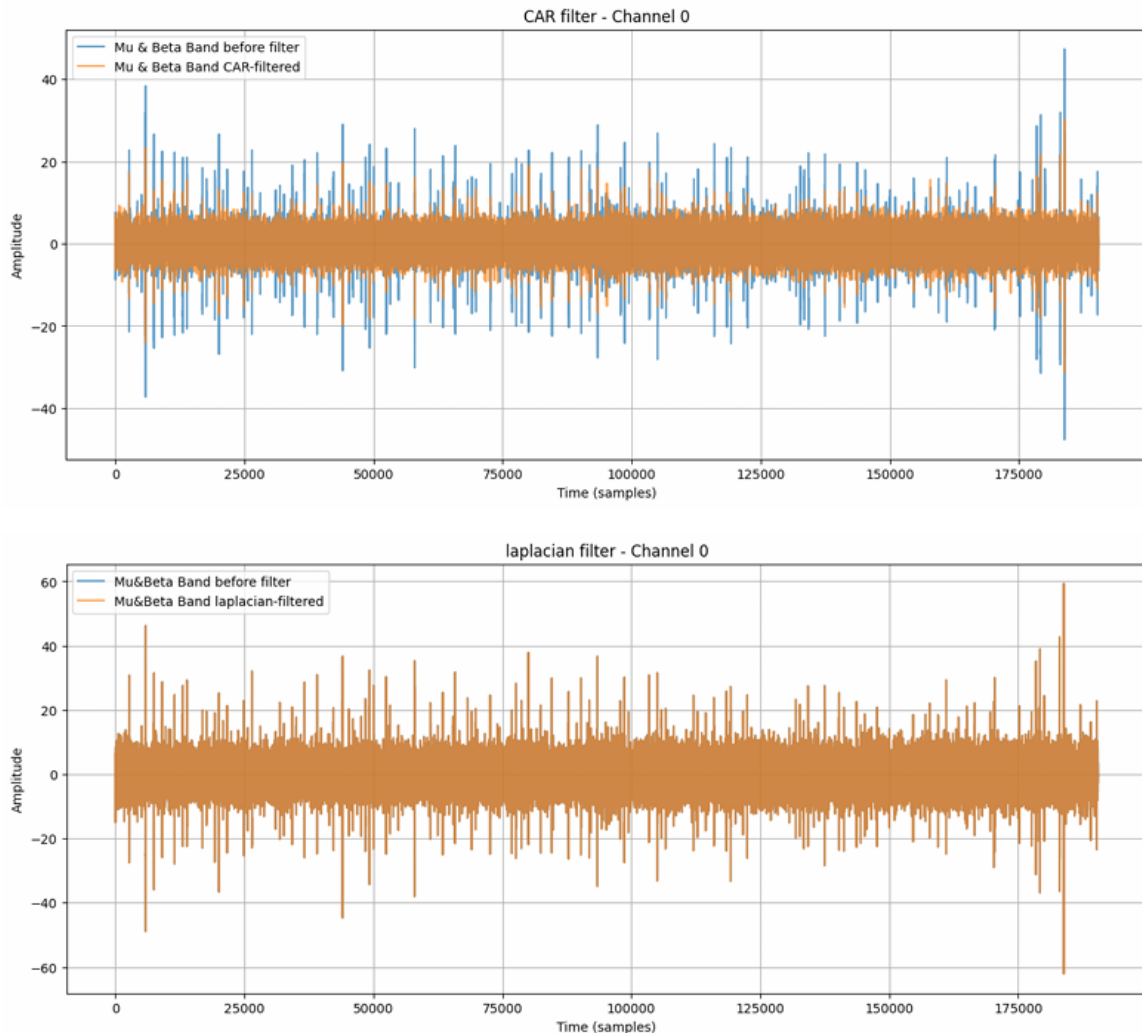
```
[ ] channel_to_plot = 0

plot_signals(EEG_mu, EEG_mu_beta_car, 'Mu & Beta Band', 'CAR', channel_to_plot)
plot_signals(EEG_mu_beta, EEG_mu_beta_laplacian, 'Mu&Beta Band', 'laplacian', channel_to_plot)
```

- Dữ liệu A:



- **Dữ liệu B:**



2. Vẽ Các Kênh Khác

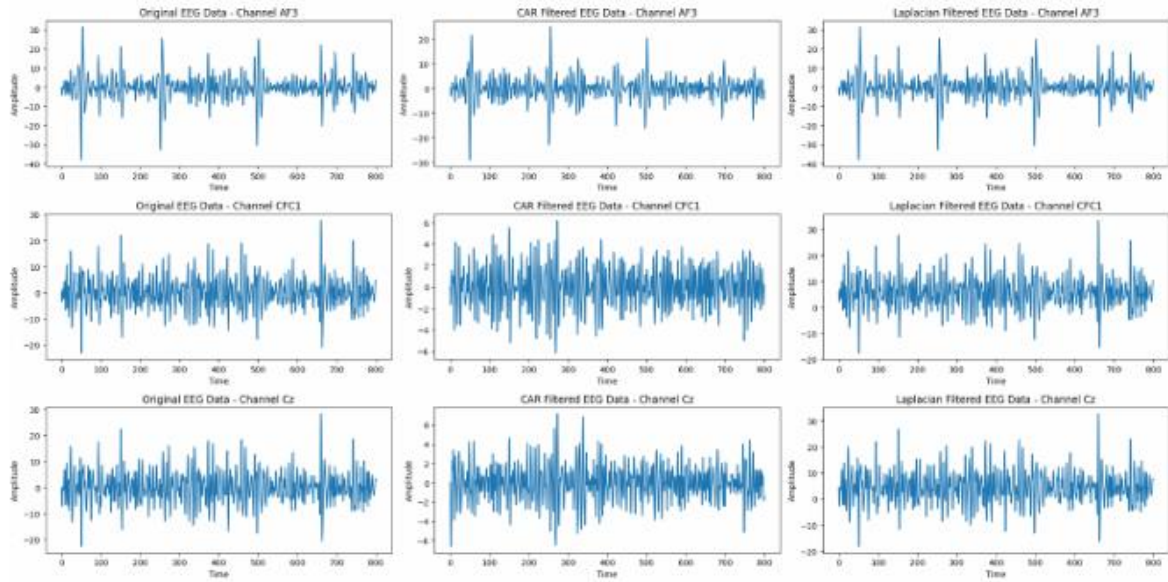
Các biểu đồ này trực quan hóa tín hiệu gốc, CAR-filtered và Laplacian-filtered cho nhiều kênh, cung cấp cái nhìn tổng quan về tác động của việc lọc trên các dải tần khác nhau.

- Mu+Beta Band:

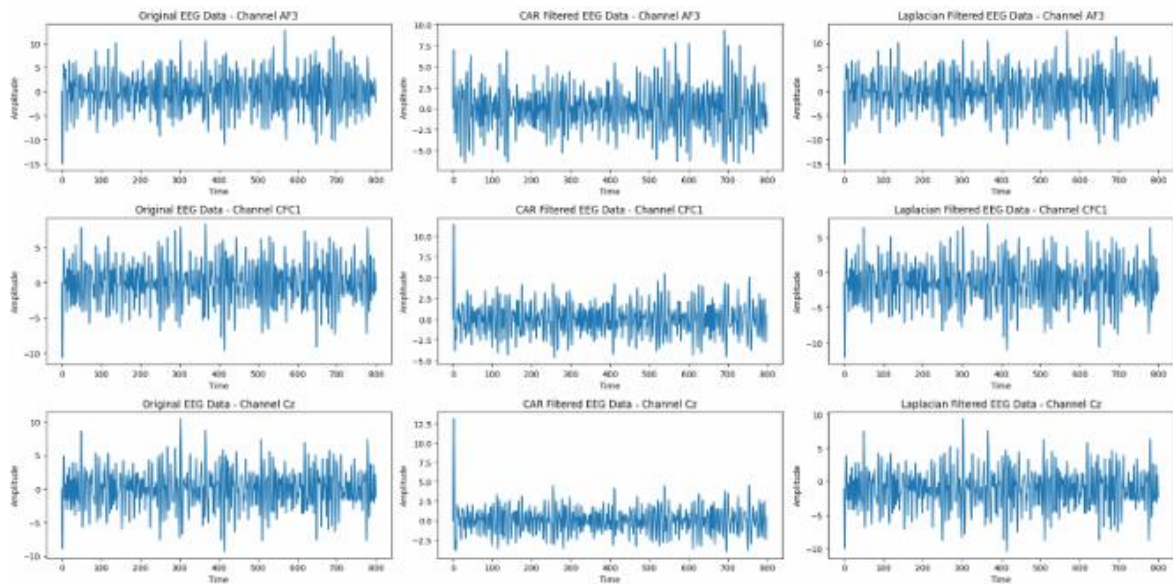
Hình dạng tín hiệu ban đầu có độ tương tự cao hơn với Laplacian-filtered và ít tương đồng hơn với CAR-filtered.

```
[ ] amplitude_plot(EEG_mu_beta, EEG_mu_beta_car, EEG_mu_beta_laplacian)
```

- Dữ liệu A:



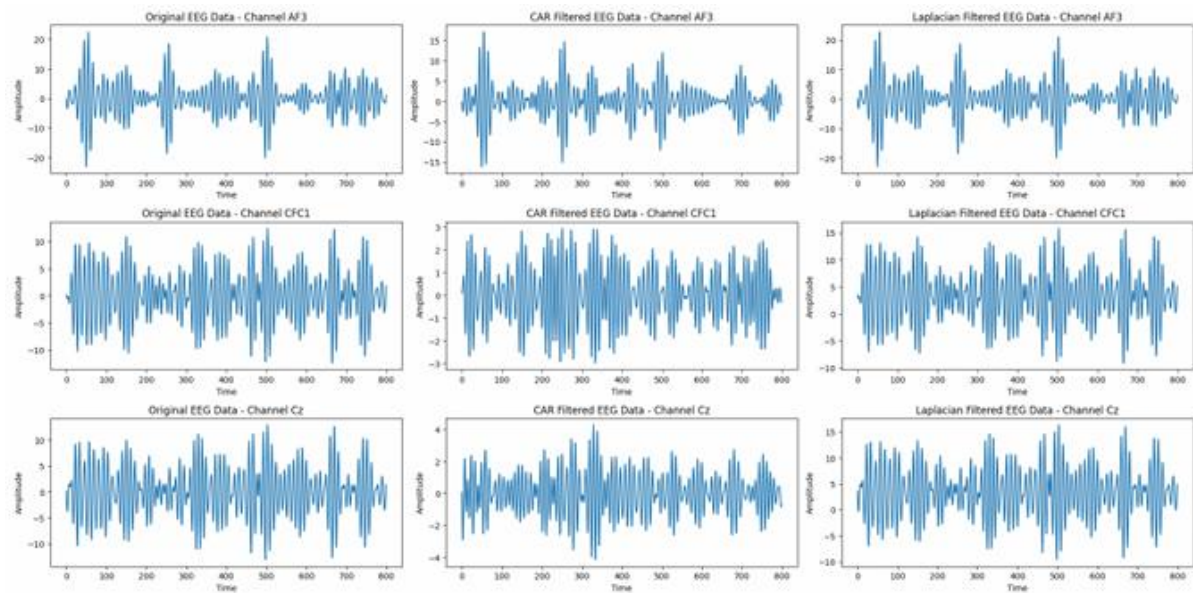
- **Dữ liệu B:**



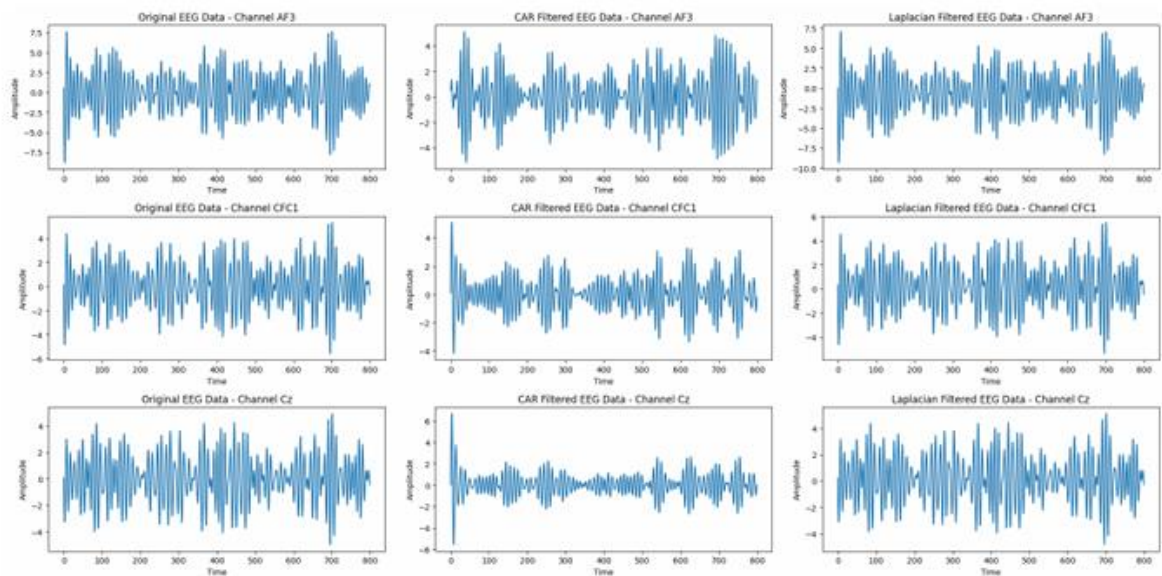
- **Mu Band:**

```
[ ] amplitude_plot(EEG_mu, EEG_mu_car, EEG_mu_laplacian)
```

- **Dữ liệu A:**



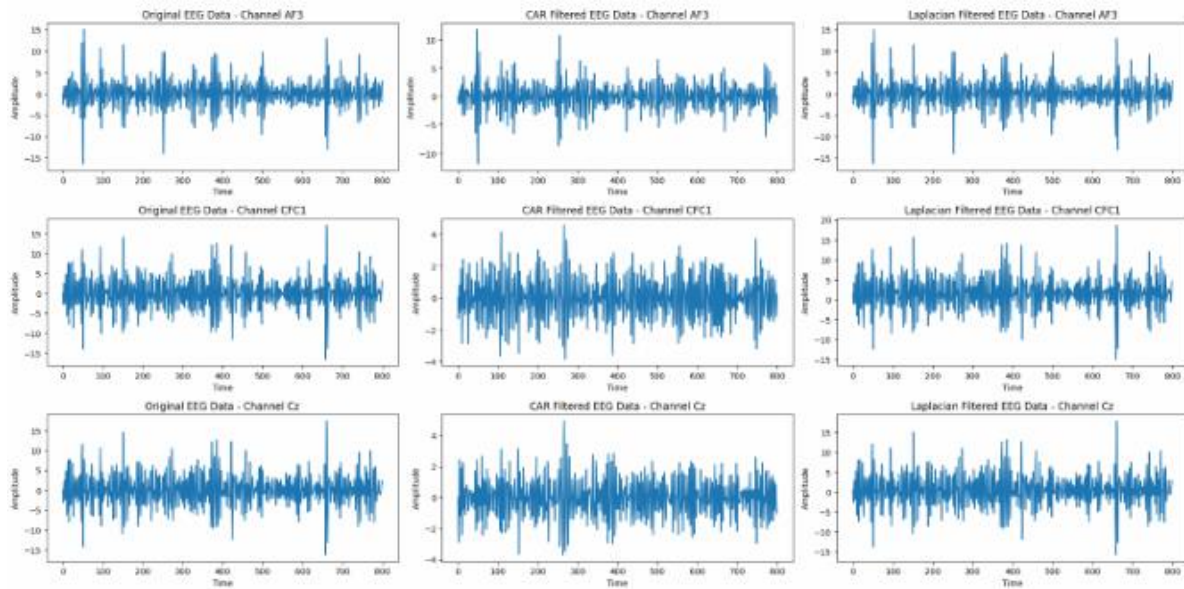
● **Dữ liệu B:**



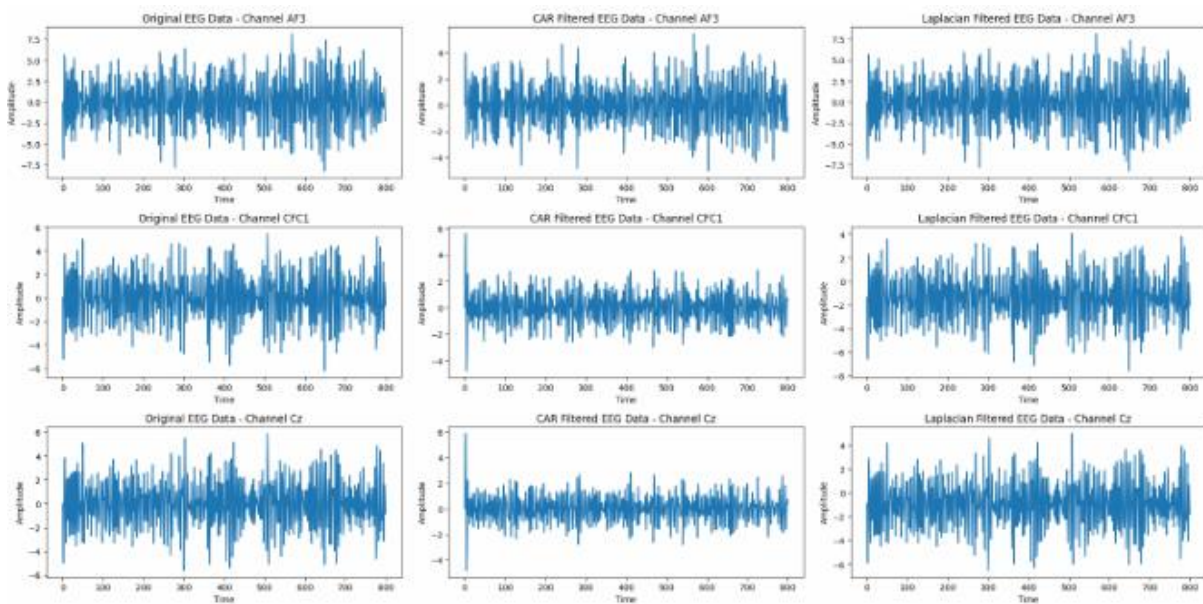
- **Beta Band:**

Tần số của tín hiệu Beta cao hơn tín hiệu MU.

● **Dữ liệu A:**



- **Dữ liệu B:**



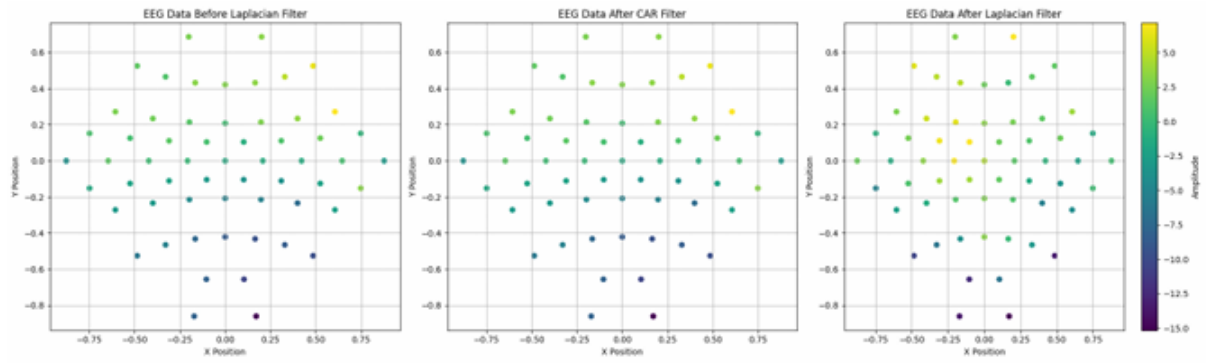
3. Scatter Plot Electrodes

Sự phân bố không gian của biên độ tín hiệu trên dây điện cực tại một thời điểm cụ thể, trước và sau khi lọc.

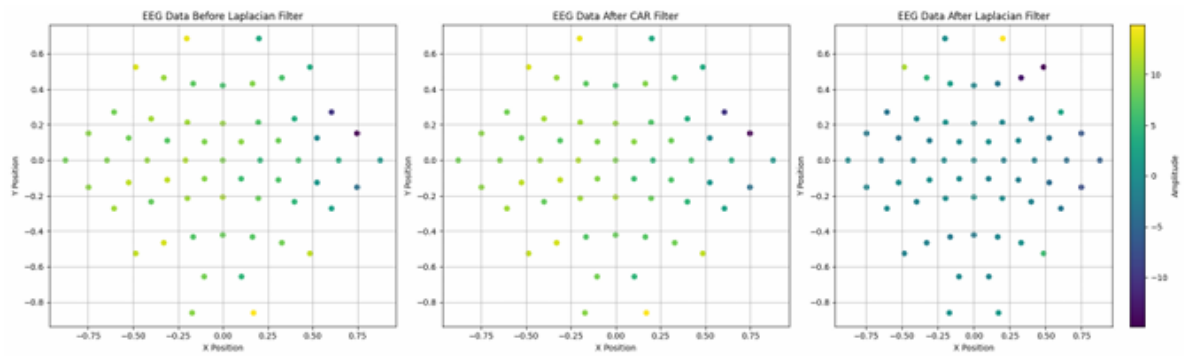
- **Mu+Beta Band:**

```
[ ] amplitude_scatter_plot(EEG_mu_beta, EEG_mu_beta_car, EEG_mu_beta_laplacian)
```

- **Dữ liệu A:**



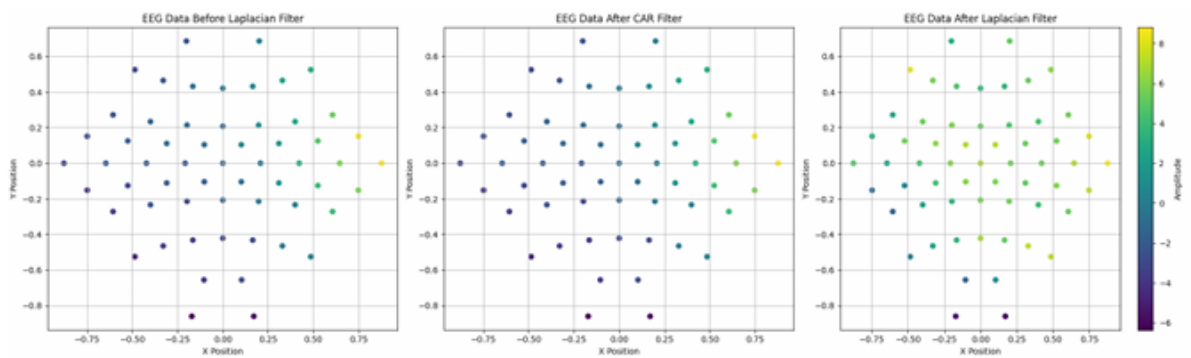
• Dữ liệu B:



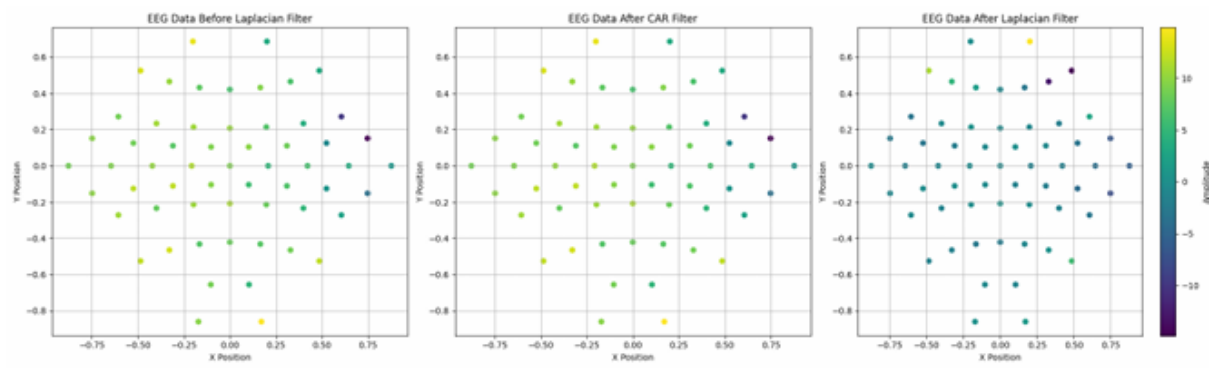
- Mu Band:

```
[ ] amplitude_scatter_plot(EEG_mu, EEG_mu_car, EEG_mu_laplacian)
```

• Dữ liệu A:



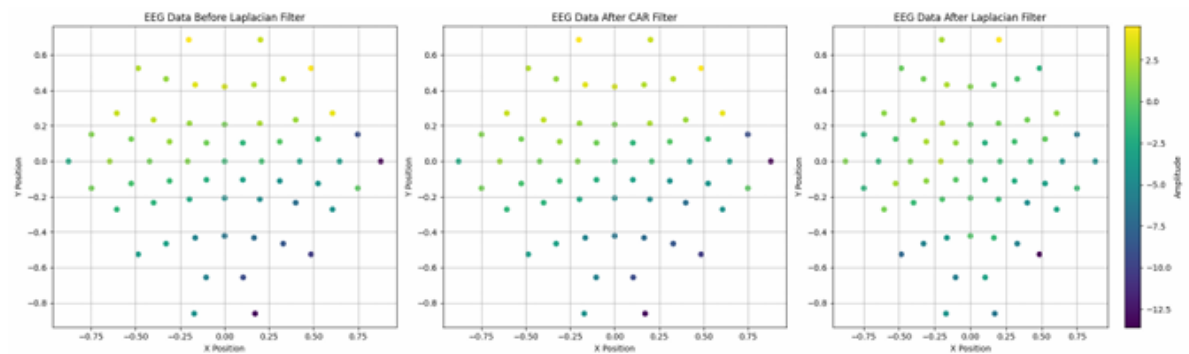
• Dữ liệu B:



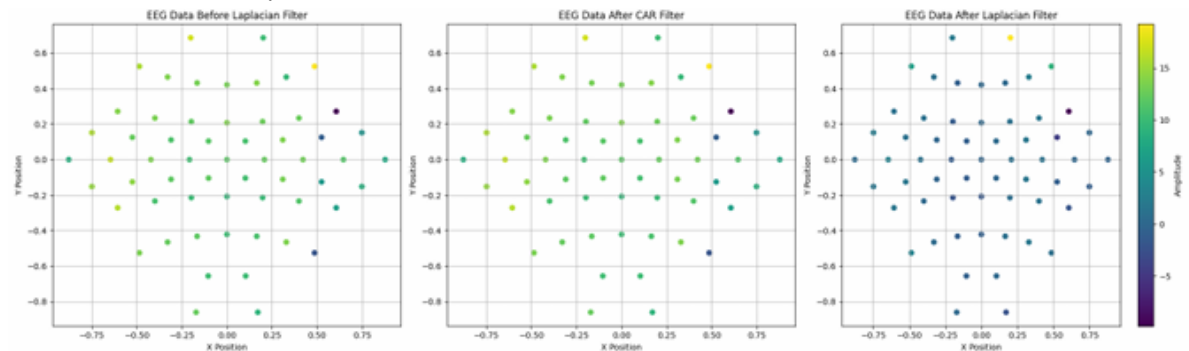
- Beta Band:

```
[ ] amplitude_scatter_plot(EEG_beta, EEG_beta_car, EEG_beta_laplacian)
```

• Dữ liệu A:



• Dữ liệu B:



4. Apply and Plot t-SNE

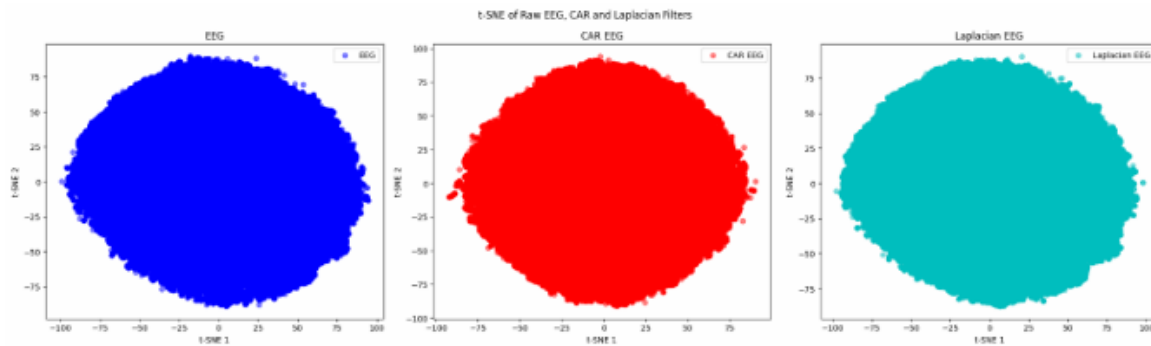
- Áp dụng t-SNE cho dữ liệu EEG dải Mu gốc, CAR-filtered và Laplacian-filtered và vẽ kết quả. Kỹ thuật trực quan hóa này giúp hiểu cấu trúc và sự phân nhóm của dữ liệu EEG đa chiều sau các phương pháp lọc khác nhau.
- Sử dụng các hình ảnh trực quan này, chúng ta có thể rút ra những hiểu biết về tác động của các kỹ thuật lọc khác nhau lên tín hiệu EEG, điều này hỗ trợ trong việc phân tích và giải thích dữ liệu thần kinh.

```
[ ] tsne_mu_beta = apply_tsne(EEG_mu_beta)
    tsne_mu_beta_car = apply_tsne(EEG_mu_beta_car)
    tsne_mu_beta_laplacian = apply_tsne(EEG_mu_beta_laplacian)

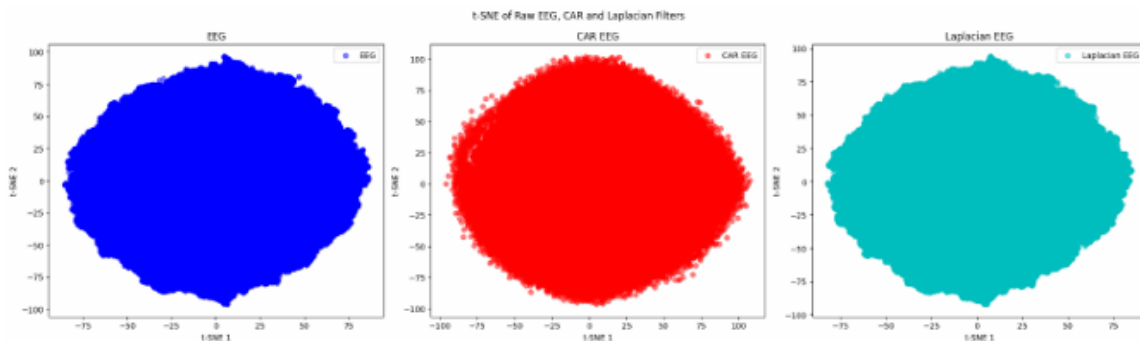
    plot_tsne(tsne_mu_beta, tsne_mu_beta_car, tsne_mu_beta_laplacian, 't-SNE of Raw EEG, CAR and Laplacian Filters')
```

- Các tín hiệu gốc t-SNE có độ tương tự cao hơn với các tín hiệu Laplacian-filtered và ít tương đồng hơn với các tín hiệu CAR-filtered.

- Dữ liệu A:



- Dữ liệu B:



Phần III. Feature Extraction

Trước khi thực hiện phân loại và phân cụm, việc trích xuất và giảm đặc trưng là rất quan trọng để đạt được độ chính xác tốt nhất. Em đã sử dụng ba phương pháp khác nhau để trích xuất đặc trưng: CSP, PCA và LDA. Em đã so sánh kết quả của các phương pháp này để xác định phương pháp nào cung cấp các đặc trưng tốt nhất cho việc phân loại và phân cụm tiếp theo.

3.1. Data preparation

1. Reshape and transpose:

- CSP yêu cầu dữ liệu 3D theo định dạng (samples, channels, window data). Dữ liệu chứa các mẫu với tần số 100Hz, dẫn đến 190.473 điểm dữ liệu từ 59 kênh theo thời gian, có 200 vị trí bắt đầu cho mỗi lớp. Cần phải tách mỗi cửa sổ dựa trên các vị trí bắt đầu của chúng. Để xác định kích thước cửa sổ, tìm khoảng cách tối thiểu giữa hai vị trí cửa sổ liên tiếp.

```
[ ] window_size = 800

EEG_3D_mu_beta = np.array([EEG_mu_beta[start:start + window_size, :] for start in window_pos])
EEG_3D_mu_beta_T = EEG_3D_mu_beta.transpose(0,2,1)

EEG_3D_mu_beta_car = np.array([EEG_mu_beta_car[start:start + window_size, :] for start in window_pos])
EEG_3D_mu_beta_car_T = EEG_3D_mu_beta_car.transpose(0,2,1)

EEG_3D_mu_beta_laplacian = np.array([EEG_mu_beta_laplacian[start:start + window_size, :] for start in window_pos])
EEG_3D_mu_beta_laplacian_T = EEG_3D_mu_beta_laplacian.transpose(0,2,1)

print(EEG_mu_beta_car.shape)
print(EEG_3D_mu_beta_car.shape)
print(EEG_3D_mu_beta_car_T.shape)
```

- Kết quả
 - Dữ liệu A:
 - Original data shape: (190,473, 59)
 - Reshaped data shape: (200, 800, 59)
 - Transposed data shape: (200, 59, 800)
 - Dữ liệu B:
 - Original data shape: (190,473, 59)
 - Reshaped data shape: (200, 785, 59)
 - Transposed data shape: (200, 59, 785)

2. Train & Test split:

- Trước khi áp dụng các phương pháp trích xuất đặc trưng, điều quan trọng là phải chia dữ liệu thành các tập training và testing. Điều này đảm bảo rằng mô hình được đánh giá trên dữ liệu chưa thấy, cung cấp một thước đo tốt hơn về hiệu suất của nó.
- Hàm **train_test_split** từ **sklearn.model_selection** được sử dụng để chia dữ liệu. Chúng tôi phân chia dữ liệu theo tỷ lệ lớp để duy trì tỷ lệ lớp giống nhau trong cả tập huấn luyện và tập kiểm tra.

```
from sklearn.model_selection import train_test_split

EEG_3D_mu_beta_T_train, EEG_3D_mu_beta_T_test, y_mu_beta_train,
y_mu_beta_test = train_test_split(EEG_3D_mu_beta_T, window_label,
                                  test_size=0.25, stratify=window_label, random_state=30)
print(EEG_3D_mu_beta_T_train.shape)

EEG_3D_mu_beta_car_T_train, EEG_3D_mu_beta_car_T_test, y_mu_beta_car_train,
y_mu_beta_car_test = train_test_split(EEG_3D_mu_beta_car_T, window_label,
                                       test_size=0.25, stratify=window_label, random_state=30)
print(EEG_3D_mu_beta_car_T_train.shape)

EEG_3D_mu_beta_laplacian_T_train, EEG_3D_mu_beta_laplacian_T_test, y_mu_beta_laplacian_train,
y_mu_beta_laplacian_test = train_test_split(EEG_3D_mu_beta_laplacian_T, window_label,
                                             test_size=0.25, stratify=window_label, random_state=30)
print(EEG_3D_mu_beta_laplacian_T_train.shape)
```

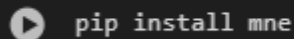
- Kết quả sau khi tách dữ liệu A

- For the original data:
 - Training data shape: (150, 59, 800)
 - Testing data shape: (50, 59, 800)
- For the car-filtered data:
 - Training data shape: (150, 59, 800)
 - Testing data shape: (50, 59, 800)
- For the Laplacian-filtered data:
 - Training data shape: (150, 59, 800)
 - Testing data shape: (50, 59, 800)

3.2. Using Common Spatial Patterns (CSP)

1. Install mne

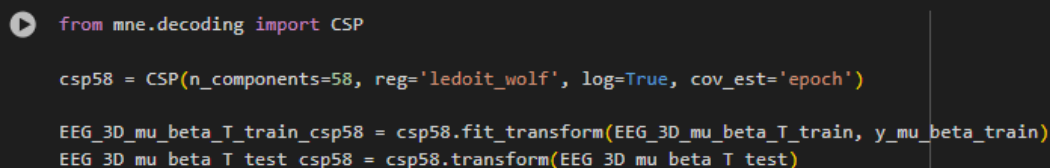
- CSP cố gắng tối đa hóa phương sai giữa hai lớp và giảm thiểu phương sai trong mỗi lớp. CSP được áp dụng cho các đoạn sóng EEG đã phân đoạn để trích xuất các bộ lọc không gian giúp phân biệt các tác vụ hình ảnh động cơ khác nhau.
- Để thực hiện trích xuất đặc trưng sử dụng CSP (Common Spatial Patterns), chúng ta đầu tiên cài đặt thư viện mne:



```
pip install mne
```

2. Apply CSP

- Thử nghiệm với các giá trị khác nhau cho **n_components**, đại diện cho số lượng đặc trưng sau khi trích xuất. Chúng tôi xác định rằng việc đặt **n_components** bằng số lượng kênh trừ đi một (58) mang lại kết quả tốt nhất trong phân loại.
- Áp dụng hàm CSP cho các bộ dữ liệu đã được lọc khác nhau của mình:
 - EEG_3D_mu_beta_T_train (dữ liệu gốc)
 - EEG_3D_mu_beta_car_T_train (dữ liệu đã lọc CAR)
 - EEG_3D_mu_beta_laplacian_T_train (dữ liệu đã lọc Laplacian)
- Để trực quan hóa 2D, em sử dụng 2 thành phần. Khi áp dụng CSP, em tiến hành fit và transform trên dữ liệu huấn luyện, và chỉ transform (không fit) trên dữ liệu kiểm tra để tránh rò rỉ dữ liệu.



```
from mne.decoding import CSP

csp58 = CSP(n_components=58, reg='ledoit_wolf', log=True, cov_est='epoch')

EEG_3D_mu_beta_T_train_csp58 = csp58.fit_transform(EEG_3D_mu_beta_T_train, y_mu_beta_train)
EEG_3D_mu_beta_T_test_csp58 = csp58.transform(EEG_3D_mu_beta_T_test)
```

- Các tham số của CSP:
 - **n_components=58:**

- Tham số này chỉ định số lượng thành phần cần trích xuất bằng CSP. Trong trường hợp này, nó được đặt thành 58, là số kênh trừ đi một. Điều này có nghĩa là CSP sẽ trích xuất 58 bộ lọc không gian hoặc đặc trưng từ dữ liệu EEG.
- **reg='ledoit_wolf':**
 - Tham số **reg** xác định loại chuẩn hóa (regularization) để áp dụng cho ma trận hiệp phương sai.
 - Chuẩn hóa giúp làm cho ước lượng ma trận hiệp phương sai trở nên mạnh mẽ hơn, đặc biệt là khi xử lý dữ liệu có chiều cao.
 - '**ledoit_wolf**' là một loại ước lượng thu nhỏ giúp cải thiện ước lượng ma trận hiệp phương sai bằng cách giảm sự biến thiên của nó. Điều này đặc biệt hữu ích cho dữ liệu EEG, vốn có thể bị nhiễu và có chiều cao.
- **log=True:**
 - Tham số **log** cho biết liệu có nên áp dụng biến đổi logarithm đối với các đặc trưng sau khi lọc CSP hay không.
 - Việc đặt **log=True** có nghĩa là logarithm của các phương sai của các tín hiệu đã được lọc CSP sẽ được tính toán.
 - Biến đổi này thường giúp ổn định phương sai và làm cho dữ liệu thích hợp hơn cho các tác vụ phân loại sau này.
- **cov_est='epoch':**
 - Tham số **cov_est** xác định cách thức ước lượng các ma trận hiệp phương sai.
 - '**epoch**' có nghĩa là các ma trận hiệp phương sai sẽ được tính toán từ các epochs (các cửa sổ thời gian riêng biệt) của dữ liệu EEG. Phương pháp này phù hợp với dữ liệu được phân đoạn thành các epoch, như các thử nghiệm trong một thí nghiệm EEG.

3. Concatenate Results

- Sau khi áp dụng CSP vào dữ liệu EEG, chúng ta muốn trực quan hóa và kiểm tra kết quả. Để làm điều này, chúng ta cần kết hợp dữ liệu CSP-transformed training and testing thành một bộ dữ liệu duy nhất. Điều này cho phép chúng ta vẽ đồ thị và so sánh các đặc trưng được trích xuất từ cả hai bộ dữ liệu.

```
[34] EEG_3D_mu_beta_T_csp58 = np.concatenate((EEG_3D_mu_beta_T_train_csp58, EEG_3D_mu_beta_T_test_csp58), axis=0)
train_test_labels = np.concatenate((y_mu_beta_train, y_mu_beta_test), axis=0)
print("EEG CSP feature Shape", EEG_3D_mu_beta_T_csp58.shape)
```

EEG CSP feature Shape (200, 58)

4. Plot results

Sử dụng ba phương pháp khác nhau để trực quan hóa tác động của CSP. Vì các đặc trưng có nhiều chiều và mục tiêu là đại diện chúng trong không gian hai chiều, các phương pháp này được chọn vì khả năng mô tả hiệu quả dữ liệu đã được chuyển đổi qua CSP-transformed.

a. Plotting using the First Two Components of CSP


```

import matplotlib.pyplot as plt
def plot_csp(csp_features, labels , title ):
    plt.figure(figsize=(8, 6))
    n_components = EEG_3D_mu_beta_car_T_csp58.shape[1]

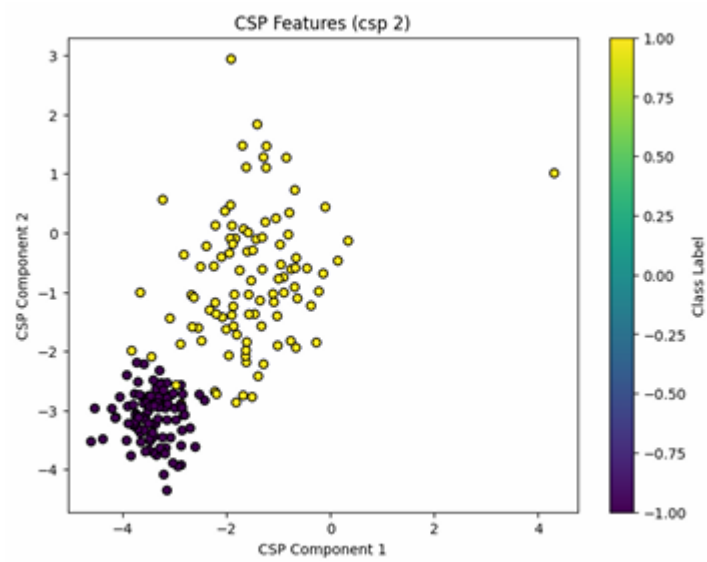
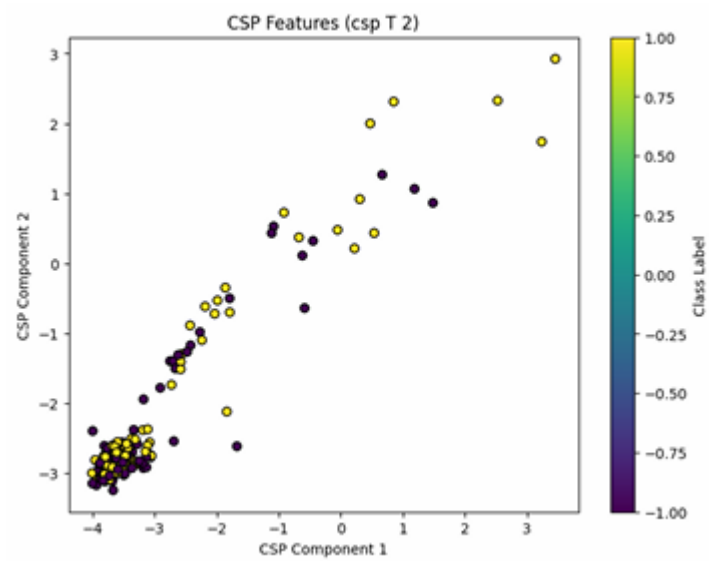
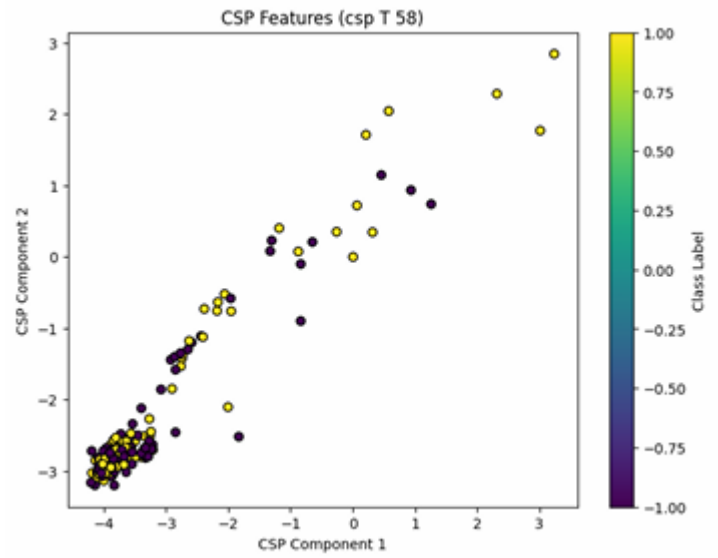
    if n_components == 1:
        plt.scatter(csp_features[:, 0], np.zeros_like(csp_features[:, 0]), c=labels, cmap='viridis', edgecolor='k')
        plt.xlabel('CSP Component 1')
        plt.yticks([])
    else:
        plt.scatter(csp_features[:, 0], csp_features[:, 1], c=labels, cmap='viridis', edgecolor='k')
        plt.xlabel('CSP Component 1')
        plt.ylabel('CSP Component 2')

    plt.title('CSP Features ' + title)
    plt.colorbar(label='Class Label')
    plt.show()

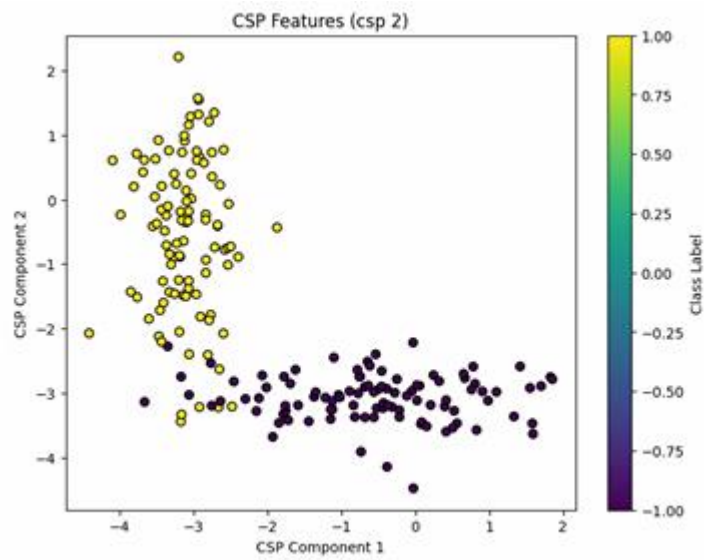
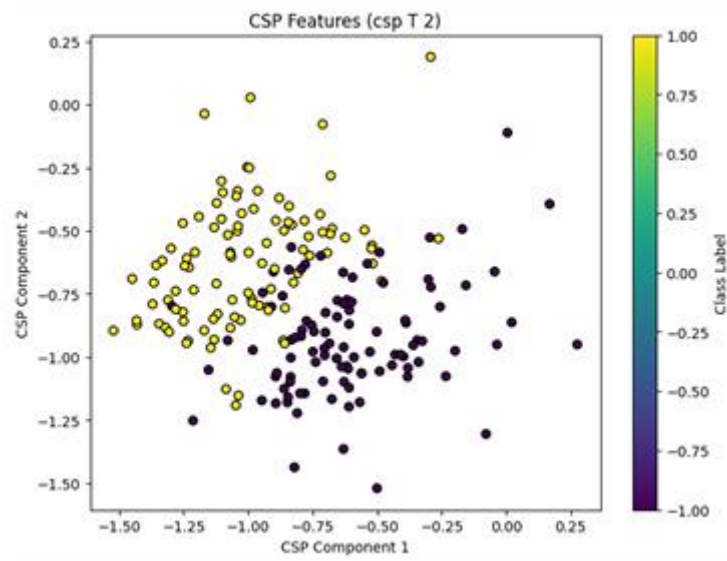
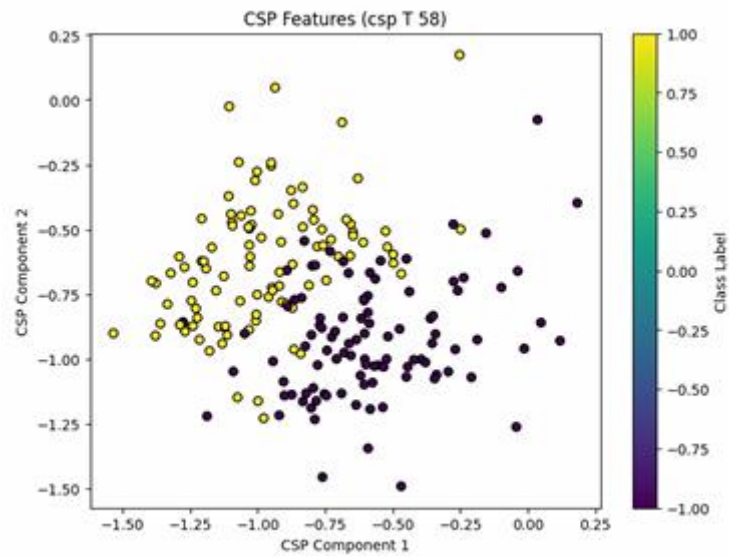
```

- Mô tả:
Biểu đồ này trực quan hóa các đặc trưng CSP sử dụng hai thành phần đầu tiên được rút ra từ thuật toán Common Spatial Patterns (CSP). Hàm kiểm tra số lượng thành phần đã được rút ra (`n_components`). Nếu chỉ có một thành phần, nó sẽ vẽ thành phần này so với giá trị bằng không để làm rõ. Ngược lại, nó vẽ thành phần đầu tiên trên trục x và thành phần thứ hai trên trục y. Màu sắc của mỗi điểm thể hiện nhãn lớp tương ứng của nó, như được chỉ ra bởi thanh màu.
- Biểu đồ:
Sử dụng ba phương pháp trực quan hóa để khám phá tác động của CSP. Biểu đồ đầu tiên trình bày CSP với 58 đặc trưng sau khi áp dụng bộ lọc car. Các biểu đồ sử dụng bộ lọc Laplacian và không lọc cho kết quả tương tự với bộ lọc car, vì vậy chúng tôi chỉ tập trung vào dữ liệu đã lọc bằng car trong báo cáo này. Biểu đồ thứ hai minh họa CSP với 2 đặc trưng sau khi áp dụng bộ lọc car. Biểu đồ thứ ba hiển thị CSP với 2 đặc trưng sau khi áp dụng bộ lọc car, sử dụng một biểu diễn giả định có 800 kênh với mỗi kênh chứa 59 điểm dữ liệu.
- Trong dataset B, việc rút trích đặc trưng CSP thể hiện hiệu suất vượt trội, phân tách hiệu quả các lớp khác nhau dựa trên các đặc trưng của chúng. Ngược lại, trong dataset A, kết quả có lợi hơn cho dữ liệu không biến đổi, trong khi hiệu suất trên dữ liệu 59 kênh kém nổi bật hơn.

- **Dữ liệu A:**



- Dữ liệu B:



b. Plotting using PCA (Principal Component Analysis)

```
[41] from sklearn.decomposition import PCA

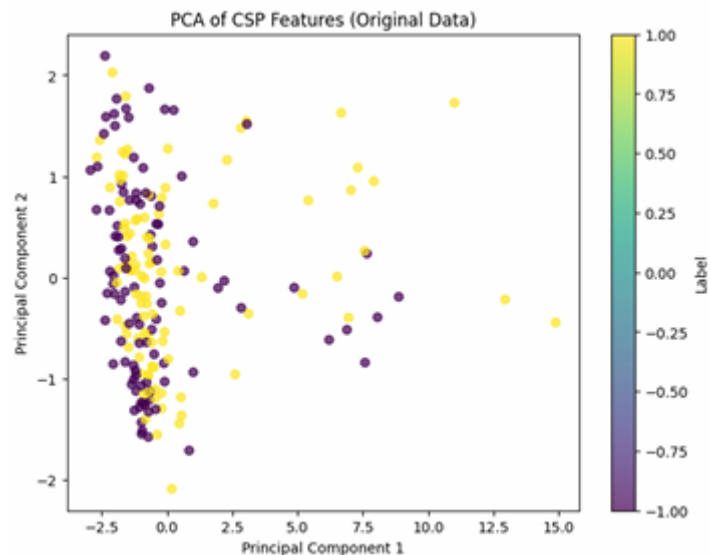
def plot_csp_using_pca(EEG, labels, text):

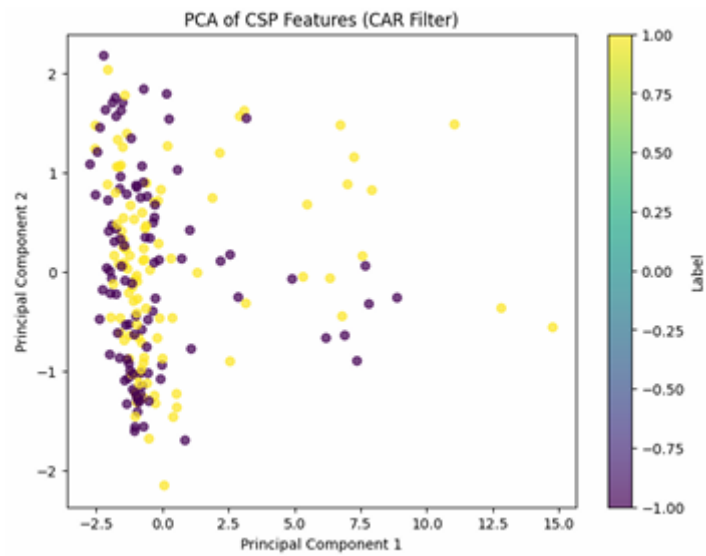
    pca = PCA(n_components=2)
    reduced_features = pca.fit_transform(EEG)

    plt.figure(figsize=(8, 6))
    scatter = plt.scatter(reduced_features[:, 0], reduced_features[:, 1], c=labels, cmap='viridis', alpha=0.7)
    plt.title('PCA of CSP Features'+ text)
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.colorbar(scatter, label='Label')
    plt.show()
```

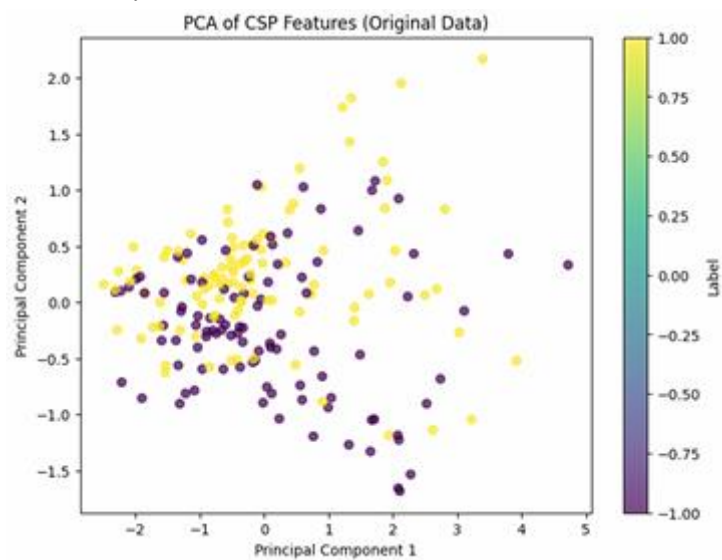
- Hàm này khởi tạo một đối tượng PCA với hai thành phần và áp dụng nó lên dữ liệu EEG. Nó chuyển đổi các đặc trưng đã được chuyển đổi bởi CSP thành hai thành phần chính bằng PCA.
- Các đặc trưng đã được giảm chiều sau đó được vẽ dưới dạng biểu đồ phân tán, trong đó mỗi điểm đại diện cho một mẫu EEG.
- Các điểm được tô màu theo nhãn lớp của chúng, giúp đánh giá trực quan khả năng phân tách các lớp trong không gian đặc trưng đã giảm chiều.
- Biểu đồ bao gồm nhãn cho các thành phần chính, tiêu đề phản ánh việc sử dụng PCA trên các đặc trưng CSP, và một thanh màu chỉ ra các nhãn lớp.

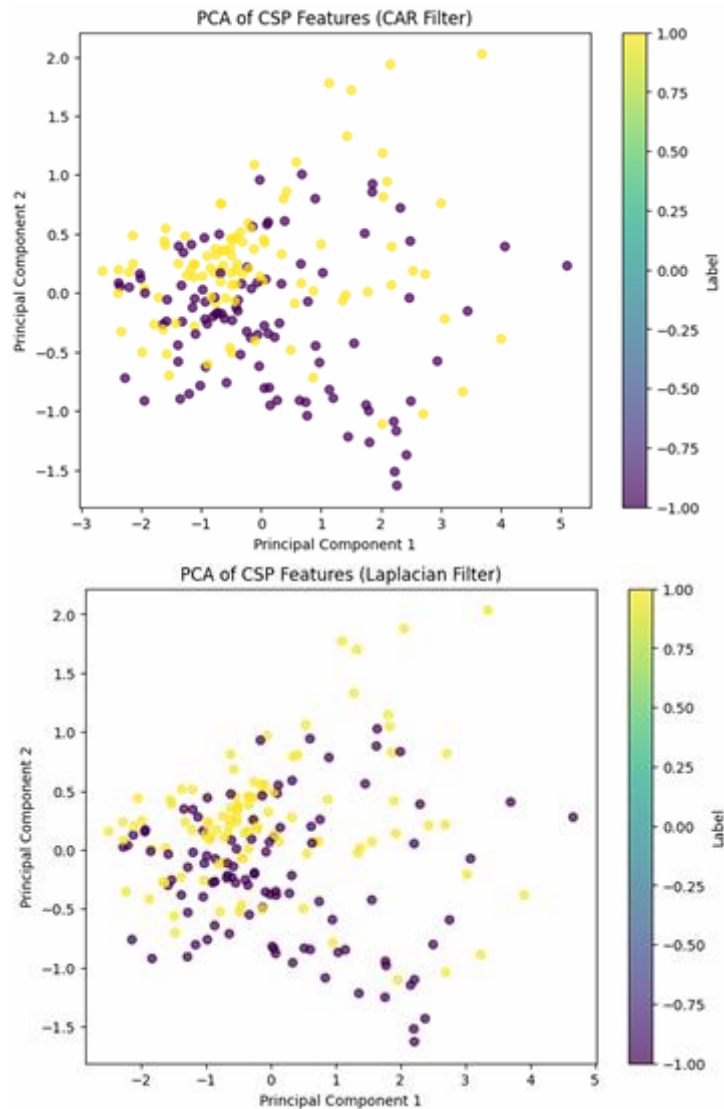
- Dữ liệu A:





- Dữ liệu B:





c. plot using t-SNE

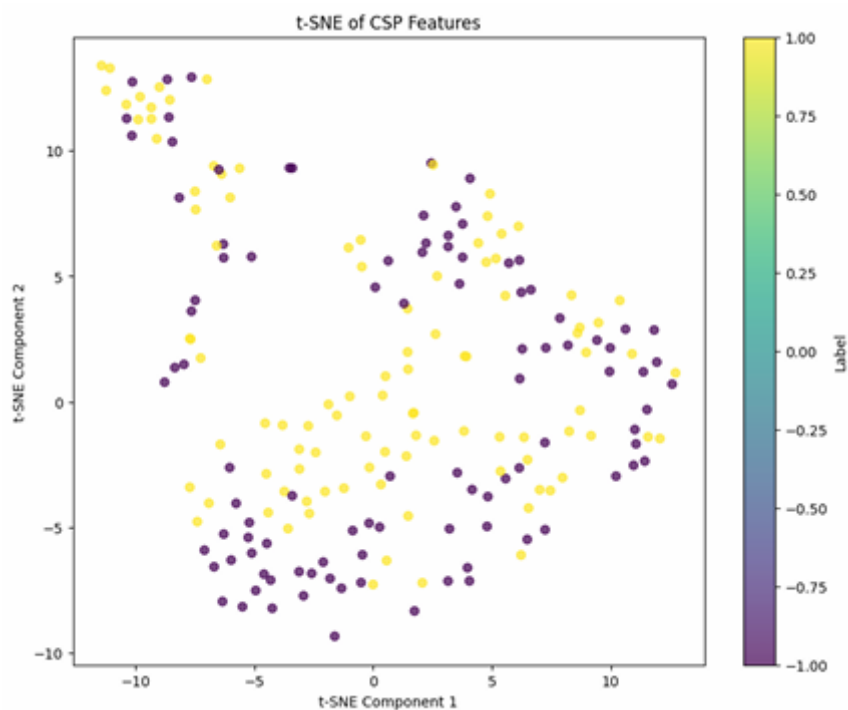
- Hàm **plot_csp_using_tsne** sử dụng phương pháp Nhúng Lân cận Stochastic phân tán (t-SNE) để trực quan hóa các đặc trưng EEG đã được biến đổi bằng CSP trong không gian có số chiều thấp hơn. Hàm này nhận ba tham số:
 - **EEG**: Đại diện cho các đặc trưng đã được biến đổi bởi CSP.
 - **labels**: Một mảng chỉ định nhãn lớp cho từng vector đặc trưng.
 - **text**: Văn bản bổ sung được sử dụng trong tiêu đề của biểu đồ.

```
from sklearn.manifold import TSNE

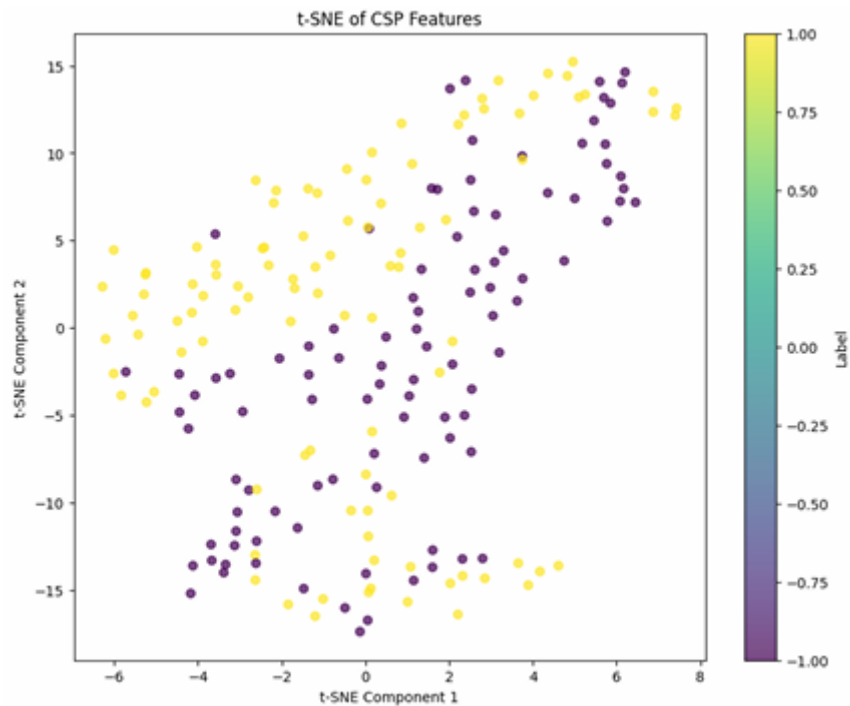
tsne = TSNE(n_components=2, random_state=42)
reduced_features_tsne = tsne.fit_transform(EEG_3D_mu_beta_car_T_csp58)

plt.figure(figsize=(10, 8))
scatter = plt.scatter(reduced_features_tsne[:, 0], reduced_features_tsne[:, 1], c=train_test_labels,
                    cmap='viridis', alpha=0.7)
plt.title('t-SNE of CSP Features')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.colorbar(scatter, label='Label')
plt.show()
```

- Hàm này khởi tạo một đối tượng t-SNE với hai thành phần và một trạng thái ngẫu nhiên cố định để đảm bảo tính tái lập.
 - Hàm áp dụng t-SNE để biến đổi các đặc trưng đã được chuyển đổi bằng CSP vào không gian hai chiều.
 - Các đặc trưng sau khi biến đổi được hiển thị dưới dạng biểu đồ tán xạ, trong đó mỗi điểm đại diện cho một mẫu EEG.
 - Các điểm được tô màu theo nhãn lớp của chúng, giúp kiểm tra trực quan sự phân bố lớp trong không gian đặc trưng đã biến đổi.
- Kết quả:
 - Dữ liệu A:



- Dữ liệu B:



3.3. Principal Component Analysis (PCA)

(Principal Component Analysis - PCA) là một kỹ thuật được sử dụng để giảm số chiều trong các tập dữ liệu. PCA xác định các thành phần trực giao, giúp giải thích phương sai lớn nhất trong dữ liệu, từ đó tạo ra một biểu diễn đơn giản hơn.

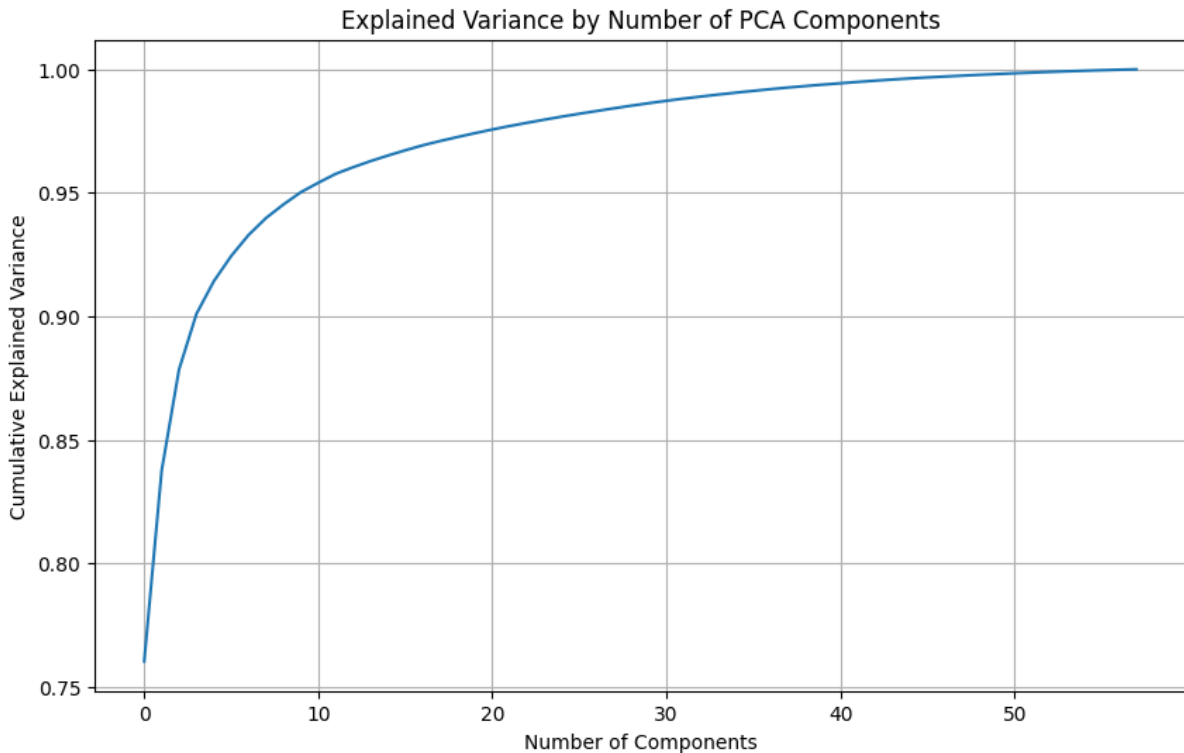
1. Find best n-component:

Xác định số lượng thành phần PCA tối ưu đảm bảo giải thích ít nhất 95% phương sai,

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

pca = PCA()
pca.fit(EEG_3D_mu_beta_car_T_train_csp58)

plt.figure(figsize=(10, 6))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Explained Variance by Number of PCA Components')
plt.grid(True)
plt.show()
```



```
desired_variance = 0.95
cumulative_variance = np.cumsum(pca.explained_variance_ratio_)
n_components = np.argmax(cumulative_variance >= desired_variance) + 1
print(f'Number of components to explain {desired_variance*100}% variance: {n_components}')
```

Number of components to explain 95.0% variance: 10

2. Applying PCA with Optimal Components:

```
pca = PCA(n_components=10)
EEG_3D_mu_beta_car_T_train_csp58_pca10 = pca.fit_transform(EEG_3D_mu_beta_car_T_train_csp58)
EEG_3D_mu_beta_car_T_test_csp58_pca10 = pca.transform(EEG_3D_mu_beta_car_T_test_csp58)
print("train set shape:", EEG_3D_mu_beta_car_T_train_csp58_pca10.shape)
print("test set shape:", EEG_3D_mu_beta_car_T_test_csp58_pca10.shape)
```

- Đoạn mã này áp dụng CSP-transformed lên dữ liệu EEG đã được chuyển đổi bằng CSP, sử dụng số lượng thành phần tối ưu đã được xác định trước (`n_components`). Mã này thực hiện giảm chiều dữ liệu cho cả tập huấn luyện và tập kiểm tra, đồng thời vẫn giữ được phần phương sai quan trọng.

- Dữ liệu A:

```
train set shape: (150, 10)
test set shape: (50, 10)
```

- Dữ liệu B:

```
train set shape: (150, 25)
test set shape: (50, 25)
```

- **PCA trên Dữ Liệu Gốc Đã Lọc CAR:**

Đối với dữ liệu EEG gốc đã được lọc bằng CAR, chúng tôi áp dụng PCA bằng cách sử dụng phương pháp sau:

- Với số lượng thành phần tối ưu (best n-components):

```

n_trials, n_channels, n_samples = EEG_3D_mu_beta_car.shape
EEG_3D_mu_beta_car_resaped = EEG_3D_mu_beta_car.reshape(n_trials, -1)
print(EEG_3D_mu_beta_car_resaped.shape)

n_components = 160

pca = PCA(n_components=n_components)
EEG_3D_mu_beta_car_pca160 = pca.fit_transform(EEG_3D_mu_beta_car_resaped)

print(EEG_3D_mu_beta_car_pca160.shape)

```

(200, 47200)
(200, 160)

- for 2-components:

```

[52] n_components = min(EEG_3D_mu_beta_car_resaped.shape[1], 2)

pca = PCA(n_components=n_components)
EEG_3D_mu_beta_car_pca2 = pca.fit_transform(EEG_3D_mu_beta_car_resaped)

print(EEG_3D_mu_beta_car_pca2.shape)

```

(200, 2)

3. Plot PCA

```

import numpy as np
from sklearn.decomposition import PCA
def plot_pca(pca_features, labels, title):
    plt.figure(figsize=(8, 6))

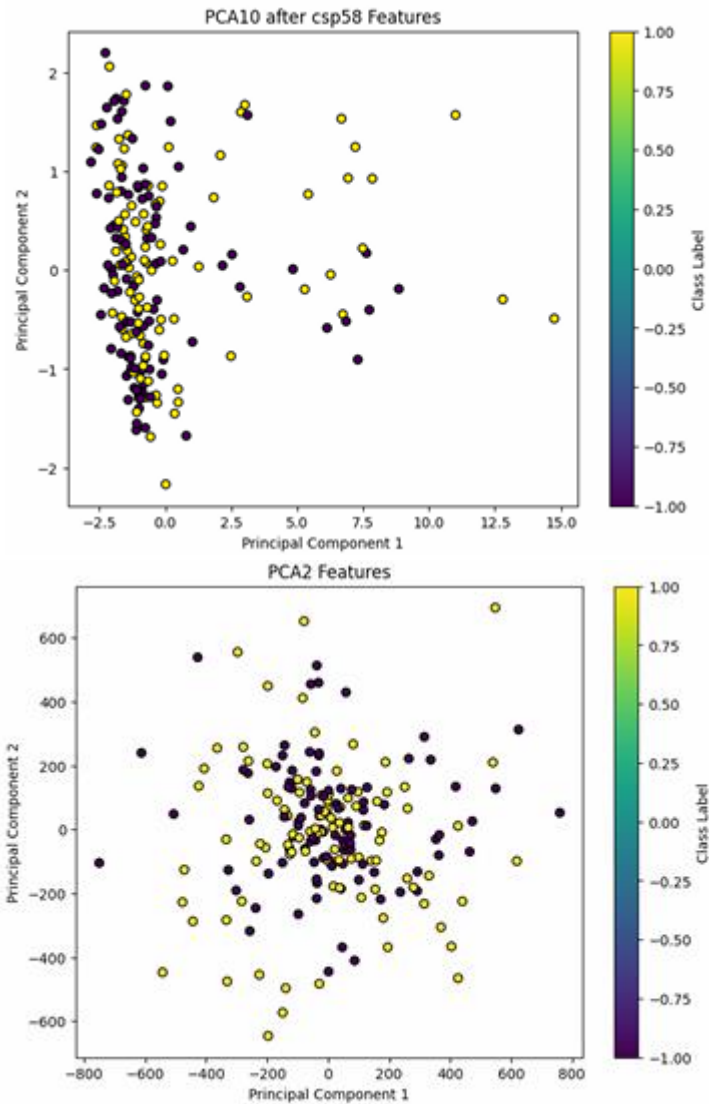
    if n_components == 1:
        plt.scatter(pca_features[:, 0], np.zeros_like(pca_features[:, 0]), c=labels, cmap='viridis', edgecolor='k')
        plt.xlabel('Principal Component 1')
        plt.yticks([])
    else:
        plt.scatter(pca_features[:, 0], pca_features[:, 1], c=labels, cmap='viridis', edgecolor='k')
        plt.xlabel('Principal Component 1')
        plt.ylabel('Principal Component 2')

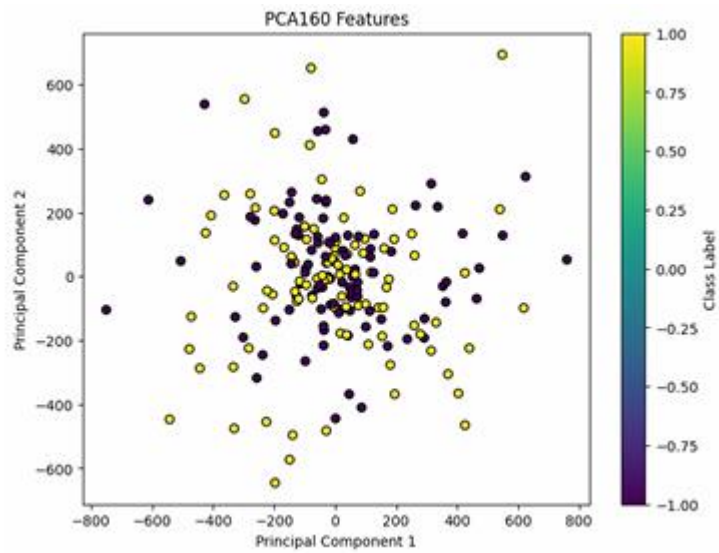
    plt.title(title)
    plt.colorbar(label='Class Label')
    plt.show()

```

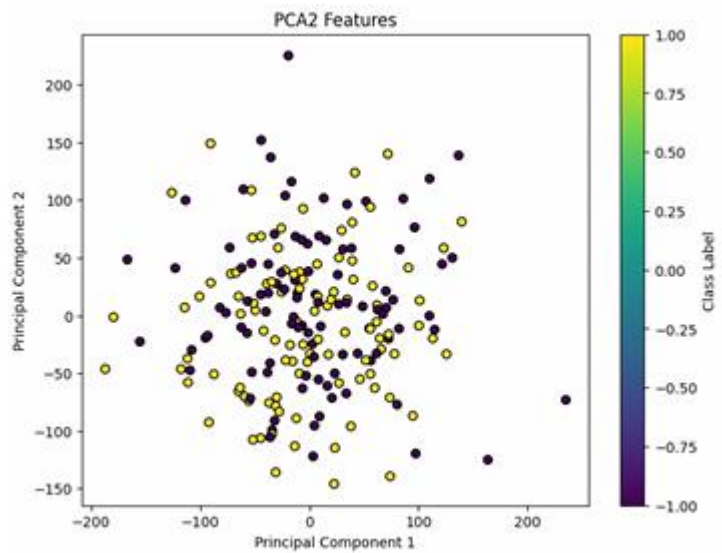
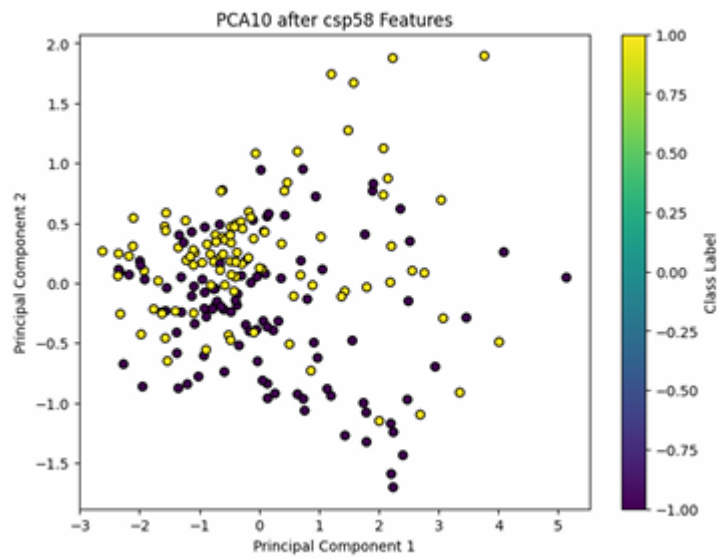
- Hàm **plot_pca** được thiết kế để trực quan hóa kết quả của Phân Tích Thành Phần Chính (PCA), một kỹ thuật được sử dụng để giảm số chiều của dữ liệu trong khi vẫn bảo toàn phương sai quan trọng. Hàm này nhận các đặc trưng đã được PCA chuyển đổi (**pca_features**) và nhãn lớp tương ứng (**labels**) để vẽ chúng trong không gian một chiều hoặc hai chiều.
- Nếu **pca_features** chỉ có một thành phần chính (**n_components == 1**), hàm sẽ tạo một biểu đồ phân tán dọc theo trục x, được gắn nhãn là 'Thành Phần Chính 1' (Principal Component 1). Trục y được ẩn đi (**yticks=[]**), và mỗi điểm dữ liệu được mã màu dựa trên nhãn lớp của nó, sử dụng bảng màu 'viridis'.
- Trong trường hợp có hai thành phần chính (**n_components == 2**), hàm sẽ vẽ các điểm dữ liệu trong một biểu đồ phân tán hai chiều, với 'Thành Phần Chính 1' trên trục x và 'Thành Phần Chính 2' trên trục y. Tương tự, mỗi điểm được mã màu dựa trên nhãn lớp.

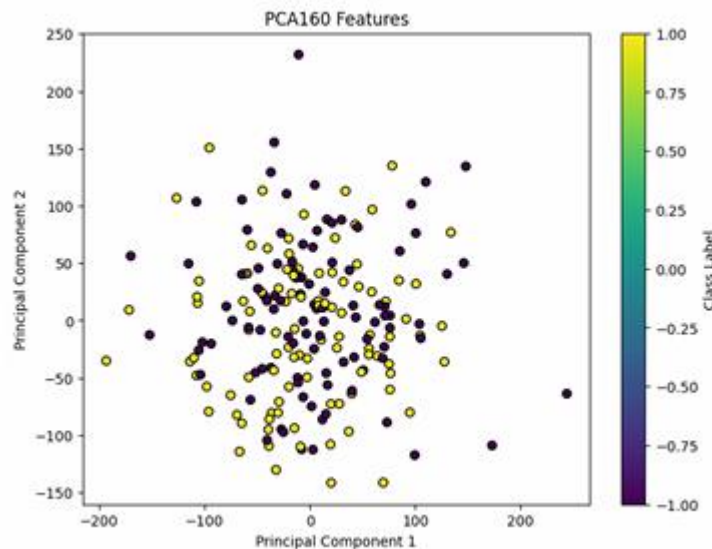
- Tiêu đề của biểu đồ (**title**) cung cấp ngữ cảnh hoặc mô tả liên quan đến việc trực quan hóa PCA. Một thanh màu (color bar) cũng được bao gồm trong biểu đồ, chỉ ra các nhãn lớp tương ứng của các điểm dữ liệu.
- Biểu đồ:
 - Dữ liệu A:





- Dữ liệu B:





3.4. Linear Discriminant Analysis (LDA)

Phân Tích Phân Biệt Tuyến Tính (Linear Discriminant Analysis - LDA) là một kỹ thuật giảm số chiều nhằm tối đa hóa sự phân tách giữa các lớp trong dữ liệu bằng cách chiếu nó lên một không gian có số chiều thấp hơn.

1. LDA after CSP

```
[57] from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

n_classes = len(np.unique(window_label))
n_features = EEG_3D_mu_beta_car_T_train_csp58.shape[1]
n_components = min(n_features, n_classes - 1)
lda = LDA(n_components=n_components)
EEG_3D_mu_beta_car_T_train_csp58_lda = lda.fit_transform(EEG_3D_mu_beta_car_T_train_csp58, y_mu_beta_car_train)
EEG_3D_mu_beta_car_T_test_csp58_lda = lda.transform(EEG_3D_mu_beta_car_T_test_csp58)
print(EEG_3D_mu_beta_car_T_train_csp58_lda.shape)
```

(150, 1)

- Mã này áp dụng LDA trên các đặc trưng CSP để chiếu dữ liệu vào một không gian phân biệt phù hợp cho việc phân loại.
- Số lớp là số nhãn duy nhất và số đặc trưng là chiều thứ hai của dữ liệu. Số thành phần (n_components) thích hợp cho LDA là "Số lớp - 1" nếu nó nhỏ hơn số đặc trưng hiện tại.

2. LDA on original data

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

n_trials, n_samples, n_channels = EEG_3D_mu_beta_car.shape
EEG_3D_mu_beta_car_resaped = EEG_3D_mu_beta_car.reshape(n_trials, -1)
n_classes = len(np.unique(window_label))
n_features = EEG_3D_mu_beta_car_resaped.shape[1]
n_components = min(n_features, n_classes - 1)
lda = LDA(n_components=n_components)
EEG_3D_mu_beta_car_lda = lda.fit_transform(EEG_3D_mu_beta_car_resaped, window_label)

print(EEG_3D_mu_beta_car_lda.shape)
```

(200, 1)

- Sử dụng dữ liệu 3-D chưa được biến đổi, sau đó chuyển đổi thành 2 chiều để phù hợp với LDA. Sau đó, áp dụng LDA trên dữ liệu đó.
- Vì bộ dữ liệu của chỉ có hai lớp (trái và phải/chân), nên đặc trưng đầu ra của LDA là một chiều trong cả hai bộ dữ liệu.

3. Plot LDA

Chức năng này tương tự như hàm **plot-pca** đã được đề cập trước đó.

```
[59] def plot_lda(pca_features, labels, title):
    plt.figure(figsize=(8, 6))

    if n_components == 1:
        plt.scatter(pca_features[:, 0], np.zeros_like(pca_features[:, 0]), c=labels, cmap='viridis', edgecolor='k')
        plt.xlabel('Linear Discriminant 1')
        plt.yticks([])
    else:
        plt.scatter(lda_features[:, 0], pca_features[:, 1], c=labels, cmap='viridis', edgecolor='k')
        plt.xlabel('Linear Discriminant 1')
        plt.ylabel('Linear Discriminant 2')

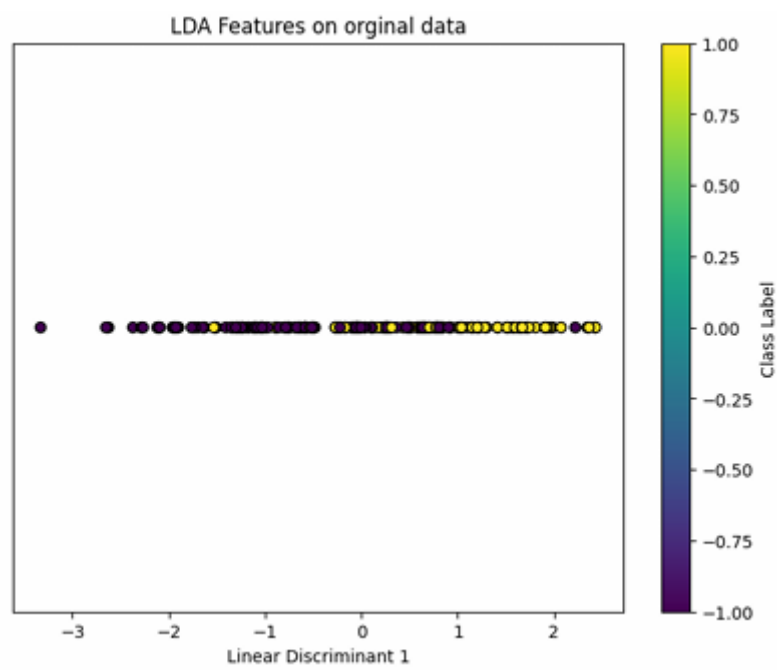
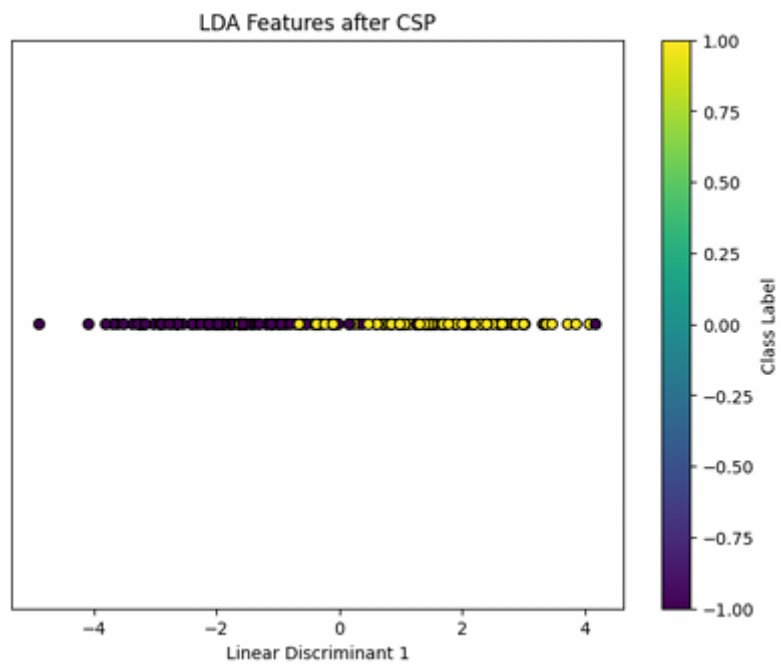
    plt.title('LDA Features'+ title)
    plt.colorbar(label='Class Label')
    plt.show()
```

4. Các kết quả kiểm tra và huấn luyện của lda rồi vẽ kết quả:

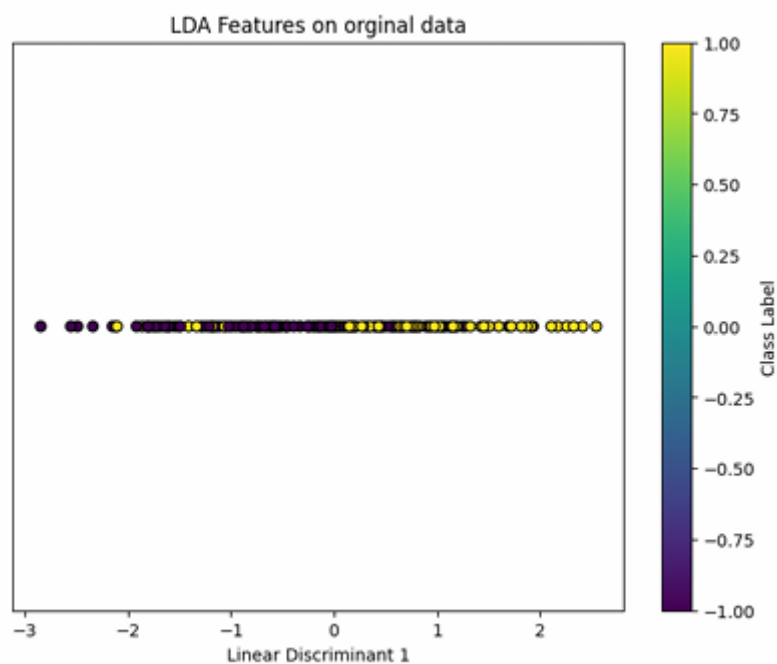
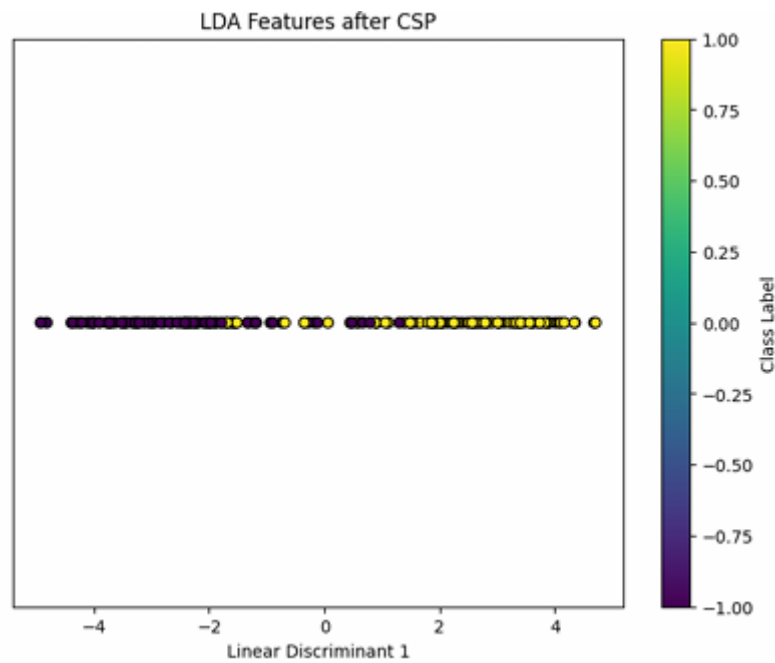
```
[60] EEG_3D_mu_beta_car_T_csp58_lda = np.concatenate((EEG_3D_mu_beta_car_T_train_csp58_lda,
    EEG_3D_mu_beta_car_T_test_csp58_lda), axis=0)
car_train_test_labels = np.concatenate((y_mu_beta_car_train, y_mu_beta_car_test), axis=0)
print("EEG LDA feature Shape", EEG_3D_mu_beta_car_T_csp58_lda.shape)
```

EEG LDA feature Shape (200, 1)

- Kết quả:
 - Dữ liệu A:



- Dữ liệu B:



- LDA phân tách dữ liệu một cách hiệu quả trong cả tập dữ liệu gốc và tập dữ liệu CSP-extracted, cho thấy hiệu quả vượt trội hiệu suất khi được áp dụng sau quá trình tiền xử lý CSP. Điều thú vị là, không giống như PCA và CSP, nó thực hiện tốt hơn trên tập dữ liệu A so với tập dữ liệu B.

3.5. Visual Comparisons

- CSP, PCA và LDA mang lại những lợi thế khác biệt tùy thuộc vào đặc điểm của tập dữ liệu và mục tiêu phân loại. CSP tỏ ra hiệu quả trong việc phân tách các lớp hình ảnh động cơ theo không gian, trong khi PCA cung cấp những hiểu biết sâu sắc về

phân bố phương sai, nhưng nó không hoạt động tốt trong việc trích xuất các tính năng đặc biệt.

- LDA, mặc dù chủ yếu được sử dụng trong 1D do hạn chế về dữ liệu, đã nêu bật những thách thức trong giảm kích thước cho dữ liệu EEG.
- Vì vậy, trong số các phương pháp trích xuất đặc trưng này, CSP được chứng minh là phương pháp tốt nhất.

Phần IV. Classification

4.1. Apply 8 Classifiers

- Trong phần này, sau khi áp dụng các bộ lọc khác nhau, chúng ta sử dụng các tín hiệu được lọc có tên “EEG_3D_mu_beta_car_T_csp58_pca10” để thực hiện phân loại. Đối với điều này, theo mã bên dưới, chúng tôi điều chỉnh 8 phân loại khác nhau trên dữ liệu tàu và sau đó lấy số liệu đánh giá bằng cách sử dụng thử nghiệm dữ liệu.


```
def classification(X_train, y_train, X_test, y_test):
    models = {
        'Logistic Regression': LogisticRegression(),
        'Decision Tree': DecisionTreeClassifier(),
        'Random Forest': RandomForestClassifier(),
        'SVM': SVC(),
        'KNN': KNeighborsClassifier(),
        'Gradient Boosting': GradientBoostingClassifier(),
        'AdaBoost': AdaBoostClassifier(),
        'Naive Bayes': GaussianNB(),
        'MLP Neural Network': MLPClassifier()
    }

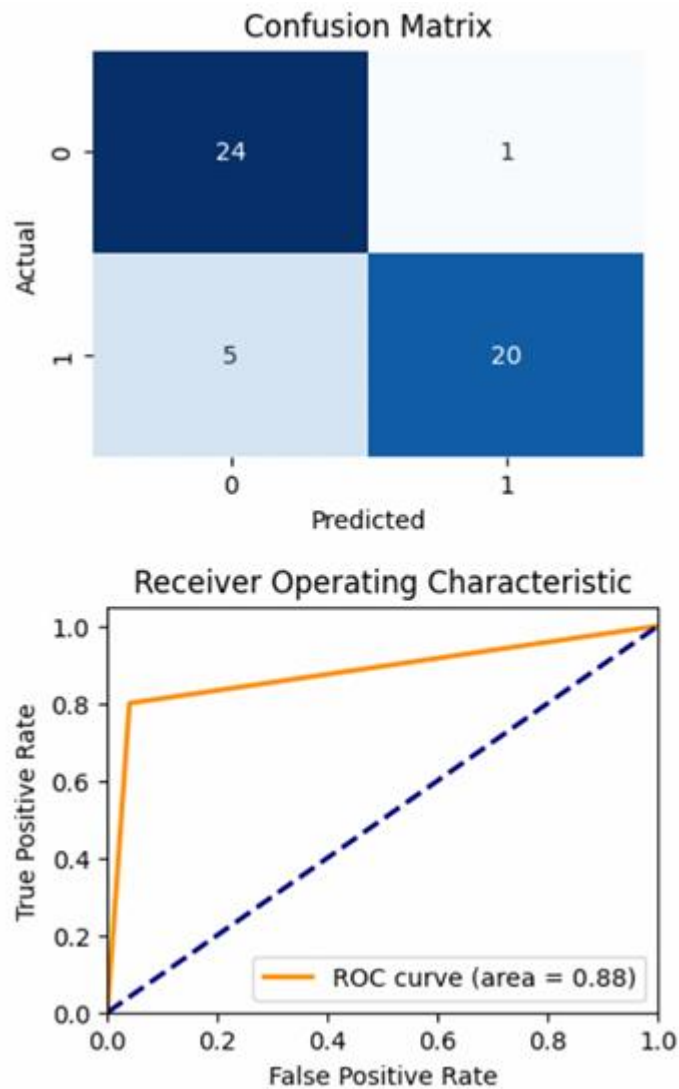
    results = {}
    for name, model in models.items():
        print()
        print(f"\033[38;5;208m{name}\033[0m")
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        results[name] = accuracy
        print(f"\033[32mAccuracy: {accuracy:.2f}\033[0m")
        print(classification_report(y_test, y_pred))
        confusion = confusion_matrix(y_test, y_pred)
        tn, fp, fn, tp = confusion.ravel()
        specificity = tp / (tp + fn)
        print(f"\033[95mSpecificity: {specificity:.2f}\033[0m")
```

- Dữ liệu A:
 - Bảng này dành cho dữ liệu có bộ lọc CAR và áp dụng PCA:

classifier	Accuracy	Precision	Recall	F1-SCORE
Logistic Regression	82	94	68	79
Decision Tree	72	79	60	68
Random Forest	80	100	60	75
SVM	84	90	76	83
KNN	68	76	52	62
Gradient Boosting	82	94	68	79
AdaBoost	78	94	60	73
Naïve Bayes	80	94	64	76
MLP	88	95	80	87

Table 1 – Evaluation Matrix of Classifiers with PCA

- Đường cong ROC và Ma trận nhầm lẫn cho tất cả các phân loại đã được vẽ. Ví dụ, sau đây kết quả thu được cho MLP:



- Bảng này dành cho dữ liệu có bộ lọc CAR và áp dụng LDA:
Như chúng ta có thể thấy kết quả phân loại bằng dữ liệu PCA tốt hơn nhiều so với phân loại bằng dữ liệu LDA.

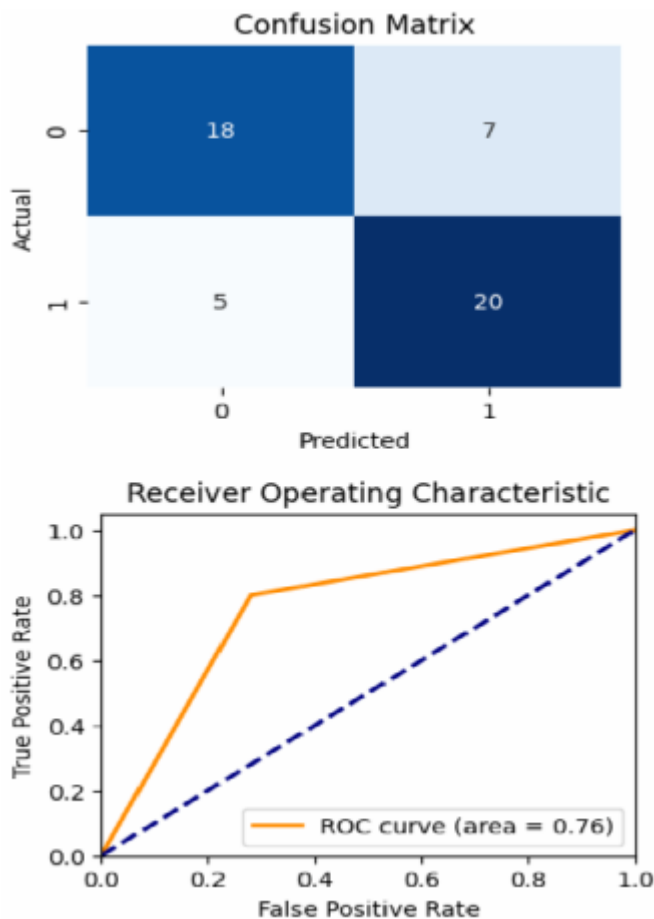
classifier	Accuracy	Precision	Recall	F1-SCORE
Logistic Regression	74	80	64	71
Decision Tree	72	79	60	68
Random Forest	72	79	60	68
SVM	74	80	64	71
KNN	78	89	64	74
Gradient Boosting	72	79	60	68
AdaBoost	72	79	60	68
Naïve Bayes	74	80	64	71
MLP	74	80	64	71

- Dữ liệu B:
 - Bảng này dành cho dữ liệu có bộ lọc CAR và áp dụng PCA:

classifier	Accuracy	Precision	Recall	F1-SCORE
Logistic Regression	84	81	88	85
Decision Tree	70	71	68	69
Random Forest	74	73	76	75
SVM	80	83	76	79
KNN	78	75	84	79
Gradient Boosting	74	75	72	73
AdaBoost	66	68	60	64
Naïve Bayes	82	83	80	82
MLP	76	74	80	77

Table 3 – Evaluation Matrix of Classifiers with PCA

- Đường cong ROC và Ma trận nhầm lẫn cho tất cả các phân loại đã được vẽ. Ví dụ, sau đây kết quả thu được cho MLP:



- Bảng này dành cho dữ liệu có bộ lọc CAR và áp dụng LDA:
Độ chính xác trung bình của dữ liệu sử dụng PCA là 76 và độ chính xác trung bình của dữ liệu sử dụng LDA là 80. Trong bộ dữ liệu này độ chính xác của phân loại bằng LDA cao hơn phân loại bằng PCA.

classifier	Accuracy	Precision	Recall	F1-SCORE
Logistic Regression	82	83	80	82
Decision Tree	78	82	72	77
Random Forest	78	82	72	77
SVM	80	80	80	80
KNN	82	83	80	82
Gradient Boosting	78	82	72	77
AdaBoost	78	82	72	77
Naïve Bayes	82	83	80	82
MLP	82	83	80	82

Table 4 – Evaluation Matrix of Classifiers with LDA

4.2. Một số bộ phân loại (với dữ liệu PCA) khác

1. MLP

```

import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, accuracy_score

mlp = MLPClassifier(hidden_layer_sizes=(100,), max_iter=200, random_state=42)
mlp.fit(EEG_3D_mu_beta_car_T_train_csp58_pca10, y_mu_beta_car_train)
y_pred = mlp.predict(EEG_3D_mu_beta_car_T_test_csp58_pca10)

# Evaluate the model
print("Accuracy:", accuracy_score(y_mu_beta_car_test, y_pred))
print("Classification Report:\n", classification_report(y_mu_beta_car_test, y_pred))

```

```

Accuracy: 0.9
Classification Report:
              precision    recall  f1-score   support

     -1         0.86      0.96      0.91         25
       1         0.95      0.84      0.89         25

 accuracy          0.90         50
  macro avg         0.91         50
 weighted avg         0.91         50

```

Bằng cách điều chỉnh các tham số của bộ phân loại MLP, em đã đạt được độ chính xác 90%

2. Gradient Boosting Classifier

Trình phân loại tăng cường độ dốc (Gradient Boosting Classifier - GBC) là một phương pháp học máy tổng hợp kết hợp các dự đoán của một số mô hình yếu (thường là cây quyết định) để cải thiện độ chính xác dự đoán. Nó thực hiện việc huấn luyện tuần tự, với mỗi mô hình mới được điều chỉnh để cải thiện độ chính xác ở những khu vực mà các mô hình trước đó hoạt động kém. Trong trường hợp này, chúng tôi sử dụng GBC để phân loại dữ liệu EEG sau khi đã trích xuất các đặc trưng.

```

from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

def gradient_boosting(X_train, y_train, X_test, y_test):

    param_grid = {
        'n_estimators': [100, 200],
        'learning_rate': [0.01, 0.1],
        'max_depth': [3,4,5],
        'min_samples_split': [6],
        'min_samples_leaf': [4],
        'subsample': [0.9],
        'max_features': ['log2']
    }

    gbc = GradientBoostingClassifier()
    grid_search = GridSearchCV(gbc, param_grid, cv=5, scoring='accuracy', verbose=2)
    grid_search.fit(X_train, y_train)

    best_params = grid_search.best_params_
    print(f'Best parameters: {best_params}')
    best_gbc = GradientBoostingClassifier(**best_params)
    best_gbc.fit(X_train, y_train)

    y_pred = best_gbc.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    print(f'Accuracy with best parameters: {accuracy}')
    return best_gbc

```

- Hàm **gradient_boosting** sử dụng **GridSearchCV** để tìm các siêu tham số tối ưu cho Trình phân loại Tăng cường Độ dốc (Gradient Boosting Classifier - GBC). Sau đó, hàm huấn luyện bộ phân loại với các tham số này trên dữ liệu huấn luyện (X_{train} , y_{train}) và đánh giá hiệu suất của nó trên dữ liệu kiểm tra (X_{test} , y_{test}). Độ chính xác của bộ phân loại với các tham số tốt nhất sẽ được in ra, và mô hình đã huấn luyện (**best_gbc**) sẽ được trả về để đánh giá hoặc dự đoán thêm.
- Để tối ưu hóa các tham số của Gradient Boosting Classifier (GBC), chúng tôi thực hiện một tìm kiếm lưới (grid search) sử dụng GPU để tính toán hiệu quả. Ban đầu, chúng tôi đã thử nghiệm một phạm vi rộng các siêu tham số để tìm cấu hình tốt nhất.
- Các tham số trong Grid Search:
 - **n_estimators**: Số lượng giai đoạn tăng cường (100, 200, 300).
 - **learning_rate**: Bước nhảy cho mỗi vòng lặp tăng cường (0.01, 0.1, 0.2).
 - **max_depth**: Độ sâu tối đa của các cây quyết định (3, 4, 5).
 - **min_samples_split**: Số lượng mẫu tối thiểu cần có để chia một nút trong cây (2, 5, 10).
 - **min_samples_leaf**: Số lượng mẫu tối thiểu cần có tại một nút lá (1, 2, 4).
 - **subsample**: Tỷ lệ phần trăm mẫu được sử dụng để huấn luyện mỗi cây (0.8, 0.9, 1.0).
 - **max_features**: Số lượng tính năng tối đa được xem xét để chia một nút ('auto', 'sqrt', 'log2').
- Sau khi xác định được các tham số hoạt động tốt nhất, chúng tôi tinh chỉnh lưới tìm kiếm để tập trung vào các kết hợp hiệu quả nhất, nhằm giảm tải tính toán trong khi vẫn duy trì hiệu suất tối ưu.

một cách tuần tự, trong đó mỗi học viên tiếp theo chú ý nhiều hơn đến các trường hợp bị phân loại sai bởi các học viên trước. Quá trình lặp lại này giúp AdaBoost tập trung vào các trường hợp khó, từ đó cải thiện độ chính xác tổng thể.

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

def ada_boost(X_train, y_train, X_test, y_test):

    weak_classifier = GaussianNB()

    ada_boost = AdaBoostClassifier(base_estimator=weak_classifier, n_estimators=100,
                                   learning_rate=.01, random_state=42)

    ada_boost.fit(X_train, y_train)
    y_pred = ada_boost.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy: {accuracy:.2f}")
ada_boost(EEG_3D_mu_beta_car_T_train_csp58_pca10, y_mu_beta_car_train,
          EEG_3D_mu_beta_car_T_test_csp58_pca10, y_mu_beta_car_test)
```

Accuracy: 0.80

- Các tham số:
 - **base_estimator**: Bộ phân loại cơ bản để được tăng cường. Trong trường hợp này, GaussianNB() được sử dụng như bộ phân loại yếu.
 - **n_estimators**: Số vòng tăng cường hoặc số học viên yếu được huấn luyện (100 trong trường hợp này).
 - **learning_rate**: Trọng số áp dụng cho sự đóng góp của mỗi bộ phân loại trong quá trình huấn luyện (0.01 trong trường hợp này).
 - **random_state**: Hạt giống để tạo ra số ngẫu nhiên, đảm bảo tính tái sản xuất.
- Hàm **ada_boost** triển khai AdaBoost với Gaussian Naive Bayes (GaussianNB) làm bộ phân loại yếu. Nó huấn luyện bộ phân loại AdaBoost với **n_estimators=100** vòng tăng cường, mỗi học viên yếu được ảnh hưởng bởi các trường hợp bị phân loại sai với trọng số theo **learning_rate=0.01**. Sau khi huấn luyện, nó dự đoán nhãn cho bộ dữ liệu kiểm tra và tính toán độ chính xác của các dự đoán bằng cách sử dụng **accuracy_score**.
- Việc thiết lập tham số được thực hiện thông qua tìm kiếm lưới trên các tham số khác nhau và triển khai với các tham số tốt nhất có sẵn trong tìm kiếm lưới.
- **Kết quả**:
 - Dữ liệu A:

```
ada_boost(EEG_3D_mu_beta_car_T_train_csp58_pca10, y_mu_beta_car_train,
          EEG_3D_mu_beta_car_T_test_csp58_pca10, y_mu_beta_car_test)
```

Accuracy: 0.80


```
[ ] ada_boost(EEG_3D_mu_beta_car_T_train_csp58_lda, y_mu_beta_car_train,
              EEG_3D_mu_beta_car_T_test_csp58_lda, y_mu_beta_car_test)
```

```
⇒ Accuracy: 0.74
```

4. K-Nearest Neighbors (KNN) Classifier

- K-Nearest Neighbors (KNN) là một thuật toán học dựa trên cá thể, không tham số được sử dụng để nhiệm vụ phân loại. Nó phân loại các điểm dữ liệu dựa trên lớp đa số trong số k gần nhất của chúng. lân cận trong không gian đặc trưng. Việc lựa chọn k ảnh hưởng đến sự đánh đổi độ lệch-phương sai của mô hình: nhỏ hơn các giá trị làm cho mô hình nhạy cảm hơn với nhiễu, trong khi các giá trị lớn hơn làm mịn ranh giới quyết định.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

def KNN_classifier(X_train, y_train, X_test, y_test, k):

    param_grid = {'n_neighbors': np.arange(1, 31)}
    knn = KNeighborsClassifier()
    grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')
    grid_search.fit(X_train, y_train)
    best_k = grid_search.best_params_['n_neighbors']

    print(f'The best value of k is: {best_k}')

    best_knn = KNeighborsClassifier(n_neighbors=best_k)
    best_knn.fit(X_train, y_train)

    y_pred = best_knn.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    print(f'Accuracy with k={k}: {accuracy}')
```

- Nếu tham số k không được cung cấp thủ công, hàm sẽ tự động chọn số lượng láng giềng tốt nhất (n_neighbors) thông qua tìm kiếm lưới. Hàm sau đó dự đoán nhãn cho dữ liệu kiểm tra (X_test) và đánh giá độ chính xác của các dự đoán bằng cách sử dụng **accuracy_score**. Cách tiếp cận này đảm bảo rằng mô hình KNN được điều chỉnh để hoạt động tối ưu trên bộ dữ liệu EEG, tận dụng ưu thế của phân loại dựa trên láng giềng để thực hiện các nhiệm vụ phân loại hình ảnh động cơ chính xác.
- Kết quả:
 - Dữ liệu A:

```
[ ] KNN_classifier(EEG_3D_mu_beta_car_T_train_csp58_pca10, y_mu_beta_car_train,
                  EEG_3D_mu_beta_car_T_test_csp58_pca10, y_mu_beta_car_test, 5)
```

```
⇒ The best value of k is: 24
   Accuracy with k=5: 0.72
```



```

KNN_classifier(EEG_3D_mu_beta_car_T_train_csp58_lda, y_mu_beta_car_train,
               EEG_3D_mu_beta_car_T_test_csp58_lda, y_mu_beta_car_test,5)

```

The best value of k is: 8
 Accuracy with k=5: 0.74

- KNN được huấn luyện với ít đặc trưng hơn sẽ hoạt động tốt hơn. Trước đây, có khoảng cách 10% về độ chính xác giữa LDA và PCA, với LDA hoạt động tốt hơn khi chỉ sử dụng một đặc trưng. Tuy nhiên, sau khi thay đổi hạt giống (seed), chúng tôi không còn quan sát được sự khác biệt này, và cả hai phương pháp đều hoạt động gần như tương đương.

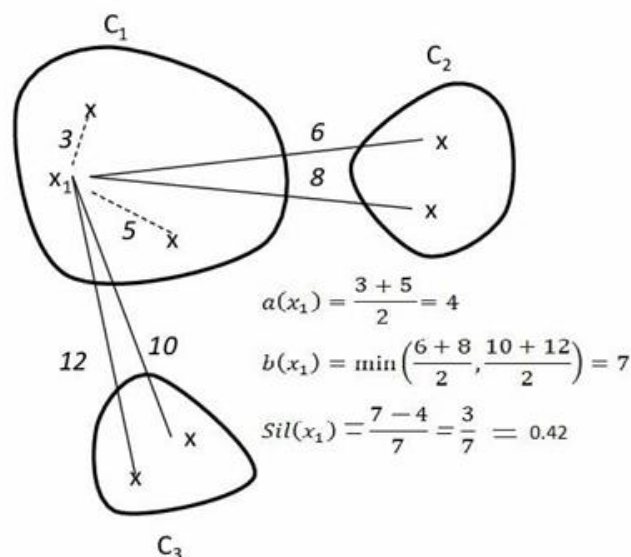
Phần V. Clustering

5.1. Silhouette Score and Plot

- Điểm hình bóng là thước đo được sử dụng để đánh giá chất lượng phân cụm. Nó đo lường mức độ giống nhau của một đối tượng nằm trong cụm riêng của nó so với các cụm khác.
- Đầu tiên hãy tính khoảng cách trung bình tới tất cả các điểm khác trong cùng một cụm. Điều này được gọi là a_i . Sau đó tính khoảng cách trung bình tới tất cả các điểm trong cụm lân cận gần nhất (b_i). Điểm hình bóng (s_i) cho mỗi điểm sau đó được đưa ra:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

An example of silhouette score:



- Biểu đồ silhouette là một biểu diễn đồ họa của điểm số silhouette của từng điểm dữ liệu trong các nhóm. Dưới đây là cách xác định biểu đồ silhouette tốt nhất:
 1. **Điểm số Silhouette Trung Bình Cao:** Biểu đồ silhouette tốt nhất sẽ có điểm số silhouette trung bình cao trên tất cả các nhóm, lý tưởng là gần +1. Điều này cho thấy các nhóm được phân tách tốt và các điểm dữ liệu được phân nhóm hợp lý.
 2. **Độ Rộng Nhất Quán của Silhouette:** Độ rộng silhouette nên đồng đều giữa các nhóm. Sự khác biệt lớn trong độ rộng silhouette có thể chỉ ra rằng một số nhóm có độ gọn hơn những nhóm khác, điều này có thể chỉ ra chất lượng phân nhóm không đồng đều.
 3. **Giá trị Silhouette Dương:** Hầu hết các điểm dữ liệu nên có giá trị silhouette dương, chỉ ra rằng chúng gần hơn với nhóm của mình hơn là với các nhóm láng giềng. Một số lượng lớn các giá trị âm có thể chỉ ra sự phân loại sai.
 4. **Kích Thước Nhóm Cân Đối:** Biểu đồ nên hiển thị các nhóm có kích thước gần bằng nhau. Một giải pháp phân nhóm tốt không có nhóm nào lớn hoặc nhỏ hơn đáng kể so với các nhóm khác trừ khi có lý do hợp lý cho điều đó.
 5. **Tối Thiểu Sự Chồng Lấn:** Không nên có sự chồng lấn giữa silhouette của các nhóm khác nhau. Sự chồng lấn silhouette chỉ ra rằng các nhóm không được phân tách tốt.
- Điểm số silhouette và biểu đồ silhouette của chúng.

```
def plot_silhouette(data, max_clusters=10):
    range_n_clusters = range(2, max_clusters + 1)
    silhouette_avg_scores = []

    for n_clusters in range_n_clusters:
        fig, ax1 = plt.subplots(1, 1)
        fig.set_size_inches(4, 3)

        clusterer = KMeans(n_clusters=n_clusters, random_state=10)
        cluster_labels = clusterer.fit_predict(data)

        silhouette_avg = silhouette_score(data, cluster_labels)
        silhouette_avg_scores.append(silhouette_avg)
        print(f"\033[95mFor n_clusters = {n_clusters}, the average silhouette_score is : {silhouette_avg}\n")

        sample_silhouette_values = silhouette_samples(data, cluster_labels)

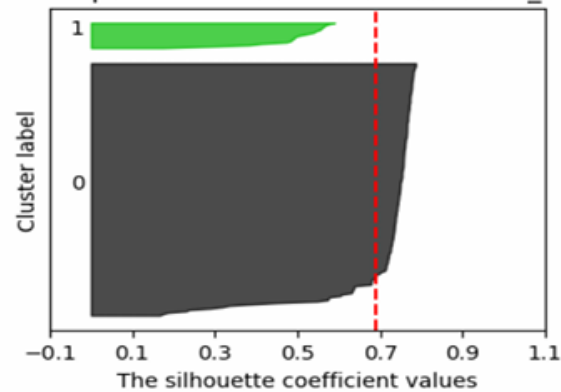
        y_lower = 10
        for i in range(n_clusters):
            ith_cluster_silhouette_values = sample_silhouette_values[cluster_labels == i]
            ith_cluster_silhouette_values.sort()
```

- Dữ liệu A:
 - Biểu đồ silhouette cho 2 cụm:

Một trong các cụm lớn hơn nhiều so với cụm còn lại và các cụm không cân bằng.

For $n_clusters = 2$, the average silhouette_score is : 0.6914144956357596

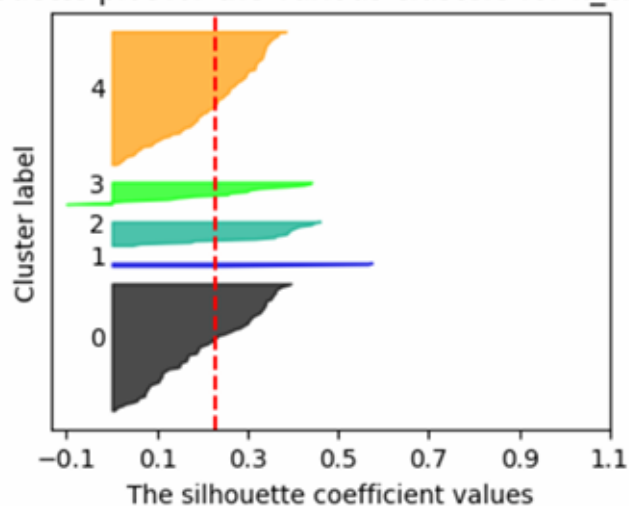
The silhouette plot for the various clusters for $n_clusters = 2$



- Biểu đồ silhouette cho 5 cụm:
Các cụm mất cân bằng và điểm bóng thấp hơn điểm của 2 cụm.

For $n_clusters = 5$, the average silhouette_score is : 0.22917696041580043

The silhouette plot for the various clusters for $n_clusters = 5$

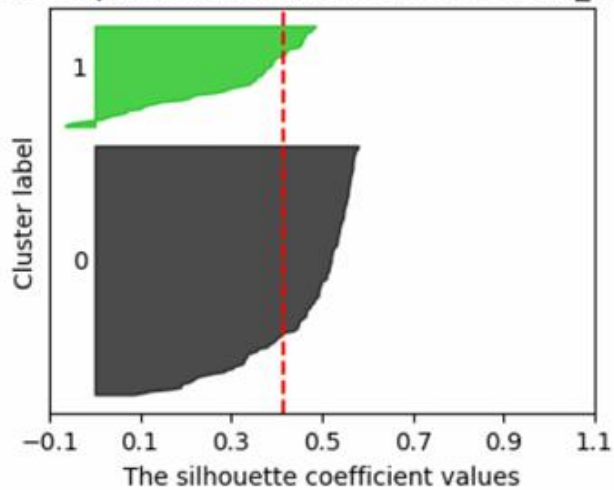


55

- Dữ liệu B:
 - Biểu đồ silhouette cho 2 cụm:
Một trong các cụm lớn hơn nhiều so với cụm còn lại và các cụm không cân bằng.

For `n_clusters = 2`, the average silhouette_score is : 0.4166689924158162

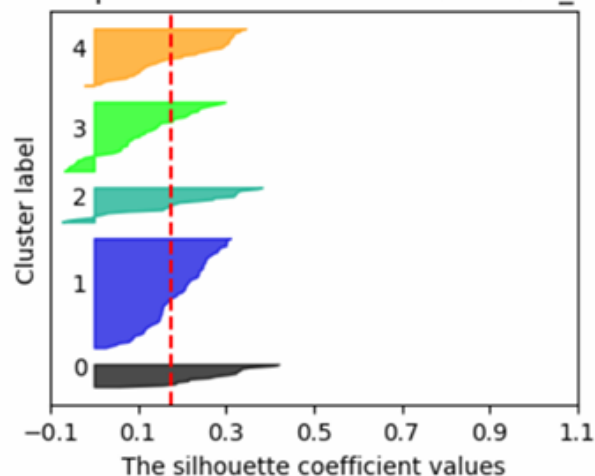
The silhouette plot for the various clusters for `n_clusters = 2`



- Biểu đồ silhouette cho 5 cụm:
Các cụm mất cân bằng và điểm bóng thấp hơn điểm của 2 cụm.

For `n_clusters = 5`, the average silhouette_score is : 0.17382354907673617

The silhouette plot for the various clusters for `n_clusters = 5`



5.2. K-Means:

- Áp dụng phương pháp K-mean để tìm nhãn và tâm của cụm.
- Sau đó, bằng cách sử dụng mã tiếp theo, chúng tôi giảm tính chiều của dữ liệu thành hai chiều bằng PCA để vẽ chúng.

```

import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans

def apply_kmeans(data, n_clusters):

    kmeans = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = kmeans.fit_predict(data)
    cluster_centers = kmeans.cluster_centers_

    return cluster_labels, cluster_centers

def plot_clusters(data, cluster_labels, cluster_centers):

    pca = PCA(n_components=2)
    reduced_data = pca.fit_transform(data)

    plt.figure(figsize=(6, 4))

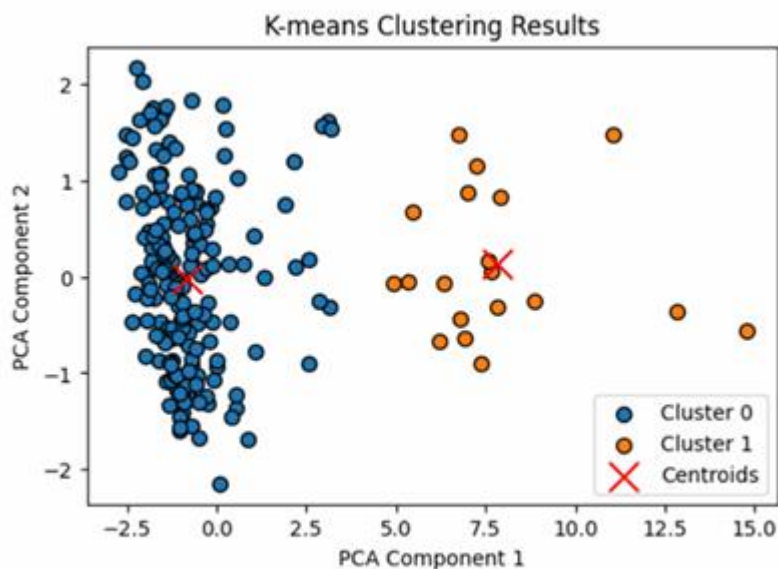
    unique_labels = np.unique(cluster_labels)
    colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
    for k, col in zip(unique_labels, colors):
        class_member_mask = (cluster_labels == k)
        xy = reduced_data[class_member_mask]
        plt.scatter(xy[:, 0], xy[:, 1], label=f'Cluster {k}', edgecolor='k', s=50)

    reduced_centers = pca.transform(cluster_centers)
    plt.scatter(reduced_centers[:, 0], reduced_centers[:, 1], marker='x', s=200, c='red', label='Centroids')

    plt.title("K-means Clustering Results")
    plt.xlabel("PCA Component 1")
    plt.ylabel("PCA Component 2")
    plt.legend()
    plt.show()

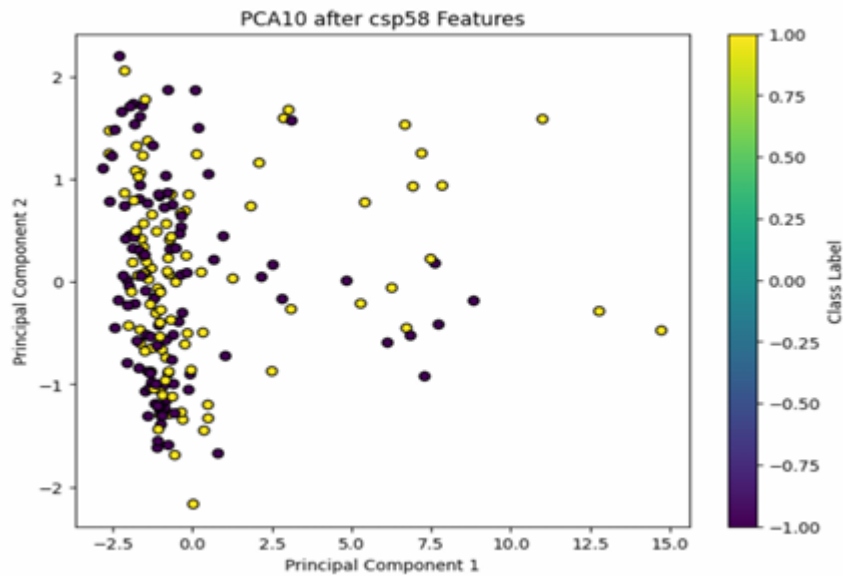
```

- Kết quả:
 - Dữ liệu A:
 - sơ đồ 2 cụm sử dụng k-means:

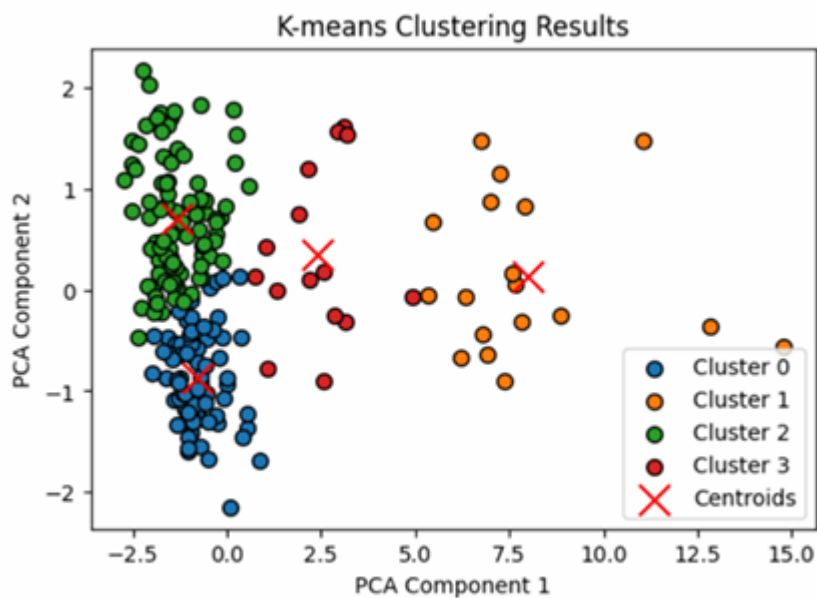


- Nhãn dữ liệu:

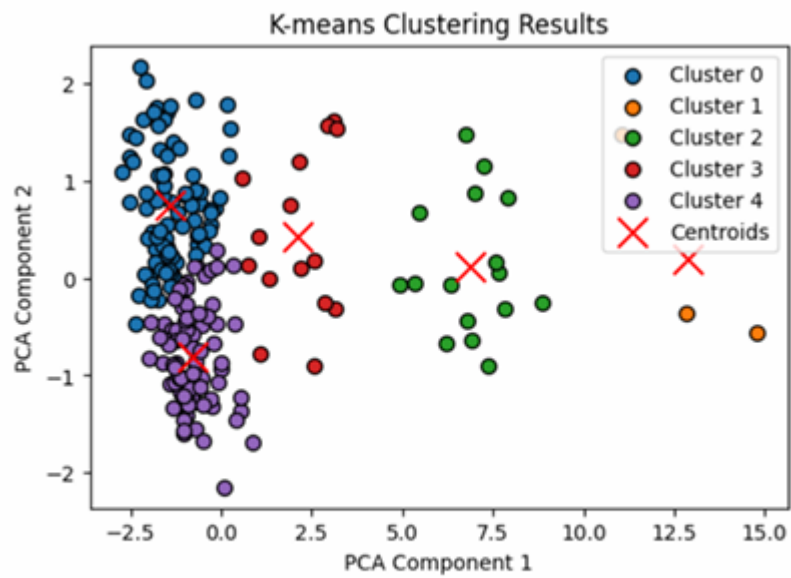
Do tính chất của dữ liệu, không thể tách dữ liệu bằng cách phân cụm này phương pháp. Dữ liệu từ các cụm khác nhau được trộn lẫn với nhau.



- 4 clusters:



- 5 clusters:



5.3. DBSCAN

- Áp dụng phương pháp Quét DB để tìm nhãn và tâm của cụm.
- Để tìm các tham số tối ưu cho phương pháp DBSCAN, sử dụng tìm kiếm dạng lưới

```

import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.decomposition import PCA
import matplotlib.cm as cm

def apply_dbscan(data, eps=0.5, min_samples=5):

    dbscan = DBSCAN(eps=eps, min_samples=min_samples)
    cluster_labels = dbscan.fit_predict(data)

    return cluster_labels

def plot_dbscan_clusters(data, cluster_labels):

    pca = PCA(n_components=2)
    reduced_data = pca.fit_transform(data)

    plt.figure(figsize=(7, 6))

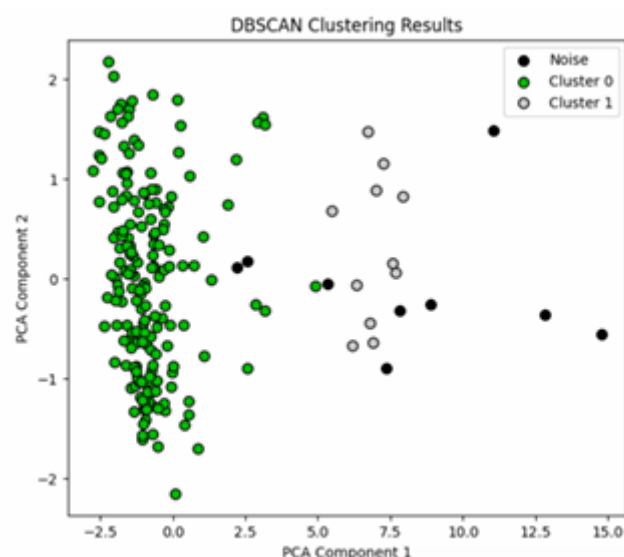
    unique_labels = np.unique(cluster_labels)
    colors = [cm.nipy_spectral(each) for each in np.linspace(0, 1, len(unique_labels))]
    for k, col in zip(unique_labels, colors):
        if k == -1:
            col = [0, 0, 0, 1]

        class_member_mask = (cluster_labels == k)
        xy = reduced_data[class_member_mask]
        plt.scatter(xy[:, 0], xy[:, 1], c=[col], label=f'Cluster {k}' if k != -1 else 'Noise', edgecolor=

    plt.title("DBSCAN Clustering Results")
    plt.xlabel("PCA Component 1")
    plt.ylabel("PCA Component 2")
    plt.legend()
    plt.show()

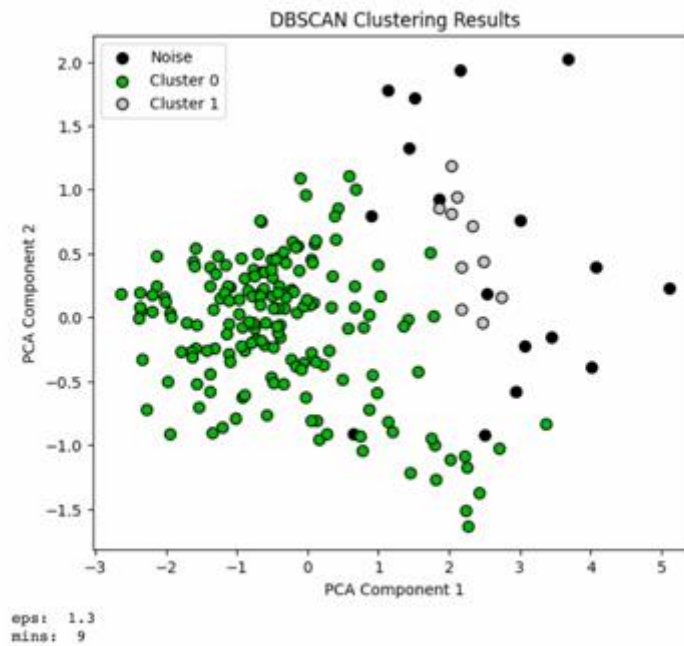
```

- Kết quả:
 - Dữ liệu A:
- Sau khi thực hiện tìm kiếm dạng lưới, các tham số phân cụm dữ liệu hiệu quả là: Epsilon = 3, mẫu tối thiểu = 12



- Dữ liệu B:

Sau khi thực hiện tìm kiếm dạng lưới, các tham số phân cụm dữ liệu hiệu quả là: Epsilon = 1,3, mẫu tối thiểu = 9

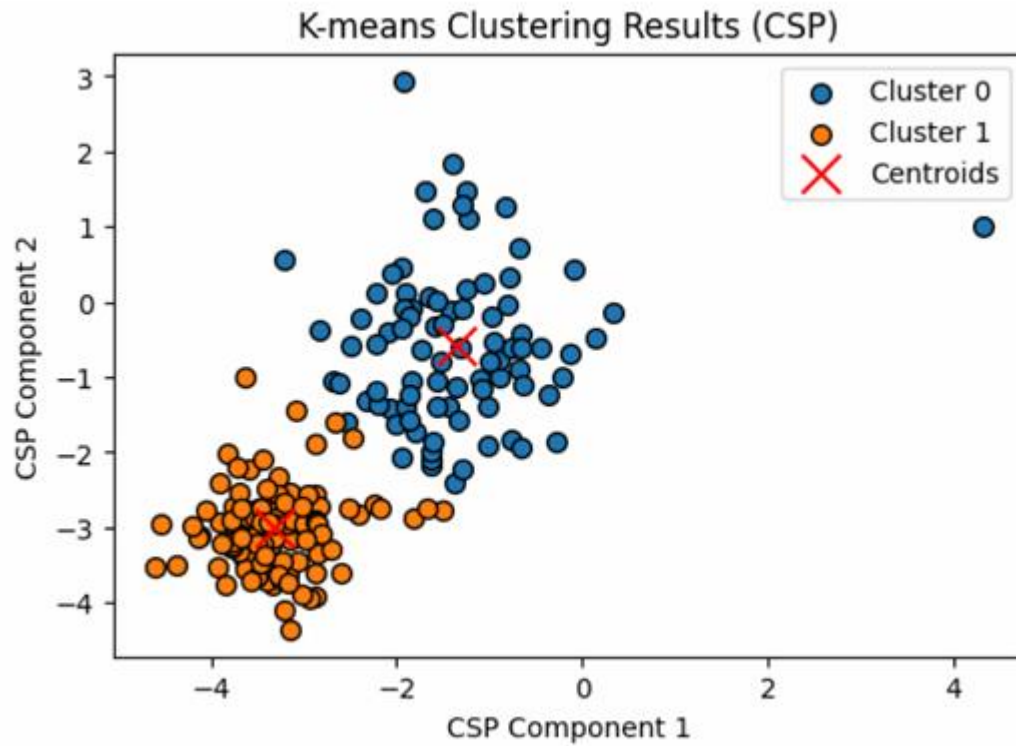


- Dữ liệu A:

Như chúng ta đã biết, trong CSP, trước tiên chúng ta chỉ định số lượng cửa sổ, sau đó là số kênh và cuối cùng là số lượng mẫu trên mỗi cửa sổ. Tuy nhiên, chúng tôi vô tình phát hiện ra rằng nếu thay đổi số của các kênh và số lượng mẫu trên mỗi cửa sổ trong CSP, dữ liệu sẽ được phân tách tốt để phân cụm.

Việc phân cụm được áp dụng thành công.

- Kết quả phân cụm:



○ Real labels:

