

گزارش پروژه درس BSS

علیرضا حسینی

شماره دانشجویی : ۸۱۰۱۰۱۱۴۲

جداسازی کور منابع

دکتر اخوان

بهار 1401

فهرست مطالب

۱-۱- چکیده.....	5
۱-۲- مقدمه.....	5
۱-۳- مجموعه داده.....	7
۱-۴- الگوی فضایی مشترک (CSP).....	8
1-4-1- مقدمه و توضیحات.....	8
1-4-2- پیاده سازی CSP.....	9
۱-۵- تحلیل تشخیص خطی.....	11
1-5-1- مقدمه و توضیحات.....	11
1-5-2- پیاده سازی LDA.....	13
۱-۶- متود بهینه انتخابی.....	13
۱-۷- لود دیتاست و پیش پردازش.....	14
۱-۸- الگوریتم leave one out برای جداسازی تست و ترین.....	17
۱-۹- رویکرد های متفاوت آموزش طبقه بند ها.....	19
1-9-1- استفاده تمام داده ها و طراحی فقط 3 مدل طبقه بند و جداسازی LOO.....	19
1-9-2- استفاده تمام داده ها و طراحی فقط 3 مدل طبقه بند و جداسازی رندوم.....	27
1-9-3- استفاده تمام داده ها و طراحی فقط 1 طبقه بند با SVM و جداسازی رندوم.....	29
1-9-4- آموزش 15 مدل برای 15 تا دیتا با جداسازی با روش LOO.....	30
۱-۱۰- نتیجه گیری.....	43
۱-۱۱- ضمیمه : توضیح و نحوه ران کد ارسالی.....	44

فهرست اشکال

شکل (۱-۱) پیاده سازی CSP در متلب.....	10
شکل (۲-۱) پیاده سازی CSP با استفاده از کتاب خانه های آماده.....	10
شکل (۳-۱) پیاده سازی الگوریتم CSP در پایتون بدون استفاده از الگوریتم های آماده.....	11
شکل (۴-۱) کانال های استفاده شده برای حل مساله طبقه بندی تصور حرکتی.....	14
شکل (۵-۱) انتخاب کانال های مورد علاقه برای بررسی و ایجاد متغیر فیچر ها و لیل ها.....	15
شکل (۶-۱) کد پایتون خواندن دیتا ها.....	15
شکل (۷-۱) چسباندن فیچر ها در یک ماتریس X و transpose آن.....	16
شکل (۸-۱) کد متلب بخش لود کردن و پیش پردازش داده ها.....	16
شکل (۹-۱) کد پایتون پیاده سازی الگوریتم LOO.....	18
شکل (۱۰-۱) کد پایتون ایجاد دیتاست داده های هر کلاس و تمیز کردن داده ها.....	19
شکل (۱۱-۱) کد متلب پیاده سازی الگوریتم LOO.....	19
شکل (۱۲-۱) کد پایتون آماده سازی X و Y برای داده های آموزشی و آزمایشی طبقه بند اول.....	20
شکل (۱۳-۱) اعمال CSP رو X و Y های طبقه بند اول.....	21
شکل (۱۴-۱) اعمال LDA بر طبقه بند اول.....	21
شکل (۱۵-۱) گرفتن پیشبینی مدل روی داده های تست و ترین.....	21
شکل (۱۶-۱) محاسبه دقت و ماتریس آشفتگی ، طبقه بند اول.....	21
شکل (۱۷-۱) طراحی طبقه بند دوم (مقایسه کلاس سوم در مقابل کلاس اول و دوم).....	22
شکل (۱۸-۱) طراحی طبقه بند سوم (مقایسه کلاس 1 و 2).....	23
شکل (۱۹-۱) تعریف فانکشن CSP در متلب.....	23
شکل (۲۰-۱) کد متلب طراحی و خروجی گرفتن از طبقه بند سوم.....	23
شکل (۲۱-۱) ماتریس آشفتگی و دقت در طبقه بند اول.....	24
شکل (۲۲-۱) ماتریس آشفتگی و دقت در طبقه بند اول.....	24
شکل (۲۳-۱) ماتریس آشفتگی و دقت در طبقه بند اول.....	24
شکل (۲۴-۱) چسباندن دیتای تمام کلاس ها به هم.....	25
شکل (۲۵-۱) پیاده سازی الگوریتم تشخیص 4 کلاس.....	26
شکل (۲۶-۱) دقت و ماتریس آشفتگی مدل با 3 طبقه بند روی تمام داده های آموزشی و آزمایش.....	27
شکل (۲۷-۱) کد پایتون مشاهده وزن های CSP و LDA.....	27
شکل (۲۸-۱) نمونه نحوه تقسیم رندوم داده ها.....	28
شکل (۲۹-۱) دقت و ماتریس آشفتگی طبقه بند اول با دیتای تمام 15 نفر و تقسیم رندوم داده ها.....	28
شکل (۳۰-۱) دقت و ماتریس آشفتگی طبقه بند دوم با دیتای تمام 15 نفر و تقسیم رندوم داده ها.....	28
شکل (۳۱-۱) دقت و ماتریس آشفتگی طبقه بند سوم با دیتای تمام 15 نفر و تقسیم رندوم داده ها.....	29
شکل (۳۲-۱) آموزش مدل SVM.....	29

- شکل (۳۳-۱) ماتریس آشفته‌گی و دقت در حالت استفاده از SVM 4 کلاسه 30
- شکل (۳۴-۱) بخش‌های مختلف نوت بوک ارسالی 44
- شکل (۳۵-۱) کد متلب دانلود دیتاست 45

**** توضیح اصلی کد ها در بخش 1-9-1 میباشد و به علت تکراری بودن کد در بخش های دیگر فقط**

نتایج در بخش های دیگر آمده است.

۱-۱- چکیده

طبقه‌بندی تصور حرکتی^۱ با استفاده از سیگنال‌های EEG در سال‌های اخیر به دلیل کاربردهای بالقوه آن در رابط‌های مغز و کامپیوتر (BCIs) و توانبخشی عصبی توجه قابل توجهی را به خود جلب کرده است. این گزارش به بررسی مساله طبقه‌بندی تصور حرکتی با استفاده از سیگنال‌های EEG می‌پردازد و روشی را شامل الگوریتم الگوهای فضایی مشترک (CSP) برای استخراج ویژگی و تحلیل تشخیصی خطی (LDA) برای طبقه‌بندی پیشنهاد می‌کند. مجموعه داده مورد استفاده در این مطالعه شامل چهار کلاس است: کلاس یک حرکات تصویری انگشت شست را نشان می‌دهد، کلاس دو حرکات تصویری بازو را نشان می‌دهد، کلاس سوم حرکات تصویری پا را نشان می‌دهد و کلاس چهار نشان دهنده هیچ حرکتی است.

۱-۲- مقدمه

طبقه‌بندی تصور حرکتی با استفاده از سیگنال‌های EEG به عنوان یک حوزه تحقیقاتی مهم در زمینه رابط‌های مغز و کامپیوتر (BCIs) و توانبخشی عصبی ظاهر شده است. توانایی رمزگشایی و طبقه‌بندی اهداف پشت حرکات تصویری، این پتانسیل را دارد که تعامل انسان و رایانه را متحول کند و به بهبودی افراد دارای اختلالات حرکتی کمک کند. با تجزیه و تحلیل سیگنال‌های الکتروانسفالوگرام (EEG) که فعالیت الکتریکی مغز را ثبت

¹ Motor Imagary

می کند، محققان می توانند الگوهای را کشف کنند که با وظایف تصور حرکتی خاصی مطابقت دارد.

تصور حرکتی به شبیه سازی ذهنی حرکت بدون هیچ گونه اجرای فیزیکی اشاره دارد. مشاهده شده است که وقتی افراد به وضوح انجام یک کار حرکتی را تصور می کنند، فعالیت مغز آنها الگوهای مشخصی را نشان می دهد که شبیه الگوهای مشاهده شده در طول اجرای واقعی حرکتی است. این الگوها را می توان از سیگنال های EEG شناسایی و رمزگشایی کرد و ابزاری برای درک فرآیندهای عصبی اساسی درگیر در تصاویر حرکتی فراهم می کند.

طبقه بندی وظایف تصویرسازی حرکتی نوید زیادی برای توسعه رابط های مغز و کامپیوتر دارد. BCI به افراد اجازه می دهد تا دستگاه های خارجی را کنترل کنند یا با محیط های مجازی به طور مستقیم از طریق فعالیت مغز خود تعامل داشته باشند و نیاز به روش های ورودی فیزیکی سنتی را دور بزنند. با استفاده از طبقه بندی تصاویر حرکتی، BCI می تواند افراد را قادر سازد تا اندام مصنوعی را کنترل کنند، در محیط های واقعیت مجازی حرکت کنند یا بدون تکیه بر عملکردهای حرکتی آسیب دیده خود ارتباط برقرار کنند.

علاوه بر این، طبقه بندی تصور حرکتی پیامدهای مهمی برای توانبخشی عصبی دارد. افرادی که دارای اختلالات حرکتی هستند، مانند بیماران سکته مغزی یا کسانی که آسیب نخاعی دارند، اغلب در بازیابی کنترل حرکتی با چالش هایی روبرو هستند. با گنجاندن وظایف تصویرسازی حرکتی در پروتکل های توانبخشی، بیماران می توانند در تمرین ذهنی شرکت کنند که مسیرهای عصبی مرتبط با حرکت را تحریک می کند. بازخورد بلادرنگ براساس طبقه بندی حرکات تصویری می تواند اثربخشی برنامه های توانبخشی را افزایش دهد، انعطاف پذیری عصبی را تسهیل کند و بازیابی حرکتی را بهبود بخشد.

هدف این گزارش بررسی مساله طبقه بندی تصور حرکتی با استفاده از سیگنال های EEG است. به طور خاص، بر روی مجموعه داده ای متشکل از چهار کلاس تصویری حرکتی تمرکز می شود: حرکات شست، حرکات بازو، حرکات پا و عدم وجود هرگونه حرکت. برای استخراج ویژگی های متمایز از سیگنال های EEG،

از الگوریتم الگوهای فضایی مشترک (CSP) استفاده شده است. برای طبقه‌بندی، از تحلیل تشخیصی خطی (LDA) برای شناسایی دقیق کلاس تصور حرکتی از ویژگی‌های استخراج شده استفاده می‌شود.

بخش‌های بعدی این گزارش به شرح مجموعه داده، کاربرد الگوریتم CSP برای استخراج ویژگی، استفاده از LDA برای طبقه‌بندی، و کاربردهای بالقوه طبقه‌بندی تصاویر حرکتی در BCIs و توانبخشی عصبی می‌پردازد. با پرداختن به این مشکل و توسعه یک روش موثر، هدف نهایی کمک به پیشرفت تکنیک‌های طبقه‌بندی تصاویر حرکتی و اجرای عملی آنها در سناریوهای دنیای واقعی است.

۳-۱- مجموعه داده

مجموعه داده‌های مورد استفاده در این مطالعه به طور خاص به منظور اجرای دومین مسابقه ملی رابط مغز و کامپیوتر (BCI) جمع‌آوری شد. این شامل سیگنال‌های EEG است که از 15 فرد راست دست سالم، شامل 5 زن و 10 مرد، با میانگین سنی 31 سال ثبت شده است. سیگنال‌های EEG با استفاده از یک سیستم ضبط سیگنال 64 کانالی با فرکانس نمونه برداری 2400 هرتز گرفته شد.

در طول فرآیند ثبت داده‌ها، یک فیلتر حذف برق شهری برای به حداقل رساندن تداخل خارجی فعال بود. زمین سیستم دریافت سیگنال به پیشانی متصل بود و یکی از کانال‌ها با اتصال آن به گوش راست به عنوان کانال مرجع عمل می‌کرد. با این حال، داده‌های کانال مرجع متعاقباً حذف شد و منجر به ایجاد مجموعه‌ای با 63 کانال داده شد.

برای اطمینان از کیفیت سیگنال بهینه، یک فیلتر پایین‌گذر با فرکانس کات آف 50 هرتز برای تمام سیگنال‌های الکترود اعمال شد. این مرحله فیلتر کردن به حذف نویز با فرکانس بالا و مصنوعات کمک کرد که به طور

بالقوه می‌توانند بر تحلیل سیگنال‌های EEG مرتبط با تصور حرکتی تأثیر بگذارند.

پروتکل آزمایشی برای مجموعه داده شامل هر یک از شرکت‌کنندگان بود که روی یک صندلی راحت می‌نشستند در حالی که رو به صفحه‌ای در فاصله نیم متری قرار می‌گرفتند. به افراد دستور داده شد تا حرکات خاصی را به عنوان بخشی از وظایف تصویرسازی حرکتی انجام دهند. اعضای بدن هدف برای حرکات تصور شده شست، پای راست و بازوی راست بودند.

مجموعه داده به عنوان یک آرایه سلولی 1×4 قالب بندی شده است، که هر سلول حاوی سیگنال‌های EEG یک فرد است که به مدت 3 ثانیه انجام شده است، هر خانه مربوط به یک کلاس حرکتی خاص است. خانه اول نشان دهنده سیگنال‌های EEG ثبت شده در حین حرکات بازو (کلاس 1)، خانه دوم سیگنال‌های ثبت شده در حین حرکات انگشت شست (کلاس 2)، خانه سوم نشان دهنده سیگنال‌های ثبت شده در طول حرکات پا (کلاس 3) و خانه چهارم نشان دهنده سیگنال‌ها ثبت شده در حالت بدون حرکت هستند. (کلاس 4).

در دسترس بودن این مجموعه داده با ضبط‌های جامع سیگنال‌های EEG مرتبط با تصور حرکتی، ما را قادر می‌سازد تا روش‌های متعدد برای استخراج و طبقه‌بندی ویژگی‌ها، مانند استفاده از الگوریتم الگوهای فضایی مشترک (CSP) برای استخراج ویژگی و استفاده از تحلیل تشخیص خطی (LDA) برای طبقه‌بندی استفاده کرد

۱-۴- الگوی فضایی مشترک (CSP) ^۱

1-4-1- مقدمه و توضیحات

الگوی فضایی مشترک (CSP) یک تکنیک محبوب در زمینه پردازش سیگنال‌های زیست پزشکی است و به طور گسترده در کاربردهای مختلف استفاده می‌شود. این یک تکنیک فیلتر فضایی است که برای استخراج

¹ Common Spatial Pattern (CSP)

ویژگی های سیگنال های زیست پزشکی چند کانالی مانند الکتروانسفالوگرام (EEG) یا مغناطیسی مغزی (MEG) استفاده می شود. هدف CSP یافتن مجموعه ای از فیلترهای فضایی است که بتواند به طور موثر بین دو کلاس سیگنال بر اساس ماتریس های کوواریانس آنها تمایز قائل شود.

پایه ریاضی CSP بر اساس جبر خطی و روش های آماری چند متغیره است. CSP شامل تبدیل داده های EEG از حوزه زمانی به حوزه فضایی با استفاده از یک فیلتر فضایی است. هدف فیلتر فضایی یافتن مجموعه ای از وزن های فضایی است که حداکثر بین دو یا چند کلاس از داده های EEG تمایز قائل شود.

فیلتر فضایی مورد استفاده در CSP معمولاً با حل یک مسئله ارزش ویژه تعمیم یافته محاسبه می شود. مسئله generalized eigenvalue شامل یافتن بردارهای ویژه یک ماتریس است که حداکثر بین دو دسته داده EEG تمایز قائل می شود. فیلتر فضایی حاصل مجموعه ای از وزن ها است که می تواند به داده های EEG اعمال شود تا مجموعه جدیدی از داده های فیلتر شده فضایی به دست آید که بر تفاوت های بین دو کلاس تأکید می کند.

روش CSP را می توان با اعمال فیلتر فضایی برای داده های EEG اصلی و داده های فیلتر شده فضایی، بیشتر اصلاح کرد. این منجر به مجموعه ای از فیلترهای فضایی می شود که می توانند برای استخراج الگوهای فضایی فعالیت مغز که با حالات یا وظایف شناختی خاص مرتبط هستند، استفاده شوند.

2-4-1- پیاده سازی CSP

در زبان برنامه نویسی متلب میتوان به صورت زیر CSP را پیاده سازی کرد:

```

% Set the number of spatial filters (CSP components)
numComponents = 32;

% Separate the dataset into two classes
class1Data = dataset(labels == 1, :); % Replace '1' with class 1 label
class2Data = dataset(labels == 2, :); % Replace '2' with class 2 label

% Calculate the covariance matrices for each class
covMatrix1 = cov(class1Data);
covMatrix2 = cov(class2Data);

% Perform the eigenvalue decomposition
[eigVectors, eigValues] = eig(covMatrix1, covMatrix1 + covMatrix2);

% Sort the eigenvalues in descending order
[eigValues, sortIndex] = sort(diag(eigValues), 'descend');
eigVectors = eigVectors(:, sortIndex);

% Select the CSP filters
cspFilters = eigVectors(:, 1:numComponents);

```

شکل (۱-۱) پیاده سازی CSP در متلب

در زبان برنامه نویسی پایتون ، کتاب خانه های آماده ای برای این کار نوشته شده است که میتوان به صورت

زیر از آن استفاده کرد. (کتاب خانه Mullen NeuroImaging in Python)

```

from mne.decoding import CSP
# Define the CSP object
csp = CSP(n_components=numComponents)
# Concatenate the class data
concatenatedData = np.concatenate((class1Data, class2Data))
# Create labels for the concatenated data
concatenatedLabels = np.concatenate((np.ones(class1Data.shape[0]), np.ones(class2Data.shape[0]) * 2))
# Fit CSP on the concatenated data
csp.fit(concatenatedData, concatenatedLabels)

```

شکل (۱-۲) پیاده سازی CSP با استفاده از کتاب خانه های آماده

اما اگر اجازه استفاده از کتاب خانه های آماده هم نبود به صورت زیر همانند متلب میتوان CSP را پیاده

سازی کرد.

```

# Set the number of spatial filters (CSP components)
numComponents = 4

# Separate the dataset into two classes
class1Data = dataset[labels == 1, :] # Replace '1' with class 1 label
class2Data = dataset[labels == 2, :] # Replace '2' with class 2 label

# Calculate the covariance matrices for each class
covMatrix1 = np.cov(class1Data.T)
covMatrix2 = np.cov(class2Data.T)

# Perform the eigenvalue decomposition
eigValues, eigVectors = np.linalg.eig(np.dot(np.linalg.inv(covMatrix1 + covMatrix2), covMatrix1))

# Sort the eigenvalues in descending order
sortIndex = np.argsort(eigValues)[::-1]
eigValues = eigValues[sortIndex]
eigVectors = eigVectors[:, sortIndex]

# Select the CSP filters
cspFilters = eigVectors[:, :numComponents]

```

شکل (۳-۱) پیاده سازی الگوریتم CSP در پایتون بدون استفاده از الگوریتم های آماده

۱-۵- تحلیل تشخیص خطی^۱

1-5-1- مقدمه و توضیحات

تجزیه و تحلیل تشخیص خطی به عنوان ابزاری برای طبقه بندی، کاهش ابعاد و ... استفاده میشود و علیرغم سادگی، LDA اغلب نتایج طبقه بندی قوی، مناسب و قابل تفسیر تولید می کند. هنگام بررسی مساله طبقه بندی دنیای واقعی، LDA اغلب قبل از استفاده از سایر روش های پیچیده تر و انعطاف پذیر، روش معیارسنجی است.

تجزیه و تحلیل تشخیص خطی (LDA) یک الگوریتم طبقه بندی است که هدف آن یافتن ترکیبی خطی از ویژگی ها است که حداکثر بین کلاس های مختلف تمایز قائل شود. به ویژه در هنگام برخورد با داده های با ابعاد

¹ Linear Discriminant Analysis

بالا، مانند سیگنال های EEG در وظایف طبقه بندی تصاویر حرکتی، موثر است.

هدف اصلی LDA این است که داده های ورودی را در فضایی با ابعاد پایین تر نمایش دهد و در عین حال تفکیک پذیری بین کلاس ها را به حداکثر برساند. این امر با یافتن مجموعه ای از بردارهای متمایز، معروف به تفکیک کننده های خطی، به دست می آید که نسبت پراکندگی بین طبقاتی به پراکندگی درون کلاسی را به حداکثر می رساند.

الگوریتم LDA با محاسبه ماتریس های پراکندگی داده های ورودی آغاز می شود. ماتریس پراکندگی درون کلاسی تغییرات درون هر کلاس را نشان می دهد، در حالی که ماتریس پراکندگی بین کلاس ها تفاوت های بین کلاس ها را نشان می دهد. این ماتریس های پراکندگی اطلاعات مهمی را در مورد توزیع داده ها ارائه می دهند و فرآیند طرح ریزی را هدایت می کنند.

در مرحله بعد، LDA به دنبال یافتن یک ماتریس تبدیل است که در عین حفظ اطلاعات متمایز، داده های با ابعاد بالا را در فضایی با ابعاد پایین تر نگاشت می کند. این تبدیل با حل مسئله مقدار ویژه تعمیم یافته¹ به دست می آید، جایی که مقادیر ویژه نشان دهنده اهمیت هر بردار متمایز است.

بردارهای متمایز حاصل را می توان برای نمایش نقاط داده جدید و نادیده به فضای با ابعاد پایین تر استفاده کرد. سپس طبقه بندی با اختصاص برچسب کلاس بر اساس نزدیکی نقطه پیش بینی شده به مرکز کلاس یا با استفاده از قوانین تصمیم گیری اضافی انجام می شود.

یکی از مزایای LDA توانایی آن برای اندازه نمونه کوچک است که معمولاً در طبقه بندی تصور حرکتی مبتنی بر EEG به دلیل تعداد محدود آزمایش در هر کلاس ایجاد می شود. LDA از ماتریس های پراکندگی برای تخمین ماتریس های کوواریانس داده ها، حتی با اندازه های نمونه کوچک، استفاده می کند و از این اطلاعات برای

¹ generalized eigenvalue

ایجاد تمایز طبقاتی قابل اعتماد استفاده می کند.

1-5-2- پیاده سازی LDA

در متلب برای پیاده سازی LDA از `fitcdiscr` و در پایتون برای پیاده سازی از `sklearn` بخش `discriminant_analysis` میتوان `lda` را انتخاب کرده و از آن استفاده کرد.

۱-۶- متود بهینه انتخابی

رویکرد طبقه بندی پیشنهادی شامل یک روش گام به گام است که هدف آن بهینه سازی عملکرد تمایز بین کلاس های تصویر حرکتی است. به طور خاص، طبقه بندی کلاس 4 را که نشان دهنده حالت استراحت است، در مقابل کلاس های تصویر حرکتی باقی مانده (کلاس 1، کلاس 2، و کلاس 3) اولویت بندی می شود. متعاقباً، بر تمایز کلاس 3 از ترکیبی از کلاس 1 و کلاس 2 تمرکز می شود و در نهایت، کلاس 1 و کلاس 2 به طور جداگانه با استفاده از تحلیل تشخیصی خطی (LDA) طبقه بندی میشود.

منطق پشت این رویکرد ناشی از مشاهده این است که حالت استراحت (کلاس 4) یک نقطه مرجع متمایز را ارائه می دهد که نشان دهنده حالت بدون حرکت است. با اولویت بندی طبقه بندی کلاس 4 در برابر سایر کلاس ها، یک خط پایه برای تمایز بین تصر حرکتی و عدم حرکت ایجاد می شود. این مرحله طبقه بندی اولیه به کاهش تأثیر بالقوه فعالیت مغز مربوط به استراحت در طبقه بندی های بعدی کمک می کند.

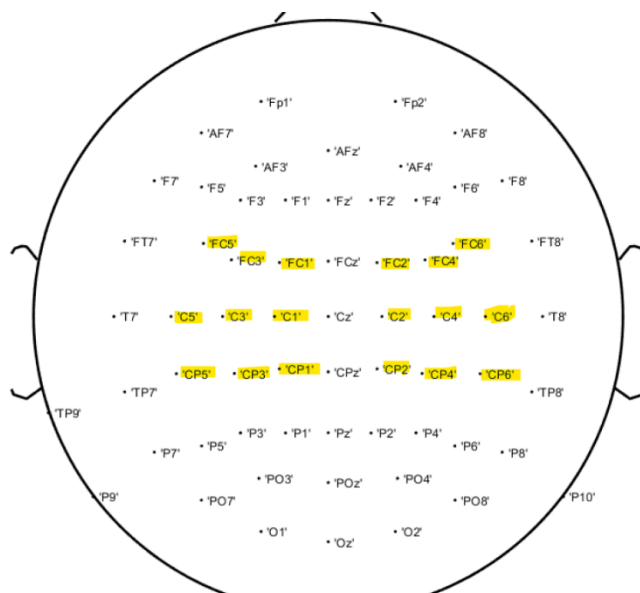
پس از طبقه بندی کلاس 4، کلاس 3 که مربوط به حرکات پا است، در مقابل مجموعه ترکیبی کلاس 1 (حرکات شست) و کلاس 2 (حرکات بازو) طبقه بندی می شود. با تمایز کلاس 3 از دو کلاس دیگر، به طور خاص تمایز تصویر حرکتی مربوط به اندام تحتانی هدف قرار داده میشود. این رویکرد متوالی این امکان را می

دهد که به تدریج فرآیند طبقه بندی با تمرکز بر انواع حرکت خاص اصلاح شود. (برای هر 2 طبقه بند قبلی از LDA استفاده میشود)

در نهایت نیز ، از تحلیل تشخیصی خطی (LDA) برای طبقه بندی کلاس 1 و کلاس 2 به طور جداگانه استفاده می شود. LDA به دلیل توانایی آن در به حداکثر رساندن جدایی بین طبقات با شناسایی ترکیبات خطی ویژگی ها مورد استفاده قرار می گیرد. با استفاده از LDA در کلاس 1 و کلاس 2، هدف ما دستیابی به تمایز بهینه بین این دو وظیفه تصور حرکتی متمایز، یعنی حرکات شست و حرکات بازو است.

۱-۷- لود دیتاست و پیش پردازش

اولین مورد انتخاب کانال های مناسب از دیتاست میباشد چرا که استفاده از هر 64 کانال دقت خوبی ندارد و توجه مناسبی هم ندارد و ترجیحا باید کانال های نزدیک به motor cortex برداشته شود که در این پروژه از کانال های زیر استفاده میشود.



شکل (۴-۱) کانال های استفاده شده برای حل مساله طبقه بندی تصور حرکتی

یکی دیگر از پیش پردازش های داده های که باید انجام شود بررسی بالانس بودن یا نبودن داده ها میباشد ک مشاهده شد آزمایش ها برای کلاس ها متفاوت میباشد و یکی دیگر از کارهایی که قبل از پردازش اصلی انجام میشود ، بالانس کردن داده هاست که بر اساس shape های موجود مینیمم تعداد trial را پیدا کرده و بر اساس آن داده ها را از نظر سمپل بالانس میکنیم.

مورد دیگر در پیش پردازش بررسی وجود مقادیر Nan یا inf میباشد که با رویکرد جایگذاری با صفر یا کلا حذف کردن آن داده ها این مورد هندل میشود.

در پایتون به کمک کد زیر میتوان دیتا ها را هم چسباند تمامی فیچر ها بر اساس کلاس های مختلف جدا کرد.

```
# Define the channels of interest
channels_of_interest = [11, 40, 12, 41, 13, 42, 14, 44, 16, 45, 17, 46, 18, 47, 20, 49, 21, 50, 22, 51, 23]

# Initialize empty lists for X and Y data
X = []
Y = []

min_num_trials = float('inf') # Initialize with a large value
```

شکل (۵-۱) انتخاب کانال های مورد علاقه برای بررسی و ایجاد متغیر فیچر ها و لیبل ها

```
# Load the .mat file for each person
data = loadmat(f'dataset/subj_{i}.mat')['data'][0]
min_num_trials = float('inf') # Initialize with a large value

# Extract the classes
class_1 = data[0][:, channels_of_interest, :]
class_2 = data[1][:, channels_of_interest, :]
class_3 = data[2][:, channels_of_interest, :]
class_4 = data[3][:, channels_of_interest, :]

# Determine the minimum number of trials among all classes
min_num_trials = min(min_num_trials, class_1.shape[2], class_2.shape[2], class_3.shape[2], class_4.shape[2])

# Append the data to X and Y
X.append(class_1[:, :, :min_num_trials])
X.append(class_2[:, :, :min_num_trials])
X.append(class_3[:, :, :min_num_trials])
X.append(class_4[:, :, :min_num_trials])

# Create labels for the classes
num_samples = min_num_trials * 4
labels = np.zeros((num_samples,))
labels[min_num_trials:min_num_trials * 2] = 1
labels[min_num_trials * 2:min_num_trials * 3] = 2
labels[min_num_trials * 3:] = 3

# Append the labels to Y
Y.append(labels)
```

شکل (۶-۱) کد پایتون خواندن دیتا ها

کد فوق در ابتدا تمامی داده های کلاس های 1 تا 4 نفر i ام را میخواند و بر اساس حداقل تعداد trial ها

داده های هر کلاس را بالانس کرده و از هر داده فقط سمپل های کانال های مشخص شده را بر میدارد.

و در نهایت X و Y ساخته میشود.

```
X = np.concatenate(X, axis=2)
Y = np.concatenate(Y, axis=0)
X = X.transpose((2, 1, 0))
```

شکل (۷-۱) چسباندن فیچر ها در یک ماتریس X و transpose آن

در متلب نیز میتوان این دستورات را به صورت زیر پیاده سازی کرد.

```
% Define the channels of interest
channels_of_interest = [11, 40, 12, 41, 13, 42, 14, 44, 16, 45, 17, 46, 18, 47, 20, 49, 21, 50, 22, 51, 23];

% Loop over each person
for i = 1:15
    %% load data

    % Initialize empty lists for X and Y data
    X = {};
    Y = {};

    % Load the .mat file for each person
    data = load(sprintf('dataset/subj_%d.mat', i));
    data = data.data{1};
    min_num_trials = inf; % Initialize with a large value

    % Extract the classes
    class_1 = data{1}(:, channels_of_interest, :);
    class_2 = data{2}(:, channels_of_interest, :);
    class_3 = data{3}(:, channels_of_interest, :);
    class_4 = data{4}(:, channels_of_interest, :);

    % Determine the minimum number of trials among all classes
    min_num_trials = min([min_num_trials, size(class_1, 3), size(class_2, 3), size(class_3, 3), size(class_4, 3)]);

    % Append the data to X and Y
    X{1} = class_1(:, :, 1:min_num_trials);
    X{2} = class_2(:, :, 1:min_num_trials);
    X{3} = class_3(:, :, 1:min_num_trials);
    X{4} = class_4(:, :, 1:min_num_trials);

    % Create labels for the classes
    num_samples = min_num_trials * 4;
    labels = zeros(num_samples, 1);
    labels(min_num_trials+1:min_num_trials*2) = 1;
    labels(min_num_trials*2+1:min_num_trials*3) = 2;
    labels(min_num_trials*3+1:end) = 3;

    % Append the labels to Y
    Y{1} = labels;
```

شکل (۸-۱) کد متلب بخش لود کردن و پیش پردازش داده ها

۸-۱- الگوریتم leave one out برای جداسازی تست و ترین

اعتبار سنجی متقابل جداسازی یکی^۱ (LOO) تکنیکی است که برای ارزیابی عملکرد یک مدل یادگیری ماشین استفاده می شود. این شامل تقسیم مجموعه داده ها به مجموعه های آموزشی و آزمایشی است، که در آن مجموعه آزمایشی فقط شامل یک نمونه واحد است و مجموعه آموزشی شامل تمام نمونه های دیگر است. این فرآیند برای هر نمونه در مجموعه داده تکرار می شود، به طوری که هر نمونه یک بار به عنوان نمونه آزمایشی عمل می کند در حالی که بقیه نمونه ها به عنوان مجموعه آموزشی عمل می کنند. در اینجا توضیحی از کد ارائه شده برای پیاده سازی الگوریتم LOO آمده است:

- لیست ها برای ذخیره مجموعه های آموزشی و آزمایشی تعریف میشود : X_{train} , X_{test} , Y_{train} و Y_{test} .

- سپس کد وارد حلقه ای می شود که min_num_trials بار تکرار می کند. این عدد نشان دهنده حداقل تعداد آزمایش یا نمونه در هر کلاس است. و بدین ترتیب می خواهیم از هر آزمایش در هر کلاس یکی را بر داریم برای آزمایش و باقی برای آموزش بماند.

- در هر تکرار حلقه، مجموعه آموزشی با الحاق نمونه های همه کلاس ها به جز کلاس فعلی i ساخته می شود. این کار با استفاده از `list comprehension` و تابع `np.concatenate` انجام می شود. نتیجه در X_{train} ذخیره می شود.

- مجموعه تست تنها با انتخاب نمونه هایی از کلاس فعلی i و ذخیره آن در X_{test} ساخته می شود.
- به طور مشابه، برچسب های مجموعه آموزشی (Y_{train}) با به هم پیوستن برچسب ها از همه کلاس ها به جز کلاس فعلی i ساخته می شوند و برچسب های مجموعه آزمایشی (Y_{test}) تنها با

¹ leave-one-out (LOO)

انتخاب برچسب‌ها از کلاس فعلی i به دست می‌آیند.

- پس از تکمیل حلقه، لیست های X_{train} ، X_{test} ، Y_{train} و Y_{test} با استفاده از

`np.concatenate` به آرایه های NumPy تبدیل می شوند و به همان متغیرها اختصاص می یابند.

کد پایتون الگوریتم توضیح داده شده در ادامه آمده است :

```
# Split the data using leave-one-out method
for i in range(min_num_trials):
    X_train.append(np.concatenate([X[Y != j, :, :] for j in range(4)], axis=0))
    X_test.append(X[Y == i, :, :])
    Y_train.append(np.concatenate([Y[Y != j] for j in range(4)], axis=0))
    Y_test.append(Y[Y == i])

# Convert lists to arrays
X_train = np.concatenate(X_train, axis=0)
X_test = np.concatenate(X_test, axis=0)
Y_train = np.concatenate(Y_train, axis=0)
Y_test = np.concatenate(Y_test, axis=0)
```

شکل (۹-۱) کد پایتون پیاده سازی الگوریتم LOO

در ادامه نیز دیتاست های جدایی از هر کلاس ساخته میشود. و همچنین بررسی میشود در صورت وجود داده

Nan یا inf آنها حذف شده (در X های کلاس 4 با توجه به اینکه حذف باعث صفر شدن shape آن میشود

مقادیر Nan to num میشود.)

```
# Create separate datasets for each class
X1_train = X_train[Y_train == 0]
X2_train = X_train[Y_train == 1]
X3_train = X_train[Y_train == 2]
X4_train = X_train[Y_train == 3]

X1_train = X1_train[np.logical_not(np.isnan(np.sum(X1_train, axis=(1, 2))) | np.isinf(np.sum(X1_train, axis=(1, 2))))]
X2_train = X2_train[np.logical_not(np.isnan(np.sum(X2_train, axis=(1, 2))) | np.isinf(np.sum(X2_train, axis=(1, 2))))]
X3_train = X3_train[np.logical_not(np.isnan(np.sum(X3_train, axis=(1, 2))) | np.isinf(np.sum(X3_train, axis=(1, 2))))]
X4_train = np.nan_to_num(X4_train)

X1_test = X_test[Y_test == 0]
X2_test = X_test[Y_test == 1]
X3_test = X_test[Y_test == 2]
X4_test = X_test[Y_test == 3]

X1_test = X1_test[np.logical_not(np.isnan(np.sum(X1_test, axis=(1, 2))) | np.isinf(np.sum(X1_test, axis=(1, 2))))]
X2_test = X2_test[np.logical_not(np.isnan(np.sum(X2_test, axis=(1, 2))) | np.isinf(np.sum(X2_test, axis=(1, 2))))]
X3_test = X3_test[np.logical_not(np.isnan(np.sum(X3_test, axis=(1, 2))) | np.isinf(np.sum(X3_test, axis=(1, 2))))]
X4_test = np.nan_to_num(X4_test)
```

شکل (۱۰-۱) کد پایتون ایجاد دیتاست داده های هر کلاس و تمیز کردن داده ها

در متلب میتوان تمام کار های فوق را به صورت زیر انجام داد:

```
% Initialize cell arrays for train and test sets
X_train = {};
X_test = {};
Y_train = {};
Y_test = {};

% Split the data using leave-one-out method
for i = 1:min_num_trials
    X_train{i} = cat(1, X(Y ~= i-1, :, :));
    X_test{i} = X(Y == i-1, :, :);
    Y_train{i} = cat(1, Y(Y ~= i-1));
    Y_test{i} = Y(Y == i-1);
end

% Convert cell arrays to arrays
X_train = cat(1, X_train{:});
X_test = cat(1, X_test{:});
Y_train = cat(1, Y_train{:});
Y_test = cat(1, Y_test{:});

% Create separate datasets for each class
X1_train = X_train(Y_train == 0, :, :);
X2_train = X_train(Y_train == 1, :, :);
X3_train = X_train(Y_train == 2, :, :);
X4_train = X_train(Y_train == 3, :, :);

X1_train = X1_train(sum(isnan(sum(X1_train, 3)), 2) == 0 & sum(isinf(sum(X1_train, 3)), 2) == 0, :, :);
X2_train = X2_train(sum(isnan(sum(X2_train, 3)), 2) == 0 & sum(isinf(sum(X2_train, 3)), 2) == 0, :, :);
X3_train = X3_train(sum(isnan(sum(X3_train, 3)), 2) == 0 & sum(isinf(sum(X3_train, 3)), 2) == 0, :, :);
X4_train(isnan(X4_train)) = 0;

X1_test = X_test(Y_test == 0, :, :);
X2_test = X_test(Y_test == 1, :, :);
X3_test = X_test(Y_test == 2, :, :);
X4_test = X_test(Y_test == 3, :, :);

X1_test = X1_test(sum(isnan(sum(X1_test, 3)), 2) == 0 & sum(isinf(sum(X1_test, 3)), 2) == 0, :, :);
X2_test = X2_test(sum(isnan(sum(X2_test, 3)), 2) == 0 & sum(isinf(sum(X2_test, 3)), 2) == 0, :, :);
X3_test = X3_test(sum(isnan(sum(X3_test, 3)), 2) == 0 & sum(isinf(sum(X3_test, 3)), 2) == 0, :, :);
X4_test(isnan(X4_test)) = 0;
```

شکل (۱۱-۱) کد متلب پیاده سازی الگوریتم LOO

۹-۱- رویکرد های متفاوت آموزش طبقه بند ها

1-9-1- استفاده تمام داده ها و طراحی فقط 3 مدل طبقه بند و جداسازی LOO

این بخش جزوی از پروژه نمیشد ولی برای آنکه استفاده از LOO ان هم روی یک مجموعه داده یک نفر

که در یک لحظه record شده است و سپس ارزیابی آن روی داده های یک نفر میتواند دقت غلط انداز و بالایی

باشد در اینجا ابتدا قبل از آموزش 15 مدل مختلف برای هر 15 نفر با دیتای تمام 15 نفر یک مدل (فقط 3 طبقه بند) آموزش داده میشود و دقت نهایی روی تمام داده های تست 15 نفر برای 4 کلاس ارزیابی میشود.

1-9-1-1- طراحی و آموزش طبقه بند ها

همانطور که در روش پیشنهادی گفته شد قرار بر این است که دیتا های کلاس 4 ام ابتدا با داده های سایر کلاس ها مقایسه شود و برای این کار ابتدا از روی X1 و X2 و .. های آموزش و آزمایش و با کانتکت کردن داده ها یک X برای طبقه بند اول درست کرده و همچنین معادل آن با مقدار دهی لیبل صفر به کلاس 4 ام و کلاس 1 به باقی کلاس های باقی مانده آرایه ای از لیبل ها هم جهت آموزش CSP و LDA تهیه میشود.

```
# Class 4 vs. Other Classes Classifier
X_classifier1_train = np.concatenate((X1_train, X2_train, X3_train, X4_train), axis=0)
y_classifier1_train = np.concatenate((np.ones(X1_train.shape[0]), np.ones(X2_train.shape[0]),
                                     np.ones(X3_train.shape[0]), np.zeros(X4_train.shape[0])))

X_classifier1_test = np.concatenate((X1_test, X2_test, X3_test, X4_test), axis=0)
y_classifier1_test = np.concatenate((np.ones(X1_test.shape[0]), np.ones(X2_test.shape[0]),
                                     np.ones(X3_test.shape[0]), np.zeros(X4_test.shape[0])))
```

شکل (۱۲-۱) کد پایتون آماده سازی X و Y برای داده های آموزشی و آزمایشی طبقه بند اول

در ادامه نیز الگوریتم CSP بر روی آن اعمال میشود تا به فیچر هایی رسیده که بتوان به کمک آن LDA را آموزش داد.

تعداد فیچر های استخراجی (n component) را بر 32 قرار داده میشود چرا که مشاهده شد که مقادیر بالاتر شبکه overfit میشود و مقادیر پایین دقت روی تست کم میشود که این مقدار به کمک الگوریتم جستجوی شبکه ای (gird search) بدست آمد.

```
# Apply CSP to X_classifier1 train and test data
csp1 = CSP(n_components=32, reg=None, log=True)
X_classifier1_train_csp = csp1.fit_transform(X_classifier1_train, y_classifier1_train)
X_classifier1_test_csp = csp1.transform(X_classifier1_test)
```

شکل (۱-۱۳) اعمال CSP رو X و Y های طبقه بند اول

Csp1 که در اینجا استفاده شد به کمک دستور زیر میتوان به وزن های W_{CSP} رسید.

csp1.filters_

در نهایت نیز پس از مرحله feature extraction باید classifier مناسبی برای هر کدام پیدا شود که از

الگوریتم LDA استفاده میشود.

به کمک دستور زیر به پیدا کردن بهترین جداسازی پرداخته میشود.

```
# Train LDA on X_classifier1 train data
lda1 = LinearDiscriminantAnalysis()
lda1.fit(X_classifier1_train_csp, y_classifier1_train)
```

شکل (۱-۱۴) اعمال LDA بر طبقه بند اول

```
# Predict on train and test data
y_classifier1_train_pred = lda1.predict(X_classifier1_train_csp)
y_classifier1_test_pred = lda1.predict(X_classifier1_test_csp)
```

شکل (۱-۱۵) گرفتن پیشبینی مدل روی داده های تست و ترین

```
# Calculate accuracy for Class 4 vs. Other Classes Classifier
accuracy_train1 = accuracy_score(y_classifier1_train, y_classifier1_train_pred)
accuracy_test1 = accuracy_score(y_classifier1_test, y_classifier1_test_pred)

# Calculate confusion matrices for Class 4 vs. Other Classes Classifier
confusion_matrix_train1 = confusion_matrix(y_classifier1_train, y_classifier1_train_pred)
confusion_matrix_test1 = confusion_matrix(y_classifier1_test, y_classifier1_test_pred)
```

شکل (۱-۱۶) محاسبه دقت و ماتریس آشفتگی ، طبقه بند اول

تمامی دستورات فوق را نیز میتوان در متلب به کمک الگوریتم هایی که قبل تر بیان شد پیاده سازی کرد کما

آنیکه در تمرین سوم درس نیز مشابه این کار انجام شده است.

برای سایر طبقه بند ها نیز مشابه کاری که انجام شد ، انجام میشود.

```
# Class 3 vs. (Class 1 and 2) Classifier
X_classifier2_train = np.concatenate((X3_train, X1_train, X2_train), axis=0)
y_classifier2_train = np.concatenate((np.zeros(X3_train.shape[0]), np.ones(X1_train.shape[0]),
                                     np.ones(X2_train.shape[0])))

X_classifier2_test = np.concatenate((X3_test, X1_test, X2_test), axis=0)
y_classifier2_test = np.concatenate((np.zeros(X3_test.shape[0]), np.ones(X1_test.shape[0]),
                                    np.ones(X2_test.shape[0])))

# Apply CSP to X_classifier2 train and test data
csp2 = CSP(n_components=32, reg=None, log=True)
X_classifier2_train_csp = csp2.fit_transform(X_classifier2_train, y_classifier2_train)
X_classifier2_test_csp = csp2.transform(X_classifier2_test)

# Train LDA on X_classifier2 train data
lda2 = LinearDiscriminantAnalysis()
lda2.fit(X_classifier2_train_csp, y_classifier2_train)

# Predict on train and test data
y_classifier2_train_pred = lda2.predict(X_classifier2_train_csp)
y_classifier2_test_pred = lda2.predict(X_classifier2_test_csp)

# Calculate accuracy for Class 3 vs. (Class 1 and 2) Classifier
accuracy_train2 = accuracy_score(y_classifier2_train, y_classifier2_train_pred)
accuracy_test2 = accuracy_score(y_classifier2_test, y_classifier2_test_pred)

# Calculate confusion matrices for Class 3 vs. (Class 1 and 2) Classifier
confusion_matrix_train2 = confusion_matrix(y_classifier2_train, y_classifier2_train_pred)
confusion_matrix_test2 = confusion_matrix(y_classifier2_test, y_classifier2_test_pred)
```

شکل (۱۷-۱) طراحی طبقه بند دوم (مقایسه کلاس سوم در مقابل کلاس اول و دوم)

```
# Class 1 vs. Class 2 Classifier
X_classifier3_train = np.concatenate((X1_train, X2_train), axis=0)
y_classifier3_train = np.concatenate((np.zeros(X1_train.shape[0]), np.ones(X2_train.shape[0])))

X_classifier3_test = np.concatenate((X1_test, X2_test), axis=0)
y_classifier3_test = np.concatenate((np.zeros(X1_test.shape[0]), np.ones(X2_test.shape[0])))

# Apply CSP to X_classifier3 train and test data
csp3 = CSP(n_components=32, reg=None, log=True)
X_classifier3_train_csp = csp3.fit_transform(X_classifier3_train, y_classifier3_train)
X_classifier3_test_csp = csp3.transform(X_classifier3_test)

# Train LDA on X_classifier3 train data
lda3 = LinearDiscriminantAnalysis()
lda3.fit(X_classifier3_train_csp, y_classifier3_train)

# Predict on train and test data
y_classifier3_train_pred = lda3.predict(X_classifier3_train_csp)
y_classifier3_test_pred = lda3.predict(X_classifier3_test_csp)

# Calculate accuracy for Class 1 vs. Class 2 Classifier
accuracy_train3 = accuracy_score(y_classifier3_train, y_classifier3_train_pred)
accuracy_test3 = accuracy_score(y_classifier3_test, y_classifier3_test_pred)

# Calculate confusion matrices for Class 1 vs. Class 2 Classifier
confusion_matrix_train3 = confusion_matrix(y_classifier3_train, y_classifier3_train_pred)
confusion_matrix_test3 = confusion_matrix(y_classifier3_test, y_classifier3_test_pred)
```

شکل (۱۸-۱) طراحی طبقه بند سوم (مقایسه کلاس 1 و 2)

در متلب نیز میتوان تمام این کارها را به کمک کد زیر انجام داد:

```
function [X_csp] = CSP(X, y, num_components)
% Compute the covariance matrices for each class
cov_matrices = cell(1, max(y));
for i = 1:max(y)
    class_data = X(y == i, :);
    cov_matrices{i} = cov(class_data);
end

% Compute the combined covariance matrix
C = sum(cat(3, cov_matrices{:}), 3);

% Perform eigenvalue decomposition
[V, D] = eig(C);

% Sort eigenvalues and eigenvectors in descending order
[~, indices] = sort(diag(D), 'descend');
V_sorted = V(:, indices);

% Select the top and bottom eigenvectors
W = [V_sorted(:, 1:num_components), V_sorted(:, end-num_components+1:end)];

% Apply CSP transformation
X_csp = X * W;
end
```

شکل (۱۹-۱) تعریف فانکشن CSP در متلب

```
% Class 4 vs. Other Classes Classifier
X_classifier1_train = [X1_train; X2_train; X3_train; X4_train];
y_classifier1_train = [ones(size(X1_train, 1), 1); ones(size(X2_train, 1), 1);
    ones(size(X3_train, 1), 1); zeros(size(X4_train, 1), 1)];

X_classifier1_test = [X1_test; X2_test; X3_test; X4_test];
y_classifier1_test = [ones(size(X1_test, 1), 1); ones(size(X2_test, 1), 1);
    ones(size(X3_test, 1), 1); zeros(size(X4_test, 1), 1)];

% Apply CSP to X_classifier1 train and test data
num_csp_components = 32;
X_classifier1_train_csp = CSP(X_classifier1_train, y_classifier1_train, num_csp_components);
X_classifier1_test_csp = CSP(X_classifier1_test, y_classifier1_test, num_csp_components);

% Train LDA on X_classifier1 train data
lda1 = fitcdiscr(X_classifier1_train_csp, y_classifier1_train);

% Predict on train and test data
y_classifier1_train_pred = predict(lda1, X_classifier1_train_csp);
y_classifier1_test_pred = predict(lda1, X_classifier1_test_csp);
```

شکل (۲۰-۱) کد متلب طراحی و خروجی گرفتن از طبقه بند سوم

1-9-1-2- نتایج و دقت هر کدام از طبقه بند ها

نتایج و دقت هر کدام از طبقه بند ها در زیر آمده است:

```
Confusion Matrix for Class 4 vs. Other Classes Classifier (Train):  
[[ 4176  8064]  
 [ 1728 34992]]  
Accuracy for Class 4 vs. Other Classes Classifier (Train): 0.8
```

```
Confusion Matrix for Class 4 vs. Other Classes Classifier (Test):  
[[  87 168]  
 [  36 729]]  
Accuracy for Class 4 vs. Other Classes Classifier (Test): 0.8
```

شکل (۱-۲۱) ماتریس آشفته‌گی و دقت در طبقه بند اول

```
Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Train):  
[[ 4464  7776]  
 [ 2448 22032]]  
Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Train): 0.7215686274509804
```

```
Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Test):  
[[  93 162]  
 [  51 459]]  
Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Test): 0.7215686274509804
```

شکل (۱-۲۲) ماتریس آشفته‌گی و دقت در طبقه بند اول

```
Confusion Matrix for Class 1 vs. Class 2 Classifier (Train):  
[[8544 3696]  
 [4224 8016]]  
Accuracy for Class 1 vs. Class 2 Classifier (Train): 0.6764705882352942
```

```
Confusion Matrix for Class 1 vs. Class 2 Classifier (Test):  
[[178  77]  
 [ 88 167]]  
Accuracy for Class 1 vs. Class 2 Classifier (Test): 0.6764705882352942
```

شکل (۱-۲۳) ماتریس آشفته‌گی و دقت در طبقه بند اول

مشاهده میشود که دقت طبقه بند اول 80 درصد و دقت طبقه بند دوم 70 درصد و دقت طبقه بند سوم 67 درصد

میباشد (نکته حایز اهمیت این است که دقت روی تست و ترین برابر است و نشان میدهد در کانال های انتخابی

و پیش پردازش های انجام شده و الگوریتم LOO که استفاده شده است اگر مدل نتوانسته یک آزمایش را

تشخیص دهد همان تک نمونه جدا شده از آن هم نمیتواند (

این مورد در بخش دوم که approach جداسازی رندوم داده ها به توجه به جایگاه آن ها در trial و الگوریتم

LOO میباشد کاملاً مشهود میشود.

1-9-1-3- ارزیابی مدل با 3 طبقه بند و گرفتن خروجی 4 کلاس

در ادامه تمام X های train و تمام X های تست را به هم چسبانده و لیبل های 0 تا 3 برای 3 کلاس برای

آن ها زده میشود.

```
# Combine all X train and test data
X_all_train = np.concatenate((X1_train, X2_train, X3_train, X4_train), axis=0)
X_all_test = np.concatenate((X1_test, X2_test, X3_test, X4_test), axis=0)

# Combine all y train and test data
y_all_train = np.concatenate((np.zeros(X1_train.shape[0]), np.ones(X2_train.shape[0]), np.ones(X3_train.shape[0])*2, np.ones(X4_train.shape[0])*3))
y_all_test = np.concatenate((np.zeros(X1_test.shape[0]), np.ones(X2_test.shape[0]), np.ones(X3_test.shape[0])*2, np.ones(X4_test.shape[0])*3))
```

شکل (۲۴-۱) چسباندن دیتای تمام کلاس ها به هم

همانطور که گفته شد الگوریتم و مدل به این صورت است که داده با طبقه بند اول مقایسه میشود و اگر

خروجی صفر شود یعنی لیبل داده کلاس 4 میشود در غیر این صورت داده وارد طبقه بند دوم شده و اگر خروجی

طبقه بند صفر باشد لیبل داده 3 میشود و در غیر این صورت وارد طبقه بند سوم شده تا لیبل کلاس 1 یا 2 را بگیرد.

برای این کار ابتدا 2 لیست تعریف میشود که خروجی های لیبل پیشبینی شده برای داده های آزمایشی و

آموزشی کلی کلاس را دربربگیرد.

و سپس الگوریتم برای داده های آزمایشی و آموزشی پیاده میشود. لازم به ذکر است که در هر مرحله برای

ورودی دادن به هر کدام از lda ها باید فیچر را به کمک وزن های csp های 1 تا 3 آموزش داده شده استخراج

کرده و آن وارد lda شود.

```

y_pred_all_train = []
y_pred_all_test = []

for x_train in X_all_train:
    x_train_csp1 = csp1.transform(np.array([x_train]))
    x_train_csp2 = csp2.transform(np.array([x_train]))
    x_train_csp3 = csp3.transform(np.array([x_train]))

    if lda1.predict(x_train_csp1) == 1:
        if lda2.predict(x_train_csp2) == 1:
            if lda3.predict(x_train_csp3) == 1:
                y_pred_all_train.append(1)
            else:
                y_pred_all_train.append(0)
        else:
            y_pred_all_train.append(2)
    else:
        y_pred_all_train.append(3)

for x_test in X_all_test:
    x_test_csp1 = csp1.transform(np.array([x_test]))
    x_test_csp2 = csp2.transform(np.array([x_test]))
    x_test_csp3 = csp3.transform(np.array([x_test]))

    if lda1.predict(x_test_csp1) == 1:
        if lda2.predict(x_test_csp2) == 1:
            if lda3.predict(x_test_csp3) == 1:
                y_pred_all_test.append(1)
            else:
                y_pred_all_test.append(0)
        else:
            y_pred_all_test.append(2)
    else:
        y_pred_all_test.append(3)

```

شکل (۲۵-۱) پیاده سازی الگوریتم تشخیص 4 کلاس

در نهایت با توجه به خروجی های 4 کلاسه میتوان ماتریس آشفتگی کلی و دقت مدل را ارزیابی کرد.

Confusion Matrix for the Cascade of Classifiers (Train):

```

[[7248 3024 1248  720]
 [3648 6864 1200  528]
 [3696 3792 4272  480]
 [3648 3264 1152 4176]]

```

Accuracy for the Cascade of Classifiers (Train): 0.46078431372549017

Confusion Matrix for the Cascade of Classifiers (Test):

```

[[151  63  26  15]
 [ 76 143  25  11]
 [ 77  79  89  10]
 [ 76  68  24  87]]

```

شکل (۱-۲۶) دقت و ماتریس آشفتگی مدل با 3 طبقه بند روی تمام داده های آموزشی و آزمایش

مشاهده میشود اگر تمام داده های 15 نفر را برای آموزش فقط یک مدل استفاده کنیم دقت کلی برابر 46 درصد میباشد که این دقت پایین به علت حدود 70 بودن دقت هر 3 طبقه بند میباشد.

1-9-1-4- مشاهده وزن های W_{LDA} و W_{CSP}

برای مشاهده وزن ها میتوان از دستور زیر استفاده کرد

```
# Retrieve the weights for CSP
csp1_weights = csp1.filters_
csp2_weights = csp1.filters_
csp3_weights = csp1.filters_

# Retrieve the weights for LDA
lda1_weights = lda1.coef_
lda2_weights = lda1.coef_
lda3_weights = lda1.coef_
```

شکل (۱-۲۷) کد پایتون مشاهده وزن های CSP و LDA

1-9-2- استفاده تمام داده ها و طراحی فقط 3 مدل طبقه بند و جداسازی رندوم

در این بخش دیگر از الگوریتم از LOO استفاده نمیشود ولی مراحل لود و پردازش همان مراحل قبل میباشد و فقط برای جداسازی کاملاً رندوم و بدون توجه به این که هر داده برای کدام trial میباشد به صورت کاملاً تصادفی 20 درصد داده ها به عنوان داده آزمایشی برداشته میشود.

در این بخش کد ها تفاوتی با قبل نمیکند و فقط به جای تقسیم اولیه همان داده های X_1 و X_2 و X_3 و X_4 کار میشود و به کمک train-test-split داده ها به صورت تصادفی در دو بخش آزمایشی و آموزشی تقسیم میشود

```
X_classifier1 = np.concatenate((X1, X2, X3, X4), axis=0)
y_classifier1 = np.concatenate((np.ones(X1.shape[0]), np.ones(X2.shape[0]), np.ones(X3.shape[0]), np.zeros(X4.shape[0])))

X_classifier1_train, X_classifier1_test, y_classifier1_train, y_classifier1_test = train_test_split(
    X_classifier1, y_classifier1, test_size=0.2, random_state=42)
```

شکل (۱-۲۸) نمونه نحوه تقسیم رندوم داده ها

در این حالت دقت ها طبقه بند ها به صورت زیر میشود.

```
Classifier 1 - Class 4 vs. Other Classes
Train Accuracy: 0.8125
Test Accuracy: 0.7647058823529411
Confusion Matrix Train :
[[ 80 127]
 [ 26 583]]
Confusion Matrix Test :
[[ 9 39]
 [ 9 147]]
```

شکل (۱-۲۹) دقت و ماتریس آشفتگی طبقه بند اول با دیتای تمام 15 نفر و تقسیم رندوم داده ها

```
Classifier 2 - Class 3 vs. (Class 1 and 2)
Train Accuracy: 0.7434640522875817
Test Accuracy: 0.5816993464052288
Confusion Matrix Train :
[[ 90 112]
 [ 45 365]]
Confusion Matrix Test :
[[ 6 47]
 [17 83]]
```

شکل (۱-۳۰) دقت و ماتریس آشفتگی طبقه بند دوم با دیتای تمام 15 نفر و تقسیم رندوم داده ها

```
Classifier 3 - Class 1 vs. Class 2
Train Accuracy: 0.7034313725490197
Test Accuracy: 0.6176470588235294
Confusion Matrix Train :
[[145 57]
 [ 64 142]]
Confusion Matrix Test :
[[35 18]
 [21 28]]
```

شکل (۱-۳۱) دقت و ماتریس آشفتگی طبقه بند سوم با دیتای تمام 15 نفر و تقسیم رندوم داده ها

مشاهده میشود که دقت طبقه بند ها روی تست کمتر میشود و در مجموع در این حالت روی داد های کلی هم دقت حدود 38 درصد دارد که نشان از نامناسب بودن آن دارد.

3-9-1- استفاده تمام داده ها و طراحی فقط 1 طبقه بند با SVM و جداسازی رندوم

در این حالت صرفا جهت مقایسه حالت 1-9-1 که به دقت نزدیک 50 بر روی تمام داده ها رسیدیم میخواهیم از یک الگوریتم دیگر (SVM) که میتوانیم مستقیم 4 تا کلاس را به آن بدهیم (استفاده کنیم و ببینیم با وجود آن که SVM طبقه بند بهتری میباشد آیا استفاده از LDA و این جرکت گام به گام و درختی درست میباشد یا خیر

برای پیاده سازی به صورت زیر اقدام میشود.

ابتدا تمام داده ها را یک جا و لیبل های 0 تا 3 آماده کرده و در ادامه به کمک train test split داده ها را تقسیم کرده و در ادامه CSP اعمال میشود.

در نهایت نیز به کمک دستور زیر از کتال خانه sklearn مدل SVM را آموزش میدهیم .

```
# Train SVM on the training set
svm = SVC()
svm.fit(X_train, y_train)
```

شکل (۱-۳۲) آموزش مدل SVM

مشاهده میشود که دقت روی داده های آموزشی حدود 50 میباشد و دقت روی داده های آزمایشی حدود

40 درصد میباشد و الگوریتم استفاده شده حتی با LDA که ضعیف تر از SVM میباشد عملکرد بهتری نسبت به 4 کلاسه جدا کردن دارد.

```
Train Accuracy: 0.49264705882352944
Test Accuracy: 0.4068627450980392
Train Confusion Matrix:
[[118  27  30  35]
 [ 59  77  41  28]
 [ 52  31  81  30]
 [ 32  21  28 126]]
Test Confusion Matrix:
[[19  9 10  7]
 [ 9 20 10 11]
 [17 11 22 11]
 [10  7  9 22]]
```

شکل (۱-۳۳) ماتریس آشفستگی و دقت در حالت استفاده از SVM 4 کلاسه

4-9-1- آموزش 15 مدل برای 15 تا دیتا با جداسازی با روش LOO

در این بخش که شاید خواسته اصلی مساله پروژه میباشد 15 تا مدل آموزش داده میشود که از نظر پیاده سازی کد آن با بخش 1 تفاوتی ندارد و تنها تفاوت این است که کل کد بخش 1 در یک loop های 15 تایی قرار میگیرد و در هر لوپ در نهایت خروجی های هر مدل پرینت میشود که به صورت txt در ادامه تمامی خروجی ها آمده است اما چیزی که مشخص میباشد به دلیل استفاده از LOO و شباهت داده های eeg خروجی تست و ترین از نظر دقت شبیه به هم میباشد و همچنین دقت طبقه بند دوم و سوم در این روش به 100 رسیده (چراکه دیتا های آدم های دیگه در آن دخیل نیست و فقط دیتای یک فرد آن هم چند آزمایش در یک لحظه بوده است اینکه به این روش بتوان به خوبی داده ها را از جدا کرد بعید به نظر نمیرسد (البته امکان اشتباه در پیاده سازی نیز وجود دارد که به همچنین دقت های بالایی رسیدیم.)

دقت طبقه بند اول نیز متفاوت و در بازه 94 تا 100 میباشد که دقت همین طبقه بند با توجه به دقت 100 درصدی طبقه بند دوم و سوم (البته نه در همه موارد مثلا 13 person برایش 100 نمیشد) دقت کلی مدل در

حالت 3 طبقه بند پشت هم را تعیین میکند.

در ادامه به صورت متنی نتایج طبقه بندی داده های تمام 15 فرد آمده است

Person 1

Confusion Matrix for Class 4 vs. Other Classes Classifier (Train):

```
[[ 864 108]
 [  0 2916]]
```

Accuracy for Class 4 vs. Other Classes Classifier (Train): 0.9722222222222222

Confusion Matrix for Class 4 vs. Other Classes Classifier (Test):

```
[[16  2]
 [ 0 54]]
```

Accuracy for Class 4 vs. Other Classes Classifier (Test): 0.9722222222222222

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Train):

```
[[ 972  0]
 [  0 1944]]
```

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Train): 1.0

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Test):

```
[[18  0]
 [ 0 36]]
```

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Test): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Train):

```
[[972  0]
 [  0 972]]
```

Accuracy for Class 1 vs. Class 2 Classifier (Train): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Test):

```
[[18  0]
 [ 0 18]]
```

Accuracy for Class 1 vs. Class 2 Classifier (Test): 1.0

Confusion Matrix for the Cascade of Classifiers (Train):

```
[[972  0  0  0]
 [  0 972  0  0]
 [  0  0 972  0]
 [108  0  0 864]]
```

Accuracy for the Cascade of Classifiers (Train): 0.9722222222222222

Confusion Matrix for the Cascade of Classifiers (Test):

```
[[18  0  0  0]
 [  0 18  0  0]
 [  0  0 18  0]
 [  2  0  0 16]]
```

Accuracy for the Cascade of Classifiers (Test): 0.9722222222222222

Person 2

Confusion Matrix for Class 4 vs. Other Classes Classifier (Train):

```
[[1026  57]
```

[0 3249]]

Accuracy for Class 4 vs. Other Classes Classifier (Train): 0.9868421052631579

Confusion Matrix for Class 4 vs. Other Classes Classifier (Test):

[[18 1]
[0 57]]

Accuracy for Class 4 vs. Other Classes Classifier (Test): 0.9868421052631579

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Train):

[[1083 0]
[0 2166]]

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Train): 1.0

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Test):

[[19 0]
[0 38]]

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Test): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Train):

[[1083 0]
[0 1083]]

Accuracy for Class 1 vs. Class 2 Classifier (Train): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Test):

[[19 0]
[0 19]]

Accuracy for Class 1 vs. Class 2 Classifier (Test): 1.0

Confusion Matrix for the Cascade of Classifiers (Train):

[[1083 0 0 0]
[0 1083 0 0]
[0 0 1083 0]
[0 0 57 1026]]

Accuracy for the Cascade of Classifiers (Train): 0.9868421052631579

Confusion Matrix for the Cascade of Classifiers (Test):

[[19 0 0 0]
[0 19 0 0]
[0 0 19 0]
[0 0 1 18]]

Accuracy for the Cascade of Classifiers (Test): 0.9868421052631579

Person 3

Confusion Matrix for Class 4 vs. Other Classes Classifier (Train):

[[969 114]
[57 3192]]

Accuracy for Class 4 vs. Other Classes Classifier (Train): 0.9605263157894737

Confusion Matrix for Class 4 vs. Other Classes Classifier (Test):

[[17 2]
[1 56]]

Accuracy for Class 4 vs. Other Classes Classifier (Test): 0.9605263157894737

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Train):

[[1083 0]
[0 2166]]

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Train): 1.0

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Test):

```
[[19 0]
 [ 0 38]]
```

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Test): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Train):

```
[[1083 0]
 [ 0 1083]]
```

Accuracy for Class 1 vs. Class 2 Classifier (Train): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Test):

```
[[19 0]
 [ 0 19]]
```

Confusion Matrix for the Cascade of Classifiers (Train):

```
[[1083 0 0 0]
 [ 0 1026 0 57]
 [ 0 0 1083 0]
 [ 57 57 0 969]]
```

Accuracy for the Cascade of Classifiers (Train): 0.9605263157894737

Confusion Matrix for the Cascade of Classifiers (Test):

```
[[19 0 0 0]
 [ 0 18 0 1]
 [ 0 0 19 0]
 [ 1 1 0 17]]
```

Accuracy for the Cascade of Classifiers (Test): 0.9605263157894737

Person 4

Confusion Matrix for Class 4 vs. Other Classes Classifier (Train):

```
[[ 918 54]
 [ 0 2916]]
```

Accuracy for Class 4 vs. Other Classes Classifier (Train): 0.9861111111111112

Confusion Matrix for Class 4 vs. Other Classes Classifier (Test):

```
[[17 1]
 [ 0 54]]
```

Accuracy for Class 4 vs. Other Classes Classifier (Test): 0.9861111111111112

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Train):

```
[[ 972 0]
 [ 0 1944]]
```

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Train): 1.0

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Test):

```
[[18 0]
 [ 0 36]]
```

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Test): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Train):

```
[[972 0]
 [ 0 972]]
```

Accuracy for Class 1 vs. Class 2 Classifier (Train): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Test):

```
[[18 0]
 [ 0 18]]
```

Accuracy for Class 1 vs. Class 2 Classifier (Test): 1.0

Confusion Matrix for the Cascade of Classifiers (Train):

```
[[972 0 0 0]
 [ 0 972 0 0]
 [ 0 0 972 0]
 [ 0 54 0 918]]
```

Accuracy for the Cascade of Classifiers (Train): 0.9861111111111112

Confusion Matrix for the Cascade of Classifiers (Test):

```
[[18 0 0 0]
 [ 0 18 0 0]
 [ 0 0 18 0]
 [ 0 1 0 17]]
```

Accuracy for the Cascade of Classifiers (Test): 0.9861111111111112

Person 5

Confusion Matrix for Class 4 vs. Other Classes Classifier (Train):

```
[[1200 0]
 [ 0 3600]]
```

Accuracy for Class 4 vs. Other Classes Classifier (Train): 1.0

Confusion Matrix for Class 4 vs. Other Classes Classifier (Test):

```
[[20 0]
 [ 0 60]]
```

Accuracy for Class 4 vs. Other Classes Classifier (Test): 1.0

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Train):

```
[[1140 60]
 [ 0 2400]]
```

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Train): 0.9833333333333333

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Test):

```
[[19 1]
 [ 0 40]]
```

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Test): 0.9833333333333333

Confusion Matrix for Class 1 vs. Class 2 Classifier (Train):

```
[[1200 0]
 [ 0 1200]]
```

Accuracy for Class 1 vs. Class 2 Classifier (Train): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Test):

```
[[20 0]
 [ 0 20]]
```

Accuracy for Class 1 vs. Class 2 Classifier (Test): 1.0

Confusion Matrix for the Cascade of Classifiers (Train):

```
[[1200 0 0 0]
 [ 0 1200 0 0]
 [ 60 0 1140 0]
 [ 0 0 0 1200]]
```

Accuracy for the Cascade of Classifiers (Train): 0.9875

Confusion Matrix for the Cascade of Classifiers (Test):

```
[[20 0 0 0]
 [ 0 20 0 0]
 [ 1 0 19 0]
 [ 0 0 0 20]]
```

Accuracy for the Cascade of Classifiers (Test): 0.9875

Person 6

Confusion Matrix for Class 4 vs. Other Classes Classifier (Train):

```
[[1080 120]
 [  0 3600]]
```

Accuracy for Class 4 vs. Other Classes Classifier (Train): 0.975

Confusion Matrix for Class 4 vs. Other Classes Classifier (Test):

```
[[18 2]
 [ 0 60]]
```

Accuracy for Class 4 vs. Other Classes Classifier (Test): 0.975

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Train):

```
[[1200  0]
 [  0 2400]]
```

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Train): 1.0

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Test):

```
[[20 0]
 [ 0 40]]
```

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Test): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Train):

```
[[1200  0]
 [  0 1200]]
```

Accuracy for Class 1 vs. Class 2 Classifier (Train): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Test):

```
[[20 0]
 [ 0 20]]
```

Accuracy for Class 1 vs. Class 2 Classifier (Test): 1.0

Confusion Matrix for the Cascade of Classifiers (Train):

```
[[1200  0  0  0]
 [  0 1200  0  0]
 [  0  0 1200  0]
 [  0  60  60 1080]]
```

Accuracy for the Cascade of Classifiers (Train): 0.975

Confusion Matrix for the Cascade of Classifiers (Test):

```
[[20 0 0 0]
 [ 0 20 0 0]
 [ 0 0 20 0]
 [ 0 1 1 18]]
```

Accuracy for the Cascade of Classifiers (Test): 0.975

Person 7

Confusion Matrix for Class 4 vs. Other Classes Classifier (Train):

```
[[ 765 102]
 [ 0 2601]]
```

Accuracy for Class 4 vs. Other Classes Classifier (Train): 0.9705882352941176

Confusion Matrix for Class 4 vs. Other Classes Classifier (Test):

```
[[15 2]
 [ 0 51]]
```

Accuracy for Class 4 vs. Other Classes Classifier (Test): 0.9705882352941176

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Train):

```
[[ 867 0]
 [ 0 1734]]
```

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Train): 1.0

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Test):

```
[[17 0]
 [ 0 34]]
```

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Test): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Train):

```
[[867 0]
 [ 0 867]]
```

Accuracy for Class 1 vs. Class 2 Classifier (Train): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Test):

```
[[17 0]
 [ 0 17]]
```

Accuracy for Class 1 vs. Class 2 Classifier (Test): 1.0

Confusion Matrix for the Cascade of Classifiers (Train):

```
[[867 0 0 0]
 [ 0 867 0 0]
 [ 0 0 867 0]
 [ 0 0 102 765]]
```

Accuracy for the Cascade of Classifiers (Train): 0.9705882352941176

Confusion Matrix for the Cascade of Classifiers (Test):

```
[[17 0 0 0]
 [ 0 17 0 0]
 [ 0 0 17 0]
 [ 0 0 2 15]]
```

Accuracy for the Cascade of Classifiers (Test): 0.9705882352941176

Person 8

Confusion Matrix for Class 4 vs. Other Classes Classifier (Train):

```
[[1026 57]
 [ 0 3249]]
```

Accuracy for Class 4 vs. Other Classes Classifier (Train): 0.9868421052631579

Confusion Matrix for Class 4 vs. Other Classes Classifier (Test):

```
[[18 1]
 [ 0 57]]
```

Accuracy for Class 4 vs. Other Classes Classifier (Test): 0.9868421052631579

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Train):

```
[[1083 0]
```

[0 2166]]

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Train): 1.0

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Test):

[[19 0]

[0 38]]

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Test): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Train):

[[1083 0]

[0 1083]]

Accuracy for Class 1 vs. Class 2 Classifier (Train): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Test):

[[19 0]

[0 19]]

Accuracy for Class 1 vs. Class 2 Classifier (Test): 1.0

Confusion Matrix for the Cascade of Classifiers (Train):

[[1083 0 0 0]

[0 1083 0 0]

[0 0 1083 0]

[0 57 0 1026]]

Accuracy for the Cascade of Classifiers (Train): 0.9868421052631579

Confusion Matrix for the Cascade of Classifiers (Test):

[[19 0 0 0]

[0 19 0 0]

[0 0 19 0]

[0 1 0 18]]

Accuracy for the Cascade of Classifiers (Test): 0.9868421052631579

Person 9

Confusion Matrix for Class 4 vs. Other Classes Classifier (Train):

[[969 114]

[0 3249]]

Accuracy for Class 4 vs. Other Classes Classifier (Train): 0.9736842105263158

Confusion Matrix for Class 4 vs. Other Classes Classifier (Test):

[[17 2]

[0 57]]

Accuracy for Class 4 vs. Other Classes Classifier (Test): 0.9736842105263158

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Train):

[[1083 0]

[57 2109]]

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Train): 0.9824561403508771

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Test):

[[19 0]

[1 37]]

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Test): 0.9824561403508771

Confusion Matrix for Class 1 vs. Class 2 Classifier (Train):

[[1083 0]

[0 1083]]

Accuracy for Class 1 vs. Class 2 Classifier (Train): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Test):

```
[[19 0]
 [ 0 19]]
```

Accuracy for Class 1 vs. Class 2 Classifier (Test): 1.0

Confusion Matrix for the Cascade of Classifiers (Train):

```
[[1026 0 57 0]
 [ 0 1083 0 0]
 [ 0 0 1083 0]
 [ 0 57 57 969]]
```

Accuracy for the Cascade of Classifiers (Train): 0.9605263157894737

Confusion Matrix for the Cascade of Classifiers (Test):

```
[[18 0 1 0]
 [ 0 19 0 0]
 [ 0 0 19 0]
 [ 0 1 1 17]]
```

Accuracy for the Cascade of Classifiers (Test): 0.9605263157894737

Person 10

Confusion Matrix for Class 4 vs. Other Classes Classifier (Train):

```
[[ 672 96]
 [ 0 2304]]
```

Accuracy for Class 4 vs. Other Classes Classifier (Train): 0.96875

Confusion Matrix for Class 4 vs. Other Classes Classifier (Test):

```
[[14 2]
 [ 0 48]]
```

Accuracy for Class 4 vs. Other Classes Classifier (Test): 0.96875

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Train):

```
[[ 768 0]
 [ 0 1536]]
```

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Train): 1.0

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Test):

```
[[16 0]
 [ 0 32]]
```

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Test): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Train):

```
[[768 0]
 [ 0 768]]
```

Accuracy for Class 1 vs. Class 2 Classifier (Train): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Test):

```
[[16 0]
 [ 0 16]]
```

Accuracy for Class 1 vs. Class 2 Classifier (Test): 1.0

Confusion Matrix for the Cascade of Classifiers (Train):

```
[[768 0 0 0]
 [ 0 768 0 0]
 [ 0 0 768 0]
```

[0 96 0 672]]
Accuracy for the Cascade of Classifiers (Train): 0.96875

Confusion Matrix for the Cascade of Classifiers (Test):

[[16 0 0 0]
[0 16 0 0]
[0 0 16 0]
[0 2 0 14]]

Accuracy for the Cascade of Classifiers (Test): 0.96875

Person 11

Confusion Matrix for Class 4 vs. Other Classes Classifier (Train):

[[867 0]
[102 2499]]

Accuracy for Class 4 vs. Other Classes Classifier (Train): 0.9705882352941176

Confusion Matrix for Class 4 vs. Other Classes Classifier (Test):

[[17 0]
[2 49]]

Accuracy for Class 4 vs. Other Classes Classifier (Test): 0.9705882352941176

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Train):

[[867 0]
[0 1734]]

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Train): 1.0

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Test):

[[17 0]
[0 34]]

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Test): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Train):

[[867 0]
[0 867]]

Accuracy for Class 1 vs. Class 2 Classifier (Train): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Test):

[[17 0]
[0 17]]

Accuracy for Class 1 vs. Class 2 Classifier (Test): 1.0

Confusion Matrix for the Cascade of Classifiers (Train):

[[765 0 0 102]
[0 867 0 0]
[0 0 867 0]
[0 0 0 867]]

Accuracy for the Cascade of Classifiers (Train): 0.9705882352941176

Confusion Matrix for the Cascade of Classifiers (Test):

[[15 0 0 2]
[0 17 0 0]
[0 0 17 0]
[0 0 0 17]]

Accuracy for the Cascade of Classifiers (Test): 0.9705882352941176

Person 12

Confusion Matrix for Class 4 vs. Other Classes Classifier (Train):

```
[[ 972  0]
 [ 54 2862]]
```

Accuracy for Class 4 vs. Other Classes Classifier (Train): 0.9861111111111112

Confusion Matrix for Class 4 vs. Other Classes Classifier (Test):

```
[[18  0]
 [ 1 53]]
```

Accuracy for Class 4 vs. Other Classes Classifier (Test): 0.9861111111111112

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Train):

```
[[ 972  0]
 [  0 1944]]
```

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Train): 1.0

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Test):

```
[[18  0]
 [ 0 36]]
```

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Test): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Train):

```
[[972  0]
 [  0 972]]
```

Accuracy for Class 1 vs. Class 2 Classifier (Train): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Test):

```
[[18  0]
 [ 0 18]]
```

Accuracy for Class 1 vs. Class 2 Classifier (Test): 1.0

Confusion Matrix for the Cascade of Classifiers (Train):

```
[[972  0  0  0]
 [  0 972  0  0]
 [  0  0 918 54]
 [  0  0  0 972]]
```

Accuracy for the Cascade of Classifiers (Train): 0.9861111111111112

Confusion Matrix for the Cascade of Classifiers (Test):

```
[[18  0  0  0]
 [ 0 18  0  0]
 [ 0  0 17  1]
 [ 0  0  0 18]]
```

Accuracy for the Cascade of Classifiers (Test): 0.9861111111111112

Person 13

Confusion Matrix for Class 4 vs. Other Classes Classifier (Train):

```
[[1200  0]
 [ 60 3540]]
```

Accuracy for Class 4 vs. Other Classes Classifier (Train): 0.9875

Confusion Matrix for Class 4 vs. Other Classes Classifier (Test):

```
[[20  0]
 [ 1 59]]
```

Accuracy for Class 4 vs. Other Classes Classifier (Test): 0.9875

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Train):

```
[[1140 60]
 [ 60 2340]]
```

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Train): 0.9666666666666667

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Test):

```
[[19 1]
 [ 1 39]]
```

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Test): 0.9666666666666667

Confusion Matrix for Class 1 vs. Class 2 Classifier (Train):

```
[[1200 0]
 [ 0 1200]]
```

Accuracy for Class 1 vs. Class 2 Classifier (Train): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Test):

```
[[20 0]
 [ 0 20]]
```

Accuracy for Class 1 vs. Class 2 Classifier (Test): 1.0

Confusion Matrix for the Cascade of Classifiers (Train):

```
[[1140 0 60 0]
 [ 0 1140 0 60]
 [ 60 0 1140 0]
 [ 0 0 0 1200]]
```

Accuracy for the Cascade of Classifiers (Train): 0.9625

Confusion Matrix for the Cascade of Classifiers (Test):

```
[[19 0 1 0]
 [ 0 19 0 1]
 [ 1 0 19 0]
 [ 0 0 0 20]]
```

Accuracy for the Cascade of Classifiers (Test): 0.9625

Person 14

Confusion Matrix for Class 4 vs. Other Classes Classifier (Train):

```
[[1083 0]
 [ 0 3249]]
```

Accuracy for Class 4 vs. Other Classes Classifier (Train): 1.0

Confusion Matrix for Class 4 vs. Other Classes Classifier (Test):

```
[[19 0]
 [ 0 57]]
```

Accuracy for Class 4 vs. Other Classes Classifier (Test): 1.0

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Train):

```
[[1083 0]
 [ 0 2166]]
```

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Train): 1.0

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Test):

```
[[19 0]
 [ 0 38]]
```

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Test): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Train):

```
[[1083  0]
 [  0 1083]]
```

Accuracy for Class 1 vs. Class 2 Classifier (Train): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Test):

```
[[19  0]
 [  0 19]]
```

Accuracy for Class 1 vs. Class 2 Classifier (Test): 1.0

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Train): 1.0

Accuracy for Class 1 vs. Class 2 Classifier (Train): 1.0

Confusion Matrix for the Cascade of Classifiers (Train):

```
[[1083  0  0  0]
 [  0 1083  0  0]
 [  0  0 1083  0]
 [  0  0  0 1083]]
```

Accuracy for the Cascade of Classifiers (Train): 1.0

Confusion Matrix for the Cascade of Classifiers (Test):

```
[[19  0  0  0]
 [  0 19  0  0]
 [  0  0 19  0]
 [  0  0  0 19]]
```

Accuracy for the Cascade of Classifiers (Test): 1.0

Person 15

Confusion Matrix for Class 4 vs. Other Classes Classifier (Train):

```
[[1083  0]
 [  0 3249]]
```

Accuracy for Class 4 vs. Other Classes Classifier (Train): 1.0

Confusion Matrix for Class 4 vs. Other Classes Classifier (Test):

```
[[19  0]
 [  0 57]]
```

Accuracy for Class 4 vs. Other Classes Classifier (Test): 1.0

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Train):

```
[[1083  0]
 [  0 2166]]
```

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Train): 1.0

Confusion Matrix for Class 3 vs. (Class 1 and 2) Classifier (Test):

```
[[19  0]
 [  0 38]]
```

Accuracy for Class 3 vs. (Class 1 and 2) Classifier (Test): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Train):

```
[[1083  0]
 [  0 1083]]
```

Accuracy for Class 1 vs. Class 2 Classifier (Train): 1.0

Confusion Matrix for Class 1 vs. Class 2 Classifier (Test):

```
[[19  0]
 [  0 19]]
```

Accuracy for Class 1 vs. Class 2 Classifier (Test): 1.0

Confusion Matrix for the Cascade of Classifiers (Train):

```
[[1083 0 0 0]
 [ 0 1083 0 0]
 [ 0 0 1083 0]
 [ 0 0 0 1083]]
```

Accuracy for the Cascade of Classifiers (Train): 1.0

Confusion Matrix for the Cascade of Classifiers (Test):

```
[[19 0 0 0]
 [ 0 19 0 0]
 [ 0 0 19 0]
 [ 0 0 0 19]]
```

Accuracy for the Cascade of Classifiers (Test): 1.0

در مورد وزن ها نیز با توجه به محدودیت تایپ پاسخ در نوت بوک بخش پرینت آن کامنت شده شده که

میتوان وزن های W_{LDA} و W_{CSP} را مشاهده کرد.

۱۰-۱- نتیجه گیری

در نتیجه، روش طبقه بندی پیشنهادی، که شامل اولویت بندی طبقه بندی کلاس 4 در برابر سایر کلاس های

تصور حرکتی و سپس مقایسه کلاس 3 با دو کلاس 1 و 2 و در نهایت مقایسه کلاس 1 و 2 استفاده از سه

طبقه بندی کننده با تحلیل تشخیص خطی (LDA) است، در مقایسه با استفاده از ماشین بردار پشتیبان (SVM) با

چهار کلاس کارآمدتر است. کلاس های ترکیبی علاوه بر این، با استفاده از یک رویکرد اعتبارسنجی متقاطع

LOO و آموزش مدل های فردی برای داده های هر فرد (در مجموع 15 مدل)، به نرخ های دقت بالایی دست یافتیم

که به طور متوسط حدود 96-97٪ برای هر فرد است.

تصمیم به تمرکز اولیه روی کلاس 4، که نشان دهنده حالت استراحت است، به عنوان یک گام مهم در ایجاد

یک نقطه مرجع برای تمایز بین تصور حرکتی و عدم حرکت عمل می کند. با استفاده از LDA، ما به طور موثر


ویژگی های متمایز کننده ای را که کلاس 4 را از سایر کلاس های تصاویر حرکتی متمایز می کند، ثبت می کنیم.

این رویکرد طبقه‌بندی متوالی به ما امکان می‌دهد فرآیند طبقه‌بندی را اصلاح کنیم و تمایز بین انواع حرکت خاص را بهینه کنیم.

۱۱-۱- ضمیمه : توضیح و نحوه ران کد ارسالی

کد ipynb ارسالی که نوت بوک ای است که این پروژه با زبان پایتون در آن نوشته شده است قابل استفاده در anaconda jupyter notebook و همچنین آپلود در google colab می‌باشد.

این نوت بوک شامل 4 بخش است.

Table of contents	
Load Data	
Install and import	
Train 1 Model for ALL 15 person data - Split Leave one out	
Train 1 Model for ALL 15 person data - Split Randomly	
Train 15 Models for each Person (Split LOO)	

شکل (۱-۳۴) بخش های مختلف نوت بوک ارسالی

بخش 1 : دانلود دیتاست

با توجه به اینکه دیتاست در گوگل درایو آپلود شده بود به راحتی با دستور gdown آن را دانلود کرده و

unzip می‌کنیم.

```
# download Dataset
!gdown 1h_Xi0ms4kpCvzSsM0sGfhhUqYezXk3s_

# unzip dataset
!unzip "/content/dataset.zip"
```

شکل (۱-۳۵) کد متلب دانلود دیتاست

بخش 2: نصب کتاب خانه های مورد نیاز و import انها

در این بخش در یک فایل تکست در directory به نام requirements.txt ساخته میشود و پس از آن تمام

کتاب خانه های مورد نیاز را دانلود و نصب میکند.

بخش سوم شامل کد های بخش 1-9-1 میباشد.

بخش چهارم شامل کد های 1-9-2 میباشد.

بخش پنجم نیز شامل کد های 1-9-4 میباشد.