به نام خدا

گزارش پروژه درس جدا سازی کور منابع

استاد درس: آقای دکتر اخوان بهابادی

دانشجو : امیرحسین چمی

فهرست

بررسی اولیه داده ها

در ابتدا در فایل data_review.mlx تمام آنچه در مورد ویژگی های دیتاست در فایل recording.pdf گفته شده بود را مورد بررسی قرار دادم. این داد ها را استثنائا در بخش live بررسی کردم زیرا می خواستم خروجی کد را در کنار آن داشته باشم. بقیه ی فایل ها به صورت .m خواهند بود.

In this file i just want to review the data

```
clc;clear;
cd dataset\
load subj_1.mat
cd ..
```

```
whos
```

```
  Name      Size              Bytes  Class    Attributes

  data      1x4           283853280  cell
```

```
size(data{1})
```

```
ans = 1×3
          64        7200          18
```

در این بخش داده های نفر اول یعنی subj_۱.m را load کرده و ساختار آن که یک سلول ۱*۴ است را بررسی می کنم. علی رغم اینکه فایل Recording.pdf گفته، ما ۶۴ کانال داریم یعنی کانال مرجع از آن حذف نشده است. حال باید بررسی کنی که آیا این مورد برای تمام subject ها برقرار است یا خیر.

so we have 64 channels not 63

but is it same for all ?

```
clc; clear;
cd dataset\
folder_path = '.';
num_subjects = 15;

for i = 1:num_subjects
    filename = fullfile(folder_path, sprintf('subj_%d.mat', i));
    if exist(filename, 'file')
        load(filename, 'data');

        [num_channels, num_samples, num_trials] = size(data{1});

        fprintf('Subject %2d: %d channels, %d samples, %d trials (class 1)\n', ...
            i, num_channels, num_samples, num_trials);
    else
        warning('File not found: %s', filename);
    end
end
```

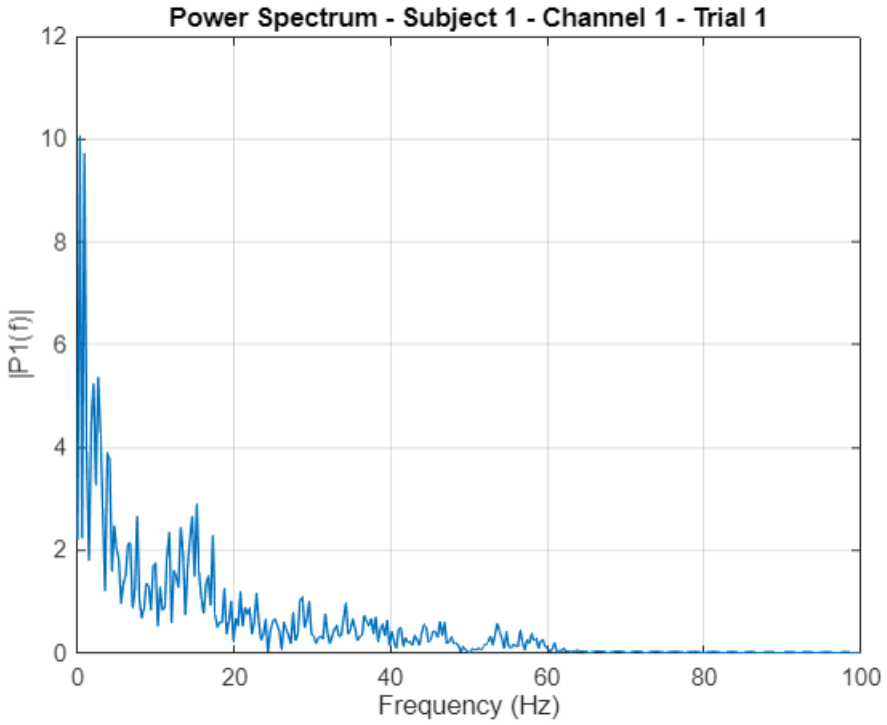
```
Subject  1: 64 channels, 7200 samples, 18 trials (class 1)
Subject  2: 64 channels, 7200 samples, 20 trials (class 1)
Subject  3: 64 channels, 7200 samples, 19 trials (class 1)
Subject  4: 64 channels, 7200 samples, 19 trials (class 1)
Subject  5: 64 channels, 7200 samples, 20 trials (class 1)
Subject  6: 64 channels, 7200 samples, 20 trials (class 1)
Subject  7: 64 channels, 7200 samples, 17 trials (class 1)
Subject  8: 64 channels, 7200 samples, 20 trials (class 1)
Subject  9: 64 channels, 7200 samples, 19 trials (class 1)
Subject 10: 64 channels, 7200 samples, 19 trials (class 1)
Subject 11: 64 channels, 7200 samples, 19 trials (class 1)
Subject 12: 64 channels, 7200 samples, 18 trials (class 1)
Subject 13: 64 channels, 7200 samples, 20 trials (class 1)
Subject 14: 64 channels, 7200 samples, 20 trials (class 1)
Subject 15: 64 channels, 7200 samples, 19 trials (class 1)
```

حال می خواهم طیف فرکانسی داده ها را بررسی کنم.

so all of them are 64 channels

what about frequency spectrum?

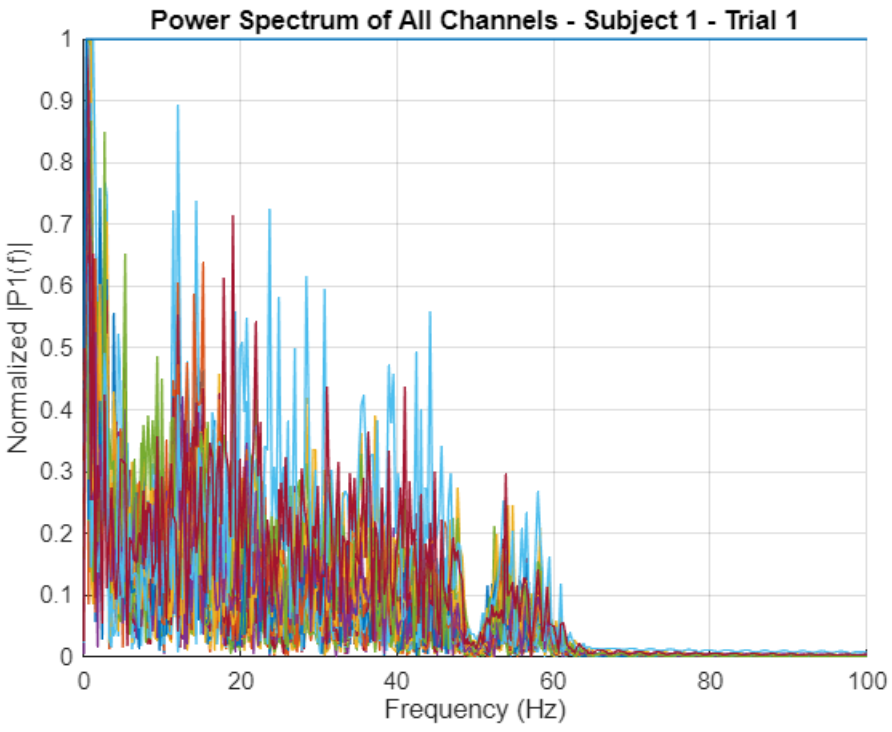
```
clc; clear;

subject_id = 1;
class_id = 1; % arm class
channel_id = 1;
trial_id = 1;
cd dataset\
load(sprintf('subj_%d.mat', subject_id));

Fs = 2400;  % sampling frequency
signal = squeeze(data{class_id}(channel_id, :, trial_id));

% FFT
L = length(signal);
NFFT = 2^nextpow2(L);
Y = fft(signal, NFFT);
P2 = abs(Y/L);
P1 = P2(1:NFFT/2+1);
P1(2:end-1) = 2*P1(2:end-1);

f = Fs*(0:(NFFT/2))/NFFT;

figure;
plot(f, P1);
xlim([0 100]);
xlabel('Frequency (Hz)');
ylabel('|P1(f)|');
title(sprintf('Power Spectrum - Subject %d - Channel %d - Trial %d', ...
    subject_id, channel_id, trial_id));
grid on;
```



پس طیف داده ی trial اول از کانال اول و subj_۱ پایین گذر است. حال ببینیم این برای تمام کانال ها برقرار است؟

so this is low pass. but is it same for all channels ?

```
clc; clear;

subject_id = 1;
class_id = 1;  % arm class
trial_id = 1;
Fs = 2400; % sampling frequency

load(sprintf('subj_%d.mat', subject_id));
[num_channels, ~, ~] = size(data{class_id});

L = size(data{class_id}, 2);
NFFT = 2^nextpow2(L);
f = Fs*(0:(NFFT/2))/NFFT;

figure;
hold on;
for ch = 1:num_channels
    signal = squeeze(data{class_id}(ch, :, trial_id));
    Y = fft(signal, NFFT);
    P2 = abs(Y/L);
    P1 = P2(1:NFFT/2+1);
    P1(2:end-1) = 2*P1(2:end-1);

    P1 = P1 / max(P1); % Normalization: Just to better understand the diagram

    plot(f, P1);
end

xlim([0 100]);
xlabel('Frequency (Hz)');
ylabel('Normalized |P1(f)|');
title(sprintf('Power Spectrum of All Channels - Subject %d - Trial %d', subject_id, trial_id));
grid on;
```



پس تمام کانال ها پایین گذر هستند.

در این جا کار من با فایل data_review.mlx به پایان می رسد. موارد مهمی که فهمیدم به شرح زیر است:

- ساختار داده ها
- تعداد کانال ها
- پایین گذر بودن طیف فرکانسی

پیش پردازش داده ها

۱. انتخاب کانال ها

تنها مورد متفاوت با ادعای فایل Recording.pdf تعداد کانال هاست که کانال مرجع از آن حذف نشده است. در کل دو رویکرد برای این مورد وجود دارد:

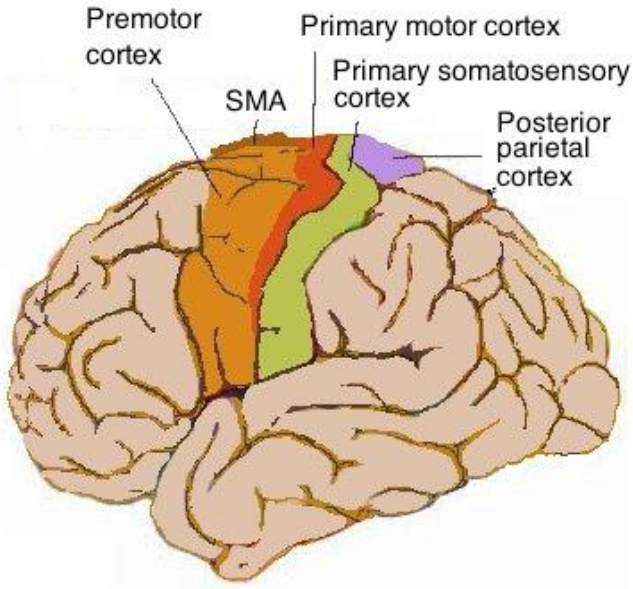
- رویکرد اول اینکه یکی از کانال ها را به عنوان مرجع انتخاب کرده و آن را از بین کانال ها حذف کنیم. انتخاب این کانال مرجع می تواند با معیار های مختلفی مثل میانگین دامنه ی آن کانال باشد.
- رویکرد دوم بر اساس یک دیدگاه متفاوت است؛ در چنین آزمایشی تمام کانال ها الزاما کاربردی و مفید نیستند پس می توانیم تنها کانال های مفید را انتخاب کرده و بقیه ی کانال ها را کنار بگذاریم. در این صورت احتمالا تعداد کانال های انتخابی از ۶۴ تا خیلی کمتر است.

من رویکرد دوم را انتخاب می کنم. این رویکرد به دو روش قابل پیاده سازی است.

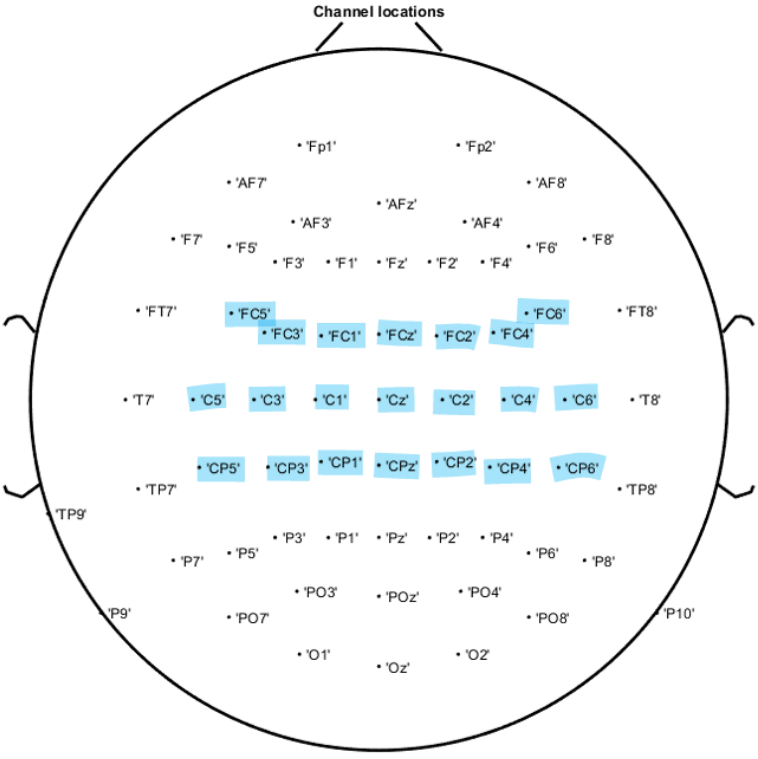
- روش اول استفاده از CSP است. با اعمال CSP روی داده ها، یک بردار از وزن ها به دست خواهیم آورد. بعضی از این وزن ها مقدار بیشتری دارند که این مقدار بیشتر نشان دهنده ی این است که کانال متناظر با آن مقدار، بخش مهم تری از مغز را شامل می شود. پس ما کانال های متناظر با این وزن ها را انتخاب می کنیم.

  واضحا برای پیاده سازی این روش از داده های کلاس چهارم نباید استفاده کنیم زیرا در کلاس چهارم داده های سکون اشخاص ذخیره شده اند در نتیجه بخش هایی که برای ما مهم هستند احتمالا در این کلاس تفاوت خاصی با بقیه ی بخش های مغز نخواهند داشت.

- روش دوم با توجه به ساختار مغز است. پس ابتدا نیاز است ساختار مغز را بشناسیم. در تصویر زیر بخش هایی که نام گذاری شده اند، همگی مسئول فرآیند حرکت اندام های بدن هستند. از تصمیم به حرکت تا حرکت اندام ها.[۱]



پس من هم کانال هایی که به نظر متناظر با همین بخش ها هستند را انتخاب می کنم. در تصویر زیر این کانال ها مشخص هستند.



شماره ی کانال های انتخاب شده به شرح زیر است.

۱۱, ۴۰, ۱۲, ۴۱, ۱۳, ۴۲, ۱۴, ۴۴, ۱۶, ۴۵, ۱۷, ۴۶, ۱۸, ۴۷, ۲۰, ۴۹, ۲۱, ۵۰, ۲۲, ۵۱, ۲۳

۲. بالانس کردن تعداد trial ها

از آنجا که در بررسی دیتا دیدم، تعداد trial ها برای کلاس های مختلف و برای افراد مختلف متفاوت است. به همین دلیل برای این که از سوگیری مدل در مرحله ی آموزش جلوگیری کنم، تعداد trial ها را در تمام کلاس ها یکسان و برابر با کمترین تعداد برای هر شخص می کنم.

۳. فیلتر کردن سیگنال ها

در بررسی اولیه در فایل data_review.mlx دیدم که سیگنال ها همگی پایین گذر بوده و طیف آن ها تا حدود ۶۰ هرتز را شامل می شود. به علت شرایط ضبط سیگنال های EEG مثل فرکانس برق شهر و یا عملکرد بخش های دیگر مغز، تمام این طیف فرکانسی شامل داده های مورد نظر ما نیست. پس باید آن را فیلتر کنیم. دو راه برای فیلتر کردن وجود دارد.

- رویکرد اول: استفاده از یک طیف فرکانسی ثابت برای تمامی اشخاص. از آنجا که داده های ما سیگنال های EEG هستند، و هدف ما تشخیص حرکت های مختلف شخص است، می توانیم با شناخت دقیق تر مغز فرکانس هایی را انتخاب کنیم که برای هدف ما مناسب هستند. مغز انواع مختلفی از موج را تولید می کند که دلتا[۲]، تتا[۳]، آلفا[۴] و بتا[۵] چهار موج اصلی هستند.

  هر کدام از این سیگنال ها نشان دهنده یک حالت خاص هستند.

  دلتا: دلتا سیگنال خواب عمیق است. شامل فرکانس های ۱ الی ۴ هرتز می شود.

  تتا: سیگنال حالت آرامش یا خواب سبک است. شامل فرکانس های ۴ الی ۷ هرتز است.

  آلفا: سیگنال حالت استراحت یا چشم بسته است. شامل فرکانس های ۸ الی ۱۲ هرتز است.

  بتا: سیگنال فعالیت مغزی بالا و تمرکز است. شامل فرکانس های ۱۲.۵ الی ۳۰ هرتز است.

  پس با توجه به چهار کلاس مورد نظر ما، بهتر است امواج آلفا و بتا را در نظر بگیریم یعنی بازه ی فرکانسی ۸ الی ۳۰ هرتز. برای اینکه از اعوجاج هم جلوگیری کنیم می توانیم کمی این بازه را بزرگ تر درنظر بگیریم. برای مثال ۷ الی ۳۵ هرتز.

- رویکرد دوم: استفاده از پهنای باند جداگانه برای هر شخص. در این روش پهنای باند برای هر شخص جداگانه تعیین شده و بر اساس این پهنای باند فیلتر طراحی می شود.

٤. رفع مقادیر nan و inf

یکی از مواردی که می تواند در اثر شرایط ضبط سیگنال ها رخ دهد، وجود داده های Nan یا inf است. می توانیم آن ها را صفر کنیم.

تمامی موارد مربوط به پیش پردازش که گفته شد در فایل preprocessing.m در پوشه ی preprocess پیاده سازی می شود.

```
clc;
clear;
close all;

% Selected channels corresponding to the motor cortex
selected_channels = [11,40,12,13,42,14,44,16,45,46,18,47,20,49,21,22,51,23,41,17,50];
% selected_channels = 1:63;

Fs = 2400;

% Define frequency ranges for each subject
filter_ranges = {
    [5 25], [5 25], [5 30], [10 25], [10 25], ...
    [5 30], [10 25], [8 70], [8 50], [5 30], ...
    [8 70], [8 20], [20 50], [8 70], [8 60]
};

cd ..\dataset\

num_subjects = 15;

for subj = 1:num_subjects
    X = cell(1,4);

    temp = load(sprintf('subj_%d.mat', subj));
    data = temp.data;

    % Design Bandpass Filter for This Subject
    freq_range = filter_ranges{subj};
    Wn = freq_range / (Fs/2);
    [b, a] = butter(4, Wn, 'bandpass');

    % Find the minimum number of trials among all 4 classes
    min_trials = inf;
    for cls = 1:4
        [~, ~, trials] = size(data{cls});
        min_trials = min(min_trials, trials);
    end
```

به علت طولانی بودن کد، ادامه ی آن در صفحه ی بعد.

```
    % Trim each class to the same number of trials, and select only motor cortex channels
    for cls = 1:4
        trials_data = data{cls}(selected_channels, :, 1:min_trials);

        for trial = 1:min_trials
            trials_data(:,:,trial) = filtfilt(b, a, trials_data(:,:,trial).').';

            % fix Nan/inf values
            temp = trials_data(:,:,trial);
            temp(isnan(temp)) = 0;
            temp(isinf(temp)) = 0;
            trials_data(:,:,trial) = temp;
        end

        X{cls} = trials_data;
    end

    cd ..\preprocess\processed_dataset
    save(sprintf('preprocessed_subj_%d.mat', subj), 'X');
    cd ..\..\dataset\
end

disp('All subjects preprocessed with subject-specific bandpass filters.');
```

در این کد داده های هر شخص لود می شود و پس از انجام مراحل پیش پردازش ذکر شده، در پوشه ی processed_dataset ذخیره می شود.

الگوی فضایی مشترک (CSP)

الگوی فضایی مشترک (CSP) یک تکنیک فیلتر فضایی است که هدف آن یافتن مجموعه ای از فیلتر های فضایی است که بتواند به طور موثر بین دو کلاس سیگنال بر اساس ماتریس های کواریانس آن ها تمایز قائل شود. برای پیاده سازی آن تابع CSP را در فایل CSP.m در پوشه ی train and evaluate نوشته ام.

```
function [W_csp] = CSP(X, Y, num_filters)
    cov1 = cov(X');
    cov2 = cov(Y');

    % whitening
    [V, D] = eig(cov1 + cov2);
    P = sqrt(inv(D)) * V';

    S1 = P * cov1 * P';
    S2 = P * cov2 * P';

    % GEVD
    [U, D] = eig(S1, S1 + S2);
    [~, idx] = sort(diag(D), 'descend');
    W_all = U(:, idx);

    W = [W_all(:, 1:num_filters/2), W_all(:, end - num_filters/2 + 1:end)];

    W_csp = W' * P; % filters
end
```

جلو تر خواهیم دید که رویکرد آموزش مدل را رویکرد، طبقه بندی cascade انتخاب می کنم. به همین دلیل باید داده هایی که برای آموزش مدل استفاده می شوند هم بر همین اساس تفکیک و ویژگی آن ها استخراج شود. بر همین اساس کد زیر در کد آموزش اصلی یعنی train_and_evaluate.m نوشته شده.

```
if cls == 4
    pos_trials = class;
    pos_trials(:,:,i) = [];

    neg_trials = cat(3, X{1}, X{2}, X{3});
    neg_trials(:,:,i) = [];

elseif cls == 3
    pos_trials = class;
    pos_trials(:,:,i) = [];

    neg_trials = cat(3, X{1}, X{2});
    neg_trials(:,:,i) = [];

elseif cls == 2
    pos_trials = class;
    pos_trials(:,:,i) = [];

    neg_trials = X{1};
    neg_trials(:,:,i) = [];

else
    pos_trials = class;
    pos_trials(:,:,i) = [];

    neg_trials = X{2};
    neg_trials(:,:,i) = [];
end
```

در هر مرحله با توجه به کلاسی که در آن هستیم داده ها را به دو دسته تقسیم کرده و آن ها را به تابع CSP دهیم. این کد را در بخش آموزش مدل خواهیم دید.

الگوریتم LDA

این الگوریتم در واقع ابزاری است که برای طبقه بندی داده ها استفاده می شود. هدف آن یافتن یک ترکیب خطی از ویژگی هاست که بیشترین تماین را بین کلاس های مختلف قائل شود. می توان آن را مستقیما نوشت و یا از دستور fitcdiscr() استفاده کرد.

الگوریتم grid search

الگوریتم Grid Search[۶] یک روش جستجوی فراگیر برای پیدا کردن بهترین ترکیب از ابرپارامترها در مدل های یادگیری ماشین است. در این روش، با تعریف یک شبکه (grid) از مقادیر ممکن برای هر پارامتر، همه ی ترکیب های ممکن بررسی می شوند و عملکرد مدل برای هر ترکیب با استفاده از یک معیار ارزیابی (مثل دقت یا میانگین خطا) سنجیده می شود.

برای پیاده سازی آن در پوشه ی grid search فایلی با همین نام نوشتم و الگوریتم آن را پیاده سازی کردم. تنها تفاوت کد آن با کد اصلی که در بخش آموزش مدل -رویکرد اول- خواهیم دید، این است که تعداد فیلتر ها در این کد یک range است که برای هر عدد در این بازه یکبار مدل برای هر شخص اموزش داده می شود و دقت آن اندازه گیری می شود. به دلیل اینکه کد طولانی و تکراری است، تصویر آن را قرار ندادم. برای بررسی دقیق تر به فایل آن مراجعه فرمایید.

طبق این الگوریتم تعداد فیلتر های مکانی برای هر subject به صورت زیر انتخاب شد.

Subj۱: ۱٤, subj۲: ۸, subj۳: ۱٤, subj٤: ٦, subj٥: ۱٤, subj٦: ۱۰, subj۷: ۱٤, subj۸: ۱۰ ,
subj۹: ۱٤, subj۱۰: ۸, subj۱۱: ۱٤, subj۱۲: ۱۲, subj۱۳: ۱۰, subj۱٤: ۱۲, subj۱٥: ٦

من یکبار با تعداد فیلتر های ثابت مدل ها را آموزش دادم و دقت های بدست آمده خوب بود. اما به نظرم راضی کننده نبود برای همین از الگوریتم grid search استفاده می کنم.

رویکرد طبقه بندی

۱. پیاده سازی تابع طبقه بندی با LDA

از بین روش های ممکن برای طبقه بندی چهار کلاس مختلف که در کلاس به آن اشاره شد، من از روش پیشنهادی فایل Report.pdf استفاده می کنم. این رویکرد طبقه بندی به صورت آبشاری (cascade) است.[۷]

در این روش ابتدا کلاسی که می دانیم تشخیص آن نسبت به بقیه ساده تر است را انتخاب می کنیم. سپس طبقه بند اول را برای تشخیص بین این کلاس (در اینجا کلاس چهارم) و سه کلاس دیگر آموزش می دهیم. و همین فرآیند را برای سه کلاس دیگر تکرار می کنیم.(اطلاعات کافی در کلاس در این مورد ارائه شد. برای آشنایی بیشتر به مرجع شماره ۶ مراجعه نمایید.)

تابع آن را در فایل train_cascade_LDA.m در پوشه ی train and evaluate پیاده سازی کردم.

```
function cascade_model = train_cascade_LDA(X, Y)
    cascade_model = {};

    % class 4 vs rest
    Y1 = (Y == 4);
    model4 = fitcdiscr(X, Y1, 'DiscrimType', 'diagLinear');
    cascade_model{end+1} = struct('model', model4, 'class', 4);

    % class 3 vs 2 and 1
    idx = Y ~= 4;
    X2 = X(idx, :);
    Y2 = Y(idx) == 3;
    model3 = fitcdiscr(X2, Y2, 'DiscrimType', 'diagLinear');
    cascade_model{end+1} = struct('model', model3, 'class', 3);

    % class 2 vs class 1
    idx = Y == 1 | Y == 2;
    X3 = X(idx, :);
    Y3 = Y(idx) == 2;
    model2 = fitcdiscr(X3, Y3, 'DiscrimType', 'diagLinear');
    cascade_model{end+1} = struct('model', model2, 'class', 2);
end
```

۲. پیاده سازی تابع پیش بینی

از آنجا که از طبقه بندی آبشاری استفاده شد، باید تابع پیش بینی هم مطابق با آن باشد. در این پیش بینی هر داده ابتدا با استفاده از یک مدل آموزش داده، پیش بینی می شود. اگر پیش بینی آن مدل ۱ منطقی بود پس به آن کلاس مربوط است و اگر ۰ منطقی بود به کلاس بعدی می رویم و این روال را تا طبقه بند سوم ادامه می دهیم و اگر داده ای در پیش بینی طبقه بند سوم هم ۰ منطقی بود پس طبق فرض باید آن داده برای کلاس ۱ باشد.

تابع آن را در فایل predict_cascade_LDA.m و در پوشه ی train and evaluate نوشتم.

```
function preds = predict_cascade_LDA(cascade_model, X)
    preds = zeros(size(X,1),1);
    for i = 1:size(X,1)
        for j = 1:length(cascade_model)
            model = cascade_model{j}.model;
            cls   = cascade_model{j}.class;
            label = predict(model, X(i,:));
            if label == 1
                preds(i) = cls;
                break;
            end
        end
        if preds(i) == 0
            preds(i) = 1;  % default to class 1
        end
    end
end
```

رویکرد های آموزش مدل

۱. آموزش یک مدل برای هر subject

مطابق کدی که در فایل train_one_model_per_subj.m در پوشه ی train and evaluate نوشتم، داده های هر شخص لود شده و پس از اعمال روش Leave One Out آموزش مدل شروع شده و در انتها مدل آموزش دیده ارزیابی شده، سپس مدل آموزش دیده برای هر شخص در مسیر models/one model per subj ذخیره می شود و ماتریس کانفیوژن اموزش و تست در یک fig رسم شده و این ماتریس های کانفیوژن با دو فرمت jpg. و fig. در مسیر confusion matrix/cm model per subj ذخیره می شود.

```
clc;
clear;
close all;

num_filters_list = [14 8 14 6 14 10 14 10 14 8 14 12 10 12 6];
num_subjects = 15;

for subj= 2:num_subjects
    load(sprintf('../preprocess/processed_dataset/preprocessed_subj_%d.mat', subj));
    num_filters = num_filters_list(subj);

    num_trials = size(X{1}, 3); % The number of trials is the same in all classes due to preprocessing.
    train_confmat = zeros(4,4);
    test_confmat = zeros(4,4);

    for counter = 1 : num_trials * 4
        X_train = [];
        Y_train = [];
        X_test = [];
        Y_test = [];

        for cls = 4 : -1 : 1
            class = X{cls};
```

ادامه کد در تصویر بعدی.

```
        for cls = 4 : -1 : 1
            class = X{cls};

            for i = 1: num_trials % LOO
                % Data preparation for cascade CSP
                if cls == 4
                    pos_trials = class;
                    pos_trials(:,:,i) = [];

                    neg_trials = cat(3, X{1}, X{2}, X{3});
                    neg_trials(:,:,i) = [];

                elseif cls == 3
                    pos_trials = class;
                    pos_trials(:,:,i) = [];

                    neg_trials = cat(3, X{1}, X{2});
                    neg_trials(:,:,i) = [];

                elseif cls == 2
                    pos_trials = class;
                    pos_trials(:,:,i) = [];

                    neg_trials = X{1};
                    neg_trials(:,:,i) = [];

                else
                    pos_trials = class;
                    pos_trials(:,:,i) = [];

                    neg_trials = X{2};
                    neg_trials(:,:,i) = [];
                end
```

```
                train_trials_reshaped = reshape(pos_trials, size(pos_trials,1), []);
                ref_trials_reshaped = reshape(neg_trials,  size(neg_trials,1),  []);

                [W_csp] = CSP(train_trials_reshaped, ref_trials_reshaped, num_filters);

                train_trials = pos_trials;
                test_trial = class(:,:,i);

                % train features
                X_features = zeros(num_trials - 1, num_filters);
                labels = zeros(num_trials - 1, 1);

                for i = 1 : num_trials - 1
                    Xi = train_trials(:, :, i);
                    X_train_csp = W_csp * Xi;
                    % feature = var(X_csp, 0, 2)';

                    % if i want to use log feature for better performance
                    train_feature = log(var(X_train_csp, 0, 2))';

                    X_features(i, :) = train_feature;
                    labels(i, :) = cls;
                end

                X_train = [X_train; X_features];
                Y_train = [Y_train; labels];

                % Extract test features
                X_test_csp = W_csp * test_trial;
                % feature = var(X_csp, 0, 2)';

                % if i want to use log feature for better performance
                test_feature = log(var(X_test_csp, 0, 2))';
                X_test = [X_test; test_feature];
                Y_test = [Y_test; cls];

            end
        end
```

در اینجا الگوریتم LOO و CSP برای جداسازی داده ها و استخراج ویژگی های آن ها به پایان می رسد. ادامه ی کد آموزش مدل و ارزیابی آن است.

```
    % For better performance. Can be commented.
    [X_train, mu, sigma] = zscore(X_train);
    X_test = (X_test - mu) ./ sigma;

    % Train cascade LDA
    cascade_model = train_cascade_LDA(X_train, Y_train);

    % Predict using cascade
    train_preds = predict_cascade_LDA(cascade_model, X_train);
    test_preds  = predict_cascade_LDA(cascade_model, X_test);


    train_cm = confusionmat(Y_train, train_preds, 'Order', 1:4);
    test_cm  = confusionmat(Y_test,  test_preds,  'Order', 1:4);

    train_confmat = train_confmat + train_cm;
    test_confmat  = test_confmat + test_cm;
end
```

```
    % Accuracy
    acc_train = sum(diag(train_confmat)) / sum(train_confmat(:));
    acc_test  = sum(diag(test_confmat))  / sum(test_confmat(:));

    fprintf('Subject %d\n', subj);
    fprintf('Train Accuracy: %.2f%%\n', acc_train * 100);
    fprintf('Test  Accuracy: %.2f%%\n', acc_test * 100);

    save(fullfile('../models/one model per subj', sprintf('cascade_model_subj_%d.mat', subj)), 'cascade_model');

    fig = figure('Name', sprintf('Confusion Matrices - Subject %d', subj), 'NumberTitle', 'off');
    % Train
    subplot(1,2,1);
    confusionchart(train_confmat, 1:4, 'Title', 'Train Confusion Matrix subject', ...
                   'RowSummary','row-normalized', 'ColumnSummary','column-normalized');

    % Test
    subplot(1,2,2);
    confusionchart(test_confmat, 1:4, 'Title', 'Test Confusion Matrix subject ', ...
                   'RowSummary','row-normalized', 'ColumnSummary','column-normalized');

    saveas(fig, fullfile('../confusion matrix/cm model per subj', sprintf('subj%d.fig',subj)));
    saveas(fig, fullfile('../confusion matrix/cm model per subj', sprintf('subj%d.jpg',subj)));
end
```

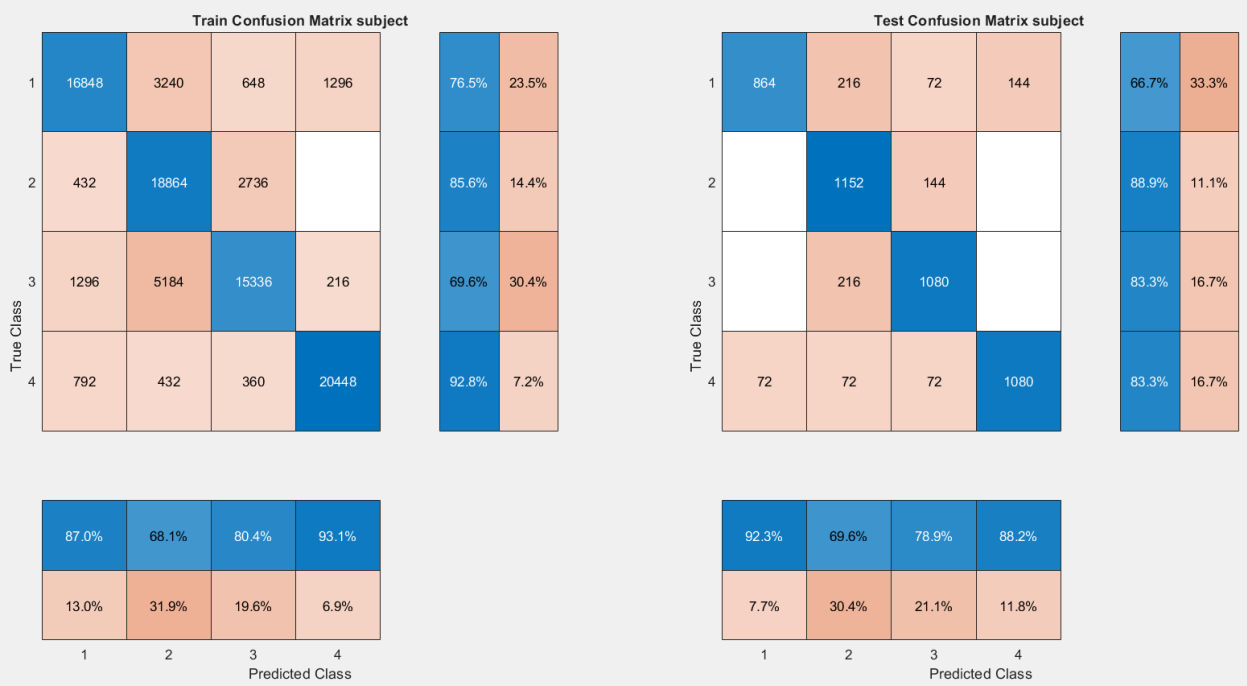
نتایج آموزش

ماتریس کانفیوژن اموزش و تست هر شخص به صورت زیر است.

Subject ۱

Train Accuracy: ۸۱.۱۳٪

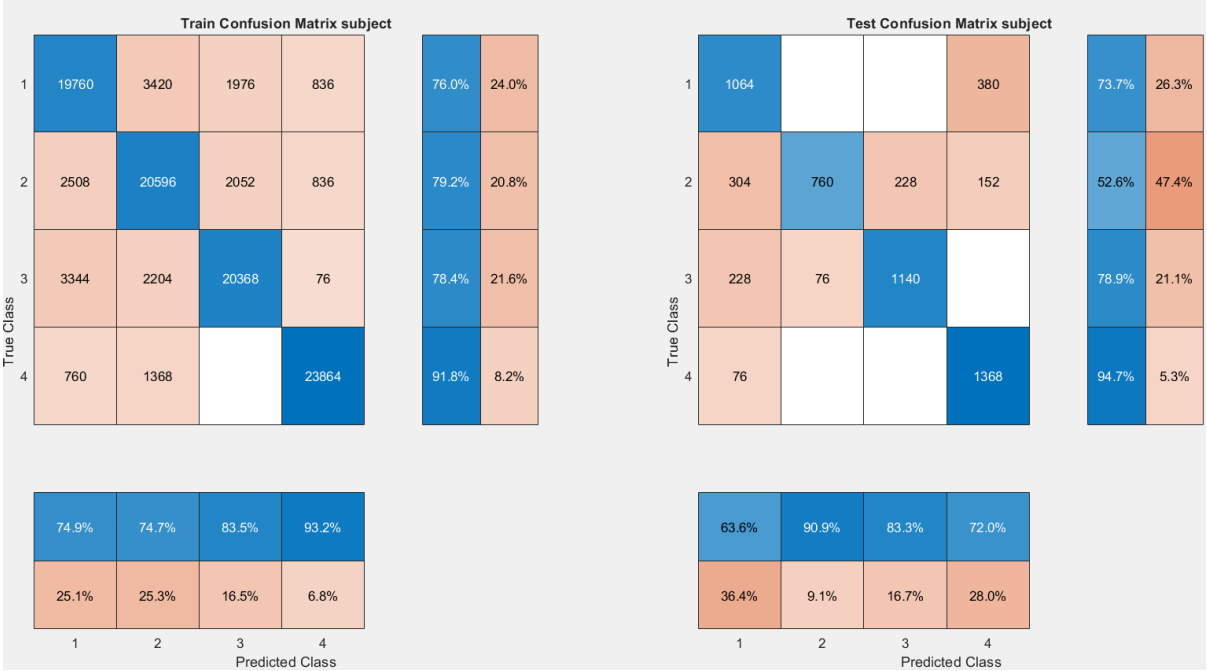
Test Accuracy: ۸۰.۵۶٪



Subject ۲

Train Accuracy: ۸۱,۳٦٪

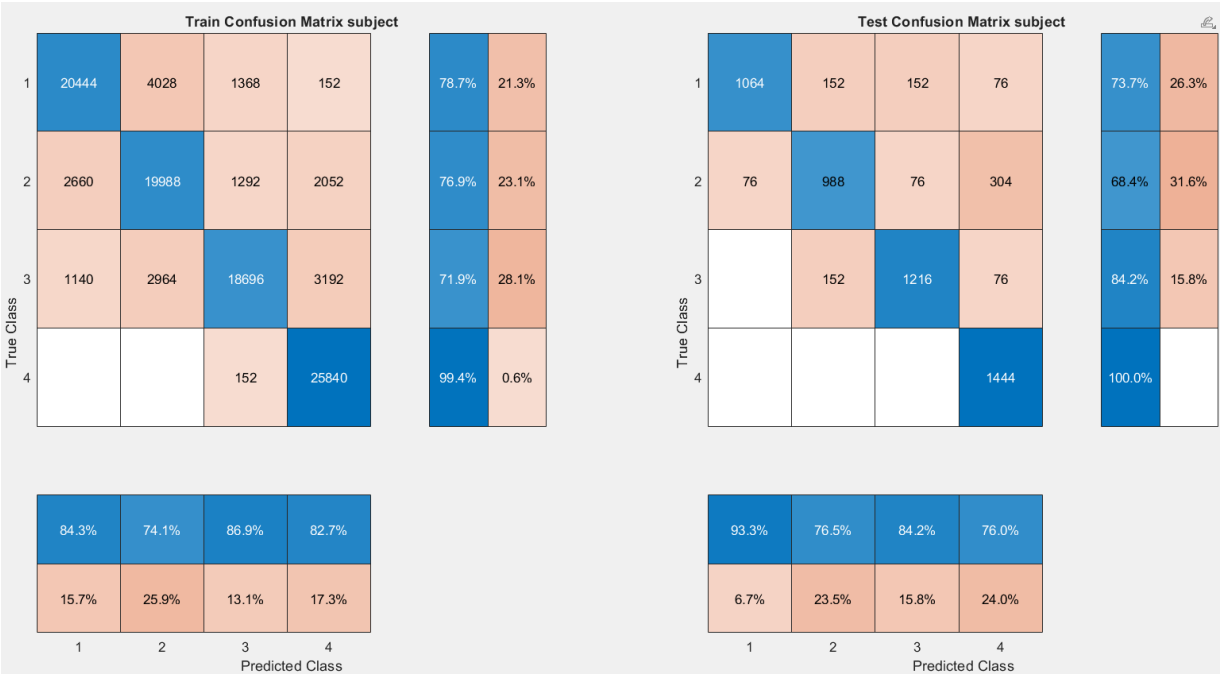
Test Accuracy: ۷۵,۰۰٪



Subject ۳

Train Accuracy: ۸۱,۷۳٪

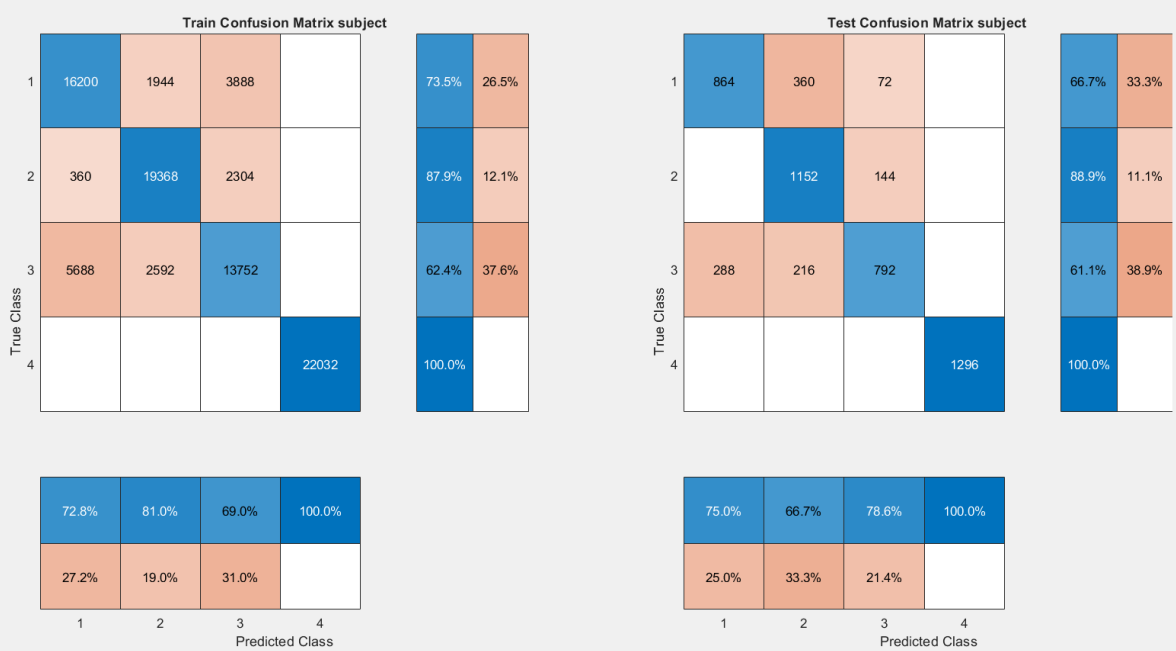
Test Accuracy: ۸۱,۵۸٪



Subject ٤

Train Accuracy: ٨٠,٩٦٪

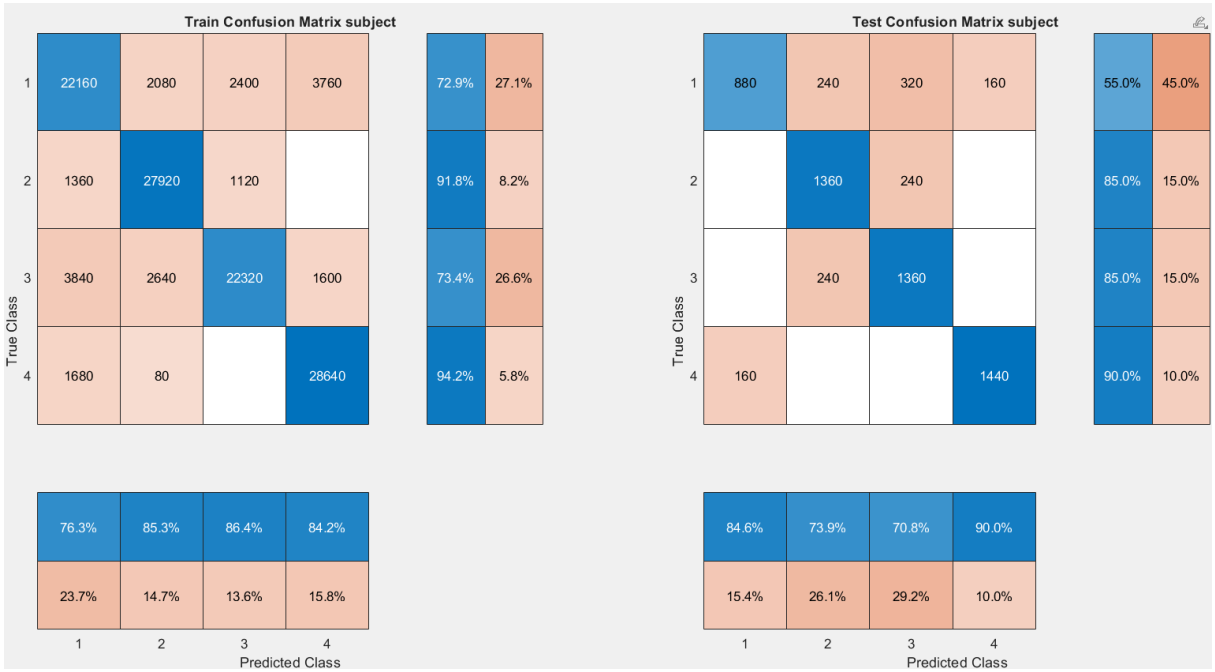
Test Accuracy: ٧٩,١٧٪



Subject ٥

Train Accuracy: ٨٣.٠٩٪

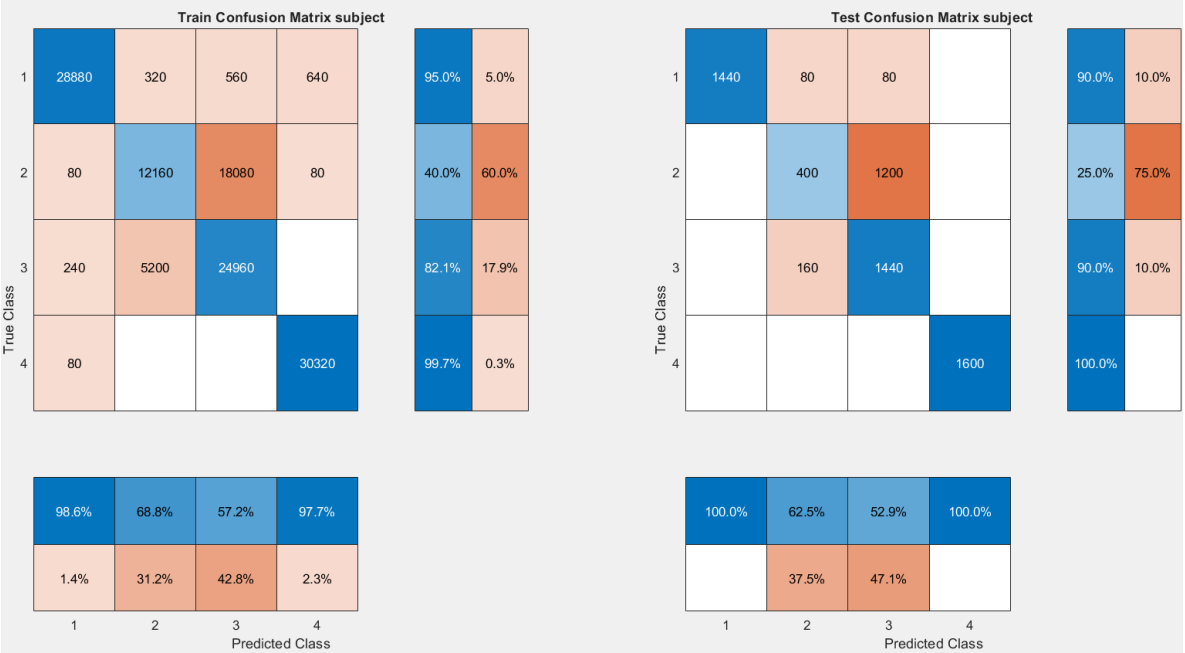
Test Accuracy: ٧٨.٧٥٪



Subject ۶

Train Accuracy: ۷۹.۲۱٪

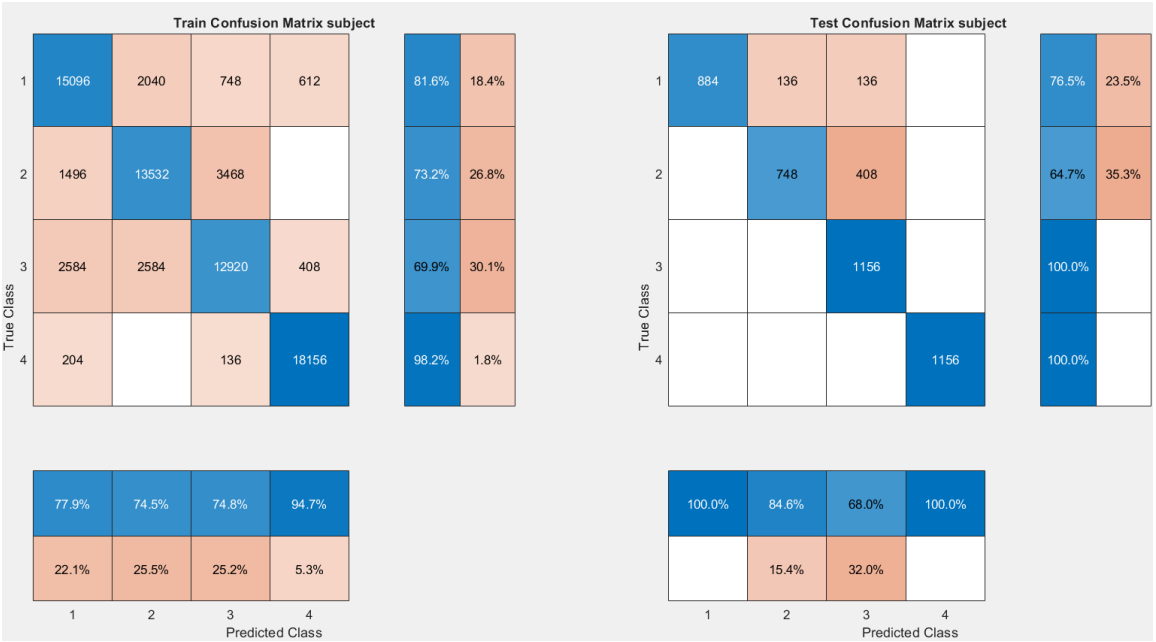
Test Accuracy: ۷۶.۲۵٪



Subject ۷

Train Accuracy: ۸۰.۷۰٪

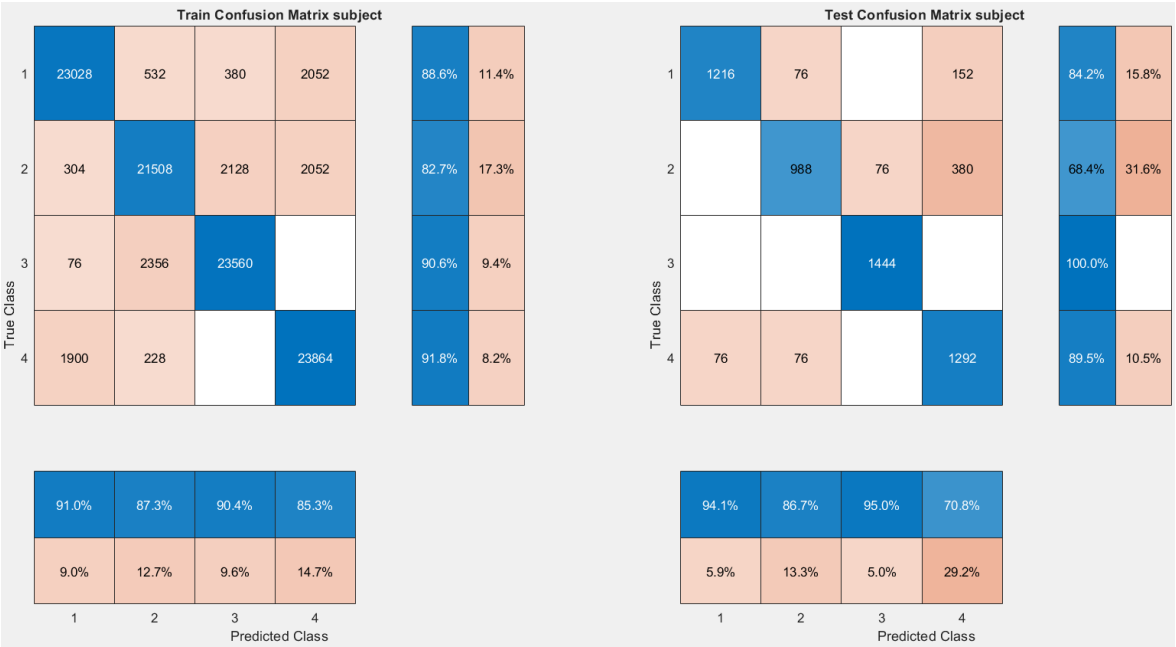
Test Accuracy: ۸۵.۲۹٪



Subject ۸

Train Accuracy: ۸۸.۴۵٪

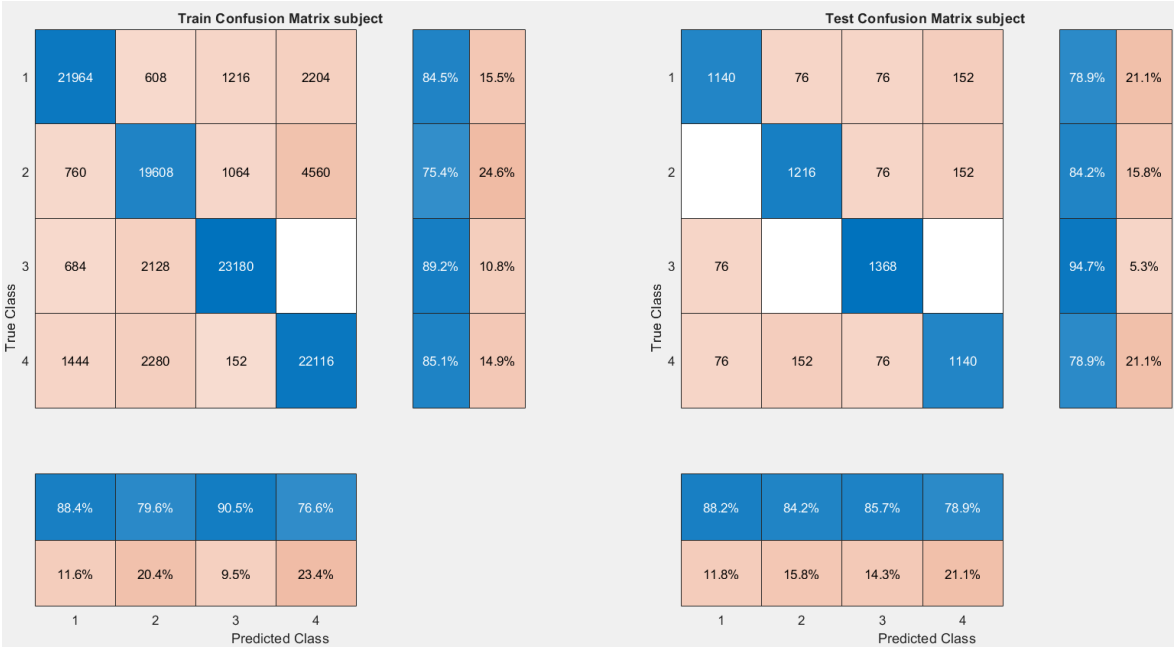
Test Accuracy: ۸۵.۵۳٪



Subject ۹

Train Accuracy: ۸۳.۵۵٪

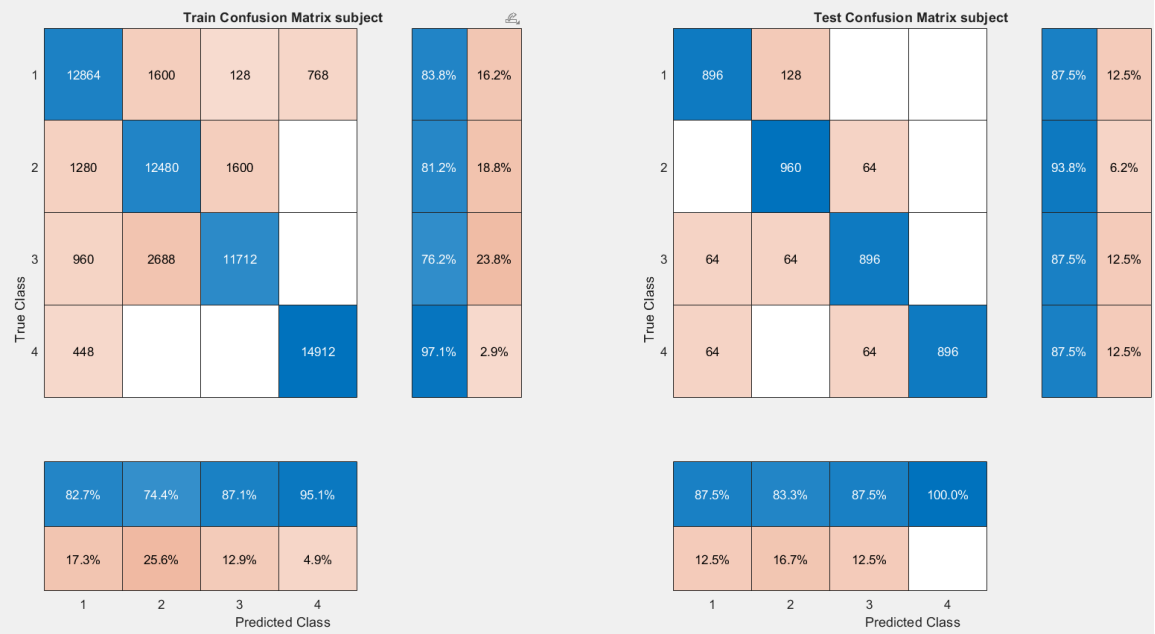
Test Accuracy: ۸۴.۲۱٪



Subject ۱۰

Train Accuracy: ۸۴.۵۸٪

Test Accuracy: ۸۹.۰۶٪



Subject ۱۱

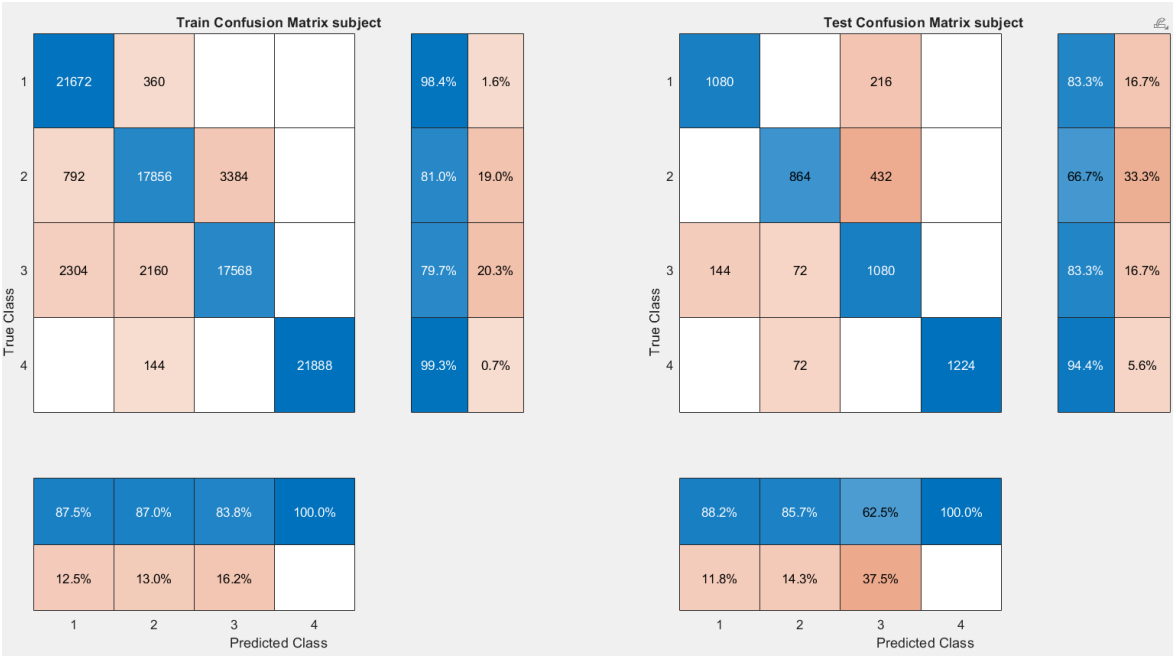
Train Accuracy: ۷۷.۴۸٪

Test Accuracy: ۸۰.۸۸٪

Subject ۱۲

Train Accuracy: ۸۹.۶۲٪

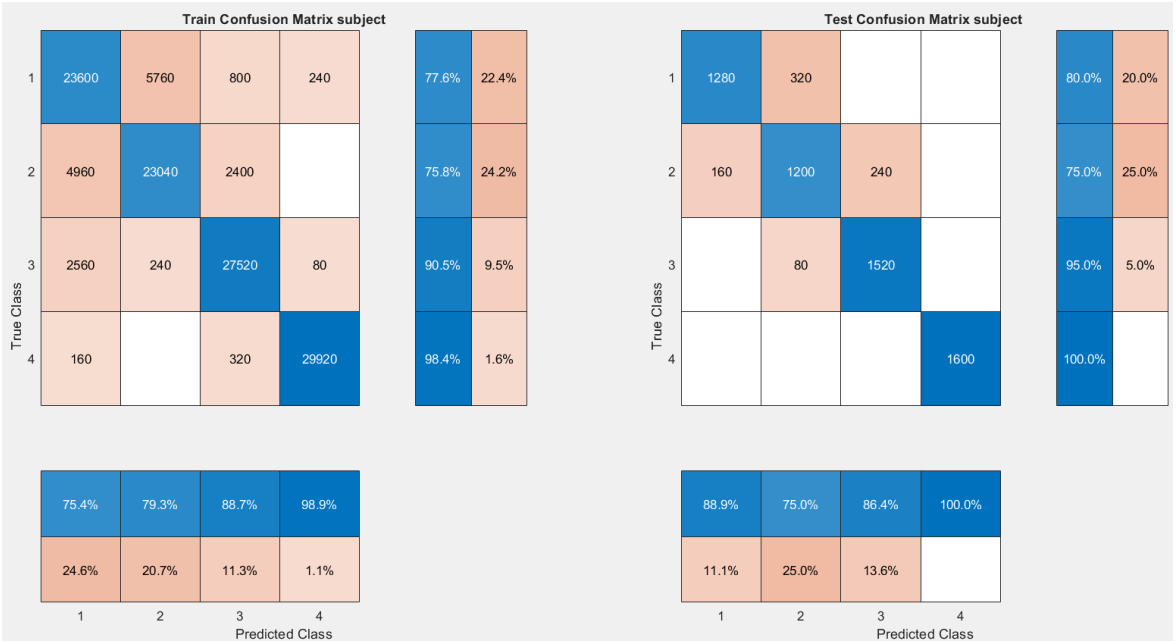
Test Accuracy: ۸۱.۹۴٪



Subject ۱۳

Train Accuracy: ۸۵.۵۹٪

Test Accuracy: ۸۷.۵۰٪



Subject ۱۴

Train Accuracy: ۸۳.۷۷٪

Test Accuracy: ۸۴.۲۱٪

Subject ۱۵

Train Accuracy: ۸۰.۴۱٪

Test Accuracy: ۸۲.۸۹٪

۲. آموزش یک مدل برای تمام subject ها

مطابق کدی که در فایل train_model_all_subj.m در پوشه ی train and evaluate نوشتم، در این رویکرد داده های تمام subject ها را برای آموزش فقط یک مدل با سه طبقه بند به صورت آبشاری استفاده می کنم. نهایتا ماتریس کانفیوژن آن در مسیر confusion matrix/cm model all subj و مدل آن آموزش دیده در مسیر model/one model for all subj ذخیره می شوند.

کد این بخش شبیه کد رویکرد قبل است تنها تفاوت آن در این است که داده های هر subj در به ازای هر کلاس در یک cell جدید ریخته می شوند و تمامی مراحل از جمله LOO، CSP و LDA روی این داده های جمع شده اجرا می شود.

```
clc;
clear;
close all;

num_subjects = 15;
num_filters = 10;

all_trials = cell(4, 1);

for subj = 1 : num_subjects
    load(sprintf('../preprocess/processed_dataset/preprocessed_subj_%d.mat', subj));
    for cls = 1:4
        all_trials{cls} = cat(3, all_trials{cls}, X{cls});
    end
end

num_trials_total = size(all_trials{1}, 3);

train_confmat = zeros(4,4);
test_confmat = zeros(4,4);

for i = 1 : 100
    X_train = [];
    Y_train = [];
    X_test  = [];
    Y_test = [];

    for cls = 4 : -1 : 1
        pos_trials = all_trials{cls};
        test_trial = pos_trials(:,:,i);
        pos_trials(:,:,i) = [];
        % Data preparation for cascade CSP
        if cls == 4
            neg_trials = cat(3, all_trials{1}, all_trials{2}, all_trials{3});
            neg_trials(:,:,i) = [];
        elseif cls == 3
            neg_trials = cat(3, all_trials{1}, all_trials{2});
            neg_trials(:,:,i) = [];
        elseif cls == 2
            neg_trials = all_trials{1};
            neg_trials(:,:,i) = [];
        else
            neg_trials = all_trials{2};
            neg_trials(:,:,i) = [];
        end
```

```
        pos_reshaped = reshape(pos_trials, size(pos_trials,1), []);
        neg_reshaped = reshape(neg_trials, size(neg_trials,1), []);

        [W_csp] = CSP(pos_reshaped, neg_reshaped, num_filters);

        % train features
        num_train_trials = size(pos_trials,3);
        train_features = zeros(num_train_trials, num_filters);

        for j = 1:num_train_trials
            Xf = W_csp * pos_trials(:,:,j);
            train_features(j,:) = log(var(Xf, 0, 2))';
        end
        X_train = [X_train; train_features];
        Y_train = [Y_train; cls * ones(num_train_trials,1)];

        % Extract test features
        Xf_test = W_csp * test_trial;
        test_feature = log(var(Xf_test, 0, 2))';
        X_test = [X_test; test_feature];
        Y_test = [Y_test; cls];
    end

    % For better performance. Can be commented.
    [X_train, mu, sigma] = zscore(X_train);
    X_test = (X_test - mu) ./ sigma;

    % Train cascade LDA
    model = train_cascade_LDA(X_train, Y_train);

    % Predict using cascade
    Y_train_pred = predict_cascade_LDA(model, X_train);
    Y_test_pred  = predict_cascade_LDA(model, X_test);

    train_cm = confusionmat(Y_train, Y_train_pred, 'Order', 1:4);
    test_cm  = confusionmat(Y_test,  Y_test_pred,  'Order', 1:4);

    train_confmat = train_confmat + train_cm;
    test_confmat  = test_confmat + test_cm;
end
```

ادامه کد در تصویر بعدی.

```
% Accuracy
acc_test  = sum(diag(test_confmat)) / sum(test_confmat(:));
class_acc = diag(test_confmat) ./ sum(test_confmat, 2);
acc_train = sum(diag(train_confmat)) / sum(train_confmat(:));

fprintf('Accuracy of one model for all subjects\n');
fprintf('Train Accuracy: %.2f%%\n', acc_train * 100);
fprintf('Test  Accuracy: %.2f%%\n', acc_test * 100);
for c = 1:4
    fprintf('Class %d Accuracy: %.2f%%\n', c, class_acc(c) * 100);
end

fig = figure('Name', 'All Subjects Confusion Matrices', 'NumberTitle', 'off');

% Train
subplot(1,2,1);
confusionchart(train_confmat, 1:4, 'Title', 'Train Confusion Matrix subject', ...
               'RowSummary','row-normalized', 'ColumnSummary','column-normalized');

% Test
subplot(1,2,2);
confusionchart(test_confmat, 1:4, 'Title', 'Test Confusion Matrix - All Subjects', ...
               'RowSummary','row-normalized', 'ColumnSummary','column-normalized');


saveas(fig, fullfile('../confusion matrix/cm model all subj', 'model_all_subj.fig'));
saveas(fig, fullfile('../confusion matrix/cm model all subj', 'model_all_subj.jpg'));

save('../models/one model for all subj/one_model_all_subjects.mat', 'model');
```

نتایج آموزش

ماتریس کانفیوژن آموزش و تست و همچنین دقت مدل روی داده های تست برای هر کلاس به شرح زیر است.

Accuracy of one model for all subjects

Train Accuracy: ۷۴.۶۹٪

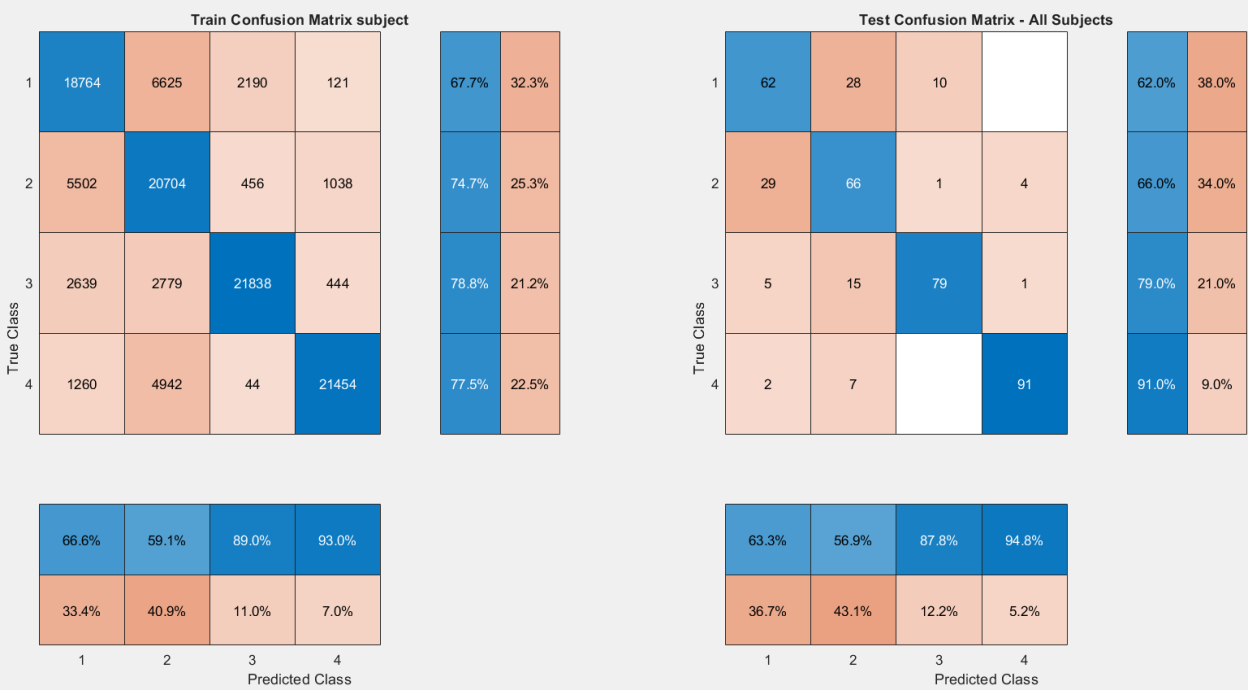
Test Accuracy: ۷۴.۵۰٪

Class ۱ Accuracy: ۶۲.۰۰٪

Class ۲ Accuracy: ۶۶.۰۰٪

Class ۳ Accuracy: ۷۹.۰۰٪

Class ۴ Accuracy: ۹۱.۰۰٪



همان طور که می بینیم دقت مدل برای هر کلاس نزدیک آن اعدادی شد که طبق فایل Report.pdf انتظار می رفت.

مراجع

[١] Motor cortex - Wikipedia

[٢] Delta wave - Wikipedia

[٣] Theta wave - Wikipedia

[٤] Alpha wave - Wikipedia

[٥] Beta wave - Wikipedia

[٦] Hyperparameter optimization - Wikipedia

[٧] Cascading classifiers - Wikipedia