Spatial Filtering of EEG data

Introduction

Common Spatial Pattern (CSP) is a signal processing technique that uses spatial filtering, and is commonly used in EEG. The goal of CSP is to find a set of spatial filters that can effectively differentiate between two classes of signals based on their covariance matrices. By solving a generalized eigenvalue problem, CSP enhances the signal variance of one class while reducing it for the other class, which allows us to separate the two classes as they become more easily discriminable.

For this experiment, we differentiate between right and left activities. The subject has a 16-electrodes cap placed on their hand, and is placed in front of a computer. For 8 trials separated in 2 different sessions, the subject is asked to imaginate squeezing an object (such as a tennis ball) into their right or left hand, in various orders. The EEG data is then recorded, filtered, and analyzed as to separate the trials between left and right.

The key objective of this experiment is to differentiate the distinct spatial patterns. We are more specifically analyzing the Mu waves, which are between 7.5 and 12.5Hz, and reflect the synchronous firing of motor neurons in rest state. When performing or imagining a movement, they become desynchronized. The differentiation is based on the topographical organization of the brain, where specific locations are associated with corresponding body parts. For instance, the right and left motor cortex correspond to the left hand and right hand, respectively.

Stimuli and Procedure

The sampling frequency used in this experiment is 256Hz, we have 8 runs each separated in 80 different trials, 40 left imagined movements and 40 right imagined movements. We verified the signal quality during the experinment by looking at the scope of the data during various artifacts performed by the subject (blinking, tensing their neck).

We start by importing the data and separating between left and right trials. The 18th row of the data represents 2 seconds after the start of each trial. For each onset, we place the data in a left or right array depending on the informations of the session.

Filtering plays an important role in the quality of the CSP filters. We tried different ways of filtering: before separating the trials, after separating them, using different bandpass frequencies. We found that the LDA accuracy was the best when we filtered after separation, and when using a bandpass filter more narrow than what was recommended previously, of 8-20 Hz. This filtering method is too broad and does not focus on the mu waves, which results in a less effective discrimination between left and right trials. Instead, we used a more narrow 9-15 Hz bandpass filter.

```
% Remove the first 10s for settling

fs = 256;

data1 = load('run1.mat').y;
eeg1_unscaled_0 = reshape(data1,size(data1,1),size(data1,3));
eeg1_unscaled = transpose(zscore(transpose(eeg1_unscaled_0(:,fs*10:end))));
```

```
data2 = load('run2.mat').y;
eeg2_unscaled_0 = reshape(data2,size(data2,1),size(data2,3));
eeg2_unscaled = transpose(zscore(transpose(eeg2_unscaled_0(:,fs*10:end))));

data3 = load('run3.mat').y;
eeg3_unscaled_0 = reshape(data3,size(data3,1),size(data3,3));
eeg3_unscaled = transpose(zscore(transpose(eeg3_unscaled_0(:,fs*10:end))));

data4 = load('run1.mat').y;
eeg4_unscaled_0 = reshape(data4,size(data4,1),size(data4,3));
eeg4_unscaled = transpose(zscore(transpose(eeg4_unscaled_0(:,fs*10:end))));

data5 = load('run5.mat').run5;
eeg5_unscaled_0 = reshape(data5,size(data5,1),size(data5,3));
eeg5_unscaled = transpose(zscore(transpose(eeg5_unscaled_0(:,fs*10:end))));

data6 = load('run6.mat').run6;
eeg6_unscaled_0 = reshape(data6,size(data6,1),size(data6,3));
eeg6_unscaled = transpose(zscore(transpose(eeg6_unscaled_0(:,fs*10:end))));

data7 = load('run7.mat').run7;
eeg7_unscaled_0 = reshape(data7,size(data7,1),size(data7,3));
eeg7_unscaled = transpose(zscore(transpose(eeg7_unscaled_0(:,fs*10:end))));

data8 = load('run8.mat').run8;
eeg8_unscaled_0 = reshape(data8,size(data8,1),size(data8,3));
eeg8_unscaled = transpose(zscore(transpose(eeg8_unscaled_0(:,fs*10:end))));
```

```
z1 = load('classrun1.mat').z1;
z2 = load('classrun2.mat').z2;
z3 = load('classrun3.mat').z3;
z4 = load('classrun4.mat').z4;
z5 = load('classrun5.mat').z1;
z6 = load('classrun6.mat').z2;
z7 = load('classrun7.mat').z3;
z8 = load('classrun8.mat').z4;
```

```
% Center and scale the data

mean1 = mean(eeg1_unscaled, 2);
std1 = std(eeg1_unscaled, 0, 2);
eeg1 = (eeg1_unscaled - mean1)./std1;

mean2 = mean(eeg2_unscaled, 2);
std2 = std(eeg2_unscaled, 0, 2);
eeg2 = (eeg2_unscaled - mean2)./std2;

mean3 = mean(eeg1_unscaled, 2);
std3 = std(eeg3_unscaled, 0, 2);
```

```
eeg3 = (eeg3_unscaled - mean3)./std3;

mean4 = mean(eeg4_unscaled, 2);
std4 = std(eeg4_unscaled, 0, 2);
eeg4 = (eeg4_unscaled - mean4)./std4;

mean5 = mean(eeg5_unscaled, 2);
std5 = std(eeg5_unscaled, 0, 2);
eeg5 = (eeg5_unscaled - mean5)./std5;

mean6 = mean(eeg6_unscaled, 2);
std6 = std(eeg6_unscaled, 0, 2);
eeg6 = (eeg6_unscaled - mean6)./std6;

mean7 = mean(eeg7_unscaled, 2);
std7 = std(eeg7_unscaled, 0, 2);
eeg7 = (eeg7_unscaled - mean7)./std7;

mean8 = mean(eeg8_unscaled, 2);
std8 = std(eeg8_unscaled, 0, 2);
eeg8 = (eeg8_unscaled - mean8)./std8;
```

```
run1 = 5*tanh(eeg1 / 5);
run2 = 5*tanh(eeg2 / 5);
run3 = 5*tanh(eeg3 / 5);
run4 = 5*tanh(eeg4 / 5);
run5 = 5*tanh(eeg5 / 5);
run6 = 5*tanh(eeg6 / 5);
run7 = 5*tanh(eeg7 / 5);
run8 = 5*tanh(eeg8 / 5);
```

```
% The 1st and 18th channel don't need filtering, so we revert them back to
% the original

run1(1,:) = eeg1_unscaled_0(1,fs*10:end);
run1(18,:) = eeg1_unscaled_0(18,fs*10:end);
run2(1,:) = eeg2_unscaled_0(1,fs*10:end);
run2(18,:) = eeg2_unscaled_0(18,fs*10:end);
run3(1,:) = eeg3_unscaled_0(1,fs*10:end);
run3(18,:) = eeg3_unscaled_0(18,fs*10:end);
run4(1,:) = eeg4_unscaled_0(1,fs*10:end);
run4(18,:) = eeg4_unscaled_0(18,fs*10:end);
run5(1,:) = eeg5_unscaled_0(1,fs*10:end);
run5(18,:) = eeg5_unscaled_0(18,fs*10:end);
run6(1,:) = eeg6_unscaled_0(1,fs*10:end);
run6(18,:) = eeg6_unscaled_0(18,fs*10:end);
run7(1,:) = eeg7_unscaled_0(1,fs*10:end);
run7(18,:) = eeg7_unscaled_0(18,fs*10:end);
```

```
run8(1,:) = eeg8_unscaled_0(1,fs*10:end);
run8(18,:) = eeg8_unscaled_0(18,fs*10:end);
```

```
% Separate the data between left and right trials

runs = {run1, run2, run3, run4,run5, run6, run7, run8};
directions = {z1, z2, z3, z4, z5, z6, z7, z8};
fs = 256;
left_raw = {};
right_raw = {};
count_left = 0;
count_right = 0;
for i = 1:8
    run = runs{i};
onset = [];

    for col = 2:size(run, 2)
    % Find the timepoints where 18 marks 2 seconds of a new trial
        if run(18, col) == 1 && run(18, col - 1) ~= 1
            onset = [onset, col];
        end
    end

    for j = 1:size(onset,2)
        time = onset(j);
        if j == 40
            finish = size(run,2);
        else finish = onset(j+1) - 2*fs;
        end

        direction = directions{i}(j);
        if direction == 1
            count_left = count_left + 1;

            %take data between 4.5 and 8s

            left_raw{count_left} = run(2:17,time+2.5*fs:time+6*fs);
            else
            count_right = count_right + 1;
            right_raw{count_right} = run(2:17,time+2.5*fs:time+6*fs);
        end
    end
end
```

```
% Filter the data

leftcat = cat(2, left_raw{:});
leftcat = eegfilt(leftcat, 256, 9, 15);
```

```
Warning: Using firls to estimate filter coefficients. We recommend that you use fir1 instead, which
yields larger attenuation. In future, fir1 will be used by default!
eegfilt() - performing 84-point bandpass filtering.
            If a message, 'Matrix is close to singular or badly scaled,' appears,
            then Matlab has failed to design a good filter. As a workaround,
            for band-pass filtering, first highpass the data, then lowpass it.
eegfilt() - low transition band width is 1 Hz; high trans. band width, 2 Hz.
................
```

```
rightcat = cat(2, right_raw{:});
rightcat = eegfilt(rightcat, 256, 9, 15);
```

```
Warning: Using firls to estimate filter coefficients. We recommend that you use fir1 instead, which
yields larger attenuation. In future, fir1 will be used by default!
eegfilt() - performing 84-point bandpass filtering.
            If a message, 'Matrix is close to singular or badly scaled,' appears,
            then Matlab has failed to design a good filter. As a workaround,
            for band-pass filtering, first highpass the data, then lowpass it.
eegfilt() - low transition band width is 1 Hz; high trans. band width, 2 Hz.
................
```

```
submatrix_size = [16, 897];
num_repeats_row = size(leftcat, 1) / submatrix_size(1);
num_repeats_col = size(leftcat, 2) / submatrix_size(2);


left = mat2cell(leftcat, repmat(submatrix_size(1),
1, floor(num_repeats_row)), repmat(submatrix_size(2), 1,
floor(num_repeats_col)));
right = mat2cell(rightcat, repmat(submatrix_size(1),
1, floor(num_repeats_row)), repmat(submatrix_size(2), 1,
floor(num_repeats_col)));
```

```
fprintf(['There are ', num2str(count_left), ' left trials and ',
num2str(count_right), ' right trials.'])
```

```
There are 160 left trials and 160 right trials.
```

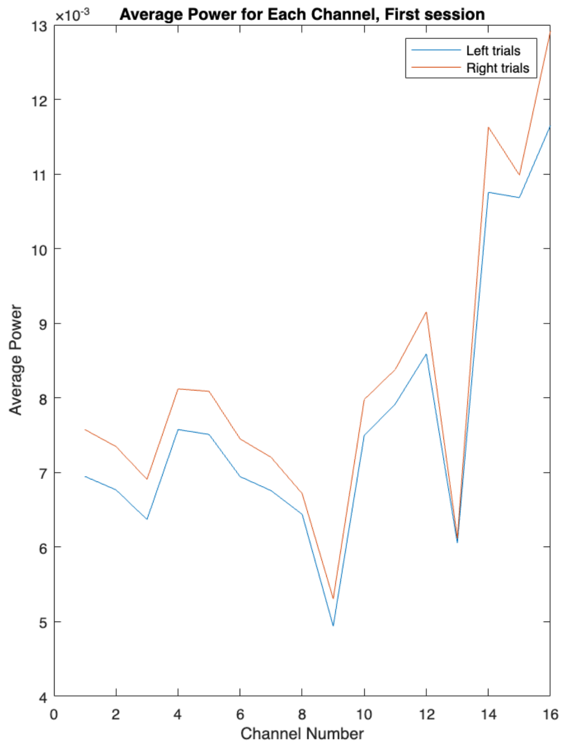
Results

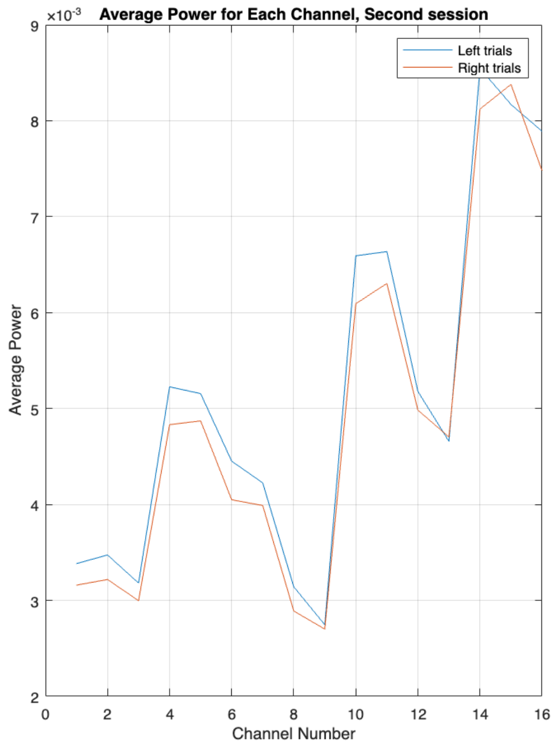
Training Imagined Movement

The subject reported an improvement of the results for session 2 compared to session 1, due to better understanding of the feedback delivered during each trial.

```
eeg_left_1 = cat(3, left{1:80});
eeg_right_1 = cat(3, right{1:80});

freq_band = [9 15];
```

```matlab
avg_power_left1 = zeros(16, 1);
avg_power_right1 = zeros(16, 1);
avg_power_left2 = zeros(16, 1);
avg_power_right2 = zeros(16, 1);

for i = 1:16

    chanleft =eeg_left_1(i, :);
    chanright =eeg_right_1(i, :);
    [pxx_left, f] = pwelch(chanleft, [], [], [], fs);
    [pxx_right, f] = pwelch(chanright, [], [], [], fs);
    freq_index = f >= freq_band(1) & f <= freq_band(2);
    avg_power_left1(i) = mean(pxx_left(freq_index));
    avg_power_right1(i) = mean(pxx_right(freq_index));
end

eeg_left_2 = cat(3, left{81:160});
eeg_right_2 = cat(3, right{81:160});

for i = 1:16

    chanleft =eeg_left_2(i, :);
    chanright =eeg_right_2(i, :);
    [pxx_left, f] = pwelch(chanleft, [], [], [], fs);
    [pxx_right, f] = pwelch(chanright, [], [], [], fs);
    freq_index = f >= freq_band(1) & f <= freq_band(2);
    avg_power_left2(i) = mean(pxx_left(freq_index));
    avg_power_right2(i) = mean(pxx_right(freq_index));
end

figure;
set(gcf,'Position',[100 100 900 500])
subplot(1,2,1)
plot(avg_power_left1);
hold on;
plot(avg_power_right1);
legend('Left trials', 'Right trials')
xlabel('Channel Number');
ylabel('Average Power');
title('Average Power for Each Channel, First session');
subplot(1,2,2)
plot(avg_power_left2);
hold on;
plot(avg_power_right2);
legend('Left trials', 'Right trials')
xlabel('Channel Number');
ylabel('Average Power');
title('Average Power for Each Channel, Second session');
grid on;
```

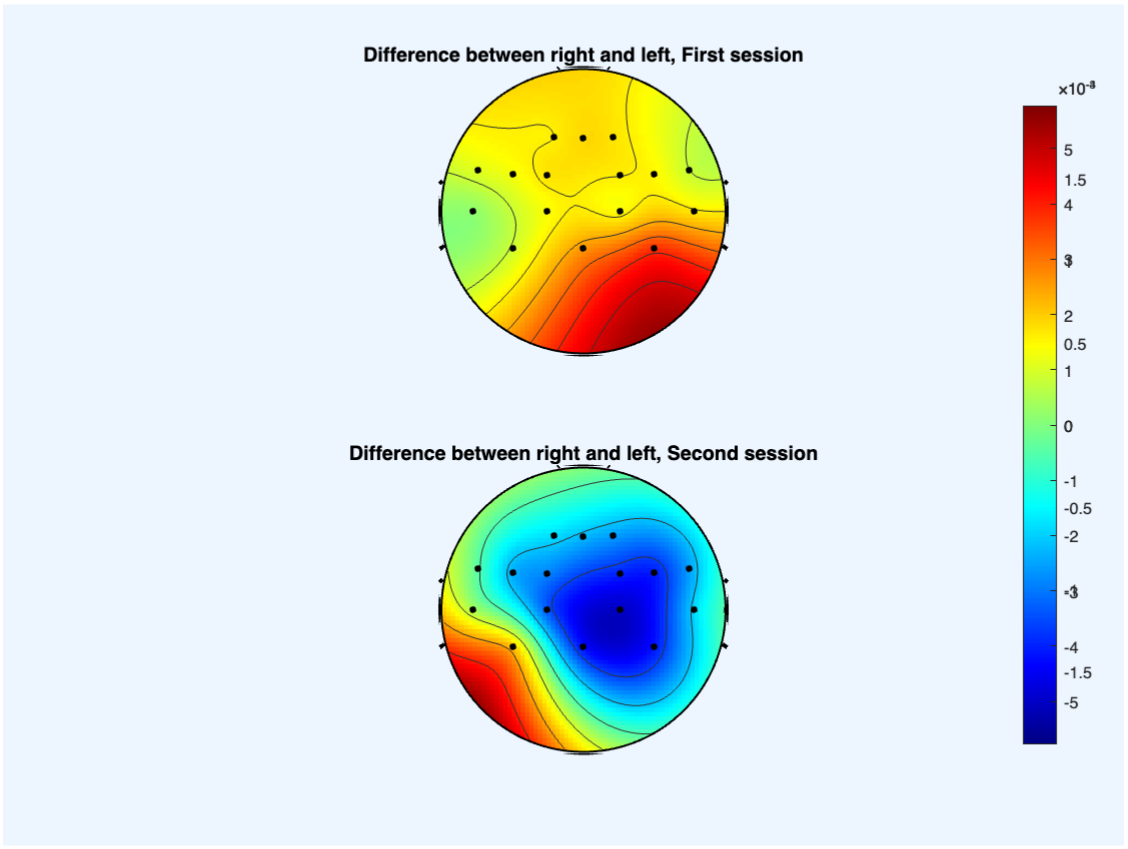




```
diff1 = zeros(16,1);
diff2 = zeros(16,1);
for i = 1:16
    diff1(i) = avg_power_right1(i) - avg_power_left1(i);
    diff2(i) = avg_power_right2(i) - avg_power_left2(i);
end
eloc1 = readlocs('CSP1.locs');
```

```
readlocs(): 'loc' format assumed from file extension
```

```
figure();
subplot(2,1,1)
topoplot(diff1, eloc1, 'style', 'both', 'headrad',0.5, 'electrodes','on');
colormap('jet');
title('Difference between right and left, First session');
h = colorbar;
set(h, 'Position', [0.91 0.12 0.03 0.76]);

subplot(2,1,2)

topoplot(diff2, eloc1, 'style', 'both', 'headrad',0.5, 'electrodes','on');
colormap('jet');
title('Difference between right and left, Second session');
h = colorbar;
set(h, 'Position', [0.91 0.12 0.03 0.76]);
```



We can see that the difference for the scalp map is more intense for the second session, indicating that the accuracy must improve over time. A larger number of runs would improve the classification, as the subject gets used to the experimental paradigm and learns to interpret the feedback better.

Applying Common Spatial Pattern Filters

```
% Compute the covariance for each trial, the avg covariance for left and
% right

covariance_left = zeros(16, 16, 160);
covariance_right = zeros(16, 16, 160);


for trial = 1:160

    trial_left = left{trial};
    trial_right = right{trial};

    covariance_left(:, :, trial) = cov(trial_left');
    covariance_right(:, :, trial) = cov(trial_right');
end


avg_cov_left = mean(covariance_left, 3);
```

```
avg_cov_right = mean(covariance_right, 3);
```

```
[W,L] = eig(avg_cov_left , avg_cov_right);   % Mixing matrix W (spatial
filters are columns)
lambda = diag(L)
```

```
lambda = 16×1
    0.4532
    0.4760
    0.4792
    0.4811
    0.4863
    0.4931
    0.4949
    0.4965
    0.5018
    0.5044
      ⋮
```

• We choose the first 6 Eigen vectors (columns of W) that correspond to the largest Eigen values: CSP projections

```
CSP = W(:,end-5:end)'
```

```
CSP = 6×16
    1.8076  -13.1686    3.4640   10.5452    6.0748   -9.4961  -13.2953    9.7783 ···
    6.7362    1.0422  -19.3277  -12.5085   25.7223    1.0434    6.5818    4.7846
   -0.0573    0.1735    7.6830    5.7157  -21.1876   -3.8520    8.3049    2.7805
   -1.2986  -10.5936    9.1579   12.0282    3.3288   -2.3958   -2.8076    5.6539
    4.1282  -11.1975   19.3265   -3.2609    4.0402   -1.1871   -3.0698   -4.2554
   -0.0472    1.9647   -6.7645   -2.5601    1.4069    3.6548    3.2011    5.6346
```

• We project the data (channel x Gme) using the CSP projections (6 x Gme)

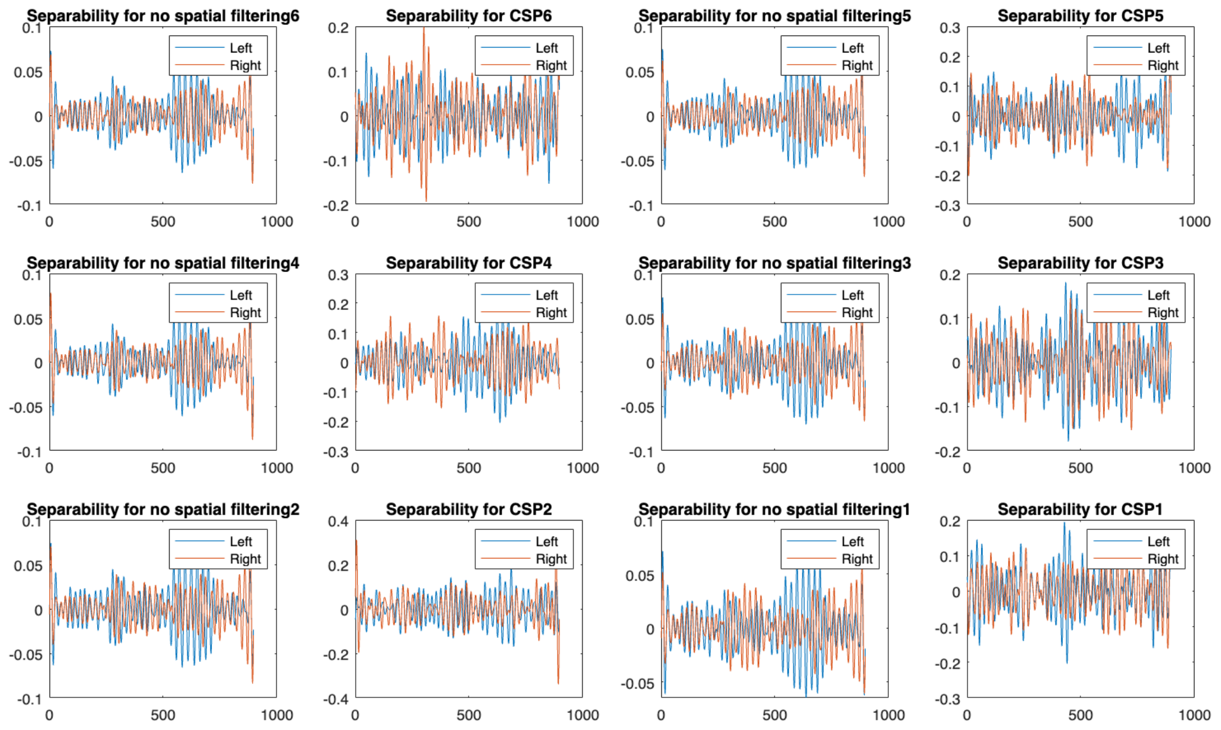
```
reshaped_left_1 = cell2mat(left);
reshaped_left_2 = reshape(reshaped_left_1, 16, 897, 160);

reshaped_right_1 = cell2mat(right);
reshaped_right_2 = reshape(reshaped_right_1, 16, 897, 160);

%figure;
%plot(reshaped_left_1')
```

```
CSP_proj_left = tensorprod(CSP,reshaped_left_2, 2, 1);
CSP_proj_right = tensorprod(CSP,reshaped_right_2, 2, 1);
```

We compare the separability of the two classes before and after CSP filtering.

```
figure;
set(gcf,'Position',[100 100 900 500])
index1 = [1 3 5 7 9 11];
```

```
index2 = [2 4 6 8 10 12];

for filter = 1:6

    subplot(3,4,index1(filter))
    avg_1 = mean(reshaped_left_2(filter,:,:),3);
    avg_2 = mean(reshaped_right_2(filter,:,:),3);


    plot(avg_1);
    hold on;
    plot(avg_2);
    hold off;
    legend('Left', 'Right');
    title(['Separability for no spatial filtering',num2str(7-filter)])
end

for filter = 1:6

    subplot(3,4,index2(filter))
    avg_1 = mean(CSP_proj_left(filter,:,:),3);
    avg_2 = mean(CSP_proj_right(filter,:,:),3);


    plot(avg_1);
    hold on;
    plot(avg_2);
    hold off;
    legend('Left', 'Right');
    title(['Separability for CSP',num2str(7-filter)])
end
```

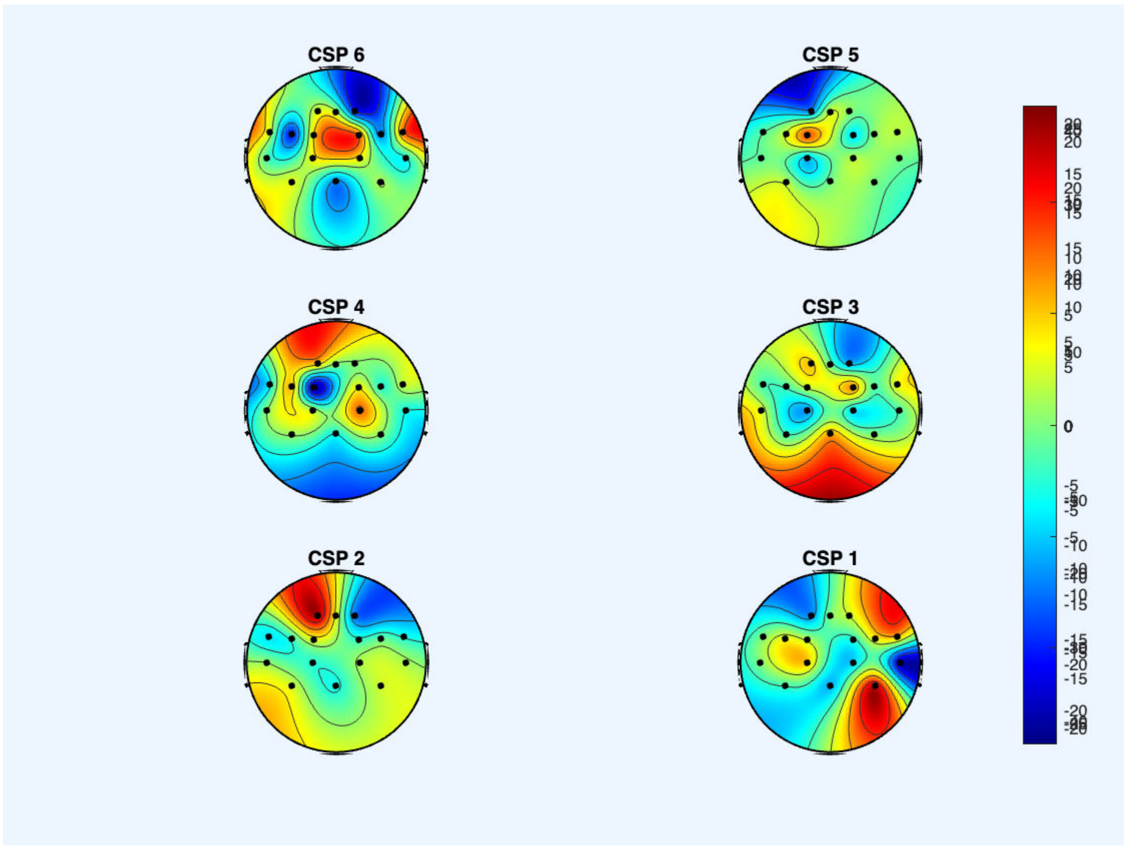


Visually, we cannot really separate appart left and right trials for with or without CSP. The effect of CSP filtering on the data should be to increasing the variance for one class while decreasing it for the other, resulting in better discrimination between the two classes. Looking at the LDA classification accuracy score should give a better indication on the effect of CSP filtering. The results are non-conclusive, we believe that is due to the accuracy of our data itself. During the experiment, when looking at the feedback, the subject often had a wrong result between left and right during the trial.

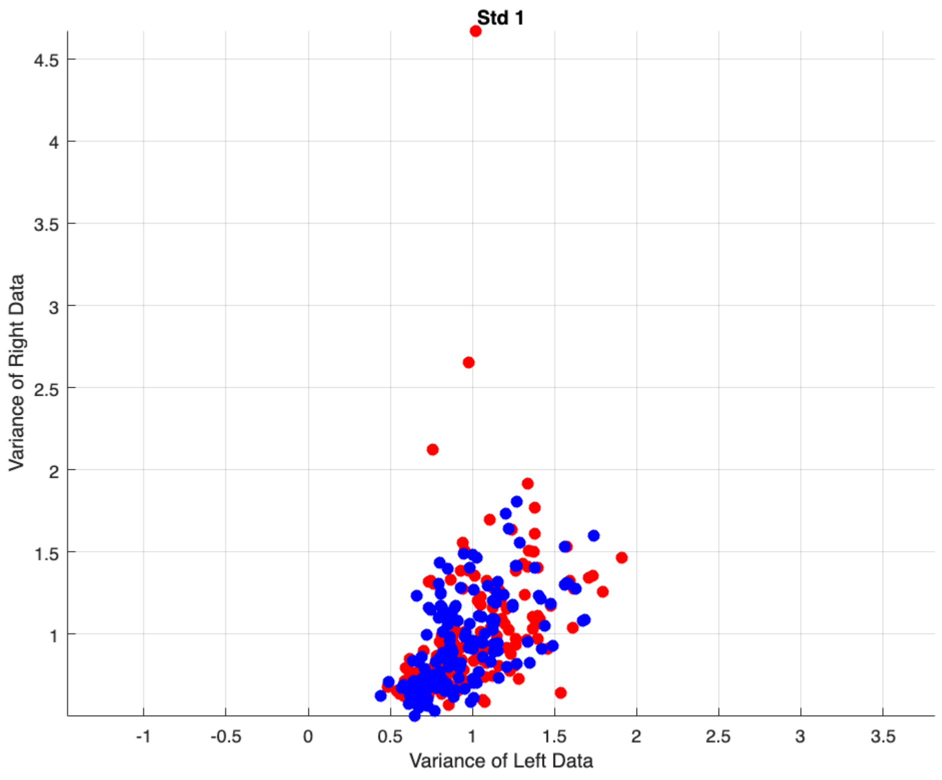
```
eloc1 = readlocs('CSP1.locs');
```

```
readlocs(): 'loc' format assumed from file extension
```

```

figure;
for i = 1:6

    subplot(3, 2, i);
    topoplot(CSP(i,:), eloc1, 'style', 'both',
'headrad',0.5,'electrodes','on');
    colormap('jet');
    title(['CSP ',num2str(7-i)]);

    h = colorbar;
    set(h, 'Position', [0.91 0.12 0.03 0.76]);
end
```



```
std_left = std(CSP_proj_left,0,2);
std_right = std(CSP_proj_right,0,2);
figure();


var_left_1 = std_left(5, :);
var_left_2 = std_left(6, :);
var_right_1 = std_right(5, :);
var_right_2 = std_right(6, :);




scatter(var_left_1, var_left_2, 'r', 'filled');
hold on;

xlabel('Variance of Left Data');
ylabel('Variance of Right Data');
title(['Std ',num2str(7-i)]);

grid on;
axis equal;

scatter(var_right_1, var_right_2, 'b', 'filled');
hold off;
```



The standard deviation is a useful measure to determine the effectiveness of the CSP filtering. As the CSP technique maximises the variance of one class while minimizing that of another class, we would expect to see two orthogonal classes when plotting the standard deviation of the first 2 CSP filters. The results are non-conclusive, we believe that is due to the accuracy of our data itself. During the experiment, when looking at the feedback, the subject often had a wrong result between left and right during the trial.

Classification in CSP-Projected Space

We divide your trials into 90% train data and 10% test data, then use our training set to train a Linear Discriminant Analysis classifier to determine the imagined movement direction for our test set.

```
rng('shuffle');

data = cat(3, CSP_proj_left, CSP_proj_right);
labels = [ones(160, 1); zeros(160, 1)];  % Labels: 1 for CSP_proj_left, 0
for CSP_proj_right


shuffled_indices = randperm(length(labels));
shuffled_data = data(:, :, shuffled_indices);
shuffled_labels = labels(shuffled_indices);

split = round(0.9 * length(shuffled_labels));
```

```
train = shuffled_data(:, :, 1:split);
train_labels = shuffled_labels(1:split);

test = shuffled_data(:, :, split+1:end);
test_labels = shuffled_labels(split+1:end);

disp(['Number of training trials: ' num2str(size(trainData,3))]);
```

```
Number of training trials: 288
```

```
disp(['Number of testing trials: ' num2str(size(testData,3))]);
```

```
Number of testing trials: 32
```

```
% LDA Classifier

ldaClassifier = fitcdiscr(reshape(train, [], size(train, 3))',
train_labels);
predicted_labels = predict(ldaClassifier, reshape(test, [], size(test,
3))');
accuracy = sum(predicted_labels == test_labels) / length(test_labels);
disp(['The percentage of trials correctly classified is  '
num2str(accuracy)]);
```

```
The percentage of trials correctly classified is  0.71875
```

We then repeat this operation ten times with different random train-test splits.

```
accuracies = zeros(1, 10);

for i = 1:10

    data = cat(3, CSP_proj_left, CSP_proj_right);
    labels = [ones(160, 1); zeros(160, 1)];  % Labels: 1 for CSP_proj_left,
0 for CSP_proj_right


    shuffled_indices = randperm(length(labels));
    shuffled_data = data(:, :, shuffled_indices);
    shuffled_labels = labels(shuffled_indices);

    split = round(0.9 * length(shuffled_labels));


    train = shuffled_data(:, :, 1:split);
    train_labels = shuffled_labels(1:split);

    test = shuffled_data(:, :, split+1:end);
    test_labels = shuffled_labels(split+1:end);
    % LDA Classifier
    ldaClassifier = fitcdiscr(reshape(train, [], size(train, 3))',
train_labels);
    predicted_labels = predict(ldaClassifier, reshape(test, [], size(test,
3))');
    accuracy = sum(predicted_labels == test_labels) / length(test_labels);


    accuracies(i) = accuracy;
    disp(['The percentage of trials correctly classified is  '
num2str(accuracy)]);
end
```

```
The percentage of trials correctly classified is  0.625
The percentage of trials correctly classified is  0.53125
The percentage of trials correctly classified is  0.65625
The percentage of trials correctly classified is  0.625
The percentage of trials correctly classified is  0.625
The percentage of trials correctly classified is  0.625
The percentage of trials correctly classified is  0.53125
The percentage of trials correctly classified is  0.71875
The percentage of trials correctly classified is  0.53125
The percentage of trials correctly classified is  0.59375
```

```
% Standard deviation

stds = std(accuracies);
std_final = stds / sqrt(10);


disp(['Standard Error: ' num2str(std_final)]);
```

```
Standard Error: 0.01932
```

The obtained classification is better than chance (>50%), but the results are not highly accurate, with the accuracy oscillating between 53% and 71% depending on the train-test accuracy. The results are non-conclusive, we believe that is due to the accuracy of our data itself. During the experiment, when looking at the feedback, the subject often had a wrong result between left and right during the trial. We believe that re-doing the experiment with a higher number of sessions and a better feedback accuracy for each trial would lead to better results on the class discrimination.