Inbar Tivon

BE521 Brain Computer Interfaces

Team Members: None

Final Report

Due: Wednesday, May 6th, 12:00 PM

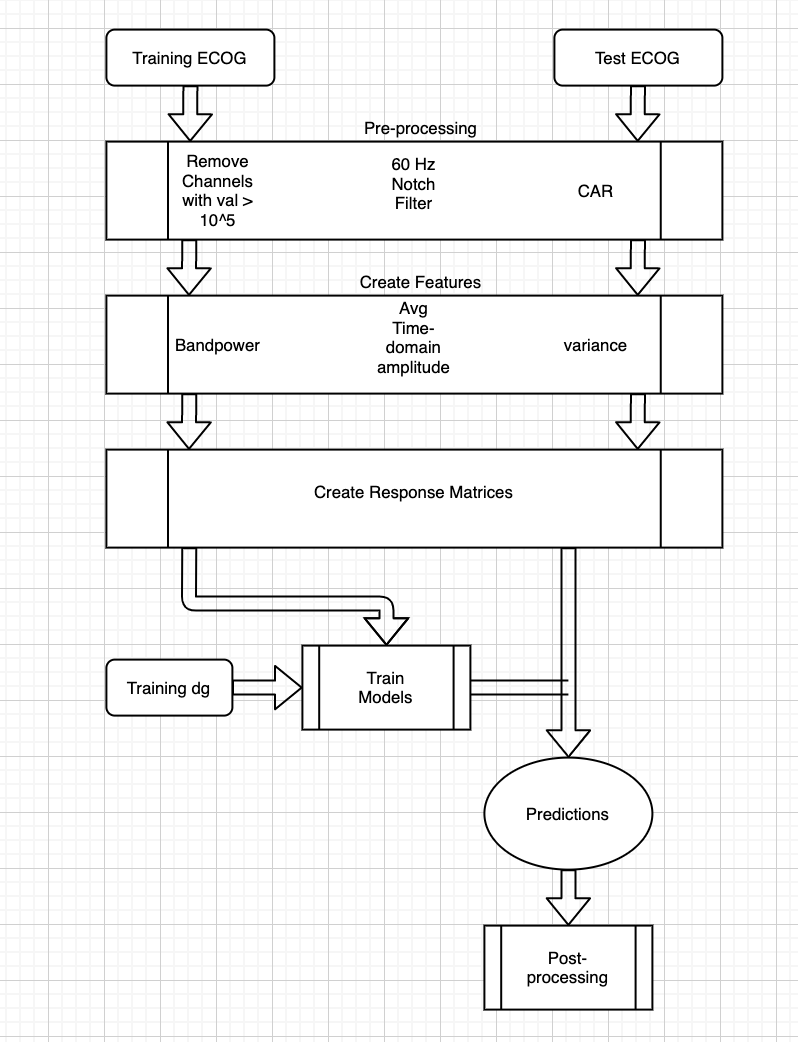
The final project of the BE521 course involved predicting finger flexion based on intracranial EEG (ECOG) in three human subjects (62, 48, and 64 channels, respectively). The Electrocorticographic signals were band pass filtered between 0.15 to 200 Hz, and sampled at 1000 Hz before being provided. The project originated from the 4th BCI Competition. My final algorithm included preprocessing of ECOG data, such as removing irrelevant channels, filtering, and applying a common average reference. The clean data was then used to calculate features in moving time windows with 80ms length and 40ms displacement. A machine learning model was created for each finger of each subject by training on 80% of, and testing on 20% of, given ECOG and finger flexion data. Finally, the models were trained using all 300 seconds of given data, post-processed, and then used to predict unknown testing data.

First, the ECOG data was pre-processed to remove channel 55 from subject 1 and channels 21 and 38 from subject 2 because they had values greater than 10^5. The next step was to get the features for each subject across all windows of length 80ms and 40ms displacement. The windows were right-aligned. Before calculating the features, the data was filtered. I designed a second-order IIR notch digital filter with the notch at frequency 60Hz and a bandwidth of 1.7143 Hz at the -3 dB level to remove power line noise. I used the MATLAB function filtfilt() to filter the data in the forward direction as well and reversed in order to avoid phase distortion. I then applied a common average reference by subtracting the average signal across all channels from each channel. The goal of this was to remove any shared artifacts across all channels of the intracranial data. The clean data was then used to calculate the windowed features. The features included getting the average bandpower in the following frequency intervals: 5-15Hz, 20-25Hz, 75-115Hz, 125-160Hz, 160-175Hz. Another feature was the average time domain voltage, or local motor potential. The last feature was the variance of the data.

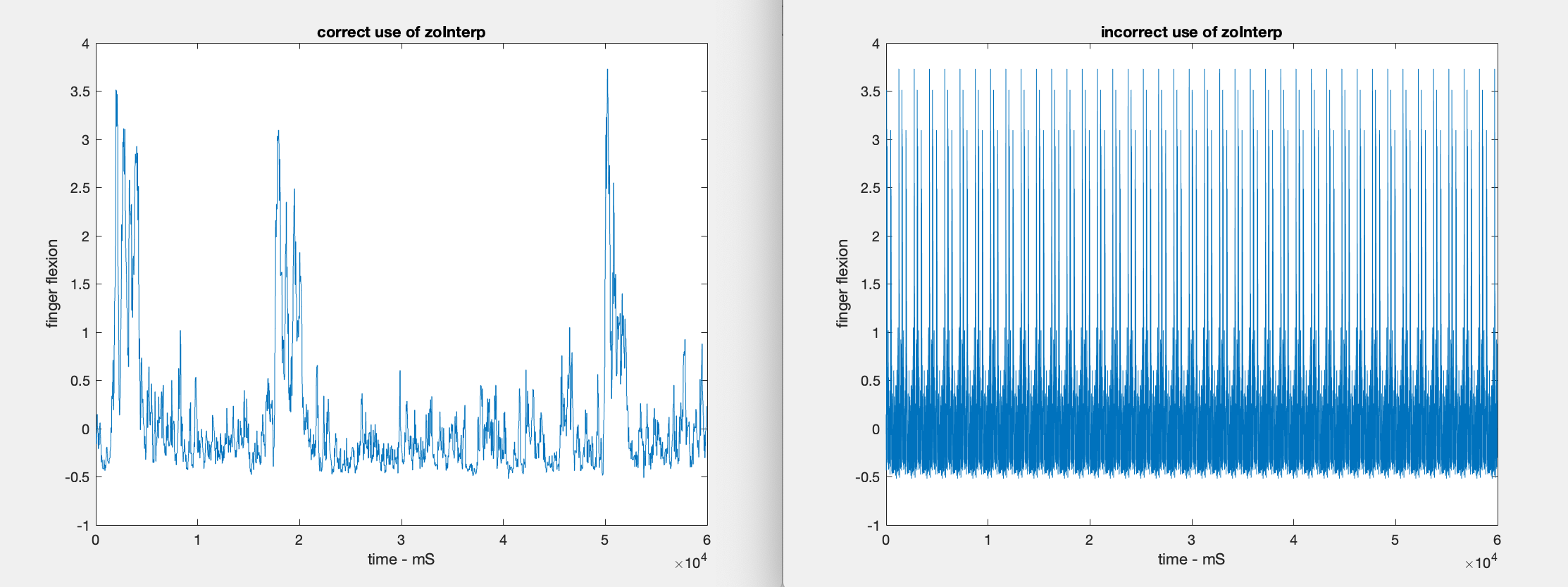
Once the features were calculated, I performed lasso (L1-constrained linear least squares fits) regularization with 5-fold cross validation to remove redundant or unimportant training features that resulted in a mean squared error greater than one standard deviation away from the minimum MSE. The exact same features were removed from the testing features. Then I created a response matrix from the reduced features. The response matrix was produced by taking the first N-1 rows of the features, flipping them, and appending them to the beginning of the feature matrix. The response matrix was of size (M, 1+ CF \* N\_wind), where M was the number of feature samples, CF was the number of channels \* number of features, and N\_wind was the number of windows to use (I specified N\_wind = 6). The response matrix is basically a timeline of features, going back (N\_wind - 1) windows. This helps improve prediction since it uses features of the current window as well as features of windows leading up to the current data for analysis. This could help with deciding whether the finger was being flexed further or was in the direction of being relaxed.

Then the response matrix of each finger of each subject was used to fit a regression ensemble model. The model used a learner aggregation method LSBoost with 40 ensemble learning cycles, learners set to be a tree with a maximum of 10 splits, and learning rate of 0.1. Each resulting model was used to predict the flexion of its corresponding finger and subject. Finally, the predictions were post-processed by sampling up using zoInterp, a function written to repeat each value of an array ‘n’ times, and then by applying a moving average with a sliding window of size 276.

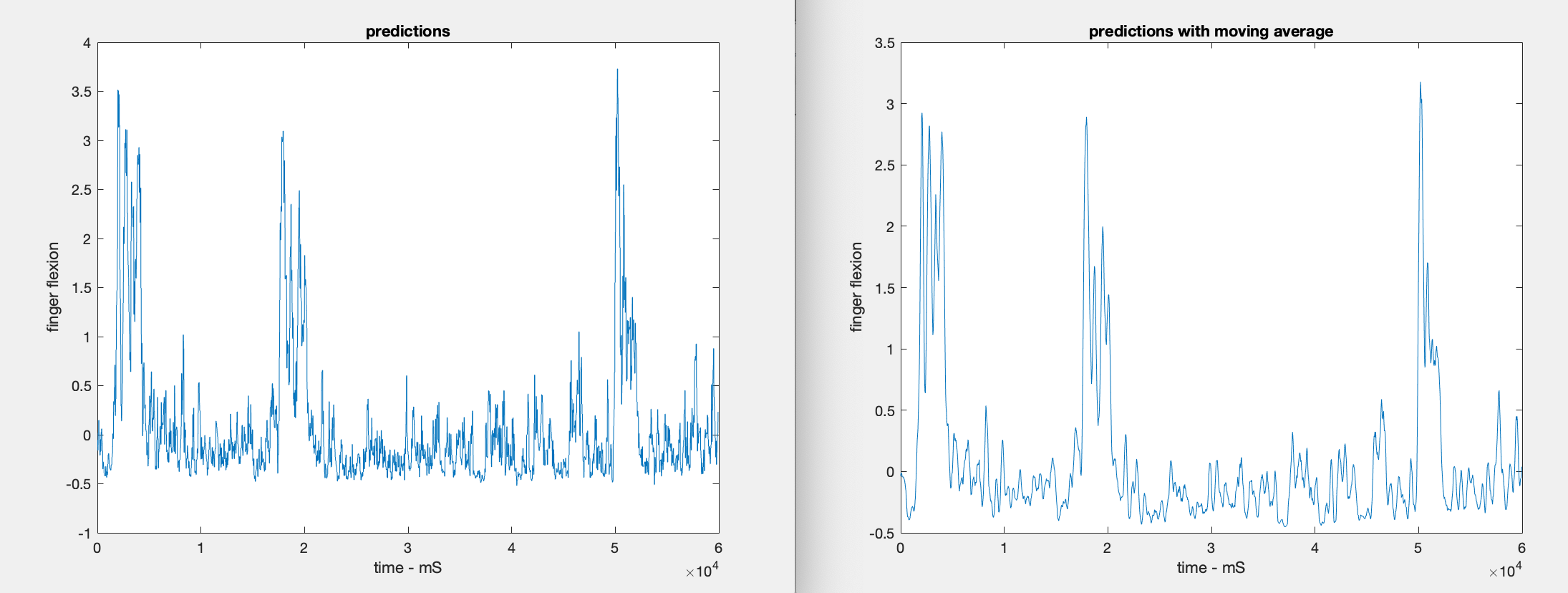
Flow chart summarizing the general steps of the algorithm:



The following pictures show how I realized I was inputting my data as a column rather than a row into my zoInterp function which initially resulted in a completely incorrect result, as seen on the right.



The following pictures show the output predictions of my 20% testing split of my training data with and without moving average. The predictions with moving average produced a plot much more similar to the given finger flexion from the data glove.



One method I tried was PCA to see if there was a better dimensionality or essential principal components of the ECOg data to focus on, however, the results of the PCA did not seem helpful or improve correlation significantly. I also initially tried using MATLAB’s fsrnca (Feature selection using neighborhood component analysis for regression). I tried to use this to check what percent of features, which were ranked by importance, I should keep. The calculated a recommendation of 34% of the function’s highest ranking features, however, this did not help improve correlation much. A similar failed attempt happened with using ReliefF algorithm which ranks importance of predictors. Other failed methods including a high pass filter at 0.3Hz for the ECOG data (when the data was already high passed at 0.15Hz), trying very small or very large values for N\_wind for the R-matrix (small values did not provide enough past feature timeline while too large values possibly caused overfitting to the training data), trying cross validation with a high number of folds for lasso (which makes sense since the data is time continuous), applying moving variance, as well as trying logistic regression and linear regression. I also tried the average frequency domain amplitudes of the frequency ranges mentioned in one of the provided papers (8-12, 18-24, 75-115, 125-159, 159-175Hz) but since the rest of my algorithm was so different, this also worsened my correlation.

Some initial thoughts about why the fourth (ring) finger’s flexion was generally so correlated with the third (middle) and fifth’s (little) can be seen first hand as it is difficult to move the third or fifth finger without moving the fourth. This could be a result of tied tendons in the hand or shared muscles of the hand.

Overall, I really enjoyed this experience and feel like I did a good job on it. By plotting the residuals or predictions vs actual, I was able to move towards a stronger model. I tried not to overfit by using cross-validation to some extent. I definitely think, given the time, there would be a lot more parameter changes I would like to try. I could play around with n\_wind, window displacement, or additional features such as line length. I could also try to ignore the last 500ms of each trial to avoid any effect of the next trial. I could also instead of padding my end predictions with the same last value, I could take a previous section of predictions and flip them to become the missing predictions. This project was very enjoyable for me and makes me want to work on similar projects in the future. One major thing I gained from this was learning to step back and plot the data/predictions to understand what’s going right or wrong.

References and Readings:

Chen, Weixuan & Liu, Xilin & Litt, Brian. (2014). Logistic-weighted regression improves decoding of finger flexion from electrocorticographic signals. 2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBC 2014. 2014. 2629-32. 10.1109/EMBC.2014.6944162.

Elgharabawy, Ayman & Wahed, Manal. (2012). Prediction of Five-Class Finger Flexion Using ECoG Signals. 2012 Cairo International Biomedical Engineering Conference, CIBEC 2012. 10.1109/CIBEC.2012.6473300.

Flamary, Rémi, and Alain Rakotomamonjy. “Decoding Finger Movements from ECoG Signals Using Switching Linear Models.” *Frontiers in Neuroscience*, Frontiers Research Foundation, 6 Mar. 2012, www.ncbi.nlm.nih.gov/pmc/articles/PMC3294271/.

Liang, Nanying, and Laurent Bougrain. “Decoding Finger Flexion from Band-Specific ECoG Signals in Humans.” *Frontiers*, Frontiers, 5 June 2012, www.frontiersin.org/articles/10.3389/fnins.2012.00091/full.

Liao, Ke, et al. “Decoding Individual Finger Movements from One Hand Using Human EEG Signals.” *PLOS ONE*, Public Library of Science, journals.plos.org/plosone/article?id=10.1371%2Fjournal.pone.0085192.

Appendix

Code:

function [predicted\_dg] = make\_predictions(test\_ecog)

% INPUTS: test\_ecog - 3 x 1 cell array containing ECoG for each subject, where test\_ecog{i}

% to the ECoG for subject i. Each cell element contains a N x M testing ECoG,

% where N is the number of samples and M is the number of EEG channels.

% OUTPUTS: predicted\_dg - 3 x 1 cell array, where predicted\_dg{i} contains the

% data\_glove prediction for subject i, which is an N x 5 matrix (for

% fingers 1:5)

% Run time: The script has to run less than 1 hour.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%

load('saved\_final\_models.mat');

%%

ltest1 = test\_ecog{1};

ltest1(:,55) = [];

ltest2 = test\_ecog{2};

ltest2(:,21) = [];

ltest2(:,38) = [];

ltest3 = test\_ecog{3};

%% Get Features

% run getWindowedFeats function

window\_length = 80e-3;

window\_overlap = 40e-3;

fs = 1000;

%train features for each subject

%% features loaded above, uncomment to run again

[final\_test\_feats1]=getWindowedFeats(ltest1, fs, window\_length, window\_overlap);

[final\_test\_feats2]=getWindowedFeats(ltest2, fs, window\_length, window\_overlap);

[final\_test\_feats3]=getWindowedFeats(ltest3, fs, window\_length, window\_overlap);

%% create Rmatrix and train model

%%

%make sure to load Mdls and Bs and is

N\_wind = 6;

%%

ta = final\_test\_feats1(:,Ba1(:,ia1)~=0);

tb = final\_test\_feats1(:,Bb1(:,ib1)~=0);

tc = final\_test\_feats1(:,Bc1(:,ic1)~=0);

td = final\_test\_feats1(:,Bd1(:,id1)~=0);

Rta1 = create\_R\_matrix(ta, N\_wind);

Rtb1 = create\_R\_matrix(tb, N\_wind);

Rtc1 = create\_R\_matrix(tc, N\_wind);

Rtd1 = create\_R\_matrix(td, N\_wind);

yfita1 = predict(Mdla1, Rta1);

yfitb1 = predict(Mdlb1, Rtb1);

yfitc1 = predict(Mdlc1, Rtc1);

yfitd1 = predict(Mdld1, Rtd1);

%%

ta = final\_test\_feats2(:,Ba2(:,ia2)~=0);

tb = final\_test\_feats2(:,Bb2(:,ib2)~=0);

tc = final\_test\_feats2(:,Bc2(:,ic2)~=0);

td = final\_test\_feats2(:,Bd2(:,id2)~=0);

Rta1 = create\_R\_matrix(ta, N\_wind);

Rtb1 = create\_R\_matrix(tb, N\_wind);

Rtc1 = create\_R\_matrix(tc, N\_wind);

Rtd1 = create\_R\_matrix(td, N\_wind);

yfita2 = predict(Mdla2, Rta1);

yfitb2 = predict(Mdlb2, Rtb1);

yfitc2 = predict(Mdlc2, Rtc1);

yfitd2 = predict(Mdld2, Rtd1);

%%

ta = final\_test\_feats3(:,Ba3(:,ia3)~=0);

tb = final\_test\_feats3(:,Bb3(:,ib3)~=0);

tc = final\_test\_feats3(:,Bc3(:,ic3)~=0);

td = final\_test\_feats3(:,Bd3(:,id3)~=0);

Rta1 = create\_R\_matrix(ta, N\_wind);

Rtb1 = create\_R\_matrix(tb, N\_wind);

Rtc1 = create\_R\_matrix(tc, N\_wind);

Rtd1 = create\_R\_matrix(td, N\_wind);

yfita3 = predict(Mdla3, Rta1);

yfitb3 = predict(Mdlb3, Rtb1);

yfitc3 = predict(Mdlc3, Rtc1);

yfitd3 = predict(Mdld3, Rtd1);

%%

predicted\_dg = cell(3,1);

%

[x,~] = size(test\_ecog{1});

[y,~] = size(final\_test\_feats1);

ya = floor(x/y);

yb = rem(x,y);

a = {yfita1, yfitb1, yfitc1, yfitd1, yfitd1};

T = zeros(x, 5);

for i = [1 2 3 5]

temp = zoInterp(cell2mat(a(i))', ya);

temp = [temp, temp(end) \*ones(yb,1)']';

T(:,i) = temp;

end

predicted\_dg{1} = T;

a = {yfita2, yfitb2, yfitc2, yfitd2,yfitd2};

T = zeros(x, 5);

for i = [1 2 3 5]

temp = zoInterp(cell2mat(a(i))', ya);

temp = [temp, temp(end) \*ones(yb,1)']';

T(:,i) = temp;

end

predicted\_dg{2} = T;

a = {yfita3, yfitb3, yfitc3, yfitd3,yfitd3};

T = zeros(x, 5);

for i = [1 2 3 5]

temp = zoInterp(cell2mat(a(i))', ya);

temp = [temp, temp(end) \*ones(yb,1)']';

T(:,i) = temp;

end

predicted\_dg{3} = T;

%% moving mean

for i = 1:3

temp = predicted\_dg{i};

for j = [1 2 3 5]

mv = movmean(temp(:,j),276);

temp(:,j) = mv;

end

predicted\_dg{i} = temp;

end

End

%LeaderboardCode

%load data

load('final\_proj\_part1\_data.mat');

load('leaderboard\_data.mat');

%% Extract dataglove and ECoG data

% Dataglove should be (samples x 5) array

% ECoG should be (samples x channels) array

% use all the training data

%training set of all 3 subjects

train\_dg\_mat = cell2mat(train\_dg);

final\_traindg\_1 = train\_dg\_mat(:,1:5);

final\_traindg\_2 = train\_dg\_mat(:,6:10);

final\_traindg\_3 = train\_dg\_mat(:,11:15);

final\_trainecog\_1 = train\_ecog{1}(:,:);

final\_trainecog\_2 = train\_ecog{2}(:,:);

final\_trainecog\_3 = train\_ecog{3}(:,:);

ltest1 = leaderboard\_ecog{1};

ltest1(:,55) = [];

ltest2 = leaderboard\_ecog{2};

ltest2(:,21) = [];

ltest2(:,38) = [];

ltest3 = leaderboard\_ecog{3};

%% Get Features

% run getWindowedFeats function

%

window\_length = 80e-3;

window\_overlap = 40e-3;

fs = 1000;

%train features for each subject

%% features loaded above, uncomment to run again

% [final\_test\_feats1]=getWindowedFeats(ltest1, fs, window\_length, window\_overlap);

%

% [final\_test\_feats2]=getWindowedFeats(ltest2, fs, window\_length, window\_overlap);

%

% [final\_test\_feats3]=getWindowedFeats(ltest3, fs, window\_length, window\_overlap);

%

% %train features

% [final\_train\_feats1]=getWindowedFeats(final\_trainecog\_1, fs, window\_length, window\_overlap);

%

% [final\_train\_feats2]=getWindowedFeats(final\_trainecog\_2, fs, window\_length, window\_overlap);

%

% [final\_train\_feats3]=getWindowedFeats(final\_trainecog\_3, fs, window\_length, window\_overlap);

%% create Rmatrix and train model

final\_downsampdg1 = downsample(final\_traindg\_1,40);

final\_downsampdg2 = downsample(final\_traindg\_2,40);

final\_downsampdg3 = downsample(final\_traindg\_3,40);

final\_Y1 = final\_downsampdg1(2:end,:);

final\_Y2 = final\_downsampdg2(2:end,:);

final\_Y3 = final\_downsampdg3(2:end,:);

[yfita1, yfitb1, yfitc1, yfitd1, Mdla1, Mdlb1, Mdlc1, Mdld1,Ba1, Bb1, Bc1, Bd1, ia1, ib1, ic1, id1] = final\_model(final\_train\_feats1, final\_Y1, final\_test\_feats1);

[yfita2, yfitb2, yfitc2, yfitd2, Mdla2, Mdlb2, Mdlc2, Mdld2,Ba2, Bb2, Bc2, Bd2,ia2, ib2, ic2, id2] = final\_model(final\_train\_feats2, final\_Y2, final\_test\_feats2);

[yfita3, yfitb3, yfitc3, yfitd3, Mdla3, Mdlb3, Mdlc3, Mdld3,Ba3, Bb3, Bc3, Bd3,ia3, ib3, ic3, id3] = final\_model(final\_train\_feats3, final\_Y3, final\_test\_feats3);

%%

predicted\_dg = cell(3,1);

%

a = {yfita1, yfitb1, yfitc1, yfitd1, yfitd1};

T = zeros(147500, 5);

for i = [1 2 3 5]

temp = zoInterp(cell2mat(a(i))', 40);

temp = [temp, temp(end) \*ones(60,1)']';

T(:,i) = temp;

end

predicted\_dg{1} = T;

a = {yfita2, yfitb2, yfitc2, yfitd2,yfitd2};

T = zeros(147500, 5);

for i = [1 2 3 5]

temp = zoInterp(cell2mat(a(i))', 40);

temp = [temp, temp(end) \*ones(60,1)']';

T(:,i) = temp;

end

predicted\_dg{2} = T;

a = {yfita3, yfitb3, yfitc3, yfitd3,yfitd3};

T = zeros(147500, 5);

for i = [1 2 3 5]

temp = zoInterp(cell2mat(a(i))', 40);

temp = [temp, temp(end) \*ones(60,1)']';

T(:,i) = temp;

end

predicted\_dg{3} = T;

%% moving mean

for i = 1:3

temp = predicted\_dg{i};

for j = [1 2 3 5]

mv = movmean(temp(:,j),276);

temp(:,j) = mv;

end

predicted\_dg{i} = temp;

end

function [all\_feats]=getWindowedFeats(raw\_data, fs, window\_length, window\_overlap)

%

% getWindowedFeats\_release.m

%

% Instructions: Write a function which processes data through the steps

% of filtering, feature calculation, creation of R matrix

% and returns features.

%

% Points will be awarded for completing each step

% appropriately (note that if one of the functions you call

% within this script returns a bad output you won't be double

% penalized)

%

% Note that you will need to run the filter\_data and

% get\_features functions within this script. We also

% recommend applying the create\_R\_matrix function here

% too.

%

% Inputs: raw\_data: The raw data for all patients

% fs: The raw sampling frequency

% window\_length: The length of window

% window\_overlap: The overlap in window

%

% Output: all\_feats: All calculated features

%

%% Your code here (3 points)

%raw\_data = ltest1;

[~,nchannels] = size(raw\_data);

% First, filter the raw data

clean\_data = filter\_data(raw\_data);

%displacement

d = window\_length - window\_overlap;

%number of windows

%NumWins = @(xLen, fs, winLen, winDisp) (xLen-winLen\*fs+winDisp\*fs)/(winDisp\*fs);

NumWins = @(xLen, fs, winLen, winDisp) (xLen-winLen\*fs+winDisp\*fs)/(winDisp\*fs);

%nw = NumWins(length(clean\_data), fs, window\_length, d);

nw = floor(NumWins(length(clean\_data), fs, window\_length, d));

% Then, loop through sliding windows

all\_feats = zeros(nw, 7\*nchannels);

for i = 0:nw-1

ind = i\*d\*fs;

% Within loop calculate feature for each segment (call get\_features)

all\_feats(end-i, :) = get\_features(clean\_data(end-ind-window\_length\*fs+1:end-ind,:),fs);

end

% Finally, return feature matrix

end

function clean\_data = filter\_data(raw\_eeg)

%

% filter\_data\_release.m

%

% Instructions: Write a filter function to clean underlying data.

% The filter type and parameters are up to you.

% Points will be awarded for reasonable filter type,

% parameters, and correct application. Please note there

% are many acceptable answers, but make sure you aren't

% throwing out crucial data or adversely distorting the

% underlying data!

%

% Input: raw\_eeg (samples x channels)

%

% Output: clean\_data (samples x channels)

%

%% Your code here (2 points)

%raw\_eeg = final\_trainecog\_1;

[nsamples,nchannels] = size(raw\_eeg);

% high pass filter 0.3hz

%hp = raw\_eeg;

% hp = highpass(raw\_eeg, 0.3, 1000);

% disp("highpass");

% d = designfilt('highpassiir', ... % Response type

% 'StopbandFrequency',0.15, ... % Frequency constraints

% 'PassbandFrequency',0.3, ...

% 'DesignMethod','ellip', ... % Design method

% 'MatchExactly','stopband', ... % Design method options

% 'SampleRate',1000); % Sample rate

% hp = filtfilt(d, raw\_eeg);

hp = raw\_eeg;

%60hz notch filter for power line noise

wo = 60/(1000/2);

bw = wo/35;

[b,a] = iirnotch(wo,bw);

ff = filtfilt(b,a,hp);

disp("60 notch");

%common average reference

%should i align signals??

% %average across all channels

% ac = mean(ff,2);

% %subtract ac from all channels

% A = repmat(ac,1,nchannels);

% ac = ac - A;

% %reshape into trials

% ac = reshape(ac, 4000,[]);

% ac = mean(ac,2);

% %subtract from all trials across all channels

% A = repmat(ac,nsamples/4000,nchannels);

% clean\_data = ff - A;

% % clean\_data = ac;

% disp("CAR");

%average across all channels

ac = mean(ff,2);

%subtract ac from all channels

A = repmat(ac,1,nchannels);

clean\_data = ff - A;

%reshape into trials

% m = mod(length(raw\_eeg),4000);

% ac = reshape(ac(1:end - m,:), 4000,[]);

% ac = mean(ac,2);

% %subtract from all trials across all channels

% A = repmat(ac,nsamples,nchannels);

% clean\_data = ff - A;

% % clean\_data = ac;

end

function [features] = get\_features(clean\_data,fs)

%

% get\_features\_release.m

%

% Instructions: Write a function to calculate features.

% Please create 4 OR MORE different features for each channel.

% Some of these features can be of the same type (for example,

% power in different frequency bands, etc) but you should

% have at least 2 different types of features as well

% (Such as frequency dependent, signal morphology, etc.)

% Feel free to use features you have seen before in this

% class, features that have been used in the literature

% for similar problems, or design your own!

%

% Input: clean\_data: (samples x channels)

% fs: sampling frequency

%

% Output: features: (1 x (channels\*features))

%

%% Your code here (8 points)

[L,nchannels] = size(clean\_data);

features = zeros(nchannels, 7);

for channel = 1: nchannels

% % average frequency domain feature for specific bands

% %Compute the Fourier transform of the signal.

% Y = fft(clean\_data(:,channel));

% %Compute the two-sided spectrum P2. Then compute the single-sided

% %spectrum P1 based on P2 and the even-valued signal length L.

% P2 = abs(Y/L);

% P1 = P2(1:L/2+1);

% P1(2:end-1) = 2\*P1(2:end-1);

% %if i want power instead of amplitude

% %P1 = 10\*log10(P1)

%5-15hz

% l = length(P1);

% f = fs\*(0:(L/2))/L;

% fl = length(f);

% m1 = mean(P1(f>= 5 & f<= 15));

m1 = bandpower(clean\_data(:,channel),1000,[5 15]);

features(channel, 1) = m1;

%20-25

%m2 = mean(P1(f>= 20 & f<= 25));

m2 = bandpower(clean\_data(:,channel),1000,[20 25]);

features(channel, 2) = m2;

%75-115

%m3 = mean(P1(f>= 75 & f<= 115));

m3 = bandpower(clean\_data(:,channel),1000,[75 115]);

features(channel, 3) = m3;

%125-160

%m4 = mean(P1(f>= 125 & f<= 160));

m4 = bandpower(clean\_data(:,channel),1000,[125 160]);

features(channel, 4) = m4;

%160-175

%m5 = mean(P1(f>= 160 & f<= 175));

m5 = bandpower(clean\_data(:,channel),1000,[160 175]);

features(channel, 5) = m5;

%average time domain voltage/ local motor potential

features(channel, 6) = mean(clean\_data(:,channel));

%variance feature

features(channel, 7) =var(clean\_data(:,channel));

end

features = reshape(features',1,nchannels\*7);

end

function [yfita, yfitb, yfitc, yfitd, Mdla, Mdlb, Mdlc, Mdld, Ba, Bb, Bc, Bd, ia, ib, ic, id] = final\_model(train\_feats, Y, test\_feats)

%for n = 6:6

[Ba,FitInfoa] = lasso(train\_feats,Y(:,1),'CV',5);

[Bb,FitInfob] = lasso(train\_feats,Y(:,2),'CV',5);

[Bc,FitInfoc] = lasso(train\_feats,Y(:,3),'CV',5);

[Bd,FitInfod] = lasso(train\_feats,Y(:,5),'CV',5);

ia = find(FitInfoa.MSE>min(FitInfoa.MSE)+std(FitInfoa.MSE),1);

ib = find(FitInfob.MSE>min(FitInfob.MSE)+std(FitInfob.MSE),1);

ic = find(FitInfoc.MSE>min(FitInfoc.MSE)+std(FitInfoc.MSE),1);

id = find(FitInfod.MSE>min(FitInfod.MSE)+std(FitInfod.MSE),1);

Pa = train\_feats(:,Ba(:,ia)~=0);

Pb = train\_feats(:,Bb(:,ib)~=0);

Pc = train\_feats(:,Bc(:,ic)~=0);

Pd = train\_feats(:,Bd(:,id)~=0);

ta = test\_feats(:,Ba(:,ia)~=0);

tb = test\_feats(:,Bb(:,ib)~=0);

tc = test\_feats(:,Bc(:,ic)~=0);

td = test\_feats(:,Bd(:,id)~=0);

N\_wind = 6;

Ra =create\_R\_matrix(Pa, N\_wind);

Rb =create\_R\_matrix(Pb, N\_wind);

Rc =create\_R\_matrix(Pc, N\_wind);

Rd =create\_R\_matrix(Pd, N\_wind);

Rta = create\_R\_matrix(ta, N\_wind);

Rtb = create\_R\_matrix(tb, N\_wind);

Rtc = create\_R\_matrix(tc, N\_wind);

Rtd = create\_R\_matrix(td, N\_wind);

%corrc = zeros(1,5);

Mdla = fitrensemble(Ra,Y(:,1),'Method','LSBoost','NumLearningCycles',40, 'Learners',templateTree('MaxNumSplits',10),'LearnRate',0.1);

yfita = predict(Mdla, Rta);

%corrc(1) = corr(Y1t(:,1),yfita);

Mdlb = fitrensemble(Rb,Y(:,2),'Method','LSBoost','NumLearningCycles',40, 'Learners',templateTree('MaxNumSplits',10),'LearnRate',0.1);

yfitb = predict(Mdlb, Rtb);

%corrc(2) = corr(Y1t(:,2),yfitb);

Mdlc = fitrensemble(Rc,Y(:,3),'Method','LSBoost','NumLearningCycles',40, 'Learners',templateTree('MaxNumSplits',10),'LearnRate',0.1);

yfitc = predict(Mdlc, Rtc);

%corrc(3) = corr(Y1t(:,3),yfitc);

Mdld = fitrensemble(Rd,Y(:,5),'Method','LSBoost','NumLearningCycles',40, 'Learners',templateTree('MaxNumSplits',10),'LearnRate',0.1);

yfitd = predict(Mdld, Rtd);

%corrc(5) = corr(Y1t(:,5),yfitd);

% disp(n);

% disp(corrc);

%end

function zfvec = zoInterp(x, numInterp)

zfvec = reshape(repmat(x,numInterp,1),[1,length(x)\*numInterp]);

end

function [R]=create\_R\_matrix(features, N\_wind)

%

% get\_features\_release.m

%

% Instructions: Write a function to calculate R matrix.

%

% Input: features: (samples x (channels\*features))

% N\_wind: Number of windows to use

%

% Output: R: (samples x (N\_wind\*channels\*features))

%

%% Your code here (5 points)

[M,CF] = size(features);

%take the first N-1 rows

n1rows = features(1:N\_wind-1,:);

%append flipped to beginning of feature matrix

appFeat = [flip(n1rows);features];

%create R matrix

R = ones(M, CF\*N\_wind);

for c = 0:N\_wind-1

R(:,1+(CF\*c):CF\*(c+1)) = appFeat(1+c:c+M,:);

end

%add col of ones

o = ones(M, 1);

R = [o, R];

end