Kanika Mohan and Kashish Gupta
5/2/18

Predicting Finger Flexion Using Linear Regression, LASSO, and a 1-D Median Filter

Summary of Algorithm

To process our data we use a sliding window to gather six different types of features for each window of ECoG data. We then concatenate the features together and run a linear regression with the downsampled data glove values as labels. Our algorithm then uses LASSO to select the most important variables and perform regularization individually for each of the four target fingers. From the several models produced by LASSO, we pick the model which generates the highest training correlation and output the feature weights to our prediction step. To predict on testing data, we run the same features processing and predict based off of the feature weights from the linear regression previously obtained from the training step. Lastly, we post-process our predictions with a 1-D median filter.

Our Algorithm Step-by-Step

Pre-processing: We start by pulling the data from IEEG and separating the data into a training set and a testing set, which we would later on run cross-validation on in order to find our testing correlations. We then removed the 55th channel from subject 1 data and the 21st and 38th channels from subject 2. These channels included artifacts rather than relevant neural spikes and decreased the quality of our predictions, and removing them entirely improved our results.

Features and Moving Windows: The algorithm calculated features across 100 ms windows with 50 ms overlap. The 6 features included were the moving average of the time-domain voltage data and the average magnitude in the frequency-domain across 5 frequency bands: 5-15 Hz, 20-25 Hz, 75-115 Hz, 125-160 Hz, 160-175 Hz. These bands were chosen as they (Kubanek *et al.*). The moving average was calculated using the MovingWinFeats.m function, which pulls indices from a vector correlating to the specified moving window parameters and applies an anonymous function to the values. The function produces a vector of features with length of the number of windows. The

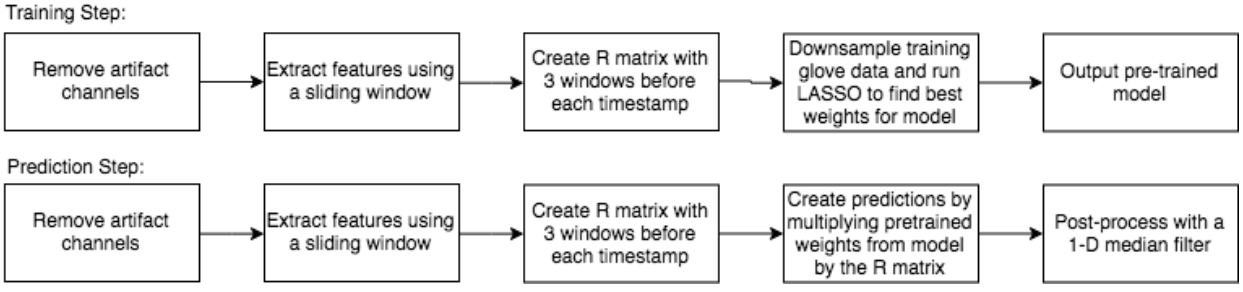
average magnitude of the frequency-domain within the 5 bands was calculated using the spectrogram function provided by MATLAB. Spectrogram was used to perform a short-time Fourier transform across a provided range of cyclical frequencies using certain window and overlap parameters. The average magnitude within each frequency band was found by pulling the correct indices from a vector of cyclical frequency values for each band and averaging over the corresponding magnitudes. This method resulted in 5 vectors with the length of the number of windows, each corresponding to a specified frequency band. These 6 features were run on all channels in the subject and then concatenated into one matrix of dimensions *num_wins x (num_feats*num_chans)*, where *num_wins* was the number of windows, *num_feats* was the 6 chosen features, and *num_chans* was the number of channels being analyzed for the subject.

Downsampling, Training, Lasso, and Choosing a model: Before training our model, we had to downsample the data from the Data Glove, because we only had features for every 100ms window with a 50ms overlap, so we downsampled the Data Glove data to get one datapoint for every 50ms. We then created an R matrix as specified in Warland *et al.*, but this time only looking at the previous three windows of features rather than 20. The method to generate the R matrix is exactly as described in the HW7 handout and in the Warland *et al.* paper, so we will not go into it here, but essentially the R matrix serves as a set of features to predict the data glove position for a given timestamp. We feed this R matrix into LASSO (the Least Absolute Shrinkage and Selection Operator), which removes weights that are less important for the prediction. The LASSO function in MATLAB outputs 100 different models (weight vectors) with the default parameters, so we then choose the model out of these 100 with the best testing correlation. To compute which model is the best, we first separate our training data into a training and validation set, and then feed the training portion into LASSO. With each of the 100 models, we calculate a testing correlation with the testing portion of the data, and choose the model (weight vector) with the best testing correlation. Our training function then outputs this optimal model as a “Beta” vector of weights.

Prediction Step: After we've already solved for the minimum least-squares difference between the stimulus (ECoG) and reconstruction (Data Glove) in order to find the output of our model (a vector of weights that can be multiplied by an input vector of features to output a prediction value for every 50ms) in our training step, we then use our trained model to generate a prediction. The prediction step involves the same pre-processing and feature extraction as the training step, but then instead of running the R matrix produced through LASSO, we simply multiply the R matrix generated in the prediction step by the Beta generated in the training step. This will give us an output prediction for each 50ms window of data. We then use the spline function to interpolate the points in between each of our prediction values so that we can output data of the same length as the input testing data. Since we only have a prediction every 50ms, the spline function finds points in between each 50ms that fit a spline.

Post-processing: Our final prediction set was then processed using a 1-D median filter to reduce noise. A 1-D median filter functionally uses a sliding window and sorts data within that window, replacing the value with the median of the values surrounding it. When the order of the filter, *n,* is specified and even, the values *x(k)* are replaced with the median of *x(k-n/2:k+(n/2)-1)*. We ran a 1-D median filter using a range of orders on our cross-validated data and graphed the correlation. We then visually chose the optimal order for each subject. The order was kept constant across all digits for one subject but varied across subjects based on the highest correlation for each subject.

Algorithm Flow Chart



A Decision We Made Along the Way

A. **B.** **C.**

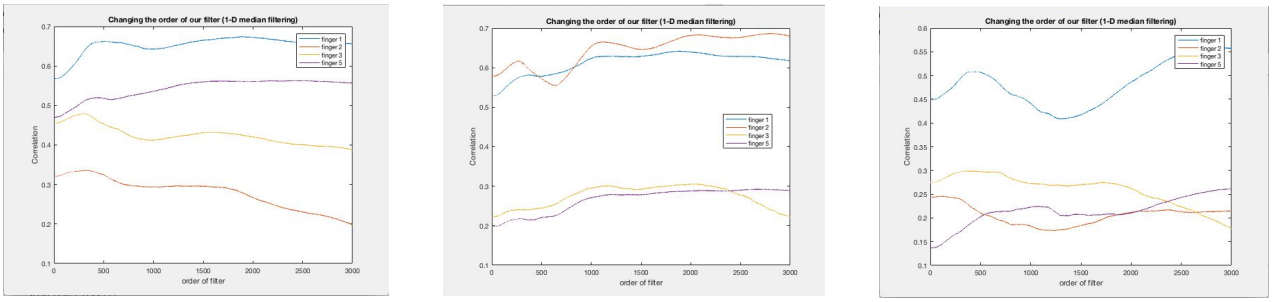


Figure X. 1-D Median Filter N-Order Parameter Scan.

A. Subject 1. The optimal order was chosen to be 2000. **B.** Subject 2. The optimal order was chosen to be 2300. **C.** Subject 3. The optimal order was chosen to be 400.

These figures motivated the decision behind choosing specific orders for each subject. While our initial algorithm applied a 400th order 1-D Median filter to the predicted data to reduce noise, which would correspond to smoothing within a window of 0.2 seconds before and after the point, we noted that each subject had a different range of order for which the correlation was maximized. While we felt that choosing a separate order to filter each finger would overfit the algorithm to our training data, we assumed that the subject would likely have the same rates of finger movement over time. Therefore, we chose to assign a specific order to each subject in our final submission. We later regretted this decision because we found that choosing a constant order 400 filter for each subject performed better than changing the order based on the subject that was being tested.

What Didn't Work

We included features such as kurtosis and skewness in our model, but both reduced the correlation values of our cross-validated data and significantly increased the runtime of the lasso algorithm. Including a 7th and 8th feature increased the size of our R-matrix and thus significantly increased processing time.

We also attempted to use Line-length and Energy as features, but these also decreased the performance of our model on a cross-validated dataset. We tried these features individually and together, but found that in both cases the performance decreased.
We found that customizing the order of our smoothing 1-D Median filter to each subject reduced our correlation values. Unfortunately, this was discovered in our last submission for the competition script, which prevented us from reverting back choosing a order-400 median filter across all subjects for our final submission.

Why the 4th Finger was Correlated with Fingers 3 and 5

The fourth finger is likely generally correlated with the third and fifth fingers given that when one bends the fourth finger, the third and fifth bend along with it. We discovered this by simply trying to bend our ring finger in isolation of the other fingers and found that it was almost impossible to move the ring finger without also moving the pinky and middle fingers. We believe this may have something to do with tendons branching and being connected across fingers. During data collection, when the subject was instructed to bend their ring finger, the data glove would have picked up movement in other fingers, leading our model to predict movement in other fingers as well since our model takes the data glove as ground truth. For this reason we omit the fourth finger entirely.

Conclusion

Throughout this project we had a very positive experience and actually felt that it was a lot easier than expected. We were able to achieve a 33% correlation by simply implementing the method that the TAs recommended within the write-up, and we were able to drastically improve the performance using LASSO several days before the first checkpoint. Furthermore, once we added a post-processing filter, we were above the checkpoint of 45% correlation. We now wish that we had spent more time reading other papers and implementing a Recursive Neural Net (RNN) in order to blow away the other teams. Overall we felt that we did quite well on this project, had a good understanding of the methodologies, and had a lot of fun along the way!

References

Kubánek, J, et al. “Decoding Flexion of Individual Fingers Using Electrocorticographic Signals in Humans.” *Journal of Neural Engineering*, vol. 6, no. 6, Jan. 2009, p. 066001., doi:10.1088/1741-2560/6/6/066001.

Warland, David K., et al. “Decoding Visual Information From a Population of Retinal Ganglion Cells.” *Journal of Neurophysiology*, vol. 78, no. 5, 2 July 1997, pp. 2336–2350., doi:10.1152/jn.1997.78.5.2336.

Appendix:

train_model.m

```
% this function will be used to train our model using the dataset for 1
% subject at a time, input a filename 'data/subject1.mat' and it outputs
% the Beta for that subject
function [betas] = train_model(filename)

%% loading the data that we downloaded from IEEG and stord as MAT files
load(filename, 'training_ecog', 'training_dg');
training_ecog = padarray(training_ecog, 50, 0, 'pre');

%training data params
num_samples = size(training_ecog, 1);
num_chans = size(training_ecog, 2);
num_feats = 6;
fs = 1000;

if num_chans == 62
    training_ecog = training_ecog(:, [1:54,56:end]);
elseif num_chans == 48
    training_ecog = training_ecog(:, [1:20,22:37,39:end]);
end

% some anonymous functions in case we need them
LLFnWins = @(x)sum(abs(diff(x)));
NumWins = @(xLen, fs, winLen, winDisp)
(((xLen/fs)-(winLen-winDisp))/winDisp);
MeanWins = @(x)mean(x);
```

```
%KurtosisWins = @(x)kurtosis(x);
num_wins = NumWins(num_samples, fs, 100/fs, 50/fs); %get number of windows
num_chans = size(training_ecog, 2);

%% compute the moving average
% compute moving average for time domain voltage using MovingWinFeats
featsVoltageAverage = zeros(num_wins, num_chans);
featsLLFn = zeros(num_wins, num_chans);

for i = 1:num_chans
    [winFeat] = MovingWinFeats(training_ecog(:,i),...
        fs, 100/fs, 50/fs,MeanWins,LLFnWins);
    featsVoltageAverage(:,i) = winFeat;
end

%% get spectrogram features
featsFreqMag_5_15 = zeros(num_wins, num_chans);
featsFreqMag_20_25 = zeros(num_wins, num_chans);
featsFreqMag_75_115 = zeros(num_wins, num_chans);
featsFreqMag_125_160 = zeros(num_wins, num_chans);
featsFreqMag_160_175 = zeros(num_wins, num_chans);
for i = 1:num_chans
    [s,f,t] =
spectrogram(training_ecog(:,i),100,50,[5:15,20:25,75:115,125:160,160:175],1
000);

    featsFreqMag_5_15 (:,i)=mean(abs(s(f>=5 & f<=15,:)));
    featsFreqMag_20_25 (:,i)= mean(abs(s(f>=20 & f<=25,:)));
    featsFreqMag_75_115 (:,i)= mean(abs(s(f>=75 & f<=115,:)));
    featsFreqMag_125_160 (:,i)= mean(abs(s(f>=125 & f<=160,:)));
    featsFreqMag_160_175(:,i)= mean(abs(s(f>=160 & f<=175,:)));
end

%%
%concatenate all features
allFeats = [featsFreqMag_5_15, featsFreqMag_20_25, ...
    featsFreqMag_75_115, featsFreqMag_125_160, featsFreqMag_160_175, ...
    featsVoltageAverage];

%% Downsample data glove to put on same time scale as features
dg_chans = size(training_dg, 2);
dec_dg = zeros(num_wins, dg_chans);
```

```
for i = 1:dg_chans
    dec_dg(:,i) = decimate(training_dg(:,i),50);
end

%% Motor Prediction
%load('data/sub1_training_feats.mat')
N = 3;
M = num_wins-(N-1);
R = ones(M, num_chans*num_feats*N + 1);

for i = 1:M
    temp = allFeats(i:i+N-1,:);
    temp = temp(:)';
    R(i, (N-1):num_chans*num_feats*N + 1) = temp;
end

%% Employ Lasso
% set training and testing fractions
fractCI = 0.6;
cross_index = ceil(M*fractCI);
Rtrain = R(1:cross_index, :);
Rtest = R(cross_index+1:end,:);
dg_train = dec_dg(N:cross_index+(N-1),[1:3,5]);
dg_test = dec_dg(cross_index+N:end,[1:3,5]);

%apply lasso
[B1, FitInfo1] = lasso(Rtrain, dg_train(:,1));
[B2, FitInfo2] = lasso(Rtrain, dg_train(:,2));
[B3, FitInfo3] = lasso(Rtrain, dg_train(:,3));
[B5, FitInfo5] = lasso(Rtrain, dg_train(:,4));

%getting predictions
preds1 = Rtest*B1;
preds2 = Rtest*B2;
preds3 = Rtest*B3;
preds5 = Rtest*B5;

 %find model with max correlation for each finger and pull beta
allpreds = {preds1, preds2, preds3, preds5};
corrs = zeros(100, 4); %100 models, 4 fingers
for j = [1,2,3,4]
    for i = 1:100
```

```
        preds = allpreds{j};
        corrs(i,j) = corr(dg_test(:,j), preds(:,i));
    end
end
[~,I] = max(corrs,[],1);

betas = [B1(:,I(1)), B2(:,I(2)), B3(:,I(3)), B5(:,I(4))];
end
```

MovingWinFeats.m

```
function [winFeat] = MovingWinFeats(x, fs, winLen, winDisp, featFn)

%find the number of windows over which to iterate
NumWins = @(xLen, fs, winLen, winDisp) ((xLen -
(winLen*fs))/(winDisp*fs)) +1;
winFeat = zeros(NumWins(length(x), fs, winLen, winDisp),1);

%begin at 1:window length, move forward by displacement numWins - 1
times, calculate feature over each window, and store
for i = 0:NumWins(length(x), fs, winLen, winDisp)-1
    winFeat(i+1) =
featFn(x((i*(winDisp*fs))+1:(i*(winDisp*fs))+(winLen*fs)));
end

end
```

predict_flexion.m

```
% this function will be used to train our model using the dataset for
1 subject at a time, input a file testdata and it outputs
% the Beta for that subject

function [full_pred] = predict_flexion(testdata, Beta)

%% loading the data that we downloaded from IEEG and stord as MAT
files
leaderboard_ecog = testdata;
```

```
%leaderboard_ecog = leaderboard_ecog(1:147500,:);
leaderboard_ecog = padarray(leaderboard_ecog, 50, 0, 'pre');
numFeats = 6;

%training data params
num_samples = size(leaderboard_ecog, 1);
num_chans = size(leaderboard_ecog, 2);
fs = 1000;

%remove certain channels
if num_chans == 62
    leaderboard_ecog = leaderboard_ecog(:,[1:54,56:end]);
elseif num_chans == 48
    training_ecog = training_ecog(:, [1:20,22:37,39:end]);
end

% some anonymous functions in case we need them
LLFnWins = @(x)sum(abs(diff(x)));
NumWins = @(xLen, fs, winLen, winDisp)
(((xLen/fs)-(winLen-winDisp))/winDisp);
MeanWins = @(x)mean(x);
%KurtosisWins = @(x)kurtosis(x);
num_wins = NumWins(num_samples, fs, 100/fs, 50/fs); %get number of
windows
num_chans = size(leaderboard_ecog, 2);

%% compute moving average for time domain voltage using
MovingWinFeats
featsVoltageAverage = zeros(num_wins, num_chans);

for i = 1:num_chans
    [winFeat] = MovingWinFeats(leaderboard_ecog(:,i),...
        fs, 100/fs, 50/fs,MeanWins,LLFnWins);
    featsVoltageAverage(:,i) = winFeat;
end

%%
featsFreqMag_5_15 = zeros(num_wins, num_chans);
```

```
featsFreqMag_20_25 = zeros(num_wins, num_chans);
featsFreqMag_75_115 = zeros(num_wins, num_chans);
featsFreqMag_125_160 = zeros(num_wins, num_chans);
featsFreqMag_160_175 = zeros(num_wins, num_chans);
for i = 1:num_chans
    [s,f,t] =
spectrogram(leaderboard_ecog(:,i),100,50,[5:15,20:25,75:115,125:160,1
60:175],1000);

    featsFreqMag_5_15 (:,i)=mean(abs(s(f>=5 & f<=15,:)));
    featsFreqMag_20_25 (:,i)= mean(abs(s(f>=20 & f<=25,:)));
    featsFreqMag_75_115 (:,i)= mean(abs(s(f>=75 & f<=115,:)));
    featsFreqMag_125_160 (:,i)= mean(abs(s(f>=125 & f<=160,:)));
    featsFreqMag_160_175(:,i)= mean(abs(s(f>=160 & f<=175,:)));
end
%concatenate all features
allFeats = [featsFreqMag_5_15, featsFreqMag_20_25, ...
    featsFreqMag_75_115, featsFreqMag_125_160, featsFreqMag_160_175,
...
     featsVoltageAverage];

%% Motor Prediction
%load('data/sub1_training_feats.mat')
M = num_wins-2;
N = 3;
R = ones(M, num_chans*numFeats*3 + 1);

for i = 1:M
    temp = allFeats(i:i+N-1,:);
    temp = temp(:)';
    R(i, (N-1):num_chans*numFeats*N + 1) = temp;
end

%% Create Predictions for the leaderboard
predictedY = R*Beta;

%% Interpolated the points back to 300,000
```

```
time_pred = 101:50:M*50 + 101 - 50;
time_real = 101:M*50 + 101-50;
interp_pred = spline(time_pred, predictedY', time_real);
full_pred = padarray(interp_pred', 100, 0, 'pre');
full_pred = padarray(full_pred, 49, 0, 'post');
end
```

make_predictions.m

```
function [predicted_dg] = make_predictions(test_ecog)

%%
% this was used to create the .mat file with all of our feature
weights (train our model and output to a .mat file)
% Beta1 = train_model('data/subject1');
% Beta2 = train_model('data/subject2');
% Beta3 = train_model('data/subject3');
% betas = cell(3,1);
% betas{1} = Beta1;
% betas{2} = Beta2;
% betas{3} = Beta3;
% save('our_best_model.mat', 'betas');

load ('our_best_model.mat', 'betas');

%%
%create cell array with one element for each subject
predicted_dg = cell(3,1);

Beta1 = betas{1};
Beta2 = betas{2};
Beta3 = betas{3};
testdata1 = test_ecog{1};
testdata2 = test_ecog{2};
testdata3 = test_ecog{3};

full_preds1 = predict_flexion(testdata1, Beta1);
full_preds1 = [full_preds1(:,1:3) zeros(size(full_preds1,1),1)
full_preds1(:,4)];
```

```
full_preds1 = medfilt1(full_preds1,400,'truncate');
sprintf('1 completed')

full_preds2 = predict_flexion(testdata2, Beta2);
full_preds2 = [full_preds2(:,1:3) zeros(size(full_preds2,1),1)
full_preds2(:,4)];
full_preds2 = medfilt1(full_preds2,400,'truncate');
sprintf('2 completed')

full_preds3 = predict_flexion(testdata3,Beta3);
full_preds3 = [full_preds3(:,1:3) zeros(size(full_preds3,1),1)
full_preds3(:,4)];
full_preds3 = medfilt1(full_preds3,400,'truncate');
sprintf('3 completed')

predicted_dg{1} = full_preds1;
predicted_dg{2} = full_preds2;
predicted_dg{3} = full_preds3;

end
```