University of Pennsylvania

Final Report

# Brain Computer Interface (BE 521)

*Submitted By:*

Christian de Abreu

Anya Hall

Alejandro Villasmil

*Submitted To:*

Dr. Brian Litt

Preya Shah

Nicki Driscoll

$$\mathbf{R} = \begin{bmatrix} 1 & r_1^1 & r_2^1 & \dots & r_N^1 & r_1^2 & r_2^2 & \dots & r_N^2 & \dots & \dots & r_1^\nu & r_2^\nu & \dots & r_N^\nu \\ 1 & r_2^1 & r_3^1 & \dots & r_{N+1}^1 & r_2^2 & r_3^2 & \dots & r_{N+1}^2 & \dots & \dots & r_2^\nu & r_3^\nu & \dots & r_{N+1}^\nu \\ 1 & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & r_M^1 & r_{M+1}^1 & \dots & r_{N+M-1}^1 & r_M^2 & r_{M+1}^2 & \dots & r_{N+M-1}^2 & \dots & \dots & r_M^\nu & r_{M+1}^\nu & \dots & r_{N+M-1}^\nu \end{bmatrix}$$

Figure 1: R Matric Representation

# 1 Final Algorithm

## 1.1 Overview

Our final algorithm consists of three major components: training, validation, and testing. In the training stage, each subject's ECOG data was loaded in and inactive channels or channels with abnormal artifacts were removed. Following this, 7 total features were extracted from each of the corresponding windows, those being the time domain average, line length, and frequency domain averages for five distinct bandwidths. The features were then allocated into a feature matrix which was then used in an optimal linear decoder to ultimately perform a linear regression and arrive at a weights vector, which could then be used to predict glove traces. Predictions were also smoothed, as it was proven to significantly improve prediction correlation by the removal of noise. Following this, we performed a 5-fold cross validation on the training feature matrix to calculate an approximate correlation that would not be subject to overfitting errors. After this checkpoint, we then used an optimal linear decoder method on the testing data implementing our weights vector acquired in the training stage.

## 1.2 Detailed Explanation

The first step in our Never Suppressing algorithm involved training. In order to accomplish this, initial training ECOG data was imported from the IEEG portal for each subject. Via a plot analysis, certain channels were removed from each subject due to complete inactivity or presence of abnormal artifacts. Specifically, channels 4, 40, 49, and 55 were removed for subject 1, channels 21 and 38 for subject 2, and none for subject 3. Following this slight modification of the ECOG data, we then turned to feature extraction. The number of windows was calculated using the NumWins function which uses the set length, sample rate, window length, and window displacement as input parameters. The feature functions include AveTimeDomain, SpecFreq3, and LLFn. AveTimeDomain simply calculates the average of a given set. SpecFreq3 utilizes the spectrogram MatLab function to extract 1000 power values for 5 distinct frequency bandwidths (5-15 Hz, 20-25 Hz, 75-115 Hz, 125-160Hz, and 160-175 Hz) and calculate the average of each. Lastly, LLFn calculates the line length of the given set of values. These feature functions were then used in conjunction with the MovingWinFeats function, which calculates the desired features across all windows present.

After extracting all 7 features across all 5999 windows of the training set and for all channels for all subjects, we then proceeded to index these into a feature matrix as described in Warland et al., 1997. It was decided that our algorithm would take into account the previous 3 time bins when making a predictions for the current time bin. Therefore, the first 3 rows of the feature matrix were omitted and every 21 columns corresponded to a set of 3 time bins for each of the 7 features. As a result, the matrix has 5996 rows and 21 times the number of channels for columns plus a column of ones. The underlying schematic of our feature matrix can be seen below, with M being equal to 5996, N equal to 3, and v equal to the number of channels. Only one feature is show, however, 7 were included in our algorithm. This matrix representation can be seen in Figure 1.

Following this, we then needed to create a target vector which would be used in the linear regression. This target vector was extracted from the glove traces corresponding to the training ECOG data. These traces had to be downsampled down to the number of windows being considered in the feature matrix. This was done using the MatLab decimate function and we then removed the first three traces corresponding to the first three time bins, as we plan on using the preceding three time bins to make predictions.

With both the feature matrix and the glove trace target vector calculated, all the necessary components to carry out the linear regression had been obtained. The following equation was used to compute the weights vector:

$$f = (R^T R)^{-1}(R^T s)$$

R corresponding to the feature matrix, s to the target vector, and f to the weights vector. This weights vector was saved as it will be used further down the line for the linear regression to be performed on the testing data.

Cross validation was also an essential component to the robustness and accuracy of our algorithm. Cross validation was necessary in order to get a sense of what our testing correlations would be and to avoid any erroneous correlations due to over-fitting. We proceeded to divide the feature matrix into 5 evenly distributed folds and iterated through the 5 folds 5 separate times, each time assigning one fold as the testing set and the remaining four as the training set. Consequently, five distinct feature matrices were obtained. The glove trace data was similarly broken up into the same five folds. Following the linear regression procedure explained above, a regression was performed for each of the five training feature matrices and its respective target vector. The corresponding weights vector was then multiplied by the testing feature matrix, as shown below:

$$u = R_{test} f$$

This calculation produces a matrix u, housing glove trace predictions for each of the time bins for each of the five fingers.

The predictions for each of the five fingers in the prediction matrix u were then squared and smoothed to aid in giving better correlation values by removing noise. Squaring the predictions assisted in making the larger values (spikes) more prominent and the low values less prominent. Smoothing was implemented by use of a local regression using weighted linear least squares and second degree polynomial model across all predictions. This smoothing operation was shown to significantly improve the resulting correlation, increasing it at times by a whole 6%. These predictions were then matched up with the actual glove traces and 25 correlation values were obtained, 5 for each fold and finger. This allowed us to gauge how well our algorithm was doing without subjecting it to any sort of bias or over fitting.

### 1.2.1 Testing

We then moved on to the testing data set. The procedure up until the formation of the feature matrix is the same as described above. To reiterate, the ECOG data was first imported, the 7 features were extracted for each time window and the feature matrix was constructed. A linear regression was then performed with this feature matrix generated from the testing ECOG and the training weights matrix. The predictions were then smoothed (as described above) and upsampled from the number of time windows back up to the original number of samples. This upsampling was executed by interpolating the predictions using a cubic spline back up to the original 1000 Hz sampling frequency.

Lastly, since the interpolated predictions were missing samples for the first three time bins (250 samples), the prediction set was padded with the appropriate number of 0s to match it up again with the original number of glove traces.

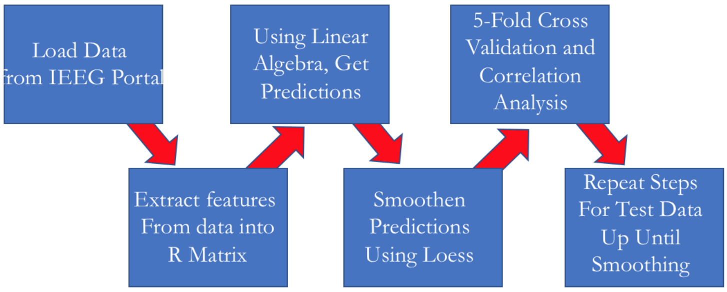


Figure 2: Flowchart of Algorithm

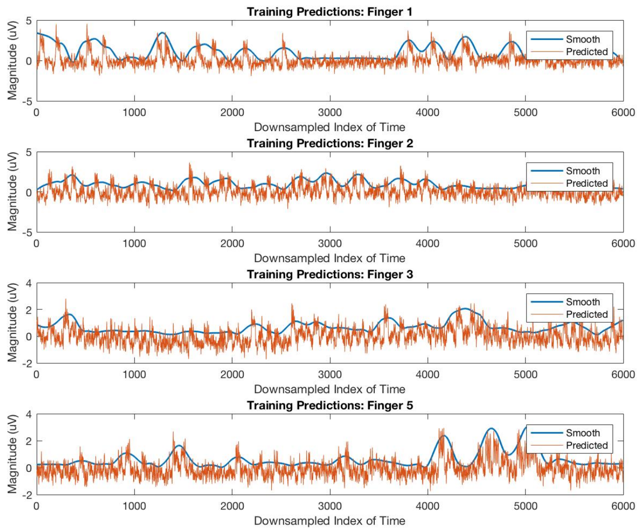


Figure 3: Smoothened Predictions and UnSmoothed Counterpart

### 1.3 Key Implementation

One of the most significant factors that led to good correlation values was the smoothing method implemented in our algorithm. A local regression using weighted linear least squares and second degree polynomial model allowed us to remove a portion of the noise present in the final predictions. In Figure 3 one can see what our predictions looked like before and after implementing this smoothing.

## 2 Challenges and Failures

### 2.1 Lasso

As learned in lecture and taken from recitation content, it was thought that lasso could be applied to our feature matrix (R matrix) such that using the Lasso() function on Matlab would yield the linear regression with least Mean-Squared Error (by reducing to only the "essential" features). However, there is a one-to-one relationship between matrix of features and output, therefore, one lasso regression was conducted for each finger (except the fourth). This led to 4 optimal feature matrices from which we took two approaches: use 4 regularized feature matrices and then combine finalized predictions for all fingers for cross-validation and for testing, or removing features that were deemed insignificant in all 4 cases and returning a feature matrix slightly slimmed to not include globally insignificant features. Note here that by features we are referring to columns in the R matrix, which has only 7 base features but that are calculated over several windows. Unfortunately, neither approach increased testing accuracy, perhaps because of over fitting to training data or simply loss of information by removing columns. Moreover, it took a fairly long time for lasso to be run 4 times so this method was not used.

### 2.2 Preprocessing

Bundy et al[2] successfully implemented prepocessing techniques to the raw data include band-passing the ECOG data between 0.1 and 260Hz through a 3rd order Butterworth Filter. Another 3rd order filter was also used to remove noise harmonics at 60Hz, 120Hz, and 180Hz. Correlations of 0.81 were achieved using signal processing and the team felt optimistic that this slight change in approach would help increase correlation values. Perhaps because the bandpass filter discarded too much ECOG data or because the team implemented a notch filter instead of a 3rd order butterworth, testing accuracy did not increase, even though crossvalidation accuracy did.

### 2.3 Binary Classification

Liao et al [1] found success in their finger flexion predictions by applying a binary classification method as part of their algorithm. This group used a pairwise binary classification, which we found confusing, but inspired our idea of how to apply binary classification. The glove trace data is essentially separated by spikes or no spikes (with some noise). We figured that if we could predict the spikes accurately, losing some accuracy on the noise would not be important. We used a threshold to decide what to consider a spike, and created a binary trace vector for each finger where the spikes are 1 and the rest of the data is 0. We then trained a binary classifier (Ensemble trees, after SVM and k-NN didn't work well) with the feature matrix to predict if a given finger was spiking. We wanted to use these binary predictions to improve the predictions from the linear model (via convolution, multiplication, or addition), as this was already working effectively at this stage of the project. Unfortunately, the binary predictions were not very accurate, and combining them with the linear regression predictions worsened the accuracy. We believe that the major limitation to this approach is that the spike activity is very jagged and oscillates between very high and very low values during finger flexion. Therefore, using a threshold and the findpeaks() function to classify spike and not spike is oversimplified.

### 2.4 Additional Features

Including features such as Area, Energy, and Zero Crossings did not improve correlation. Moreover, additional frequency bands were included to the analysis and those too did not make that much of an impact on testing accuracy.

### 2.5 Ideas for Future

Using the smoothening function, we lose some peaks that are close together, so an idea we had but did not implement was using log transform normalization to spectral data such that the power law it naturally follows would not affect measurements. Furthermore, we thought it would be interesting to investigate the effect of "squeezing" smoothed peaks such that peaks would be sharper and the base would have less spread. This was thought to be achieved by running MovingWinFeats() and using threshold promotion/demotion of values so that anything lying above a set number (finger moved), would be promoted to 150% its value whereas anything below would be demoted to 50% its value. Figure X shows this idea.

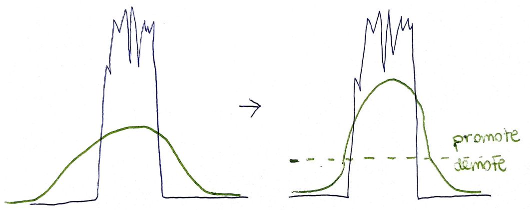


Figure 4: Promoting Demoting Idea for Peak Prominence

# 3 Correlation in 4th Finger

It's difficult to move your ring finger independently from your middle and little fingers because of the interconnectedness in the muscles and the nerves. The muscles that control finger flexion are the two long flexors in the forearm and the lubricals in the hand [3]. The ring finger has no independent flexors, so it can only move in common with the other fingers. Furthermore, the ulnar nerve connects the middle, ring, and little fingers to the brain. The fact that the ring finger lacks independent muscles or nerves leads to the difficulty of moving the ring finger separately from the ring or little fingers.

# 4 Conclusion

The beginning stages of the project were definitely the most frustrating. Not being able to pass checkpoint 1 hurt our morale, but nonetheless, we consistently continued to try and improve our correlation. Our biggest jump in correlation occurred after implementing our post-processing smoothing technique. This substantial improvement and some other tweaks in our feature matrix indexing allowed us to pass checkpoint 2 and obtain a correlation that we were satisfied with. In retrospect, the project as a whole was extremely rewarding due to our ability to overcome our struggle in passing a correlation of 0.33 by climbing all the way to the top of the leaderboard at one point. Furthermore, just to think that these novel algorithms have been/ have the potential of being incorporated in real-world brain computer interfaces such as motor prosthetics is astonishing.

# 5 References

[1] Liao, Ke, et al. "Decoding individual finger movements from one hand using human EEG signals." PloS one 9.1 (2014): e85192.

[2] Warland, David K. et al. "Decoding Visual Information From A Population Of Retinal Ganglion Cells". Journal Of Neurophysiology, vol 78, no. 5, 1997, pp. 2336-2350. American Physiological Society, doi:10.1152/jn.1997.78.5.2336.

[3] Mello, Gwyn. "This Is Why It's So Much Harder To Move Only Your Ring Finger On Its Own Than Other Fingers." Indiatimes.com, India Times, 11 Mar. 2018

[4]Bundy, David T et al. "Decoding Three-Dimensional Reaching Movements Using Electrocorticographic Signals In Humans." Journal of Neural Engineering 13.2 (2016): 026021.

# 6 Appendix

```
% NSO

```

```
% this code is for subject 1 (both training and testing). Other subjects
% used the same format, and our make_predictions.m is a generalized version
% of the testing portion of this code.


%% read in data

tic

% sub1_ecog = IEEGSession('I521_Sub1_Training_ecog', 'anyah', 'any_ieeglogin.bin');
% sub1_glove = IEEGSession('I521_Sub1_Training_dg', 'anyah', 'any_ieeglogin.bin');
%
% sub1_full = zeros(62,300000);
% for i=1:62
%     channel_i = sub1_ecog.data.getvalues(1:300000,i);
%     sub1_full(i,:) = channel_i;
% end

% sub1_new = [sub1_full(1:54,:); sub1_full(56:end,:)];
%
% sub1_glove_data = zeros(300000,5);
% for i=1:5
%     finger_i = sub1_glove.data.getvalues(1:300000,i);
%     sub1_glove_data(:,i) = finger_i;
% end

load('subject1_data.mat'); % contains sub1_glove_data, sub1_full, and sub1_new

samplerate = 1000; SR = samplerate;

sub1_no_art = [sub1_full(1:3,:); sub1_full(5:39,:);
    sub1_full(41:48,:); sub1_full(50:54,:);
    sub1_full(56:end,:);];



%% create R matrix

disp('feature extraction')
NumWins = @(xLen, fs, winLen, winDisp) floor(((xLen-winLen*fs)/(winDisp*fs))+1);
AveTimeDomain = @(x) mean(x);
LLFn = @(x) sum(abs(diff(x)));
AreaFn = @(x) sum(abs(x));
```

```
EnergyFn = @(x) sum(x.^2);
ZXFn = @(x) sum((x(2:end)-mean(x)<0 & x(1:end-1)-mean(x)>0) | ...
      (x(2:end)-mean(x)>0 & x(1:end-1)-mean(x)<0));

winLen = 0.1;
winDisp = 0.05;
xLen = length(sub1_new);
winNum = NumWins(xLen,SR,winLen,winDisp);

num_channels = 58;


time_feats = [];
freq_feats = [];
LL_feats = [];
% Area_feats = [];
for i=1:num_channels
      time_feat_i = MovingWinFeats(sub1_no_art(i,:),xLen,SR,winLen,
          winDisp,AveTimeDomain);
      freq_feat_i = SpecFreq3(sub1_no_art(i,:),SR,winLen,winDisp,winNum);
      LL_feat_i = MovingWinFeats(sub1_no_art(i,:),xLen,SR,winLen,winDisp,
          LLFn);
%       Area_i = MovingWinFeats(sub1_no_art(i,:),xLen,SR,winLen,winDisp,
      AreaFn);
      time_feats = [time_feats; time_feat_i];
      freq_feats = [freq_feats; freq_feat_i];
      LL_feats = [LL_feats; LL_feat_i];
%       Area_feats = [Area_feats; Area_i];
end


N = 3; % 3 time windows prior
F = 7; % 7 features

R = ones(winNum,num_channels*N*F+1);
for i=N+1:winNum
      for j=1:num_channels
            freq_i = freq_feats((5*j)-4:(5*j),i-N:i-1);
            R(i,((j-1)*N*F+2):(j*N*F)+1) = [time_feats(j,i-N:i-1) LL_feats(
                j,i-N:i-1) ...
                 freq_i(:)'];
      end
end
R(1:N,:) = [];

R_final = R;
```

```

% downsample
dataglove1_ds = [];
for i=1:5
    dataglove1_i = decimate(sub1_glove_data(:,i),50);
    dataglove1_ds(:,i) = dataglove1_i(N+1:end-1);
end
s = dataglove1_ds;


%%

f_a = R_final'*R_final;
f_b = R_final'*s;

f = mldivide(f_a,f_b);

u_final = R_final*f;


% preds = [];
% for i = 1:5 % smooth to remove noise
% %     b = u_final(:,i) - mean(abs(u_final(:,i)));
%     b = u_final(:,i) + min(mean(abs(u_final)));
%     b = b.^2;
%     b = smooth(b,0.06,'loess');
%     preds = [preds b];
% end



%% 5-fold cross-validation

disp('cross-validation');

% create folds
R_cross = cell(1,5);
trace_cross = cell(1,5);
intervals = [1, 1201, 2401, 3601, 4801, 5996];
% preds_final = [];
for i = 2:6
    if i == 6
        R_cross{i-1} = R_final(intervals(i-1):intervals(i), :);
        trace_cross{i-1} = s(intervals(i-1):intervals(i), :);
    else
        R_cross{i-1} = R_final(intervals(i-1):intervals(i)-1, :);
        trace_cross{i-1} = s(intervals(i-1):intervals(i)-1, :);
    end
end
```

```matlab

% perform cross-validation
correlation_test = zeros(5,5);
u_tests = [];
trace_tests = [];
for i = 1:5
    R_train = [];
    R_test = [];
    trace_train = [];
    trace_test = [];
    for j = 1:5
        if i == j
            R_test = R_cross{j};
            trace_test = trace_cross{j};
        else
            R_train = vertcat(R_train, R_cross{j});
            trace_train = vertcat(trace_train, trace_cross{j});
        end
    end

    %train test collected
    f_i = mldivide((R_train' * R_train),(R_train' * trace_train));
    u_test = R_test  * f_i;
    u_tests = [u_tests; u_test];
    trace_tests = [trace_tests; trace_test];

    preds1 = [];
    for j = 1:5
%       b = u_final(:,i) - mean(abs(u_final(:,i)));
        b = u_test(:,j) + min(mean(abs(u_test)));
        b = b.^2;
        b = smooth(b,0.06,'loess');
        preds1 = [preds1 b];
    end

    for k = 1:5
%           correlation_test(i,k) = corr(u_test(:,k),trace_test(:,k));
        correlation_test(i,k) = corr(preds1(:,k),trace_test(:,k));
    end
end

mean(mean(correlation_test))
disp('^ 5-fold val accuracy');



figure; subplot(4,1,1)
plot(preds(:,1)); hold on;
plot(s(:,1)); legend('predictions','actual'); hold off;
```

```
subplot(4,1,2)
plot(preds(:,2)); hold on;
plot(s(:,2)); legend('predictions','actual'); hold off;
subplot(4,1,3)
plot(preds(:,3)); hold on;
plot(s(:,3)); legend('predictions','actual'); hold off;
subplot(4,1,4)
plot(preds(:,5)); hold on;
plot(s(:,5)); legend('predictions','actual'); hold off;



%% testing

disp('testing');

%% read in data


% sub1_testing = IEEGSession('I521_Sub1_Leaderboard_ecog', 'anyah', '
    any_ieeglogin.bin');
%
% sub1_test_data = zeros(62,147500);
% for i=1:62
%     test_channel_i = sub1_testing.data.getvalues(1:147500,i);
%     sub1_test_data(i,:) = test_channel_i;
% end
%
% sub1_test_new = [sub1_test_data(1:54,:); sub1_test_data(56:end,:)];


load('sub1_testing_ecog.mat'); % contains sub1_test_data and
    sub1_test_new


SR_test = 1000;
xLen_test = length(sub1_test_new);
winLen_test = 0.1;
winDisp_test = 0.05;
winNum_test = NumWins(xLen_test,SR_test,winLen_test,winDisp_test);

sub1_no_art_test = [sub1_test_data(1:3,:); sub1_test_data(5:39,:);
    sub1_test_data(41:48,:); sub1_test_data(50:54,:);
    sub1_test_data(56:end,:);];

%% create R matrix

time_feats_test = [];
freq_feats_test = [];
```

```
LL_feats_test = [];
% Area_feats_test = [];
for i=1:num_channels
    time_feat_i = MovingWinFeats(sub1_no_art_test(i,:),xLen_test,
        SR_test,winLen_test,winDisp_test,AveTimeDomain);
    freq_feat_i = SpecFreq3(sub1_no_art_test(i,:),SR_test,winLen_test,
        winDisp_test,winNum_test);
    LL_feat_i = MovingWinFeats(sub1_no_art_test(i,:),xLen_test,SR_test,
        winLen_test,winDisp_test,LLFn);
%       Area_i = MovingWinFeats(sub1_no_art_test(i,:),xLen_test,SR_test,
    winLen_test,winDisp_test,AreaFn);
    time_feats_test = [time_feats_test; time_feat_i];
    freq_feats_test = [freq_feats_test; freq_feat_i];
    LL_feats_test = [LL_feats_test; LL_feat_i];
%       Area_feats_test = [Area_feats_test; Area_i];
end

R_test = ones(winNum_test,num_channels*N*F+1);
for i=N+1:winNum_test
    for j=1:num_channels
        freq_i = freq_feats_test((5*j)-4:(5*j),i-N:i-1);
        R_test(i,((j-1)*N*F+2):(j*N*F)+1) = [time_feats_test(j,i-N:i-1)
            LL_feats_test(j,i-N:i-1) ...
            freq_i(:)'];
    end
end
R_test(1:N,:) = [];


%% make predictions & upsample

u_leaderboard = R_test*f; % create predictions

preds_test = []; % smooth to remove noise
for i = 1:5
    b = u_leaderboard(:,i) + min(mean(abs(u_leaderboard))); % bring
        values above 0 so that squaring works as intended
    b = b.^2;
    b = smooth(b,0.05,'loess');
    preds_test = [preds_test b];
end


sub1_upsamp = spline(50.*(1:length(preds_test)),preds_test',(50:50*
    length(preds_test)));

sub1_pad = [zeros(5,200) sub1_upsamp zeros(5,49)];

final_predictions_sub1 = sub1_pad';
```

```
size(final_predictions_sub1)

figure; plot(final_predictions_sub1);
title('sub1 testing predictions');


toc

%% consolidate all data for submission

% load predictions for other subjects
% load('Sub2_Preds_last.mat');
% load('preds_sub3.mat');
%
% predicted_dg = cell(3,1);
% predicted_dg{1,1} = final_predictions_sub1;
% predicted_dg{2,1} = final_predictions_sub2;
% predicted_dg{3,1} = final_predictions_sub3;
```