Classification for Brain Computer Interfaces

Alex Warren
Frank Holler

05/11/2014

ECE 466/566

Professor Marefat

Knowledge Representation Systems

Term Project

Abstract

The advancements in Brain Computer Interface (BCI) open up the promise of locomotion to individuals who have lost either movement through disease or injury. As this technology becomes more advanced, it will allow users an unprecedented level of control of electronic devices. This paper explores the acquisition of Electrocorticographic (ECoG) signals and the techniques for mapping them to movement. Particularly looking at the BCI Competition IV dataset 5, where finger position is predicted from ECoG signals. A comparison of machine learning techniques is made, with analysis of the winner's methods.

1 Introduction

1.1 Brain Computer Interface

Electroencephalography (EEG) is commonly known for its use in measuring brain waves. By placing electrodes on the surface of the head, it is possible to detect the agregate activity of neurons as they fire electrochemical pulses in the cortex. Because the skull acts to insulate and diffuse the electrical activity of the brain, EEG has issues with a low signal to noise ratio. It is further limited to measuring the activity of neurons that send pulses perpendicular to the surface of the scalp. Despite this issues, it is still possible to infer brain states from the frequency and phase relationships of the electrodes.

Alternatively it is possible to place electrodes directly into the brain and get readings from individual or small groups of neurons. This has been done in animal models, but overtime as the electrodes move, scar tissue will build up and the system will become unreliable. ECoG occupies a middle ground between the limited information content of EEG and the biomaterial challenges of buried electrodes. The electrodes are placed directly on the surface of the cortex where it is possible to get a strong signal and good spatial resolution. This can be used for finding the source of epileptic fits and while the array of electrodes are present BCI the patients participate in BCI experiments.

BCI has many potential applications. EEG is well known for measuring the brain waves associated with sleep and wakefulness. It has traditionally been used in medicine and neuroscience but is also being used commercially. EEG can be used to improve marketing campaigns, architectural design, in artistic pieces, and even improve the mindset of athletes. ECoG has all the applications of EEG but does require brain surgery, therefore in the near term its use will be limited to individuals with damaged nervous systems and missing limbs.

1.2 BCI Tools

Given that access to EEG technology and good practice with EEG technology was neither the goal of this project nor a viable option, two programs were used to simulate EEG tests and signals. The two programs used are BCI2000 and bcilab. Both programs work with Matlab for signal processing. Bcilab is a

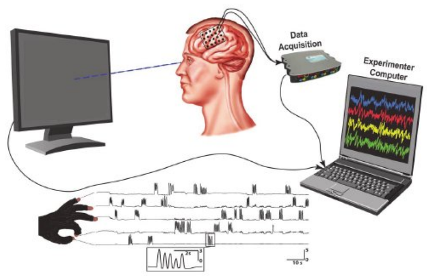

Figure 1: Experimental setup [6].

Matlab toolbox. The idea behind it is that EEG data can be processed using this program in matlab. It was intended to help facilitate research into BCI. BCILAB [2]works by essentially receiving stimulus from a user and predicting what the user is thinking.

BCI2000 [4]was another program that was intended to create an interface with the human brain. Since, we had no access to EEG technology, simulations were used in BCI2000. In the simulation, there is a ball and a rectangle on opposite sides of the screen. The objective of the test is supposed to be that a person connected to the BCI through an EEG rig tries to make the ball move to and make contact with the rectangle. Each run simulated 18 different trials, some trials were successful, others failed. At the the end of each run, a complete log of the number of failed and successful attempts is shown along with the number of transferred bits.

1.3 Acquisition of Signals

Typical acquisition of signals is done through a direct connection to a persons brain with ECoG [3] as seen in 1. The ECoG, working through an array of electrical potential sensors, records different outputs at each sensor in a matrix. Subjects connected to the ECoG rig perform a simple movement(for the competition they were instructed to flex a particular finger). The movement is supposed to last a specific amount and then the subject is supposed to rest for a period of time. This allows the BCI to distinguish between different signals. It takes a number of trials for the BCI to be trained so that it can distinguish the test movement from the other signals picked by the ECoG. After the BCI has been trained, testing can begin to determine if it can match the same movement from the subject. The signals were recorded at 1kHz with a bandpass filter from .15 to 200Hz using the BCI2000 software. The dataglove was recorded at 25Hz. 400 seconds of data was given to competition participants.

1.4 Machine Learning

Machine learning is a broad set of techniques used for interpreting and modeling data. To create a model that will map ECoG data to finger capture data, the general workflow is to start with the raw signal and convert it to a time varying feature vector. This step should reduce the dimensionality of the data, to speed up the next step. Often the goal after creating a feature vector would be to classify the data. In this case we will use linear regression to map the feature to the digit positions. The standard way to validate that a machine learning technique has been successful is to break the data into a training set and a testing set, so that the model tested on data that it was not fitted to.

2 Proir Work

Five winners were chosen that had the best correlation when their model was applied to the test with unknown digit positions. In the implementation we present, our methods are based on the winners, the Cortex Team [1]. They based their methods on the theory that the cortex uses a form of amplitude modulation to encode information, and to extract useful features they used a equiripple Finite Impulse Response (FIR) band pass filter separate the signals into bins of 1-60Hz, 60-100Hz and 100-200Hz and then they computed the power of the signal over 40ms windows. The original signal was sampled at 1kHz and the data glove was sampled at 25Hz, so by combining 40 samples of the ECoG data, the two signals had matching time steps. Finally having created three feature vectors for each channel, they used a stepwise feature selection process, to additively create a set of features to perform linear regression on. They searched for the best feature to add and continued until the correlation on test was not longer increasing or until a threshold of cycles was reached. To find the regression, they used the Wiener model.

The other winners used various methods. For creating features, Auto Regression coefficients on a moving window, band power from Wavelets, Power Spectrum Density and time-domain average of low frequency bandpass. To create a model they used, Ridge Regression, Sparse Linear Regression, Support Vector Regression, and a Directed Acyclic Graph-Support Vector Machine.

3 Methods

For this paper we decided to attempt and reproduce the results of the Cortex Team. The scripts were created in MATLAB and took advantage of it's libraries. The first task was to create feature vectors from the ECoG channels, we used two separate methods for this. In the first method we used Welch's power spectrum density estimate and summed the density over different bins, the ones they used and also alternatives. Welch's method uses FFT over subintervals and calculates an average, the resulting data for one frame can be seen in 3. Calculating this for all the data points was time consuming. The other method was faster and

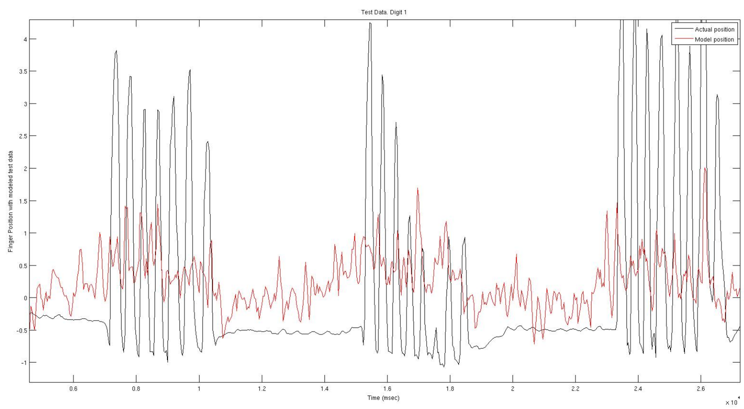

Figure 2: Comparison of predicted thumb position and actual thumb position.

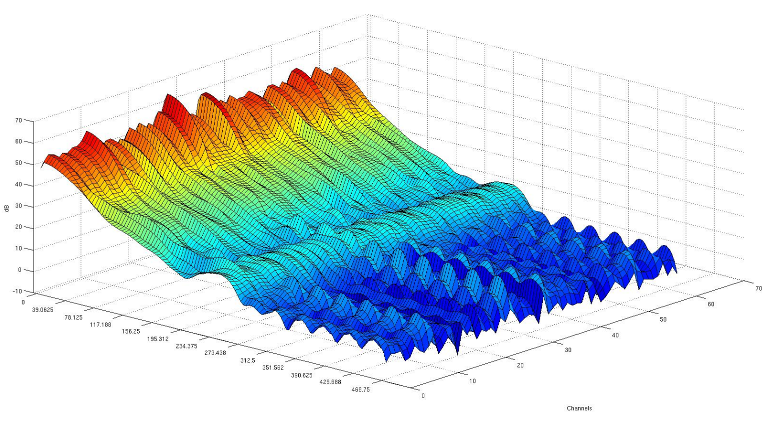

Figure 3: Spectra for one window at each of the 62 channels of ECoG data from Patient 1.

	PSD a	PSD b	PSD c	FIR original	FIR seperate
Thumb	0.40183	0.39985	0.41795	0.31557	0.32756
Index	0.346	0.32221	0.32463	0.22753	0.20718
Ring	0.3461	0.34194	0.3501	0.32231	0.31873

Table 1: Correlation of different configurations. PSD versions used different bins. a) 1-60 60-100 and 100-200Hz b) 1-30 30-90 90-120 120-230Hz c) 1-25 25-50 50-100 100-230Hz. For the Finite Impulse Response, the original feature selection was to choose features by channel and the seperate method selected feature by feature so that channels wouldn't need to go together.

based off of their methods, to use a filter to seperate the channels into band limited signals and then to find the power of each as the sum of squares.

Next, having created multiple feature vectors per channel, the task was to prune the number of features to be used in the final regression. This was done by finding a regression of each feature vector with the finger position and creating a list of those features with the best R^2 value. The top features were then combined for a final regression to determine the coefficients for each feature vector. After this it is possible to predict the finger position with the test data and compare the results to the actual finger position.

An additional issue when determining the best features to use, was also to find the appropriate delay to use. The data glove data was recorded with a delay of 37ms and the brain itself is active before the finger is moved. An estimate of appropriate delays can be gathered from [5] where the experimenters were attempting to predict movement direction. The study found that features varied between subjects with the time delay ranging from 180 to 510 ms. This was incorporated into our work by finding the best performing delay for each feature.

4 Results

The winning entry performed ith a correlation of .46 on the training data. The implementation that we put together was at best able to perform with a correlation of .4 as can be seen in table 1. However the remaining fingers were not included because the results were poor, and to save computation time. A more comprehensive comparison would require that the method be applied to the other two subjects (there were 3 sets of data from different patients). The predicted finger position can be seen in 2.

Comparing the different configurations, we find that the bins used in the third Power Spectrum Density feature create the best results, however to be thorough it would be necessary to compare the methods on each subject and with the full range of data including each finger.

5 Discussion

It is an interesting theoretical question what the best performance that could be achieved with this data would be. It is worth noting that the electrodes are placed differently on each subject and that the recording will be including signals that are unrelated to the finger movements. In each case, electrodes were positioned over the parietal cortex, which is associated with motor control, however neuroscience is not at the point where it can solve the inverse problem of determining what signals are corresponding to in regard to neural processing.

The winning solution obviously performed better than our implementation and in further work it would be interesting to determine what changes could be made to approach that level of correlation across all the data sets. One potential source of difference is in the FIR filter that is used, they did not specify the parameters and these were chosen without testing different possibilities. One interesting direction of further research would be to add meta-learning to the program so that various parameters could be optimized.

Finally other research has shown that unsupervised learning may be useful in the creation of features. In one paper [6] they compared the Band power features to a method based on prior supervised convolutional stacked auto-encoders. However this method is much more complex.

6 Bibliography

References

[1] Laurent Bougrain, Nanying Liang, et al. Band-specific features improve finger flexion prediction from ecog. In *Jornadas Argentinas sobre Interfaces Cerebro Computadora-JAICC 2009*, 2009.

[2] Arnaud Delorme, Tim Mullen, Christian Kothe, Zeynep Akalin Acar, Nima Bigdely-Shamlo, Andrey Vankov, and Scott Makeig. Eeglab, sift, nft, bcilab, and erica: new tools for advanced eeg processing. *Computational intelligence and neuroscience*, 2011:10, 2011.

[3] G Schalk, J Kubanek, KJ Miller, NR Anderson, EC Leuthardt, JG Ojemann, D Limbrick, D Moran, LA Gerhardt, and JR Wolpaw. Decoding two-dimensional movement trajectories using electrocorticographic signals in humans. *Journal of neural engineering*, 4(3):264, 2007.

[4] Gerwin Schalk, Dennis J McFarland, Thilo Hinterberger, Niels Birbaumer, and Jonathan R Wolpaw. Bci2000: a general-purpose brain-computer interface (bci) system. *Biomedical Engineering, IEEE Transactions on*, 51(6):1034–1043, 2004.

[5] Yijun Wang and Scott Makeig. Predicting intended movement direction using eeg from human posterior parietal cortex. In *Foundations of Augmented*

Cognition. Neuroergonomics and Operational Neuroscience, pages 437–446. Springer, 2009.

[6] Zuoguan Wang, Siwei Lyu, Gerwin Schalk, and Qiang Ji. Deep feature learning using target priors with applications in ecog signal decoding for bci. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 1785–1791. AAAI Press, 2013.

7 Appendix

ecog_load.m

```
%Load all the data
clear all
close all
load('../data/sub1_comp.mat');
%load('../data/sub1_testlabels.mat');
clear test_data % for speed we will just split up the training data

% train_data = train_data(1:10000,:);
% train_dg = train_dg(1:10000,:);

%Subtract Channel means to remove dc bias
train_data_means = mean(train_data);
train_data = train_data-train_data_means(ones(1,size(train_data, 1)),:);

num_train_points = floor(size(train_data, 1)*.7);
num_train_channels = size(train_data, 2);
num_test_points = size(train_data, 1) - num_train_points;

temp_a = train_data;
temp_b = train_dg;


%Split the data into training and testing
train_data = temp_a(1:num_train_points, :);
train_dg = temp_b(1:40:num_train_points, :);
test_data = temp_a(num_train_points+1:end, :);
test_dg = temp_b(num_train_points+1:40:end, :);
clear temp_a temp_b
```

ecog_learn.m

```
%Script for comparing different frequency bands
%using Welch's PSD for features
tic
disp('load')
ecog_load;
toc
disp('bins 1')
bins = [1 60; 60 100; 100 200];
features = psdFeature(train_data, bins, zeros(62,1));
ecog_regression
toc

disp('bins 2')
```

```
bins = [1 30; 30 90; 90 120; 120 230];
features = psdFeature(train_data, bins, zeros(62,1));
ecog_regression
toc

disp('bins 3')
bins = [1 25; 25 50; 50 100; 100 230];
features = psdFeature(train_data, bins, zeros(62,1));
ecog_regression
toc
```

ecog_learn_fir.m

```
% Compare different feature selection methods
% with FIR filter for features.
tic
disp('load')
ecog_load;
toc
disp('FIR with original selection')
bins = [1 60; 60 100; 100 200];
features = firFeature(train_data, bins, []);
ecog_regression
toc

disp('FIR with individual feature selection')
bins = [1 60; 60 100; 100 200];
features = firFeature(train_data, bins, []);
ecog_regression_2
toc
```

ecog_regression.m

```
%Selected the best features and test the results
time_step = 40; %ms
num_sample_result = size(features, 1);
num_bins = size(bins,1);
%we are looking at digits 1, 2 and 4
for ii = [1 2 4]
    top_regressions = zeros(6, 3);
    disp([' Digit: ' num2str(ii)])
    for delay = [160 200 240 280 320 360 400 440 480 520] %try differnt delays
        idx_delay = ceil(delay/time_step);
        constant_col = ones(num_sample_result-idx_delay+1,1); %A constant term for the
regression
```

```
    for jj = 1:num_train_channels
        [b, ~, ~, ~, stats] = regress(train_dg(1:end-idx_delay+1,ii), ...
                horzcat(features(idx_delay:end,(jj-1)*num_bins+1:(jj)*num_bins),constant_col));
        %Update the list of top regressions
        top_regressions = sortrows(top_regressions, 1);
        if stats(1)>top_regressions(1,1)
            flag = 0;
            for kk = 1:size(top_regressions,1)
                if top_regressions(kk,3) == jj
                    flag = 1;
                    if stats(1)>top_regressions(kk,1)
                        top_regressions(kk,:) = [stats(1) delay jj];
                    end
                    break;
                end
            end
            if ~flag
                top_regressions(1,:) = [stats(1) delay jj];
            end
        end
    end
end
%Get the features ready for a final regression
top_regressions = sortrows(top_regressions, 3);
selected_channels = top_regressions(:,3)';
temp = (selected_channels-1).*num_bins+1;
selected_features = zeros(num_bins, numel(selected_channels));
for jj = 1:num_bins
    selected_features(jj, :) = temp+jj-1;
end
selected_features = selected_features(:);
num_selected_features = size(selected_features,1);
delays = top_regressions(:,2)';
idx_delays = delays./time_step;
idx_delays = repmat(idx_delays, num_bins, 1);
idx_delays = idx_delays(:);
final_train_features = ones(num_sample_result, num_selected_features+1);
for jj = 1:num_selected_features
    idx_delay = idx_delays(jj);
    final_train_features(1:end-idx_delay, jj) = features(idx_delay+1:end, selected_features(jj));
    final_train_features(end-idx_delay:end, jj) = features(end, selected_features(jj));
end
%Find the best regression
[b, ~, ~, ~, stats] = regress(train_dg(:,ii), final_train_features);
disp(['R^2 ' num2str(stats(1))])
b
```

```
    % final_train_features_2 = ones(size(final_train_features));
    % final_train_features_2(1:num_train_points/40,1:end-1) = psdFeature(train_data(:,
selected_channels), bins, delays);

    %Plot the results on the training data
    expected_dg = final_train_features*b;
    figure
    x = 0:40:(size(train_dg,1)-1)*40;
    plot(x, train_dg(:,ii), 'k')
    hold all
    plot(x, expected_dg, 'r')
    xlabel('Time (msec)');
    ylabel('Finger Position with modeled training data');
    str = sprintf('Training Data. Digit %d', ii);
    title(str)
    legend('Actual position', 'Model position')
    correlation = corr(expected_dg, train_dg(:, ii));
    disp(['Training data correlation: ' num2str(correlation)])

    %Find the predicted finger position for the testing data
    test_features = ones(floor(num_test_points/40), num_selected_features+1);
    delay_per_channel = repmat(delays, num_bins, 1);
    test_features(:, 1:num_selected_features) = firFeature(test_data(:, selected_channels), bins,
delay_per_channel(:));
    %test_features(:, 1:num_selected_features) = psdFeature(test_data(:, selected_channels),
bins, delays);
    expected_dg = test_features*b;

    %Plot the results on the testing data
    figure
    x = 0:40:num_test_points-40;
    plot(x, test_dg(:,ii), 'k')
    hold all
    plot(x, expected_dg, 'r')
    xlabel('Time (msec)');
    ylabel('Finger Position with modeled test data');
    str = sprintf('Test Data. Digit %d', ii);
    title(str)
    legend('Actual position', 'Model position')
    correlation = corr(expected_dg, test_dg(:, ii));
    disp(['Testing data correlation: ' num2str(correlation)])
end

%correlation with a flat line is basically 0
%corr(mean(test_dg(:, 1)).*ones(size(test_dg,1),1), test_dg(:, 1))
```

ecog_regression_2.m

```
%The main difference between this and the first
% version is that this one allows features to be used
% seperately
format long
time_step = 40; %ms
num_sample_result = size(features, 1);
num_features = size(features, 2);
num_bins = size(bins,1);
warning('off', 'stats:regress:RankDefDesignMat')
%we are looking at digits 1, 2 and 4
for ii = [1 2 4]
   top_regressions = zeros(16, 4);
   disp([' Digit: ' num2str(ii)])
   for delay = [160 200 240 280 320 360 400 440 480 520] %try differnt delays
      idx_delay = ceil(delay/time_step);
      constant_col = ones(num_sample_result-idx_delay+1,1);
      for jj = 1:num_features
         %This will give warning about rank about ten times for 11 delays * 186 features
         [b, ~, ~, ~, stats] = regress(train_dg(1:end-idx_delay+1,ii), ...
                        horzcat(features(idx_delay:end, jj), constant_col));
         %Update the list of top regressions
         top_regressions = sortrows(top_regressions, 1);
         if stats(1)>top_regressions(1,1)
            flag = 0;
            for kk = 1:size(top_regressions,1)
               if top_regressions(kk,4) == jj
                  flag = 1;
                  if stats(1)>top_regressions(kk,1)
                     top_regressions(kk,:) = [stats(1) delay ceil(jj/3) jj];
                  end
                  break;
               end
            end
            if ~flag
               top_regressions(1,:) = [stats(1) delay ceil(jj/3) jj];
            end
         end
      end
   end
   %Get the features ready for a final regression
   top_regressions = sortrows(top_regressions, 4);
   selected_features = top_regressions(:,4);
   num_selected_features = size(selected_features,1);
```

```
    delays = top_regressions(:,2)';
    idx_delays = delays./time_step;
    final_train_features = ones(num_sample_result, num_selected_features+1);
    for jj = 1:num_selected_features
        idx_delay = idx_delays(jj);
        final_train_features(1:end-idx_delay, jj) = features(idx_delay+1:end, selected_features(jj));
        final_train_features(end-idx_delay:end, jj) = features(end, selected_features(jj));
    end

    %Find the best regression
    [b, ~, ~, ~, stats] = regress(train_dg(:,ii), final_train_features);
    disp(['R^2 ' num2str(stats(1))])
    b

    % final_train_features_2 = ones(size(final_train_features));
    % final_train_features_2(1:num_train_points/40,1:end-1) = psdFeature(train_data(:,
selected_channels), bins, delays);

    %Plot the results on the training data
    expected_dg = final_train_features*b;
    figure
    x = 0:40:(size(train_dg,1)-1)*40;
    plot(x, train_dg(:,ii), 'k')
    hold all
    plot(x, expected_dg, 'r')
    xlabel('Time (msec)');
    ylabel('Finger Position with modeled training data');
    str = sprintf('Training Data. Digit %d', ii);
    title(str)
    legend('Actual position', 'Model position')
    correlation = corr(expected_dg, train_dg(:, ii));
    disp(['Training data correlation: ' num2str(correlation)])

    %Find the predicted finger position for the testing data
    test_features = ones(floor(num_test_points/40), num_selected_features+1);
    delays = zeros(num_features, 1);
    for jj = 1:num_features
        idx = find(jj==top_regressions(:,4));
        if idx
            delays(jj) = top_regressions(idx,2);
        else
            delays(jj) = num_test_points;
        end
    end
    temp = firFeature(test_data, bins, delays);
    test_features(:, 1:num_selected_features) = temp(:, selected_features);
```

```
    expected_dg = test_features*b;

    %Plot the results on the testing data
    figure
    x = 0:40:num_test_points-40;
    plot(x, test_dg(:,ii), 'k')
    hold all
    plot(x, expected_dg, 'r')
    xlabel('Time (msec)');
    ylabel('Finger Position with modeled test data');
    str = sprintf('Test Data. Digit %d', ii);
    title(str)
    legend('Actual position', 'Model position')
    correlation = corr(expected_dg, test_dg(:, ii));
    disp(['Testing data correlation: ' num2str(correlation)])
end

%correlation with a flat line is basically 0
%corr(mean(test_dg(:, 1)).*ones(size(test_dg,1),1), test_dg(:, 1))
```

psdFeature.m

```
function features = psdFeature(data, bins, delays)
% psdFeature Creates a matrix of feature vectors based on a moving window
%            power spectra desity. Uses Welch's method to create the PSD.
%
% Currently the data is assumed to be sampled at 1kHz and this will create
% a feature for every 40 samples.
%
% INPUTS
% data    Matrix with each row an observation and each column a seperate channel.
% bins    Each row is a bin for summerizing the PSD, first column is start
%         second column is the end. In units of frequency.
% delays   Each element of this vector matches with a channel of the data
%         signal, and represents a delays in milliseconds.
%
% OUTPUTS
% features  A matrix of time varying feature vectors, each row represents
%            the features at a seperate time step. And the Channels are
%            split into one feature value for each bin.

samples_per_feature = 40;
window_size = (128-samples_per_feature)/2;
num_data_channels = size(data,2);
num_data_points = size(data,1);
```

```
time_step = 40; %ms


%bins = [1 60; 60 100; 100 200];
num_bins = size(bins,1);
num_sample_result = floor(num_data_points/samples_per_feature);
features = zeros(num_sample_result, num_data_channels*num_bins);
for ii = 1:num_sample_result
    window_start = (ii*samples_per_feature-window_size+1);
    window_end = (ii+1)*samples_per_feature+window_size;
    if window_start<1
        window_end = window_end - window_start + 1;
        window_start = 1;
    end
    if window_end > num_data_points
        window_start = window_start - (window_end - num_data_points);
        window_end = num_data_points;
    end

    for jj = 1:num_data_channels
        [spec, w] = pwelch(data(window_start:window_end,jj), [], [], [], 1000);
        idx_delay = ceil(delays(jj)/time_step);
        if idx_delay>=ii
            continue %don't start yet
        end
        for kk = 1:num_bins
            idx_s = find(w>=bins(kk,1),1);
            idx_e = find(w>=bins(kk,2),1);
            features(ii-idx_delay, num_bins*(jj-1)+kk) = sum(spec(idx_s:idx_e))/(w(idx_e)-
w(idx_s));
            if num_sample_result == ii & idx_delay ~= 0
                %copy final state to fill in remaining
                features(ii-idx_delay+1:end, num_bins*(jj-1)+kk) = features(ii-idx_delay,
num_bins*(jj-1)+kk);
            end
        end

    end

end
```

firFeature.m

```
function features = firFeature(data, bins, delays)
% psdFeature Creates a matrix of feature vectors based on the power of the
%                signal after using an equiripple bandpass filter.
```

```
%
% Currently the data is assumed to be sampled at 1kHz and this will create
% a feature for every 40 samples.
%
% INPUTS
% data    Matrix with each row an observation and each column a seperate channel.
% bins    Each row is a bin for summerizing the PSD, first column is start
%           second column is the end. In units of frequency.
% delays   Each element of this vector matches with a feature of the data
%           signal, and represents a delays in milliseconds.
%
% OUTPUTS
% features  A matrix of time varying feature vectors, each row represents
%             the features at a seperate time step. And the Channels are
%             split into one feature value for each bin.

samples_per_feature = 40;
time_step = 40; %ms
filter_transition = 10;
num_data_channels = size(data,2);
num_data_points = size(data,1);


%bins = [1 60; 60 100; 100 200];
num_bins = size(bins,1);
num_features = num_data_channels*num_bins;
num_sample_result = floor(num_data_points/samples_per_feature);
features = zeros(num_sample_result, num_features);

if numel(delays) < num_features
    delays = zeros(num_data_channels*num_bins, 1);
end

for ii = 1:num_bins
    f_pass1 = bins(ii, 1);
    f_pass2 = bins(ii, 2);
    if f_pass1-filter_transition < 1
        filter_d = fdesign.lowpass('Fp,Fst,Ap,Ast', f_pass2, f_pass2+filter_transition, 6, 60, 1000);
    else
        filter_d = fdesign.bandpass('Fst1,Fp1,Fp2,Fst2,Ast1,Ap,Ast2', f_pass1-filter_transition,
f_pass1, f_pass2, f_pass2+filter_transition,60,6,60,1000);
    end
    bin_filter = design(filter_d, 'equiripple');
    filtered_data = filter(bin_filter, data).^2;
    for jj = 1:num_data_channels
        feature_idx = (jj-1)*num_bins+ii;
```

```
        if delays(feature_idx)/time_step >= num_sample_result
            continue
        end
        idx_delay = ceil(delays(feature_idx)/time_step);
        for kk = 1:num_sample_result
            if idx_delay>=kk
                continue %don't start yet
            end
            features(kk-idx_delay, feature_idx) = sum(filtered_data((kk-
1)*samples_per_feature+1:kk*samples_per_feature,jj));
            if num_sample_result == kk & idx_delay ~= 0
                %copy final state to fill in remaining
                features(kk-idx_delay+1:end, feature_idx) = features(kk-idx_delay, feature_idx);
            end
        end
    end
end
```