BE 521 Final Report

Team: 521 Goldfinger

Team Members:
Yili Du
Aidi Liu
Xinyue Wang

05/06/2020

I. Introduction

In this final project for BE 521, we followed the guideline in the 4th Brain-Computer Interface Data Competition to create our own algorithm to predict finger flexion for three individual subjects (Miller & Schalk, 2008). The algorithm we created is called logistic-weighted random forest, which utilizes the classification results from logistic regression as a weight, and then combines the nonlinear regression results from random forest to generate a final data glove prediction. LASSO was incorporated in the logistic regression to allow variable selection and regularization. The addition of random forest is necessary in that it can provide non-linear prediction results required by the problem, while avoid overfitting commonly occurred in polynomial regression. The entire algorithm includes data pre-processing, feature extraction, feature processing, model training, prediction, cross-validation, and post-processing (Figure 1). Detailed steps were presented in the following section.

II. Algorithm Description

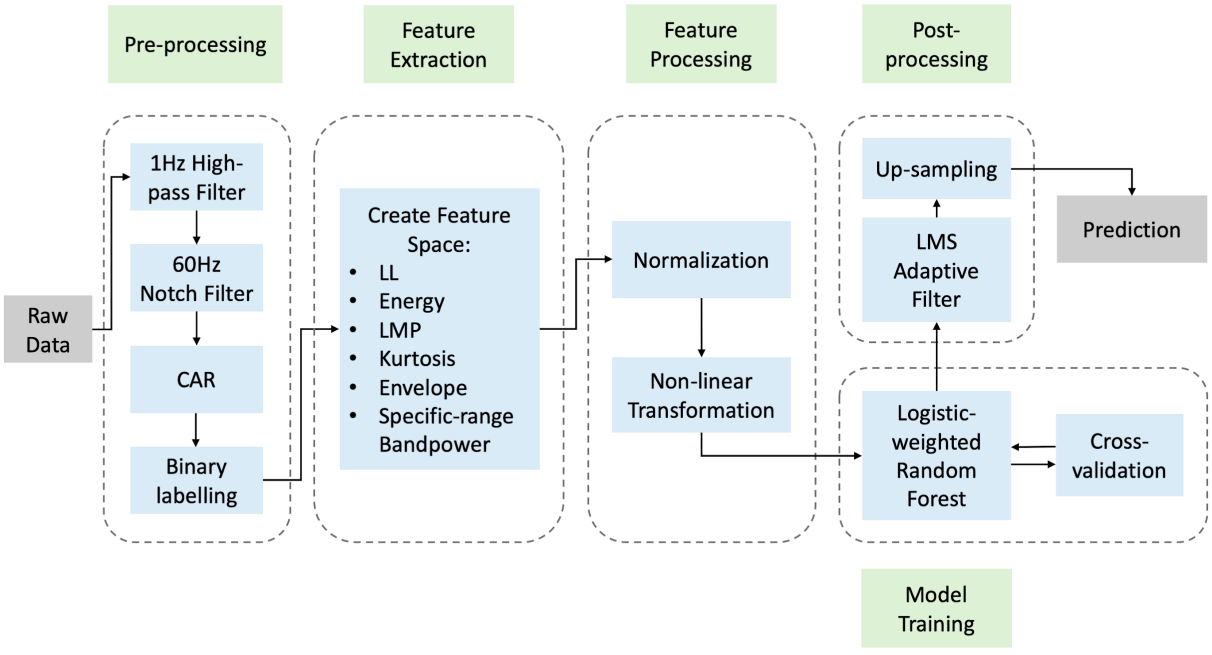

Figure 1. Algorithm flow chart

1. Pre-processing

We first filtered the raw data by applying a 1 Hz high-pass filter to remove noise caused by muscle artifacts or DC drift, and a 60 Hz notch filter to eliminate power noise. Then a computationally simple method, common average reference (CAR), was used to generate a baseline reference electrode across all channels. The data glove (DG) signals were binary labelled, where the labelling threshold for each finger was determined by optimal correlation with actual flexion angle.

2. Feature Extraction

We examined and selected 6 features: line length, energy, local motor potential, specific range band power, Kurtosis, and envelope. Line length, energy, local motor potential, and Kurtosis were calculated from the windowed data directly. The windowed data were decomposed into 5 different frequency ranges (5-15Hz, 20-25Hz, 75-115Hz, 125-160Hz, 160-175Hz) and obtained the corresponding band power in each of them (Liang & Bougrain, 2012). The upper envelope of the windowed data was calculated and the median was used in the feature space.

3. Feature Processing

First, we applied normalization on the extracted features. The normalized feature space was then nonlinear transformed through sigmoid function to better comply with the following logistic-weighted regression model.

4. Model & Prediction

The model we used was logistic-weighted random forest, which is achieved by multiplying the results obtained from logistic regression by those from random forest. This method was inspired from the logistic-weighted regression proposed by Weixuan Chen (Chen, Liu, & Litt, 2014). Regularization and variable selection through LASSO were included in the logistic regression process in order to increase prediction accuracy and model robustness.

5. Post-processing

After making the prediction, the results were filtered by a Least Mean Square (LMS) adaptive filter to mimic actual data glove recording. The filtered predictions were up-sampled to match the original length of the DG signal.

6. Cross-validation

Parameters and threshold used in random forest (nTree, minLeafSize), adaptive filter, and LASSO were optimized and determined by 5-fold cross-validation and grid search.

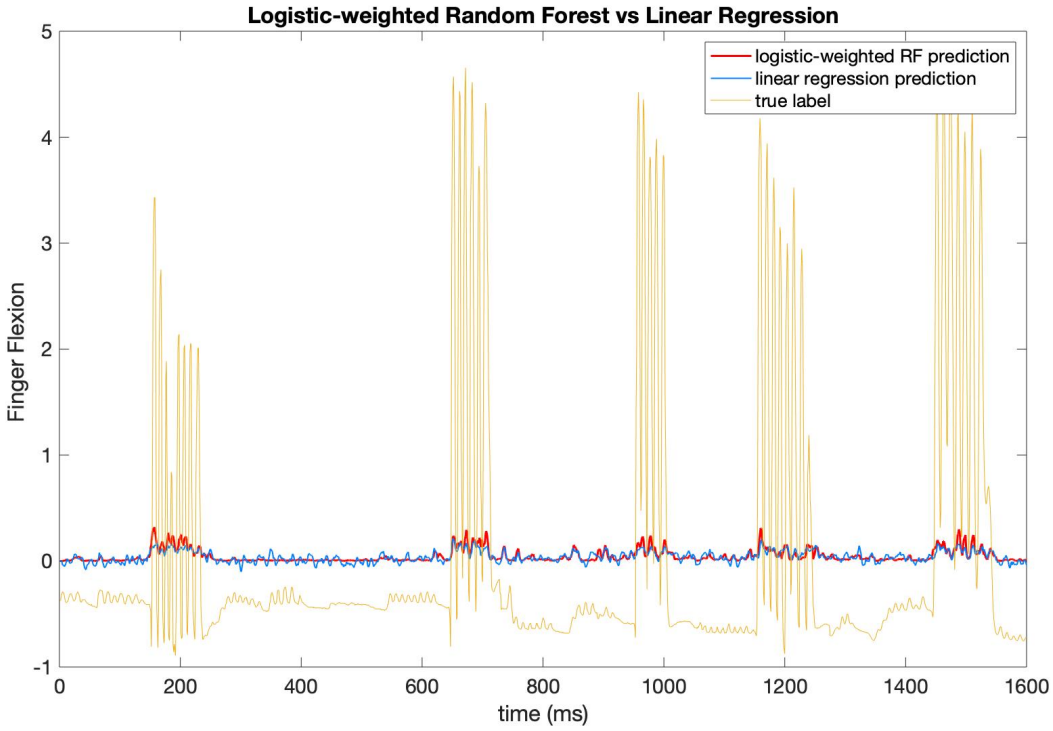

Figure 2. Logistic-weighted RF vs linear regression

III. Discussion

Before ending up using logistic-weighted random forest, we tried ensemble learning methods such as adaboosting and stacking. Forward or backward time-shift addition, channel-specific training, and channel selection were also implemented and tested. However, these methods either did not yield

ideal prediction or did not perform stable enough across all fingers in all subjects. Figure 2 demonstrates the performance of logistic-weighted random forest in comparison to linear regression. Signals predicted from logistic-weighted random forest at the non-target area (period without flexion) are less noisy than that of linear regression prediction, and thus matches closer to the true label. Figure 3 shows the effect of feature normalization as well as nonlinear transformation in the step of feature processing. Note that both normalized and transformed features were plotted according to the red axis on the right. Figure 4 presents the result of using the LMS adaptive filter to mimic real data glove signals.

The reason why the ring finger's flexion is so correlated with the movement of the little finger and the middle finger could be that the ring finger does not have an independent extensor muscle. Therefore, it will have involuntary flexion together with the intended flexion of the middle finger or the little finger upon stimulation or cue (Hager-Ross & Schieber, 2000).

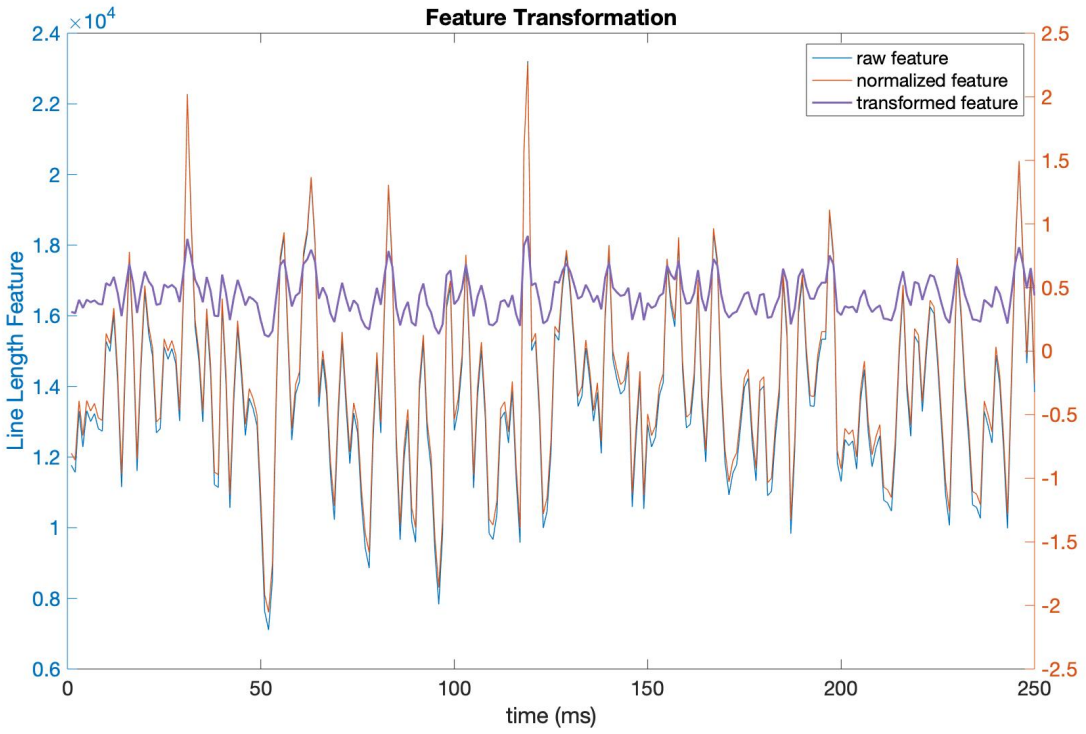

Figure 3. Feature nonlinear transformation effect

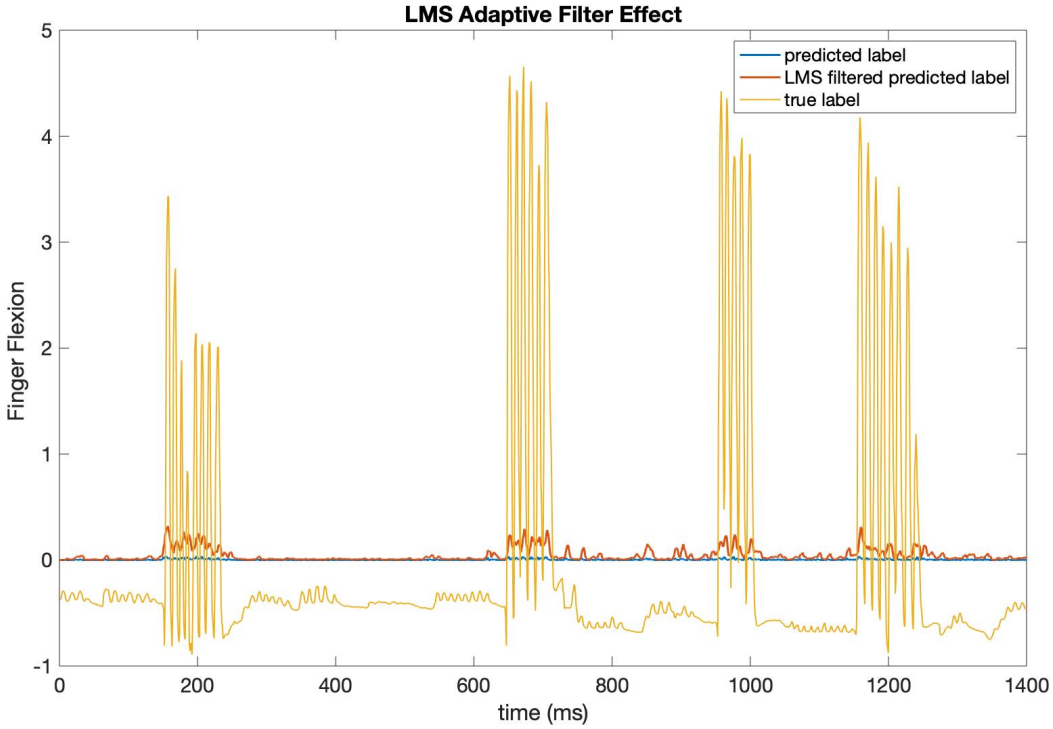

Figure 4. LMS adaptive filter effect

IV. Conclusion

Reflecting backward, we all agree that we enjoy this final project and did learn a lot during the process of implementing an algorithm by ourselves. It is sometimes painful when getting stuck on a correlation coefficient of 0.43, and disappointed after trying multiple methods but still getting zero progress. However, these difficulties made us explore more beyond simply using a certain method and to think twice about why we choose it instead of one another. During this process, we gained more thorough understanding of the entire system of brain signal processing and analysis, as well as the usage of different classification or regression models. Even after passing the second checkpoint, we kept digging into our algorithm and optimizing it. This is a valuable experience and we hope this competition can be kept in BE521 in future semesters.

V. Appendix

The code is available on github: https://github.com/YiliDu/521-Goldfinger

Data Preprocessing

```
function [ecog_CAR] = get_CAR(raw_ecog)
    % Function that re-reference EcoG data to a common average reference.
    %   Re-referencing is achieved by creating an average of all scalp channels
    %   and subtracting the resulting signal from each channel.
    %          Input: raw EcoG signal
    %         Output: EcoG signal after CAR

    %%
    % get sample number and channel number for each subject
    [~,chNum] = size(raw_ecog);

    % re-referenced the signals using a common average reference (CAR)
    ecog_CAR = zeros(size(raw_ecog));

    for ch = 1:chNum
        ecog_CAR(:,ch) = raw_ecog(:,ch) - mean(raw_ecog, 2);
    end
end
```

Feature Extraction

```
function [features] = get_features(clean_data,fs)
% Function that calculates features.
    %         Input:    clean_data: (samples x channels)
    %                         fs:  sampling frequency
    %         Output:   features:   (1 x (channels*features))

%% define features
     % Line Length
     LLFn = sum(abs(diff(clean_data)));
    % Local Motor Potential
     LMP = mean(clean_data);
    %Energy feature (reflect variance)
     E = sum(clean_data.^2);
    % Specific-range Bandpower
    BP1 = bandpower(clean_data,fs,[5 15]);
    BP2 = bandpower(clean_data,fs,[20 25]);
    BP3 = bandpower(clean_data,fs,[75 115]);
    BP4 = bandpower(clean_data,fs,[125 160]);
    BP5 = bandpower(clean_data,fs,[160 175]);
     % Kurtosis
    Kurt = kurtosis(clean_data);
    % Envelope (use median)
    [yupper,ylower] = envelope(clean_data);
    UpperEnv = median(yupper);

 %% create feature space
     features = [LLFn , LMP, E, BP1, BP2, BP3, BP4, BP5,Kurt, UpperEnv];
end
```

Feature Transformation

```
function [proFeats2] = pro_featProcess(feats1,feats2)
% Function that normalizes test features using training features, nonlinear-transform
% feature using sigmoid function.
%              Input:  feats1: features matrix of training data
%                      feats2: features matrix of testing data
%             Output: proFeats2: processed features of test data
%%
```

```
    %%  normalize using mean and std of  training data
    proFeats2 = (feats2-repmat(mean(feats1),length(feats2(:,1)),1))./std(feats1);
    %%  nonlinear transform
    proFeats2 = sigmoid(proFeats2);
end
```

Model training:

1. Get binary labels

```
function binData = finger2bin(dataDg)
% Function that creates binary label for training data  from actual  flexion label
%            Input: raw_train_dg
%            Output: binary label
%% get optimal binary threshold
    op = getThreshold(dataDg);
    binData = [];
  for kk = 1:5
      temp1 = dataDg(:,kk);
      temp1(find(temp1<op(kk))) = 0;
      temp1(find(temp1>=op(kk))) = 1;
     binData = [binData temp1];
  end
end

function op = getThreshold(tempp)
% Function that optimizes the binary threshold for each finger to have the highest correlation with actual finger flexion
%             Input: raw_train_dg
%            Output: optimal binary threshold
%% create binary labels
    for kk = 1:5
        jj = 1;
        for ii = 0:0.01:2 % range in 0-2 and have step-size of 0.01
            temp1 = tempp(:,kk);
            temp1(find(temp1<ii)) = 0;
            temp1(find(temp1>ii)) = 1;
            [rML,pMLval] = corr(temp1,tempp(:,kk));
            temp2(jj,1) = rML;
            temp2(jj,2) = ii;
            jj = jj+1;
        end
        temp3 = temp2(find(temp2(:,1)==max(temp2(:,1))),2);
        op(kk) = min(temp3);
    end
end
```

2. Logistic Regression + Random Forest Regression

```
function [testLabel,lr_train,factor] = fingerGoldening(trainFeats,trainLabel,testFeats,model,parameter)
% model:  'nlr': logistic-weighted random forest regression
% parameter: a cell that contain the model parameters
%            'lr': alpha, error-correction coefficient
%            'rf': {ntree, minLeafsize, NumPredictorsToSample}
    If strcmp(model,'nlr')
    %       '---->Logistic-weighted Random Forest  Regression'
        %% get weight from logistic regression
        alpha = parameter;
        lr_train = fitclinear(trainFeats,trainLabel(:,1),'Learner','Logistic');
        [trash,zone_lr] = predict(lr_train,testFeats);
        zone_lr = zone_lr(:,2);
        if alpha == 1
            MRpredict = zone_lr.^(1+std(zone_lr));
        elseif alpha == -1
            MRpredict = zone_lr.^(1-std(zone_lr));
        else
            MRpredict = zone_lr;
```

```
        end
        'end LR'
        %% get regression prediction from random forest
        factor = TreeBagger(500,trainFeats,trainLabel,'Method','regression','minLeafSize',50);
        [testLabel,scores] = predict(factor,testFeats);
        testLabel = testLabel.*MRpredict;
    end
end
```

3. Least Mean Squared (LMS) Adaptive filter

```
function betterResult = postProcessResult(xn,dn,order,mu)
% LMS TO make it more like the real finger
    % xn:n*1 windows' label vector(fake finger)
    % dn:the windows' binary label in the train(real finger)
 %%  get LMS filtered prediction
    rho_max = max(eig(xn*xn.'));   % max eigenvalues of testing prediction
    mu = mu*(1/rho_max);    % convergence factor 0 < mu < 1/rho
    temp = LMSFilter(xn,dn,order,mu);
    betterResult = [zeros(order-1,1);temp(order:end)];
end


function filterFinger = LMSFilter(testData,trainData,order,step)
% LMS Filter implements an adaptive filter
% input:
%     testData(xn)   testing prediction             (column vector)
%     trainData(dn)  desired binary responses from training data (column vector)
%     order      filter order                       (scalar)
%     step       convergence factor(step size)      (scalar)
%                Note: 0 < step < 1/ max_eigenvalue(xn)
%     itr        number of iterations               (scalar)
%                Note: default as length(xn),order<itr<length(xn)
% output:
%     filterFinger(yn)   filtered input signal      (column vector)

%% initialize parameters
    itr = length(testData); %default as length(xn)
    en = zeros(itr,1); %error between output signal (yn) and desired signal (dn)
    W  = zeros(order,itr); %filter weights

    %% optimize filter weights to minimize the error
    for k = order:itr
        x = testData(k:-1:k-order+1); %get order number of input signal
        y = W(:,k-1).' * x;  %output filtered signal

        %en(k) is result of subtracting the output signal from the desired
        %signal at kth iteration
        en(k) = trainData(k) - y ;

        %iterative calculation of filter weights
        W(:,k) = W(:,k-1) + step*en(k)*x;
    end

   %% return output signal using optimal weights
    filterFinger = inf * ones(size(testData));
    for k = order:length(testData)
        x = testData(k:-1:k-order+1);
        filterFinger(k) = W(:,end).'* x;
    end
end
```

4. Interpolate predicted labels back to original sampling frequency

```
function predictLabel = up_sample(down_predict,ori_signal,window_length,window_overlap,fs)
    %   Function that upsamples underlying prediction back to original sampling frequency.
```

```
    %       Input:  down_predict: prediction from downsampled signal
    %               ori_signal: original EcoG data
    %       Output: interpolated prediction
    %%
    lose = rem((length(ori_signal))-(window_length-window_overlap)*fs,window_overlap*fs);
    predict_temp = [];
    predictLabel = [];
    [~,f_num] = size(down_predict);
    for ii = 1:f_num
        predict_temp(:,ii) = zoInterp(down_predict(1:end-1,ii)',(window_length-window_overlap)*fs);
        predictLabel(:,ii) = [zeros(lose,1);...
            predict_temp(:,ii);...
            zoInterp(down_predict(end,ii),window_length * fs)'];
end
```

VI. Bibliography

Chen, W., Liu, X., & Litt, B. (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*, 2629–2632. https://doi.org/10.1109/EMBC.2014.6944162

Hager-Ross, C., & Schieber, M. H. (2000). Quantifying the independence of human finger movements: Comparisons of digits, hands, and movement frequencies. *Journal of Neuroscience*, *20*(22), 8542–8550. https://doi.org/10.1523/jneurosci.20-22-08542.2000

Liang, N., & Bougrain, L. (2012). Decoding finger flexion from band-specific ecog signals in humans. *Frontiers in Neuroscience*, *6*(JUN), 1–6. https://doi.org/10.3389/fnins.2012.00091

Miller, K. J., & Schalk, G. (2008). Prediction of Finger Flexion 4 th Brain-Computer Interface Data Competition. *New York*, 5–6. Retrieved from http://www.bbci.de/competition/iv/desc_4.pdf