# BMEG3330: Neuroengineering
# Final Project Report

**SAHA, Anujit- 1155163076**
**Kim Hee Won – 1155172023**
**Mohammad Tahmid Ul Huq - 1155165056**
**COMIA Leon Arete – 1155174165**
**Kei Shue Chak Joshua - 1155173228**

## Table of Content

## Section 1- System Design and Plan

### 1.1 Introduction

A Brain-Computer Interface (BCI) is a type of technology that bridges the brain with external devices. It is a vital tool in neuroscience research, allowing researchers to investigate brain activity patterns associated with specific tasks, cognitive processes, or neurological conditions. It can also aid individuals with severe motor disabilities to regain communication and control over their bodies and is also used in neurorehabilitation.

A brain-to-device type BCI measures and analyzes the activity of the Central Nervous system and extracts signal features that correlate with the user's intent. It then converts the signals into artificial outputs such as commands, waveforms, or graphs. A device-to-brain BCI modulates brain signals through direct physical stimulation such as Transcranial Electrical Stimulation, Transcranial magnetic stimulation, Transcranial ultrasound stimulation or direct brain stimulation, which changes the ongoing interactions between the CNS and its external or internal environment [7].

In our project, an EEG set-up is used to detect brain signals, and its output is used to control the state of a character in a video game. The BCI first detects and classifies brain signals in response to different actions. This is achieved through classification algorithms that were used in order to classify and translate the neural activities or intentions into specific outputs as explained in detail in Section 2.3. This report also covers extensively the features of received EEG data that were used to train our classification model. Our report concludes by evaluating the performance of our classifier and system as demonstrated at the BCI competition.

### 1.2: System design

Our project's first stage was to choose a subject and gather their EEG data for the purpose of training a model. The EEG data would be based on the difference in motor imagery of four distinct actions as explained more in Section 1.4. The data acquisition involved certain preprocessing procedures to make sure the two electrodes were perfectly placed and were in contact with the scalp while the reference electrodes were attached to the earlobes as elaborated in Section 1.3. The placement of the electrodes was based on the International 10-20 system, and they were directly connected to an EEG amplifier [4]. The signals are amplified, sampled at a specific sampling rate, and are converted to digital signals which can be processed for our BCI. This also involved processes such as artifact rejection and baseline correction for enhancement of the signal. These were the neural activities which worked as our input to the BCI system to control the avatar introduced in the game titled Brainrunners [4].

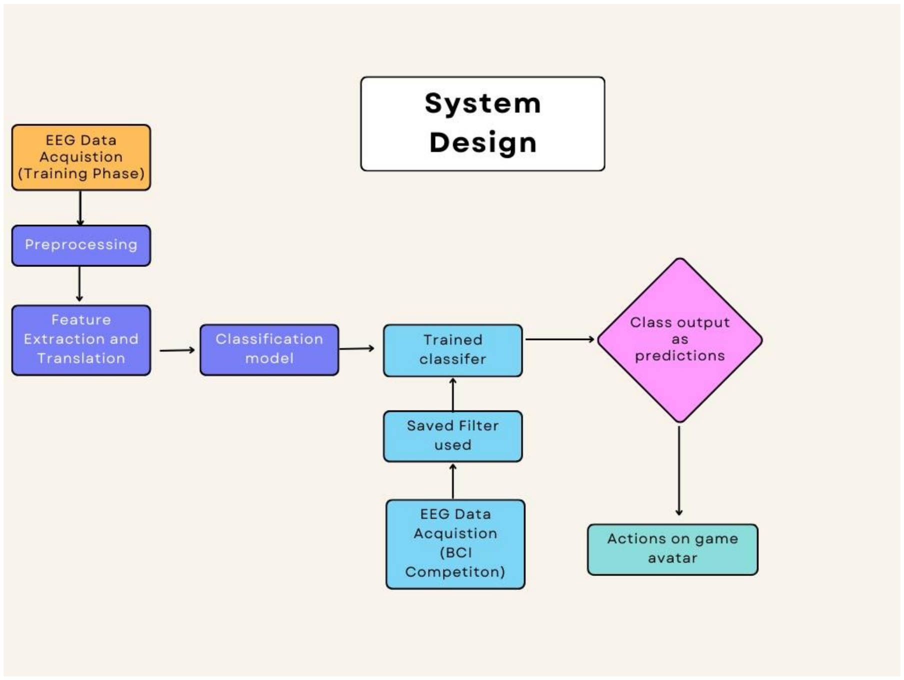
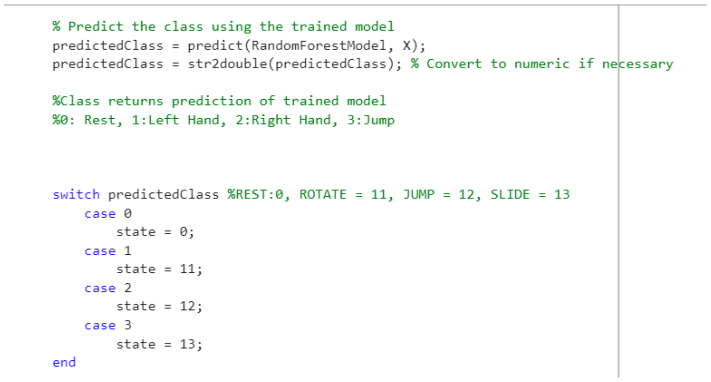

System Design
EEG Data Acquistion (Training Phase)
Preprocessing
Feature Extraction and Translation
Classification model
Trained classifer
Class output as predictions
Saved Filter used
EEG Data Acquistion (BCI Competiton)
Actions on game avatar

**Figure 1: Schematic of our System Design**

Feature extraction of the data specifically identifies and extracts the specific characteristics from the segmented EEG epochs which can be classified and used for our purposes. Classification algorithms were used to classify and translate the neural activities or intentions into specific outputs. For the model to be trained, the EEG data was split into training sets and testing sets. We used Random Forest as our machine learning model which used the training set data to learn the patterns and relationships in the data. The testing set is used for further evaluation. This training of the model allows it to make predictions specific to the class or label associated with the new unseen data. The output are the predicted labels (0,1,2 or 3) assigned as the variable Class each of which are assigned to a specific task in our our_classifer file as attached below.



```
% Predict the class using the trained model
predictedClass = predict(RandomForestModel, X);
predictedClass = str2double(predictedClass); % Convert to numeric if necessary

%Class returns prediction of trained model
%0: Rest, 1:Left Hand, 2:Right Hand, 3:Jump


switch predictedClass %REST:0, ROTATE = 11, JUMP = 12, SLIDE = 13
    case 0
        state = 0;
    case 1
        state = 11;
    case 2
        state = 12;
    case 3
        state = 13;
end
```

**Figure 2: Classifier output labeled as actions**

## 1.3: Procedures during data acquisition:

The preprocessing techniques used before the data collection involved the following steps.

1. Wearing the EEG cap and making sure it tightly fits with both ears exposed out to be attached to the reference electrode.

2. Using the blue Gel to clean the ears properly, use the gel to clean the scalp in locations C3 and C4 according to the International 10 - 20 system where the electrodes are supposed to be attached.

3. The reference electrodes must be placed in the cream captured in between the caps and then attached to the two ears. Excess cream must be cleared using tissue.

4. Using a syringe, put the right amount of gel in C3 and C4 and place the electrodes.

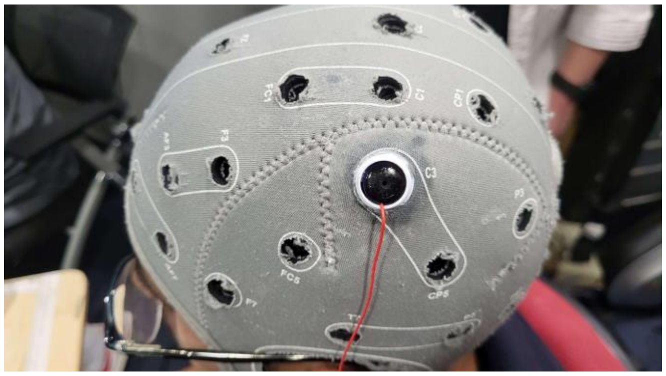


**Figure 3: An electrode placed at position C3**

5. Using the syringe, inject a little more gel through the hole in the electrode and make sure it is sealed and is perfectly in contact with the surface of the scalp.

6. Check the impedance and if it is an acceptable value (below 30 kΩ), we can tape both electrodes and move forward with the game.

7. This followed certain activities in which our subject was directed to certain activities which were to relax, switch control to left or right hand and to imagine jumping.

For our final run-through of the game, we managed to achieve an impedance value of 3.0 kΩ in channel 1 and 5.0 kΩ for channel 2.

## 1.4: Stimulus type and EEG data parameters

The training phase was based on asking the subject to imagine certain actions by following the instructions shown on the interface used for recording EEG signals strongly determined by the brain waves activated during motor imagery. There were four main stimulus types, each of which repeated 10 times at random intervals. The subject was asked to focus to the best of their ability on the task. The following two figures outline the stimuli involved and some other important parameters about the data acquired from the training phase. Each of the actions were assigned an integer value within the Stimulus_Type as shown below.

| **Stimulus** | **Stimulus Type** |
|---|---|
| Rest | 0 |
| Imagine picking up a cup with your left hand | 1 |
| Imagine picking up a cup with your right hand | 2 |
| Imagine jumping | 3 |

**Table 1: Stimuli used for EEG data acquisition.**

| Data -Type | Mat-File | Name | Value/Dimension | Function |
|---|---|---|---|---|
| Raw Data | EEGData.mat | Sample Rate | 256 | Number of EEG data acquired per second |
| | | Stimulus_TimeMark | 80x1 double | Start and end time of each stimulus |
| | | Stimuls_Type | 40x1 double | Label of stimulus, number of rows = number of stimuli |
| | | EEGData | 160784x2 double | Total number of EEG data acquired each channel. 160784/256 = 628 so we know that EEG data was taken for around 10.5 minutes. |
| Sliced/epoch-rejected data | ALLData.mat | ALLData | 2x256x200 double (3D array) | Channel x sample rate x trial:<br>For each stimulus, extract 5 seconds of EEG data for each channel.<br>So there will be 200 trials in total as there were 40 stimuli and for each stimuli, 5 seconds of data were extracted (40*5)<br>For each second, 256 EEGdata is acquired. |
| | | ALLLabel | 200 x 1 double | The number of labels, 40 stimuli was labelled but each of them was 'replaced by 5 as 5 separate data was extracted for each stimulus. |

**Table 2: Data type acquired, its dimensions, properties, and variables.**

## Section 2- System Implementation and Classification Algorithm

### 2.1: Data Pre-processing (epoch rejection)

While our main focus remained in perfecting our classifier and feature extraction method which did show a lot of improvement which will be elaborated in 2.2 and 2.3; it did not however improve the overall consistency of the data. After further research and suggestions from our professor, we decided to go over our EEG data and remove irregular and faulty epochs.

Our primary approach was to eliminate any epochs which did not fit the pattern of the intended tasks. These were mainly epochs with considerably high amplitude or frequency resembling significant distortion as shown in epoch 185 in Figure 4 given below. While we did see an improvement in accuracy, it, however, did not meet our expectations and was still inconsistent.

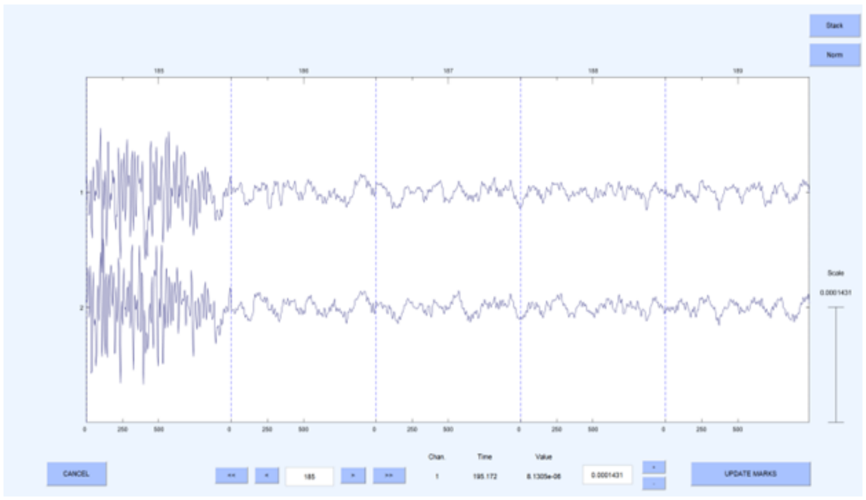


**Figure 4: Epoch 185 with significant distortion**

In our second attempt, we strategically analyzed the data containing timestamps for each task (Relax, Jump, Left hand, and Right hand). Initializing the EEG graph facilitated by EEGlab, we precisely tracked the time and epoch numbers corresponding to the execution of each task. By observing the patterns within these epochs, we identified and discarded outliers that didn't align with the expected task-specific trends. For instance, in Figure 5, during the 'Jump' action between epochs 165 - 169, Out of the five, two epochs (166 and 167) exhibited immense distortion which clearly prompted their rejection.

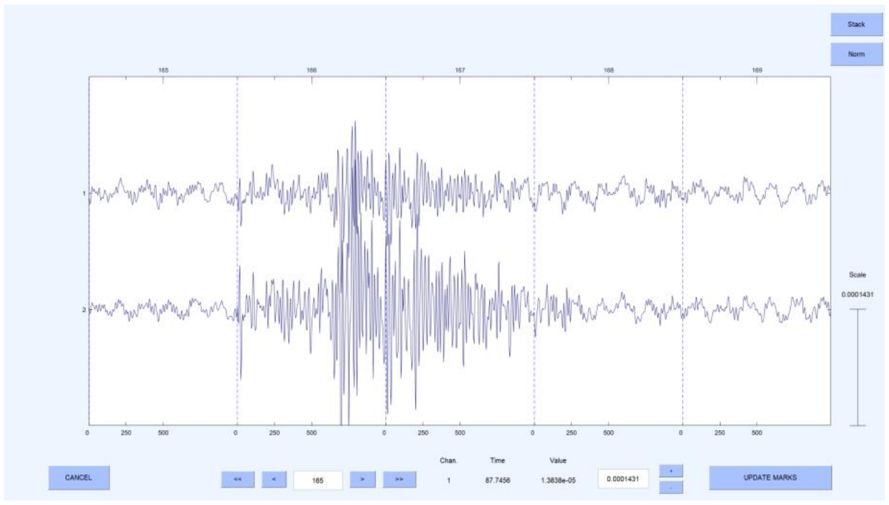
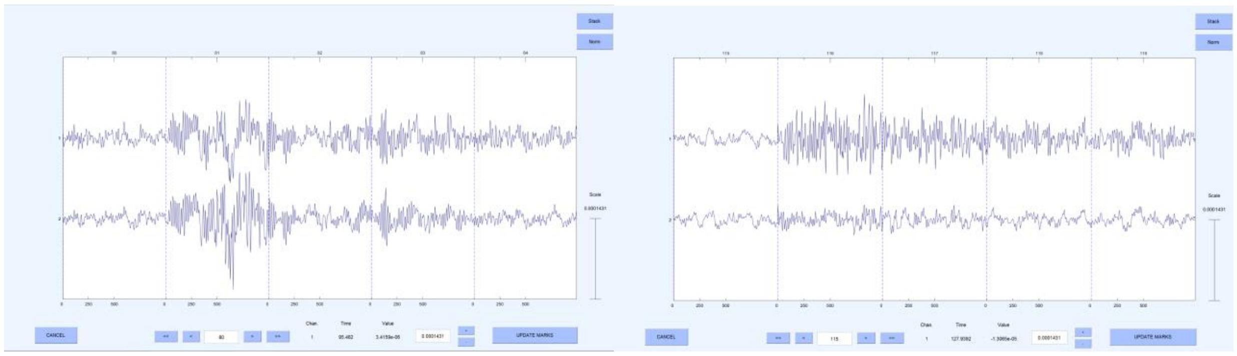


**Figure 5: Detected Artifacts between epoch 165 and 169**

Specifically focusing on the 'Right hand' task, we excluded a maximum of 15 epochs, the highest among all tasks. Overall, we disregarded around 28 epochs across all the tasks which we believed were extremely inconsistent to each established pattern. Highlighted artifacts in the figures below further underscored our detection process.



A B

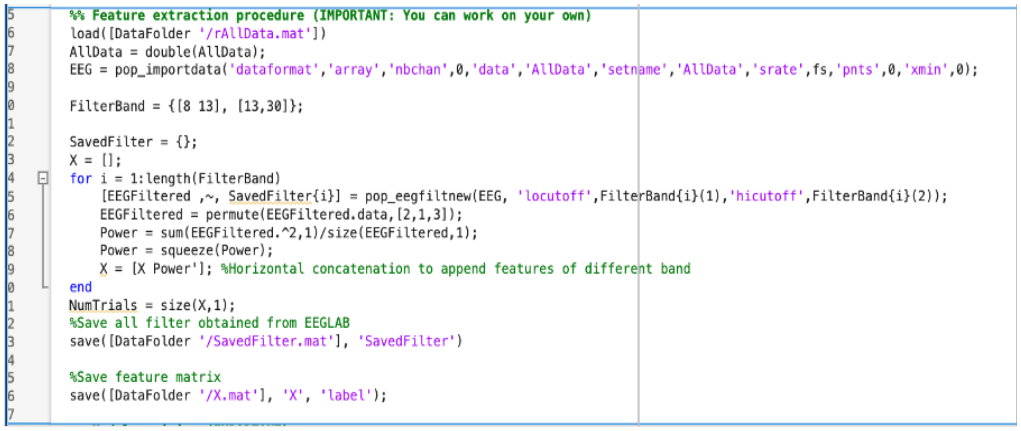
**Figure 6: Both A and B show epochs in which the Right hand was performed. A shows 80 - 84 out of which we rejected 81 - 84 and B shows 115 -119, and we rejected epochs 116 - 119.**

Precise epoch rejection was critical due to our relatively small dataset, making it highly sensitive to inaccuracies. Correct rejections were pivotal in maintaining the stability of our results, ensuring higher accuracy. Combining our epoch rejection strategy with the feature extraction methods and classifier, as detailed in sections 2.2 and 2.3, yielded consistently accurate outcomes, enabling successful gameplay.

## 2.2: Feature extraction

A major challenge for our project was to be able to identify a feature extraction method suitable for our subject's EEG data. For our feature selection method, alpha and beta frequency bands were extracted and analyzed. These are frequencies ranging from alpha: [8 13] and beta: [13 30]. Our decision to use a bandpass filter of the aforementioned range is based on previous EEG studies showing a desynchronization of the alpha(mu) band and beta band in the motor-related regions of the brain during motor imagery [5]. This allows us to take advantage of the associated power change and spectral information to be able to classify the EEG signals related to motor imagery. Spectral information is the characteristics and components of a signal across different frequencies. In our EEG data, the spectral information pertains to the frequency composition or distribution [3]. These act as features which reflect the activity of the brain while performing cognitive tasks.
Hence, we did not have a **saved_filter** mat file as our bandpass was integrated within the feature extraction method, which meant only frequencies in the aforementioned range were kept for classification purposes. Additionally, the bandpass filter acts as a notch filter for any mains line interference (around 50 Hz) while also filtering out any low frequency artifact caused by ocular movements during the process.
It is difficult to judge the feature that would have the most differentiating capability between the four different labels in our EEG data. We therefore had to implement different feature extraction methods listed below to be used for consequent feature translation.

### 2.2.1 Basic Power Calculation

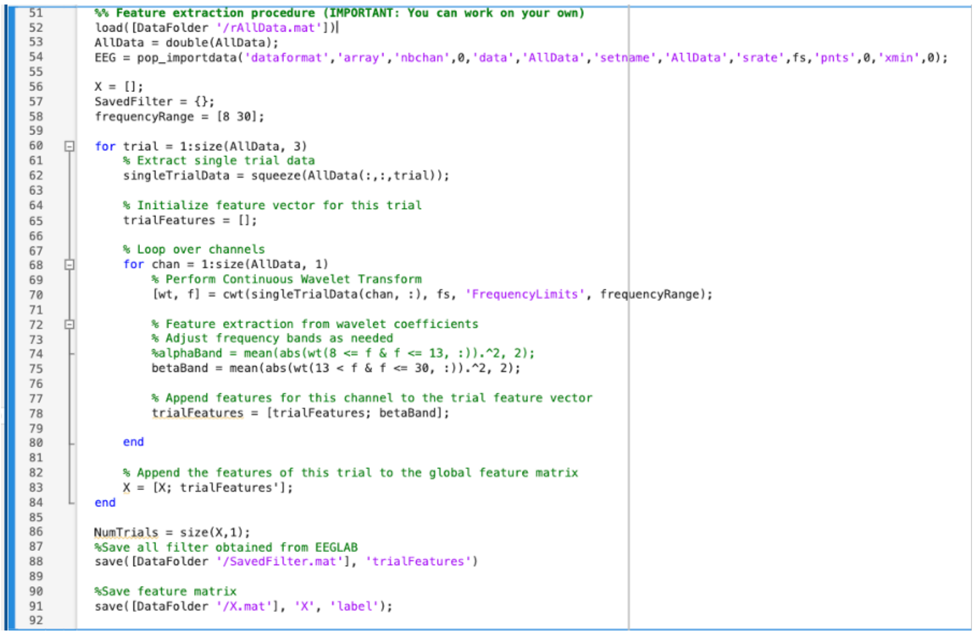


```
%% Feature extraction procedure (IMPORTANT: You can work on your own)
load([DataFolder '/rAllData.mat'])
AllData = double(AllData);
EEG = pop_importdata('dataformat','array','nbchan',0,'data','AllData','setname','AllData','srate',fs,'pnts',0,'xmin',0);

FilterBand = {[8 13], [13,30]};

SavedFilter = {};
X = [];
for i = 1:length(FilterBand)
    [EEGFiltered ,~, SavedFilter{i}] = pop_eegfiltnew(EEG, 'locutoff',FilterBand{i}(1),'hicutoff',FilterBand{i}(2));
    EEGFiltered = permute(EEGFiltered.data,[2,1,3]);
    Power = sum(EEGFiltered.^2,1)/size(EEGFiltered,1);
    Power = squeeze(Power);
    X = [X Power']; %Horizontal concatenation to append features of different band
end
NumTrials = size(X,1);
%Save all filter obtained from EEGLAB
save([DataFolder '/SavedFilter.mat'], 'SavedFilter')

%Save feature matrix
save([DataFolder '/X.mat'], 'X', 'label');
```

**Figure 7: The code for Power evaluation**

This is the provided feature extraction code using power calculation. Using EEGlab, acquired EEG data is filtered within alpha and beta range and the line Power = sum(EEGFiltered.^2,1)/size(EEGFiltered,1); is used to calculate the power values. This method does not take the spectral properties of EEG signals into account which may include important features. Additionally, this method results in a more varying, less smooth power spectrum estimation value which is not ideal for classification.

### 2.2.2. Continuous Wavelet Transform (CWT)



```matlab
%% Feature extraction procedure (IMPORTANT: You can work on your own)
load([DataFolder '/rAllData.mat'])
AllData = double(AllData);
EEG = pop_importdata('dataformat','array','nbchan',0,'data','AllData','setname','AllData','srate',fs,'pnts',0,'xmin',0);

X = [];
SavedFilter = {};
frequencyRange = [8 30];

for trial = 1:size(AllData, 3)
    % Extract single trial data
    singleTrialData = squeeze(AllData(:,:,trial));

    % Initialize feature vector for this trial
    trialFeatures = [];

    % Loop over channels
    for chan = 1:size(AllData, 1)
        % Perform Continuous Wavelet Transform
        [wt, f] = cwt(singleTrialData(chan, :), fs, 'FrequencyLimits', frequencyRange);

        % Feature extraction from wavelet coefficients
        % Adjust frequency bands as needed
        %alphaBand = mean(abs(wt(8 <= f & f <= 13, :)).^2, 2);
        betaBand = mean(abs(wt(13 < f & f <= 30, :)).^2, 2);

        % Append features for this channel to the trial feature vector
        trialFeatures = [trialFeatures; betaBand];

    end

    % Append the features of this trial to the global feature matrix
    X = [X; trialFeatures'];
end

NumTrials = size(X,1);
%Save all filter obtained from EEGLAB
save([DataFolder '/SavedFilter.mat'], 'trialFeatures')

%Save feature matrix
save([DataFolder '/X.mat'], 'X', 'label');
```

**Figure 8: The code for Continuous Wavelet Transform**

Time frequency analysis (TFA) method can be used to represent the time-varying spectral contents in EEG. It can provide better feature extraction than the traditional frequency-analysis method as the traditional method assumes EEG as stationary data when it is not. Hence, we decided to implement CWT for feature extraction, which is a type of TFA method.

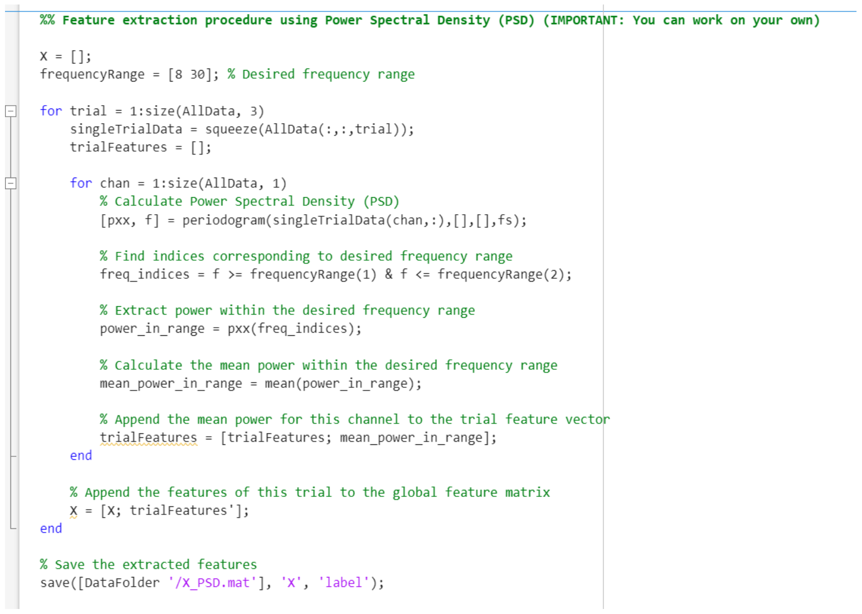
The line `for trial = 1:size(AllData, 3)` loops by the number of trials in ALLData which is 200. For each trial, `singleTrialData = squeeze(AllData(:,:,trial));` extracts the channel data and 256 EEG data points and stores it as a 2D array in singleTrialData. This is because our continuous wavelet transform function [wt,f] requires the input to be in a 2D array. `trialFeatures = [];` initializes a feature vector which will store the transformed data from the same trial but different channels.

Next, within the previous for loop, there is another for loop to loop over the channels and perform wavelet transformation. `[wt, f] = cwt(singleTrialData(chan, :), fs, 'FrequencyLimits', frequencyRange);` this line extracts the singleTrialData data for each channels and performs wavelet transform within the given FrequencyLimits which in this case is [8 30] the alpha and beta band. However, later for CWT, only beta band extraction showed better classification performance so only frequency range of 13~30 was considered for power calculation. betaBand = mean(abs(wt (13 < f & f <= 30, :)).^2, 2); This line extracts the wavelet coefficient within frequency range of 13~30 and calculates the power for each wavelet coefficient. Then it calculates the meaning of the power values. This is then stored in the trialFeatures array as 2x1 format where each row represents different channels.

X = [X; trialFeatures']; This line shows that after the power is computed for each channel, the trialFeature array is appended in the previously defined X=[]; array. However, the trialFeatures array has to be transposed to be in a 1x2 format to make sure the X array, after all the iteration of the outer loop, will be in the format of 200 x 2 array. Columns represent the channels, and the rows represent the trials. This is important as the label array is in the form of a 200 x 1 array where each row indicates the label data of that trial. (ex) label from 3rd row is the label for the 3rd trial)

### 2.2.3. Power Spectral Density (PSD)

Due to Continuous Wavelet Transformation not working as expected in the lab computer, we decided to try out Power Spectral Density (PSD). The modification we made is given below:



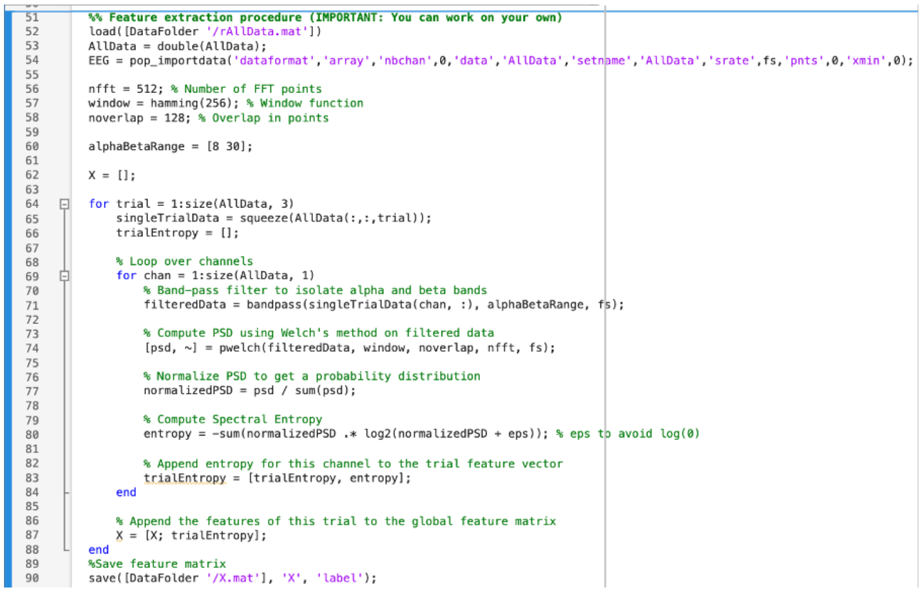
```
%% Feature extraction procedure using Power Spectral Density (PSD) (IMPORTANT: You can work on your own)

X = [];
frequencyRange = [8 30]; % Desired frequency range

for trial = 1:size(AllData, 3)
    singleTrialData = squeeze(AllData(:,:,trial));
    trialFeatures = [];

    for chan = 1:size(AllData, 1)
        % Calculate Power Spectral Density (PSD)
        [pxx, f] = periodogram(singleTrialData(chan,:),[],[],fs);

        % Find indices corresponding to desired frequency range
        freq_indices = f >= frequencyRange(1) & f <= frequencyRange(2);

        % Extract power within the desired frequency range
        power_in_range = pxx(freq_indices);

        % Calculate the mean power within the desired frequency range
        mean_power_in_range = mean(power_in_range);

        % Append the mean power for this channel to the trial feature vector
        trialFeatures = [trialFeatures; mean_power_in_range];
    end

    % Append the features of this trial to the global feature matrix
    X = [X; trialFeatures'];
end

% Save the extracted features
save([DataFolder '/X_PSD.mat'], 'X', 'label');
```

**Figure 9: Code used for Power Spectral Density**

EEG data is already loaded and organized into epochs of 5-second trial and the unwanted epochs are specified in the epochs_to_reject are removed from the dataset corresponding to their dataset. The line for chan = 1: size (AllData, 1) starts a loop over each channel on the EEG data, iterating through the first dimension.

For each of the trial and channel combinations mentioned in the EEG data, the Power Spectral Density (PSD) is calculated using the periodogram function by extracting power values across frequencies indicated by the frequency range variable. As the input for the pxx() function has to be indices corresponding to the signal, we define a freqIndices variable to store the indices where the frequency is within the alpha and beta bands. Finally, we use the mean of the obtained power values and append it to the trail feature vector. The for loop involved in this code ensures this repeat for both the channels until finally appending mean power values into the feature matrix X.

### 2.2.4. Welch Power Spectrum Estimation 1



```
%% Feature extraction procedure (IMPORTANT: You can work on your own)
load([DataFolder '/rAllData.mat'])
AllData = double(AllData);
EEG = pop_importdata('dataformat','array','nbchan',0,'data','AllData','setname','AllData','srate',fs,'pnts',0,'xmin',0);

nfft = 512; % Number of FFT points
window = hamming(256); % Window function
noverlap = 128; % Overlap in points

alphaBetaRange = [8 30];

X = [];

for trial = 1:size(AllData, 3)
    singleTrialData = squeeze(AllData(:,:,trial));
    trialEntropy = [];

    % Loop over channels
    for chan = 1:size(AllData, 1)
        % Band-pass filter to isolate alpha and beta bands
        filteredData = bandpass(singleTrialData(chan, :), alphaBetaRange, fs);

        % Compute PSD using Welch's method on filtered data
        [psd, ~] = pwelch(filteredData, window, noverlap, nfft, fs);

        % Normalize PSD to get a probability distribution
        normalizedPSD = psd / sum(psd);

        % Compute Spectral Entropy
        entropy = -sum(normalizedPSD .* log2(normalizedPSD + eps)); % eps to avoid log(0)

        % Append entropy for this channel to the trial feature vector
        trialEntropy = [trialEntropy, entropy];
    end

    % Append the features of this trial to the global feature matrix
    X = [X; trialEntropy];
end
%Save feature matrix
save([DataFolder '/X.mat'], 'X', 'label');
```

**Figure 10: Code for Welch Power Spectrum Estimation 1**

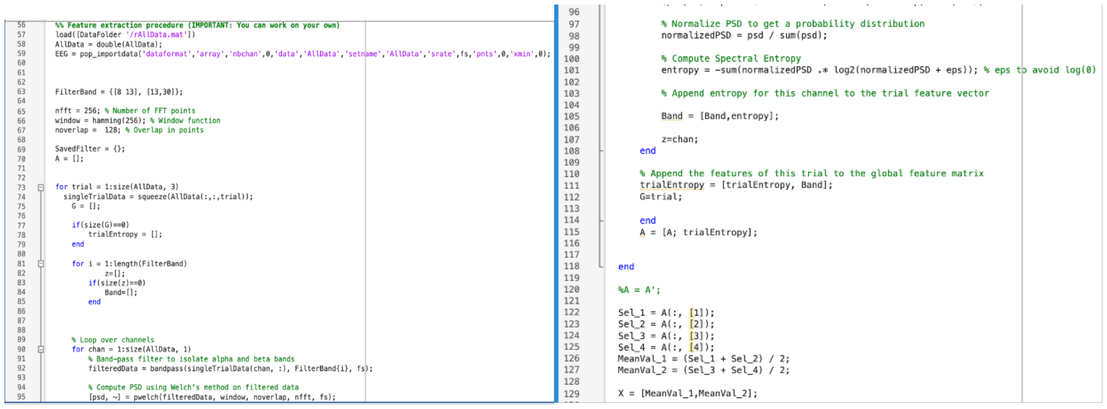
Power spectral density is how the power of a random signal is distributed across different frequencies. It can be calculated by squaring the magnitude of the Fourier transform. The main problem of the FFT-based spectral estimations is leakage. If there is a discontinuity in the EEG signal, the frequency signal will be attenuated. This can be compensated for using appropriate window functions. Welch power spectrum estimation method is based on the power periodogram. It divides the time domain EEG data into segments which overlap. Next, the window function is applied to each segment. This is done prior to the computation of the periodogram. After the computation, the values for each segment are averaged. This way, it improves the frequency resolution compared to the normal single periodograms and reduces the variance of power density estimation.

The number of FFT points, window function and non-overlap data points are initialized as: nfft = 512; window = hamming(256); noverlap = 128;

The code works similarly as the wavelet transform except for the spectral computation method. normalizedPSD = psd / sum(psd); First, the probability distribution of PSD is computed by normalizing. entropy = -sum(normalizedPSD .* log2(normalizedPSD + eps)); Next, use the normalized value to calculate the entropy data.

In general, the classification accuracy still ranged too much due to high OOB error (forest classification).

### 2.2.5. Welch Power Spectrum Estimation 2



```
%% Feature extraction procedure (IMPORTANT: You can work on your own)
load([DataFolder '/rAllData.mat'])
AllData = double(AllData);
EEG = pop_importdata('dataformat','array','nbchan',0,'data','AllData','setname','AllData','srate',fs,'pnts',0,'xmin',0);



FilterBand = {[8 13], [13,30]};

nfft = 256; % Number of FFT points
window = hamming(256); % Window function
noverlap =  128; % Overlap in points

SavedFilter = {};
A = [];


for trial = 1:size(AllData, 3)
  singleTrialData = squeeze(AllData(:,:,trial));
    G = [];

    if(size(G)==0)
        trialEntropy = [];
    end

    for i = 1:length(FilterBand)
            z=[];
        if(size(z)==0)
            Band=[];
        end



    % Loop over channels
    for chan = 1:size(AllData, 1)
        % Band-pass filter to isolate alpha and beta bands
        filteredData = bandpass(singleTrialData(chan, :), FilterBand{i}, fs);

        % Compute PSD using Welch's method on filtered data
        [psd, ~] = pwelch(filteredData, window, noverlap, nfft, fs);

        % Normalize PSD to get a probability distribution
        normalizedPSD = psd / sum(psd);

        % Compute Spectral Entropy
        entropy = -sum(normalizedPSD .* log2(normalizedPSD + eps)); % eps to avoid log(0)

        % Append entropy for this channel to the trial feature vector

        Band = [Band,entropy];

        z=chan;
    end

    % Append the features of this trial to the global feature matrix
    trialEntropy = [trialEntropy, Band];
    G=trial;

    end
    A = [A; trialEntropy];


end

%A = A';

Sel_1 = A(:, [1]);
Sel_2 = A(:, [2]);
Sel_3 = A(:, [3]);
Sel_4 = A(:, [4]);
MeanVal_1 = (Sel_1 + Sel_2) / 2;
MeanVal_2 = (Sel_3 + Sel_4) / 2;

X = [MeanVal_1,MeanVal_2];
```

**Figure 11: Code for Welch Power Spectrum Estimation 2**

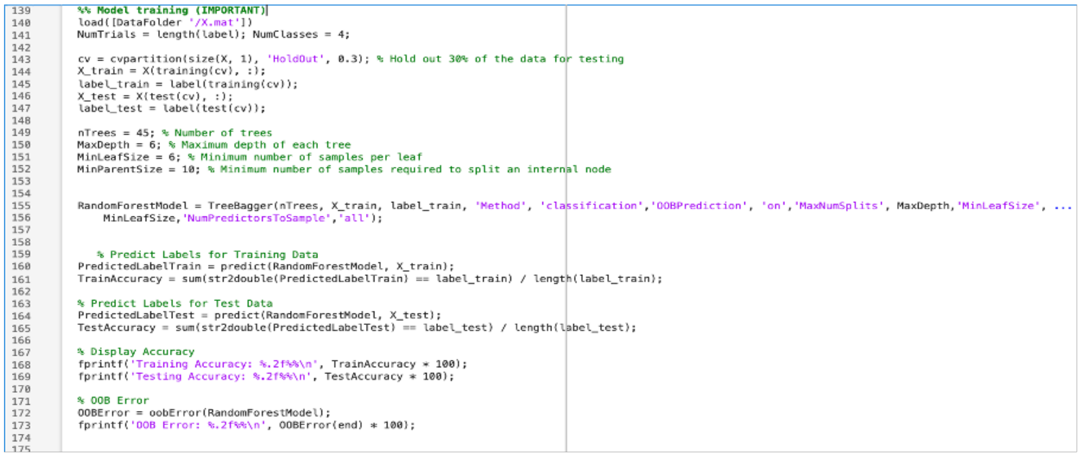
From the first feature extraction method using the given power code, using two separate frequency bands {[8 13], [13 30]} gave a better classification accuracy than the one using the alpha + beta band combined [8 30]. So, in an attempt to lower the OOB error the above welch code was modified to find the spectral value for each of the separate filter bands and the data from each channel from the same frequency range was averaged.

This was done by including an extra for loop before the for loop that loops over the channels. This loops over each filter frequency band for each trial. So, for each trial, entropy is calculated for two frequency bands and for each frequency band, there are 2 separate channels. Hence, the resulting A=[] array returns a 200 x 4 array where 200 rows represent each trial and column 1 and column 2 is the entropy value for alpha band channel 1 and 2, and column 3 and 4 is the beta band values.

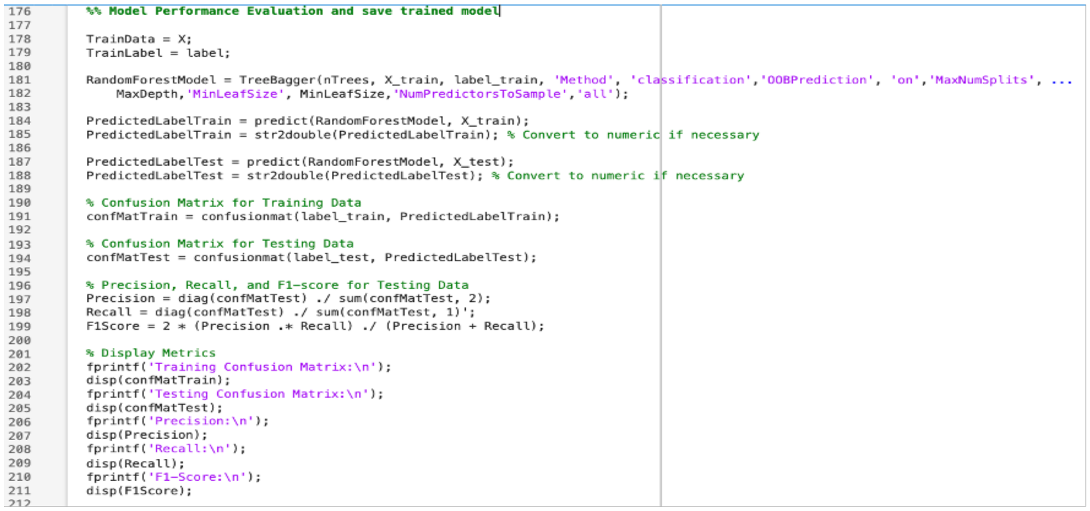
By using two separate frequency bands in distinct iterations, alpha and beta bands are separately analyzed, which allows for capturing more relevant and distinctive features for classification tasks compared to analyzing a single large frequency band of [8 30].

Next, values from channels 1 and 2 for each frequency band are averaged and stored into the X=[] array for classification. This was done in the hope of refining the spectral values. Moreover, 4 different values for each trial may cause overfitting of the classification model. X=[] array returns 200 x 2 array where again, each row represents the trials and each column represents the two frequency bands.

## 2.3: Trained Model: Random Forest



```
%% Model training (IMPORTANT)
load([DataFolder '/X.mat'])
NumTrials = length(label); NumClasses = 4;

cv = cvpartition(size(X, 1), 'HoldOut', 0.3); % Hold out 30% of the data for testing
X_train = X(training(cv), :);
label_train = label(training(cv));
X_test = X(test(cv), :);
label_test = label(test(cv));

nTrees = 45; % Number of trees
MaxDepth = 6; % Maximum depth of each tree
MinLeafSize = 6; % Minimum number of samples per leaf
MinParentSize = 10; % Minimum number of samples required to split an internal node


RandomForestModel = TreeBagger(nTrees, X_train, label_train, 'Method', 'classification','OOBPrediction', 'on','MaxNumSplits', MaxDepth,'MinLeafSize', ...
    MinLeafSize,'NumPredictorsToSample','all');


   % Predict Labels for Training Data
PredictedLabelTrain = predict(RandomForestModel, X_train);
TrainAccuracy = sum(str2double(PredictedLabelTrain) == label_train) / length(label_train);

% Predict Labels for Test Data
PredictedLabelTest = predict(RandomForestModel, X_test);
TestAccuracy = sum(str2double(PredictedLabelTest) == label_test) / length(label_test);

% Display Accuracy
fprintf('Training Accuracy: %.2f%%\n', TrainAccuracy * 100);
fprintf('Testing Accuracy: %.2f%%\n', TestAccuracy * 100);

% OOB Error
OOBError = oobError(RandomForestModel);
fprintf('OOB Error: %.2f%%\n', OOBError(end) * 100);
```



```
%% Model Performance Evaluation and save trained model

TrainData = X;
TrainLabel = label;

RandomForestModel = TreeBagger(nTrees, X_train, label_train, 'Method', 'classification','OOBPrediction', 'on','MaxNumSplits', ...
    MaxDepth,'MinLeafSize', MinLeafSize,'NumPredictorsToSample','all');

PredictedLabelTrain = predict(RandomForestModel, X_train);
PredictedLabelTrain = str2double(PredictedLabelTrain); % Convert to numeric if necessary

PredictedLabelTest = predict(RandomForestModel, X_test);
PredictedLabelTest = str2double(PredictedLabelTest); % Convert to numeric if necessary

% Confusion Matrix for Training Data
confMatTrain = confusionmat(label_train, PredictedLabelTrain);

% Confusion Matrix for Testing Data
confMatTest = confusionmat(label_test, PredictedLabelTest);

% Precision, Recall, and F1-score for Testing Data
Precision = diag(confMatTest) ./ sum(confMatTest, 2);
Recall = diag(confMatTest) ./ sum(confMatTest, 1)';
F1Score = 2 * (Precision .* Recall) ./ (Precision + Recall);

% Display Metrics
fprintf('Training Confusion Matrix:\n');
disp(confMatTrain);
fprintf('Testing Confusion Matrix:\n');
disp(confMatTest);
fprintf('Precision:\n');
disp(Precision);
fprintf('Recall:\n');
disp(Recall);
fprintf('F1-Score:\n');
disp(F1Score);
```

**Figure 12: Model training and performance evaluation**

For the classification model, a random forest method was implemented. Random forest models can identify the non-linear relationship between the label and the feature variable. It can handle complex data and map its input features to the output labels. EEG data portrays the non-linear relationship between the power of the EEG signal and the state of the brain (stimulus). It is also known to be able to perform well in complex as well as multi-class space, making it a suitable choice for our system.

Using the line cv = cvpartition(size(X, 1), 'HoldOut', 0.3); The data from the imported X array was separated where 30% of the data (200 x 0.3 = 60) was left out for testing the model.

Next the model parameters are predefined nTrees = 45; MaxDepth = 6; MinLeafSize = 6; for easier hyperparameter tuning these parameters can be set to tune between the overfitting properties and the test accuracy. Overfitting refers to the classification problem where training accuracy is too high due to small testing data sample size and data features, making the model unable to classify data unfamiliar or distinct from the training data. They have to be smartly chosen to balance between train accuracy and test accuracy. Too high train accuracy results in overfitting hence low-test accuracy but if the train accuracy is too lowered, test accuracy is also low due to under-fitting. In general, lowering the values of nTrees and MaxDepth decreases the training accuracy while increasing the MinLeafSize enhances the training accuracy.

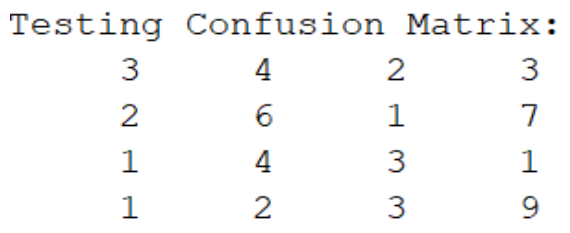
The trial-and-error method was used to find the most suitable parameter values which gave the highest test accuracy and lowest Out of Bag(OOB)Error. This was done by iterating the process through different values of nTress, MaxDepth and MinLeafSize creating vectors with suggested values. Testing and train accuracy along with OOB Error were compared before finally deciding on the parameters listed above.

## 2.4: Performance Evaluation

The performance of the model has been evaluated using certain metrics. These involved the use of F1 score and the Confusion Matrix, which gave us the accuracy of the testing and training data [1]. Confusion matrix is a table with 4 different combinations of predicted and actual values. It is a tabular representation of the performance of a classification model by comparing predicted labels against the actual true labels across different classes. It is extremely useful for measuring precision-recall, Specificity and Accuracy [1]. The main component labels of a confusion matrix are:

- **True Positives (TP):** Instances where the model predicted the class correctly, and the true label is also positive.
- **True Negatives (TN):** Instances where the model predicted the class correctly, and the true label is also negative.
- **False Positives (FP):** Instances where the model predicted the class incorrectly as positive, but the true label is negative (Type I error).
- **False Negatives (FN):** Instances where the model predicted the class incorrectly as negative, but the true label is positive (Type II error).

Based on our classifier, we obtained the following confusion matrix from our trained model.



```
Testing Confusion Matrix:
     3     4     2     3
     2     6     1     7
     1     4     3     1
     1     2     3     9
```

**Figure 13: Confusion matrix**

From inspection, the most striking observation is the success rate of our predictor to be able to predict the Class value 3 (associated with thinking of jumping) compared to any other stimulus. It could correctly classify 9 out 15 EEG segments associated with the stimulus mentioned before. For the other stimulus types, the model performed almost poorly to be able to correctly classify and label. This is reflected by low accuracy scores and F-1 scores explained below.

The confusion matrix provides a comprehensive view of how well the model is performing across different classes [1]. This can be used to determine the accuracy which determines the proportion of correctly classified predictions out of the total number of predictions made and can be represented using the given equation below.

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions\ made}$$

**Figure 14: Accuracy Equation**

A higher accuracy indicates how frequently the model is making correct predictions, thus is a quick metric to assess the overall performance of the model.

The Confusion Matrix also gives us the precision which defines the accuracy of positive prediction and the recall which are basically the proportion of true positives identified correctly.

The F1 score is a harmonic mean of the precision and recall values previously identified. This mainly works by distributing more weight towards the lower value between the two which gives us a better measure for balancing the performance of a classifier as this prevents an overly positive evaluation within the skewness of the class [1]. The F1 Score can be presented in the Equation given below:

$$F1\ Score = 2\ x\ \frac{Precision\ x\ Recall}{Precision + Recall}$$

**Figure 15: F1 Score equation**

The precision mentioned in this context mainly refers to the accuracy of the positive predictions made by the model by calculating the ratio of correctly predicted positive predictions or True Positive (TP) out of the overall total positive predictions involving True Positive and False Positive (TP + FP) [6]. The recall on the other hand measures the sensitivity of the model's predictions for a given class as it calculates the ratio of correctly predicted positive observations or True Positive (TP) to the total actual observations which involves True Positives and False Negatives (TP +FN). A higher recall indicates the model successfully captured a large proportion of positive instances [6].

$$Precision = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \qquad Recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

**Figure 16: Precision and Recall [6]**

A higher F1 score (ranging from 0 to 1) shows good balance between precision and recall and implies higher accuracy as it gets closer to 1 [1]. This gives us a much more comprehensive understanding of our Random Forest model, which also served a vital role for further tuning and perfecting the model for our BCI.

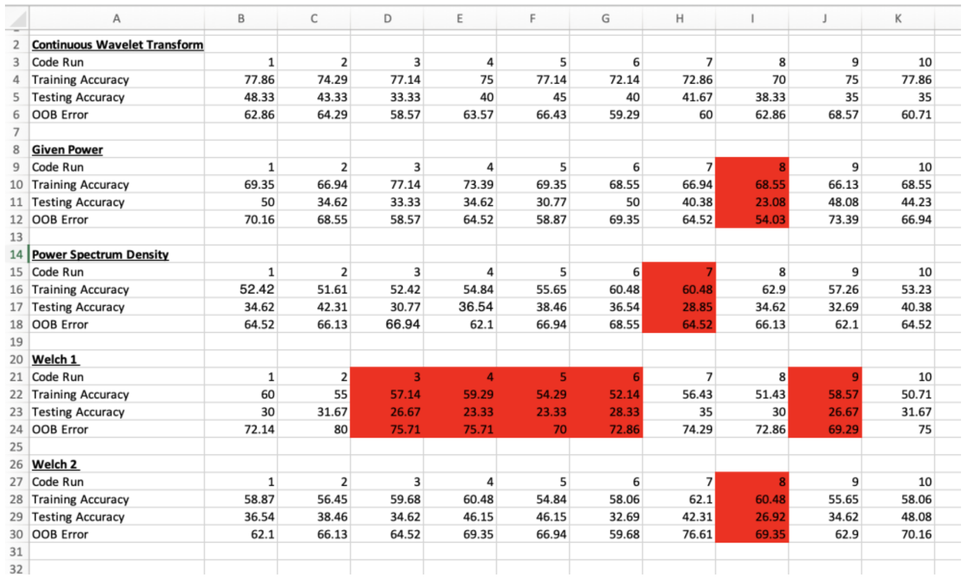
| Stimulus Type | Precision | F-1 score | Recall |
|---|---|---|---|
| 0 | 0.2500 | 0.3158 | 0.4286 |
| 1 | 0.3750 | 0.3750 | 0.3750 |
| 2 | 0.333 | 0.333 | 0.333 |
| 3 | 0.6000 | 0.5143 | 0.4500 |

**Table 3: Scoring each stimulus.**

Based on the table above, it is easy to see that our model training performs best to distinguish signals related to Stimulus Type 3 and very poorly with Stimulus Type 1.

## 2.5: Feature Translation

In addition to the test accuracy value, Out-of-Bag (OOB) error is used to assess the classification performance of each feature selection method. OOB is a metric which acts as an internal validation measure for Random Forests allowing estimation of the model's predictive performance without a dedicated validation set. This gives an idea of how well the model generalizes unseen data.



| | A | B | C | D | E | F | G | H | I | J | K |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | **Continuous Wavelet Transform** | | | | | | | | | | |
| 3 | Code Run | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 4 | Training Accuracy | 77.86 | 74.29 | 77.14 | 75 | 77.14 | 72.14 | 72.86 | 70 | 75 | 77.86 |
| 5 | Testing Accuracy | 48.33 | 43.33 | 33.33 | 40 | 45 | 40 | 41.67 | 38.33 | 35 | 35 |
| 6 | OOB Error | 62.86 | 64.29 | 58.57 | 63.57 | 66.43 | 59.29 | 60 | 62.86 | 68.57 | 60.71 |
| 7 | | | | | | | | | | | |
| 8 | **Given Power** | | | | | | | | | | |
| 9 | Code Run | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 10 | Training Accuracy | 69.35 | 66.94 | 77.14 | 73.39 | 69.35 | 68.55 | 66.94 | 68.55 | 66.13 | 68.55 |
| 11 | Testing Accuracy | 50 | 34.62 | 33.33 | 34.62 | 30.77 | 50 | 40.38 | 23.08 | 48.08 | 44.23 |
| 12 | OOB Error | 70.16 | 68.55 | 58.57 | 64.52 | 58.87 | 69.35 | 64.52 | 54.03 | 73.39 | 66.94 |
| 13 | | | | | | | | | | | |
| 14 | **Power Spectrum Density** | | | | | | | | | | |
| 15 | Code Run | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 16 | Training Accuracy | 52.42 | 51.61 | 52.42 | 54.84 | 55.65 | 60.48 | 60.48 | 62.9 | 57.26 | 53.23 |
| 17 | Testing Accuracy | 34.62 | 42.31 | 30.77 | 36.54 | 38.46 | 36.54 | 28.85 | 34.62 | 32.69 | 40.38 |
| 18 | OOB Error | 64.52 | 66.13 | 66.94 | 62.1 | 66.94 | 68.55 | 64.52 | 66.13 | 62.1 | 64.52 |
| 19 | | | | | | | | | | | |
| 20 | **Welch 1** | | | | | | | | | | |
| 21 | Code Run | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 22 | Training Accuracy | 60 | 55 | 57.14 | 59.29 | 54.29 | 52.14 | 56.43 | 51.43 | 58.57 | 50.71 |
| 23 | Testing Accuracy | 30 | 31.67 | 26.67 | 23.33 | 23.33 | 28.33 | 35 | 30 | 26.67 | 31.67 |
| 24 | OOB Error | 72.14 | 80 | 75.71 | 75.71 | 70 | 72.86 | 74.29 | 72.86 | 69.29 | 75 |
| 25 | | | | | | | | | | | |
| 26 | **Welch 2** | | | | | | | | | | |
| 27 | Code Run | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 28 | Training Accuracy | 58.87 | 56.45 | 59.68 | 60.48 | 54.84 | 58.06 | 62.1 | 60.48 | 55.65 | 58.06 |
| 29 | Testing Accuracy | 36.54 | 38.46 | 34.62 | 46.15 | 46.15 | 32.69 | 42.31 | 26.92 | 34.62 | 48.08 |
| 30 | OOB Error | 62.1 | 66.13 | 64.52 | 69.35 | 66.94 | 59.68 | 76.61 | 69.35 | 62.9 | 70.16 |
| 31 | | | | | | | | | | | |
| 32 | | | | | | | | | | | |

**Figure 17: Performance Evaluation from different feature selections**

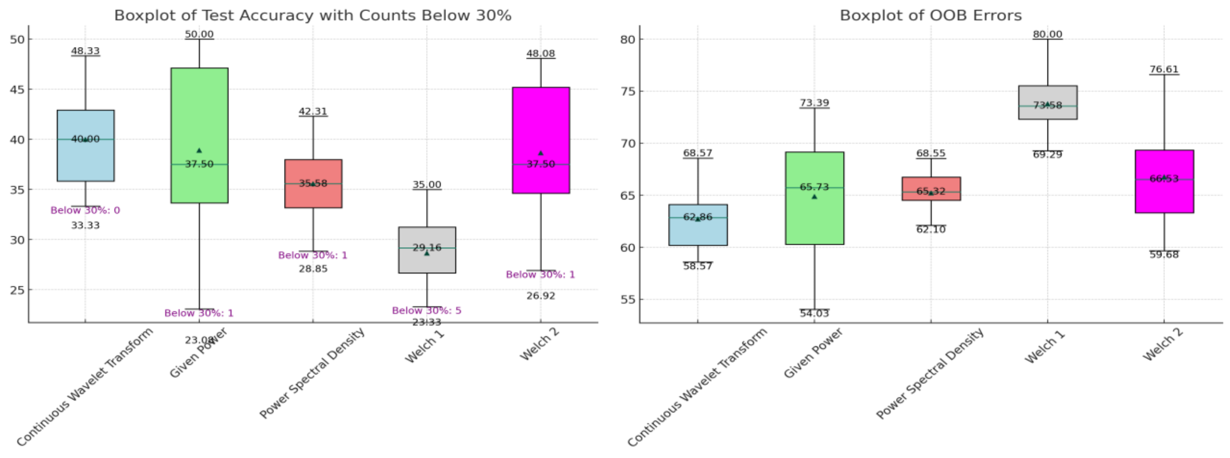


**Figure 18: Box plot of test-train accuracy and OOB error rate**

Figure 13 shows all the training accuracy, testing accuracy and OOB error values from 10 consecutive runs of Random Forest Classification Model for different feature extraction methods. The Continuous Wavelet Transform method shows appropriately higher test accuracy values with minimal variance between the highest and the lowest test accuracy. Moreover, it is the only feature extraction method that has 0 test accuracy value lower than 30% from the 10 consecutive runs. It also has the lowest median value for OOB error.

The given power method, although shows the highest test accuracy value of 50% for one of the runs, its boxplot body has the longest length which implies that it has the highest variance. This may be due to the large OOB error values and its variance. This shows that the given power method is too inconsistent for the game.

The Power Spectral Density method shows moderate performance with moderately high and consistent test accuracy values and low OOB error rate however, the boxplot shows that the test accuracy values mostly range from approximately 32%~37% which can be improved.

Welch 1 method which only used one single frequency band [8 30] showed the worst performance with the lowest test accuracy values, with 5 values lower than 30%, and highest OOB error values.

Welch 2 method, which is what was used for the game, shows high test accuracy with acceptable variance. Most of the accuracy is in the range of 35%~45% and has the highest accuracy of 48%. It also shows OOB values of acceptable range. The method with higher test accuracy then Welch 2 is the given power which is extremely inconsistent and the methods with lower OOB error values are Continuous Wavelet Transform and Power Spectrum Density for which CWT is not available and PSD has lower accuracy compared to Welch 2.

This proves that aside from Continuous Wavelet Transform, which we were not able to use due to lack of appropriate toolbox, Welch 2 method shows the best performance. The modification to the original welch function (Welch 1) has successfully improved the classification performance.

## Section 3- BCI competition

### 3.1: Initializing and playing the game

The subject for the game was the same as the member whose EEG was used for training our model. This is a natural choice as the classifier is trained based on the subject's distinct EEG waveforms to the stimuli involved in the training process.

Before playing the game, the subject was made ready for the EEG recording needed to play the game. The same procedures were followed during the data acquisition stage (Section 1.3) to ensure that there was a proper data input of EEG signal to the game_start file. Impedance was checked and the game was started if and only the impedance was at an acceptable level (below 30 kilo ohms).

On the day of the game, the Game_Start file was loaded with our trained model, saved filter and classifier. This is a crucial step as it ensures that the data input (the subject's EEG waveforms) can be correctly calibrated and labeled against the actions (JUMP, ROTATE and SLIDE) that are involved in the game.

### 3.2: Challenges encountered and the overall performance

After a few trials, the subject realized that at times the avatar is difficult to control. This may be due to the classification model having a testing accuracy of below 50% when it was tested before the competition. Another observation was a delay in the response of the avatar to perceived motor imagery changes by the subject. To address the delay, the subject tried to anticipate and foresee the next block of the track to be able to change the action of the avatar at the right time. This strategy helped in saving some time in parts of the block that transition from one action to the other.

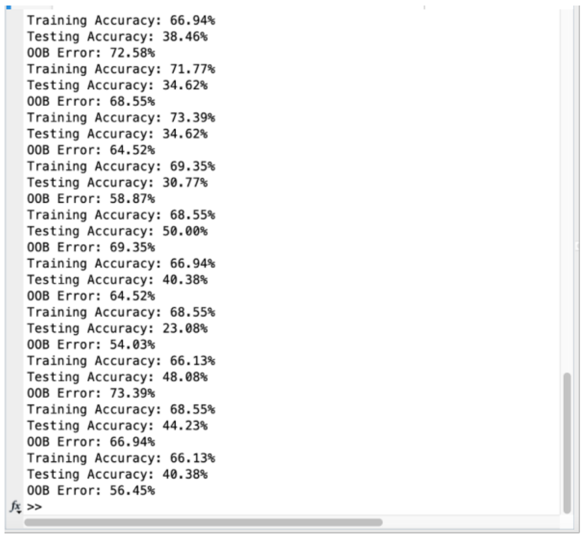
Another change made to the code was assigning the JUMP function to the motor imagery of jumping as it seemed to be easier to remember and control the avatar. Originally, the SLIDE function was assigned to the motor imagery of jumping which we felt was incongruent to the action of the game.

A major drawback in the game was the time delay caused by the first and the last blocks of the game where no actions were required. But there seemed to be random changes in state of the avatar which resulted in a significant elapse of time. From retrospect, the decision to assign the relaxing motor imagery state (Stimulus Type 0) to the action of ROTATE might have led to a poor performance in the game. This is due to the great variation in the brain wave signals when a subject is asked to relax, especially during a competitive game. This was evident as significant time elapsed at blocks where the avatar was required to ROTATE but was unable to due to the presuming variant brain wave signals when the subject wants to relax. This decision was taken as the motor imagery of jumping seemed to not produce any action on the avatar. This might have been due to the training data used for the model not having any particularly strong discriminating feature for that action. Hence the model was unable to pick up and label the brain waves obtained from such motor imagery (explained by the Confusion Matrix in Section 2.4), resulting in a great disadvantage during the game.

## Section 4: Conclusion

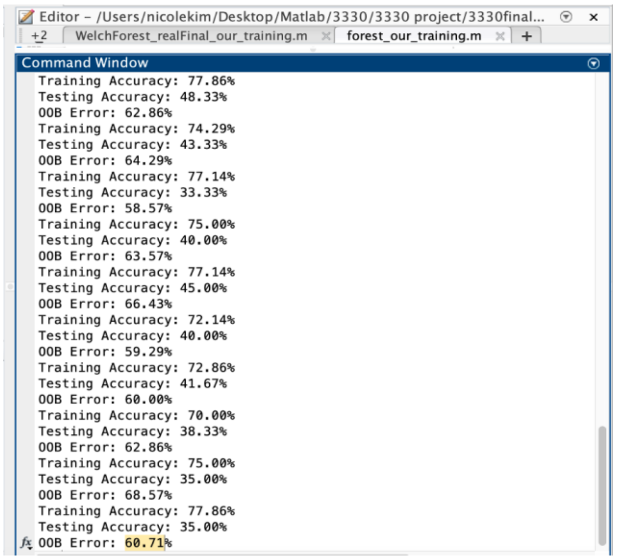
The use of EEG data derived from motor imagery to control BCI is a highly sought after technique for current advancements in virtual control of electronic devices. The project was a small simulation of the working principles involved in training and testing such models and implementing it for a BCI-run game competition. Due to time constraints and limited trails, the training data we could use for building our classifier was limited. In future applications, the training can be done by repeating the stimulus type at higher iterations to be able to properly identify trends in the waveform associated with that stimulus type. But even with the mentioned limitations, our trained model performed well in the final demonstration and was able to finish the game (5 meters track) at an average time of 137.58 seconds. The group was content with the performance and hopes to work on future projects related to EEG classification and BCI-led technologies.

**Additional Figures:**
The following figures showcase the Training Accuracy, Testing accuracy and the OOB error for each procedure.



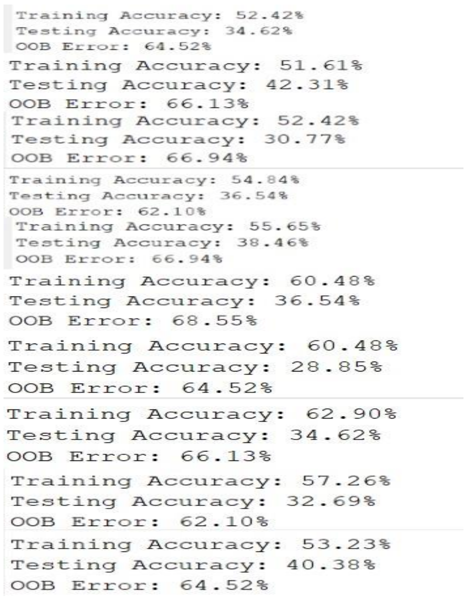
```
Training Accuracy: 66.94%
Testing Accuracy: 38.46%
OOB Error: 72.58%
Training Accuracy: 71.77%
Testing Accuracy: 34.62%
OOB Error: 68.55%
Training Accuracy: 73.39%
Testing Accuracy: 34.62%
OOB Error: 64.52%
Training Accuracy: 69.35%
Testing Accuracy: 30.77%
OOB Error: 58.87%
Training Accuracy: 68.55%
Testing Accuracy: 50.00%
OOB Error: 69.35%
Training Accuracy: 66.94%
Testing Accuracy: 40.38%
OOB Error: 64.52%
Training Accuracy: 68.55%
Testing Accuracy: 23.08%
OOB Error: 54.03%
Training Accuracy: 66.13%
Testing Accuracy: 48.08%
OOB Error: 73.39%
Training Accuracy: 68.55%
Testing Accuracy: 44.23%
OOB Error: 66.94%
Training Accuracy: 66.13%
Testing Accuracy: 40.38%
OOB Error: 56.45%
fx >>
```

**Figure 19: Results with CWT**


Editor – /Users/nicolekim/Desktop/Matlab/3330/3330 project/3330final...
+2
WelchForest_realFinal_our_training.m
forest_our_training.m
Command Window

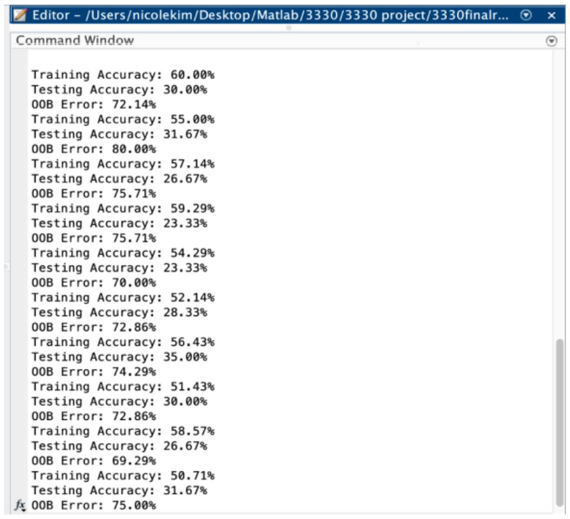
```
Training Accuracy: 77.86%
Testing Accuracy: 48.33%
OOB Error: 62.86%
Training Accuracy: 74.29%
Testing Accuracy: 43.33%
OOB Error: 64.29%
Training Accuracy: 77.14%
Testing Accuracy: 33.33%
OOB Error: 58.57%
Training Accuracy: 75.00%
Testing Accuracy: 40.00%
OOB Error: 63.57%
Training Accuracy: 77.14%
Testing Accuracy: 45.00%
OOB Error: 66.43%
Training Accuracy: 72.14%
Testing Accuracy: 40.00%
OOB Error: 59.29%
Training Accuracy: 72.86%
Testing Accuracy: 41.67%
OOB Error: 60.00%
Training Accuracy: 70.00%
Testing Accuracy: 38.33%
OOB Error: 62.86%
Training Accuracy: 75.00%
Testing Accuracy: 35.00%
OOB Error: 68.57%
Training Accuracy: 77.86%
Testing Accuracy: 35.00%
fx OOB Error: 60.71%
```

**Figure 20: Results with Given Power**



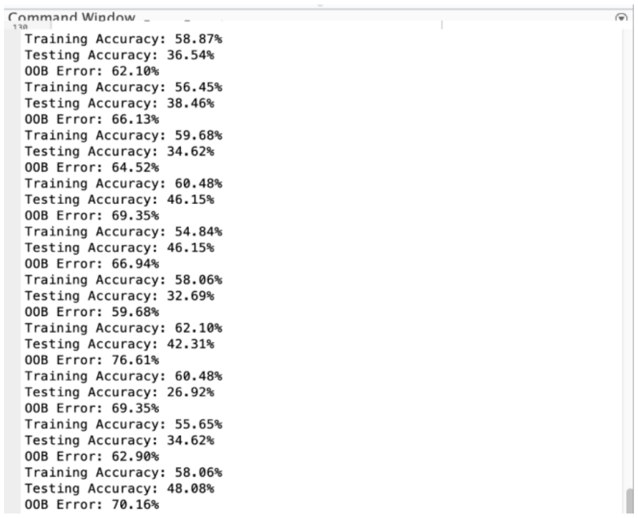
```
Training Accuracy: 52.42%
Testing Accuracy: 34.62%
OOB Error: 64.52%
Training Accuracy: 51.61%
Testing Accuracy: 42.31%
OOB Error: 66.13%
Training Accuracy: 52.42%
Testing Accuracy: 30.77%
OOB Error: 66.94%
Training Accuracy: 54.84%
Testing Accuracy: 36.54%
OOB Error: 62.10%
Training Accuracy: 55.65%
Testing Accuracy: 38.46%
OOB Error: 66.94%
Training Accuracy: 60.48%
Testing Accuracy: 36.54%
OOB Error: 68.55%
Training Accuracy: 60.48%
Testing Accuracy: 28.85%
OOB Error: 64.52%
Training Accuracy: 62.90%
Testing Accuracy: 34.62%
OOB Error: 66.13%
Training Accuracy: 57.26%
Testing Accuracy: 32.69%
OOB Error: 62.10%
Training Accuracy: 53.23%
Testing Accuracy: 40.38%
OOB Error: 64.52%
```

**Figure 21: Results with PSD**


Editor – /Users/nicolekim/Desktop/Matlab/3330/3330 project/3330finalr...
Command Window

```
Training Accuracy: 60.00%
Testing Accuracy: 30.00%
OOB Error: 72.14%
Training Accuracy: 55.00%
Testing Accuracy: 31.67%
OOB Error: 80.00%
Training Accuracy: 57.14%
Testing Accuracy: 26.67%
OOB Error: 75.71%
Training Accuracy: 59.29%
Testing Accuracy: 23.33%
OOB Error: 75.71%
Training Accuracy: 54.29%
Testing Accuracy: 23.33%
OOB Error: 70.00%
Training Accuracy: 52.14%
Testing Accuracy: 28.33%
OOB Error: 72.86%
Training Accuracy: 56.43%
Testing Accuracy: 35.00%
OOB Error: 74.29%
Training Accuracy: 51.43%
Testing Accuracy: 30.00%
OOB Error: 72.86%
Training Accuracy: 58.57%
Testing Accuracy: 26.67%
OOB Error: 69.29%
Training Accuracy: 50.71%
Testing Accuracy: 31.67%
OOB Error: 75.00%
```

**Figure 22: Results with Welch 1**



```
Training Accuracy: 58.87%
Testing Accuracy: 36.54%
OOB Error: 62.10%
Training Accuracy: 56.45%
Testing Accuracy: 38.46%
OOB Error: 66.13%
Training Accuracy: 59.68%
Testing Accuracy: 34.62%
OOB Error: 64.52%
Training Accuracy: 60.48%
Testing Accuracy: 46.15%
OOB Error: 69.35%
Training Accuracy: 54.84%
Testing Accuracy: 46.15%
OOB Error: 66.94%
Training Accuracy: 58.06%
Testing Accuracy: 32.69%
OOB Error: 59.68%
Training Accuracy: 62.10%
Testing Accuracy: 42.31%
OOB Error: 76.61%
Training Accuracy: 60.48%
Testing Accuracy: 26.92%
OOB Error: 69.35%
Training Accuracy: 55.65%
Testing Accuracy: 34.62%
OOB Error: 62.90%
Training Accuracy: 58.06%
Testing Accuracy: 48.08%
OOB Error: 70.16%
```

**Figure 23: Results with Welch 2**

## References:

[1]. T. Srivastava, “12 important model evaluation metrics for Machine Learning Everyone should know (updated 2023),” Analytics Vidhya, https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/#Confusion_Matrix (accessed Dec. 9, 2023).

[2]. J. Moini and P. Piran, “Cerebral cortex,” *Functional and Clinical Neuroanatomy*, pp. 177–240, 2020, doi: https://doi.org/10.1016/b978-0-12-817424-1.00006-9

[3]. D.-W. Kim and C. Im, “EEG Spectral Analysis,” *Biological and medical physics series*, pp. 35–53, Jan. 2018, doi: https://doi.org/10.1007/978-981-13-0908-3_3

[4]. “Main features of the EEG amplifier explained,” *Bitbrain*, Apr. 03, 2020. Available: https://www.bitbrain.com/blog/eeg-amplifier

[5] Yong X, Menon C. EEG classification of different imaginary movements within the same limb. PLoS One. 2015 Apr 1;10(4):e0121896. doi: 10.1371/journal.pone.0121896. PMID: 25830611; PMCID: PMC4382224.

[6] A. Prakash, “Precision/Recall (perfcurve),” *ww2.mathworks.cn*. Available: https://ww2.mathworks.cn/matlabcentral/answers/486122-precision-recall-perfcurve. [Accessed: Dec. 10, 2023]

[7] “What is brain-computer interface (BCI)? - Definition from WhatIs.com,” *WhatIs.com*. Available: https://www.techtarget.com/whatis/definition/brain-computer-interface-BCI