Signals and Systems Software Homework Report

Instructor: Prof. Karbalaei

Sharif University of Technology

Project(Phase 1 and Phase 2): Epileptic Seizure Prediction Using Spectral Entropy-Based Features of EEG

By Reza Nayeb Habib
Student ID# 401102694

Contents

1 Electrodes naming

the naming convention of EEG is for showing different brain parts.
the numbering starts from low numbers when near the center of the brain and goes higher when we get farther from the brain. the left side of head has odd numbers while the right side has even numbering.
F is for the frontal lobe of the brain so for example Fz is the central electrode.
P is for the Parietal lobe and O for Occipital lobe. also the left and right side shown with T are for brains Temporal lobes.

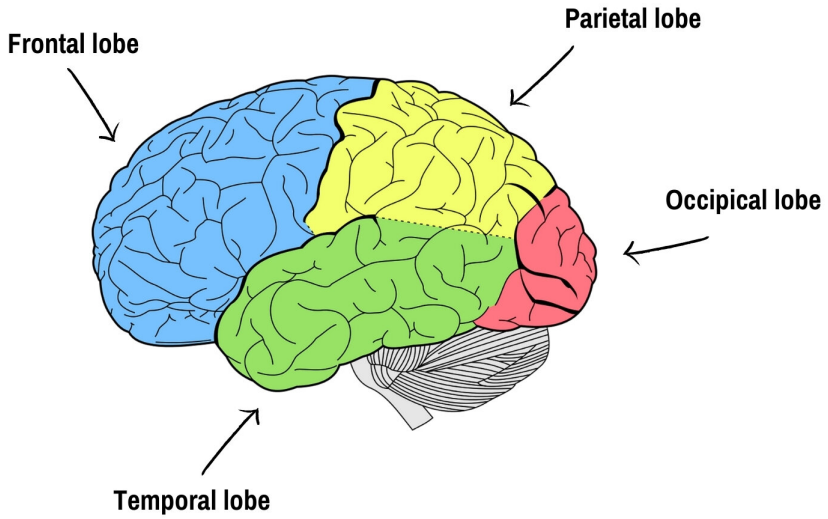


Figure 1: brain Lobes

2 Frequency Bands of EEG

Determine the activities each frequency band is associated with:

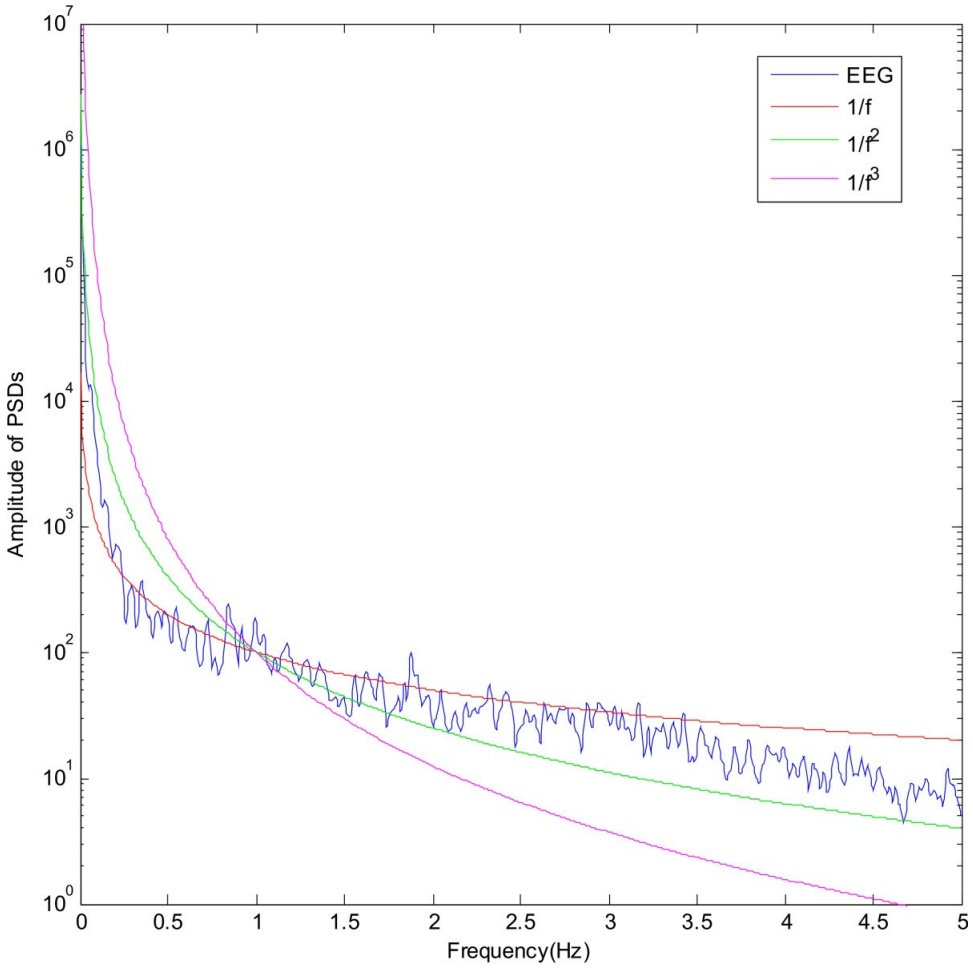


Figure 2: EEG frequency spectrum

The EEG signal spectrum usually has a frequency spectrum with the shape $\frac{1}{f}$ (see the figure above), this means that we won't likely have any frequencies higher than 100Hz from the EEG and we usually consider frequency content higher than 200Hz as noise. in figure 2 we can see different bands of EEG signal and their conventional names. we will explain each frequency's corresponding brain functionality:

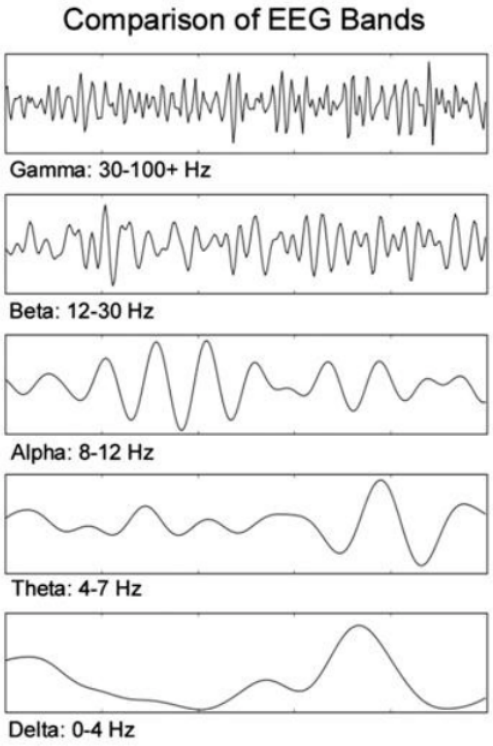


Figure 3: EEG frequency bands

Delta Waves: these waves are typically generated when the brain is at rest or deep sleep. having too little may indicate poor sleep or Inability to rejuvenate body. having too much of this frequency band's content may indicate Brain injuries, learning problems, inability to think or severe ADHD.

Theta Waves: these waves are typically associated with creative thinking, emotional connection, intuition or relaxation. having too little may indicate anxiety, stress or poor emotional awareness. having too much of this frequency band's content may indicate ADHD, depression, inattentiveness or hyperactivity.

Alpha Waves: these waves are typically associated with relaxation. having too little may indicate Anxiety, high stress, insomnia or OCD. having too much of this frequency band's content may indicate Daydreaming, inability to focus or being too relaxed.

Beta Waves: these waves are typically associated with conscious focus, memory and problem solving. having too little may indicate ADHD, daydreaming, depressionor poor cognition. having too much of this frequency band's content may indicate adrenaline, anxiety, high arousal, inability to relax, stress.

Gamma Waves: these waves are typically associated with Binding senses, cognition, information processing, learning, perception or REM sleep. having too little may indicate ADHD, depression or learning disabilities. having too much of this frequency band's content may indicate Anxiety, high arousal or stress.

3 Sampling Frequency

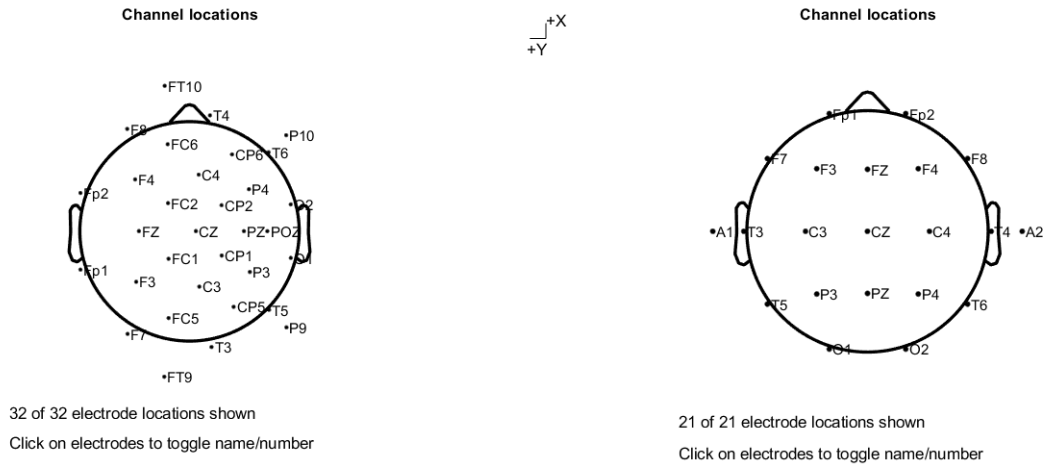
Based on frequency bands and Nyquist criterion, which sampling frequencies are preferred for EEG signals?
As mentioned in the previous section, brain signals usually are below 100Hz. Nyquist criterion states that the sampling frequency should be at least twice the highest frequency available in the signal so 200Hz sampling rate would be a reasonable choice for most EEG applications

4 Pre-processing

here the steps of Preprocessing are presented and the gui and coding tools are explained. the complete code is presented at the end of this section, also the deliverables will be presented at the end of this pdf.

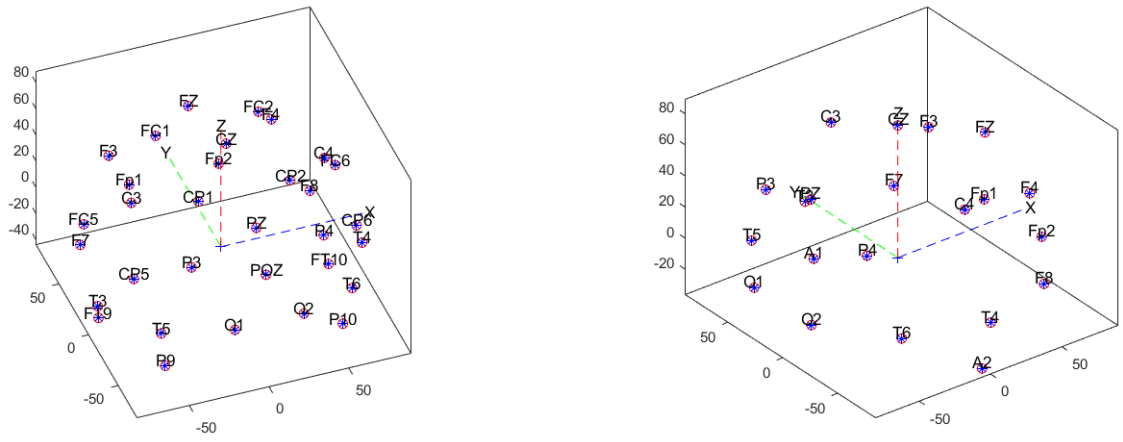
4.1 Locating Channels

using edit chanel locations in EEGlab and importing the given chanel data the channels names were attached to them(the function used is pop_chanedit())



(a) data 1 (b) data 2

Figure 4: Channel Locations



(a) data 1 (b) data 2

Figure 5: Channel Locations 3D view

4.2 High Pass filtering

here we use a high pass filter with lower band of 1 Hz to cut baseline drifts(using pop_eegfiltnew()).

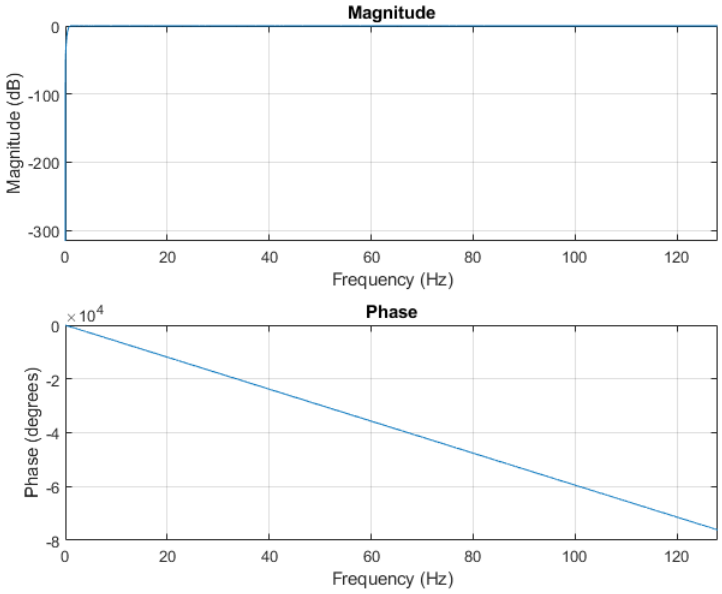


Figure 6: the high pass filter used

4.3 Notch line filter

here we use a notch filter to cut the power line frequency(usually 50 Hz or 60 Hz), using pop_cleanline() function.

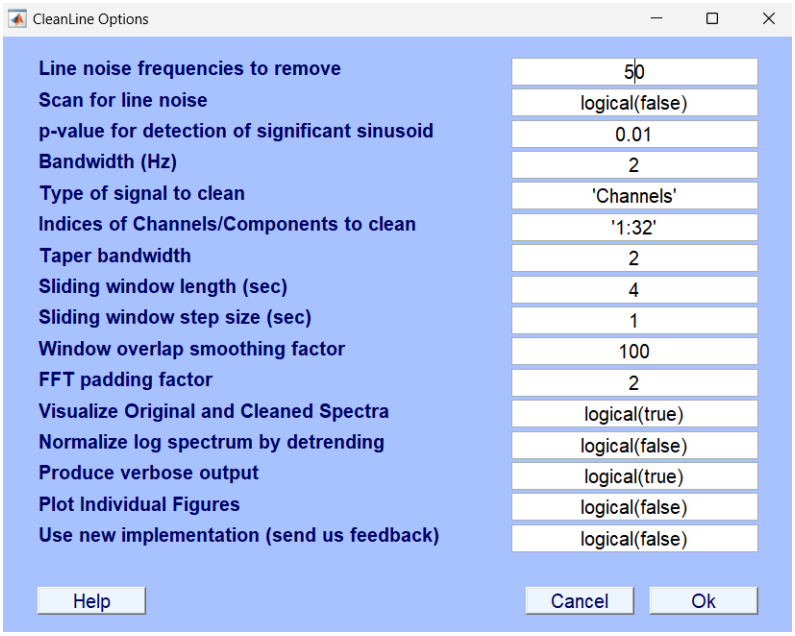
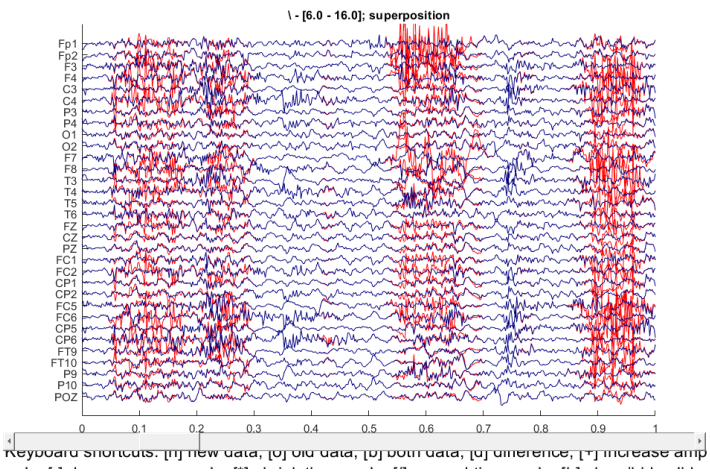


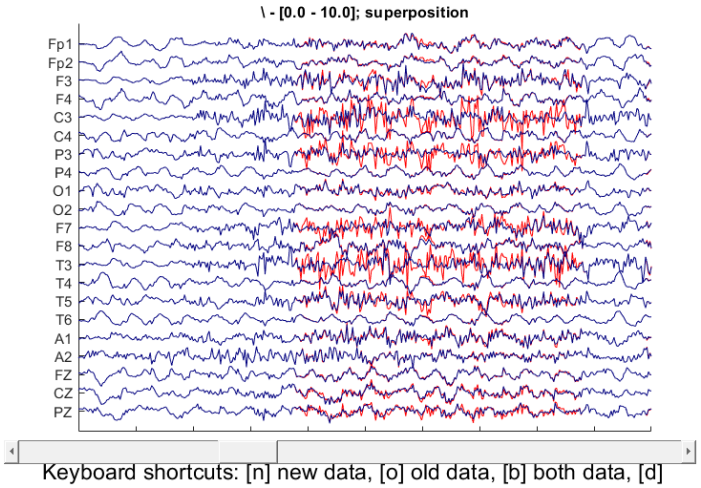
Figure 7: Clean Line panel in EEGlab

4.4 Artifact Removal(and re-referencing)

first we re-reference the data to the average meaning we take the average as the reference for all signals of the electrodes which helps remove the common noise which may be on all the electrodes using pop_reref(). then we use pop_clean_rawdata() to clean the data of any noises that can be detected without ICA such as eye activities or line noises, this method isn't as effective as ICA but will make the data cleaner for further ICA analysis.



(a) data 1



(b) data 2

Figure 8: data before and after Artifact removal

the we again re-reference the data.

4.5 Independent Component Analysis(ICA)

Independent Component Analysis(ICA) is a blind source seperation method which is a way of finding signal sources that are mixed together by analyzing the various signals received from different positions.(see the figure below)

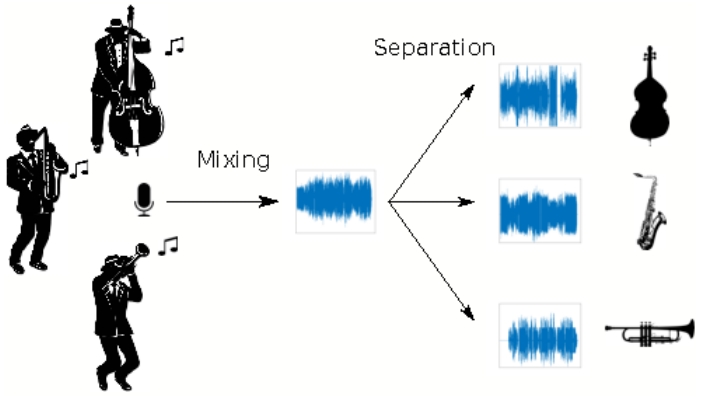


Figure 9: blind source seperation

now using this algorithm we find different components that create the signals we receive from EEG and then analyze these components by ICA_label() wich is trained on brain signals and tries to tell use which signal source is a brain signal and which are likely noise like muscle, eye blinking or line noise.
now using pop_runica() we start the ICA algorithm:

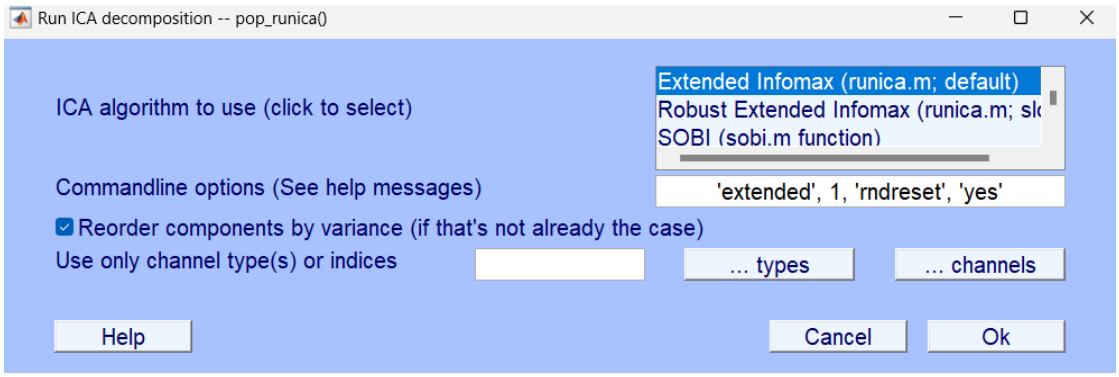


Figure 10: ICA panel in EEGlab gui

after that we need to give a head model for the dipoles to be fitted to the brain. using pop_dipfit_setting() we fit the MNI model to our brain model to get better results.

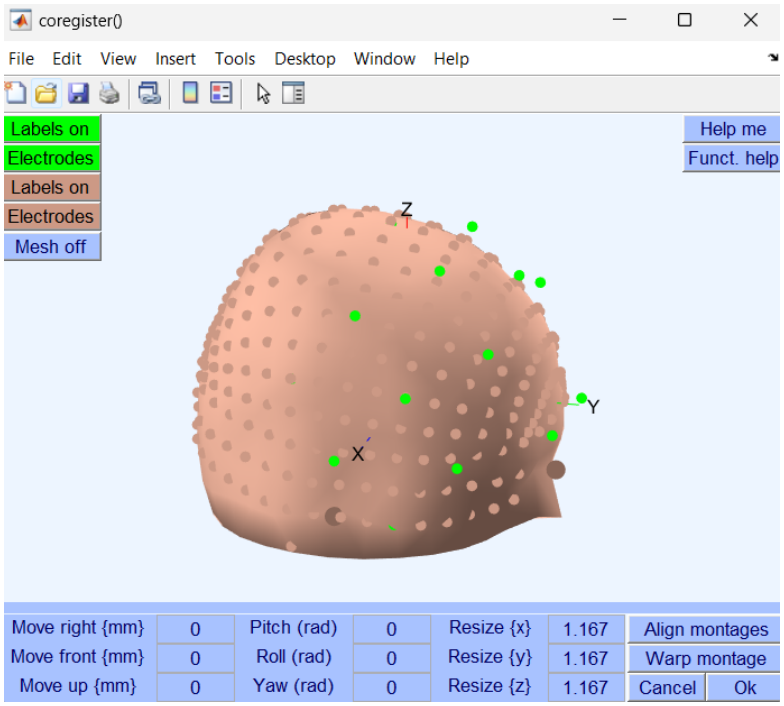
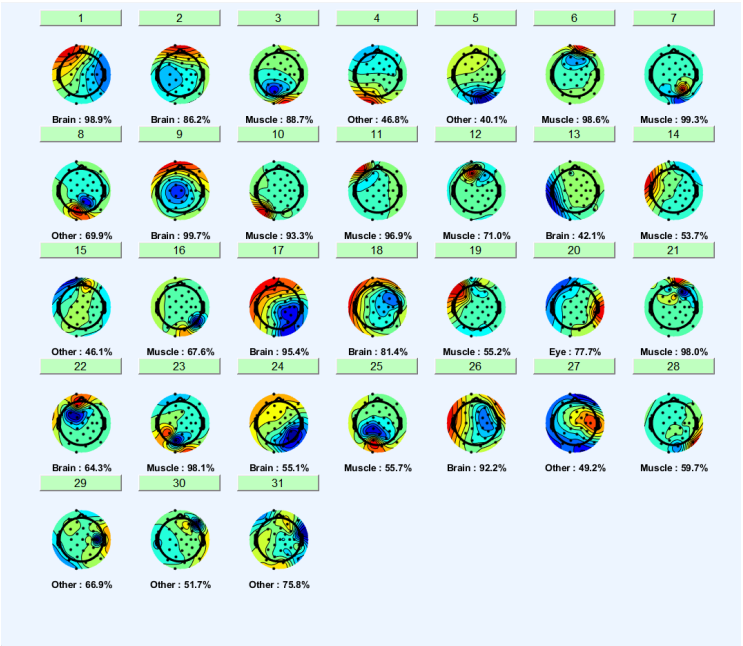
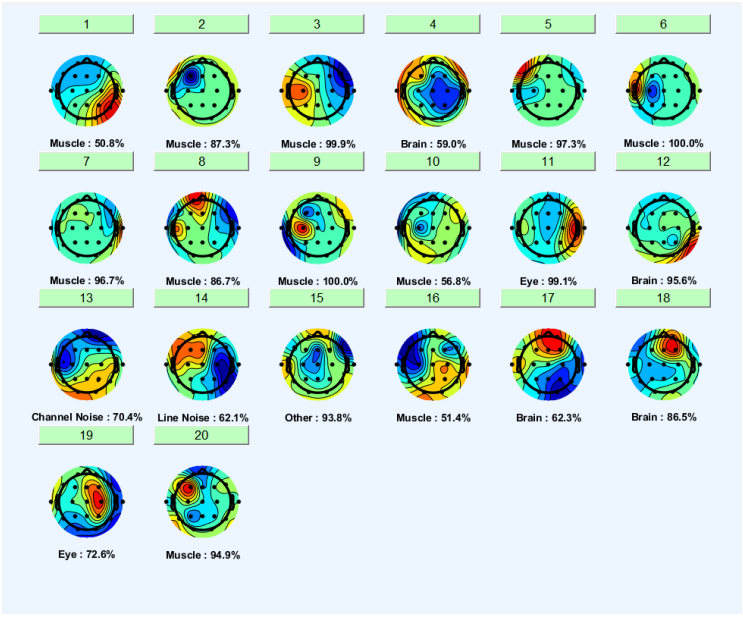


Figure 11: our electrodes before being fitted to the MNI head model

after that we run autofit() in EEGlab that does the br fits the dipoles.
then the ICA analysis is ready the ICA components can be seen below:



(a) data 1



(b) data 2

Figure 12: ICA sources

4.6 Epoching data

Epoching data is breaking the signals into multiple parts in order to better process them, a long session of EEG recording can have multiple things happening in them that should not be processed together, for example if the patient is stimulated at a special time at the session with one stimulant and at another time with another one we may want to do different processing(and even pre-processing) on the first and another on the second signal because of the different nature if the tasks.

4.7 Code

the code history is presented in a zip file that is alongside this PDF.

5 Deliverables

Frequency Spectrum:

the following figures show FZ channel before and after pre-processing on data 1 and 2.

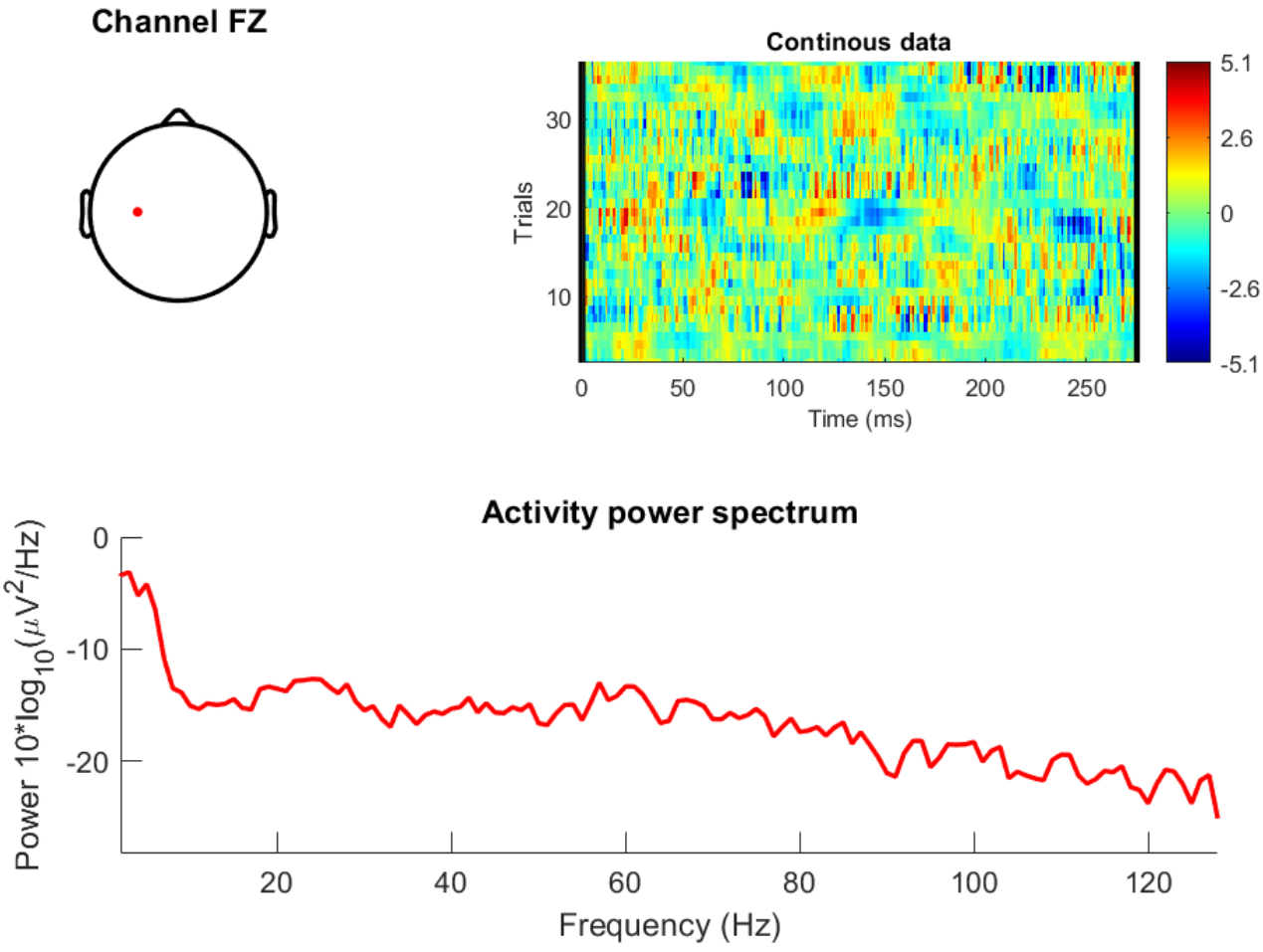


Figure 13: data 1 raw FZ channel

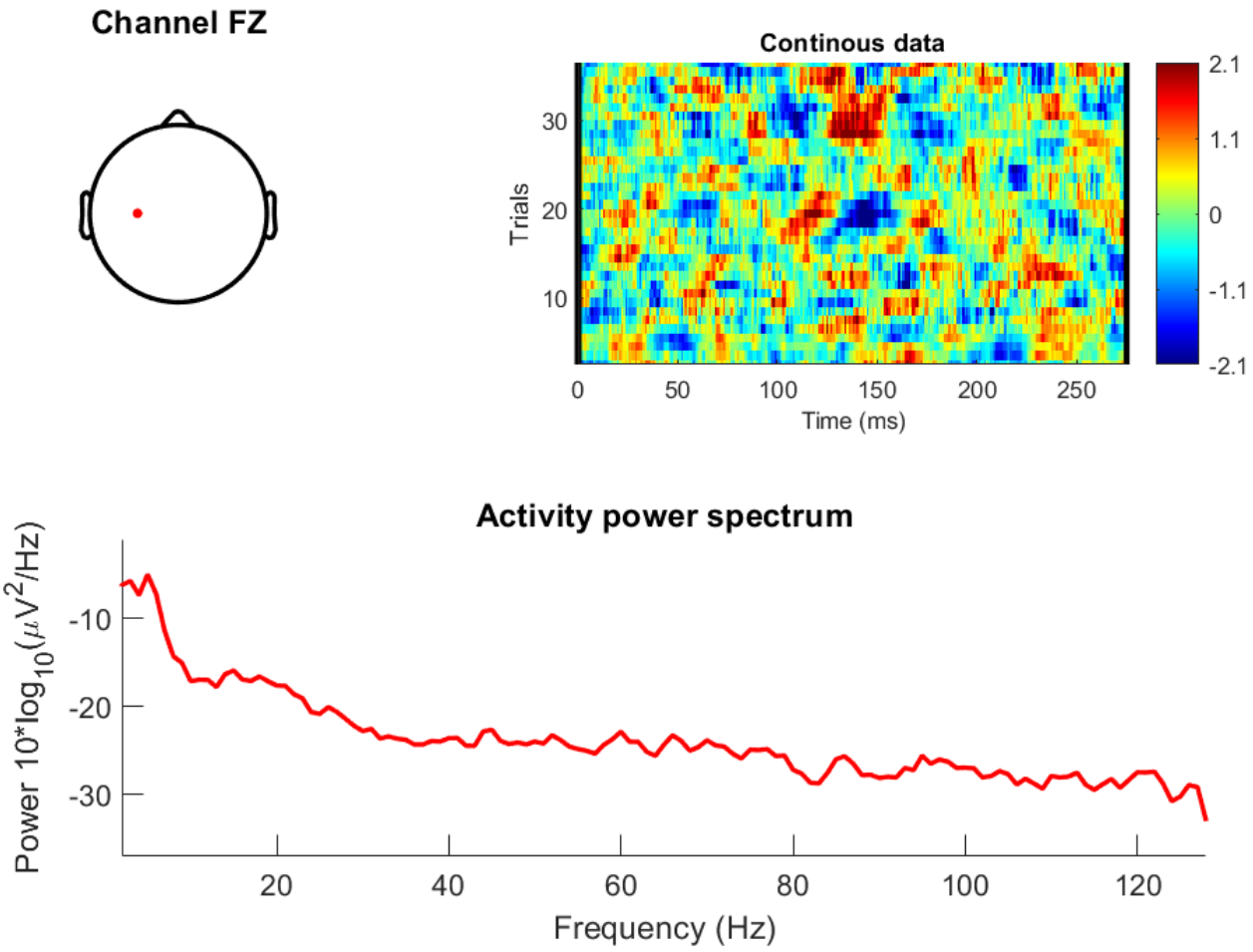


Figure 14: data 1 pre-processed FZ channel

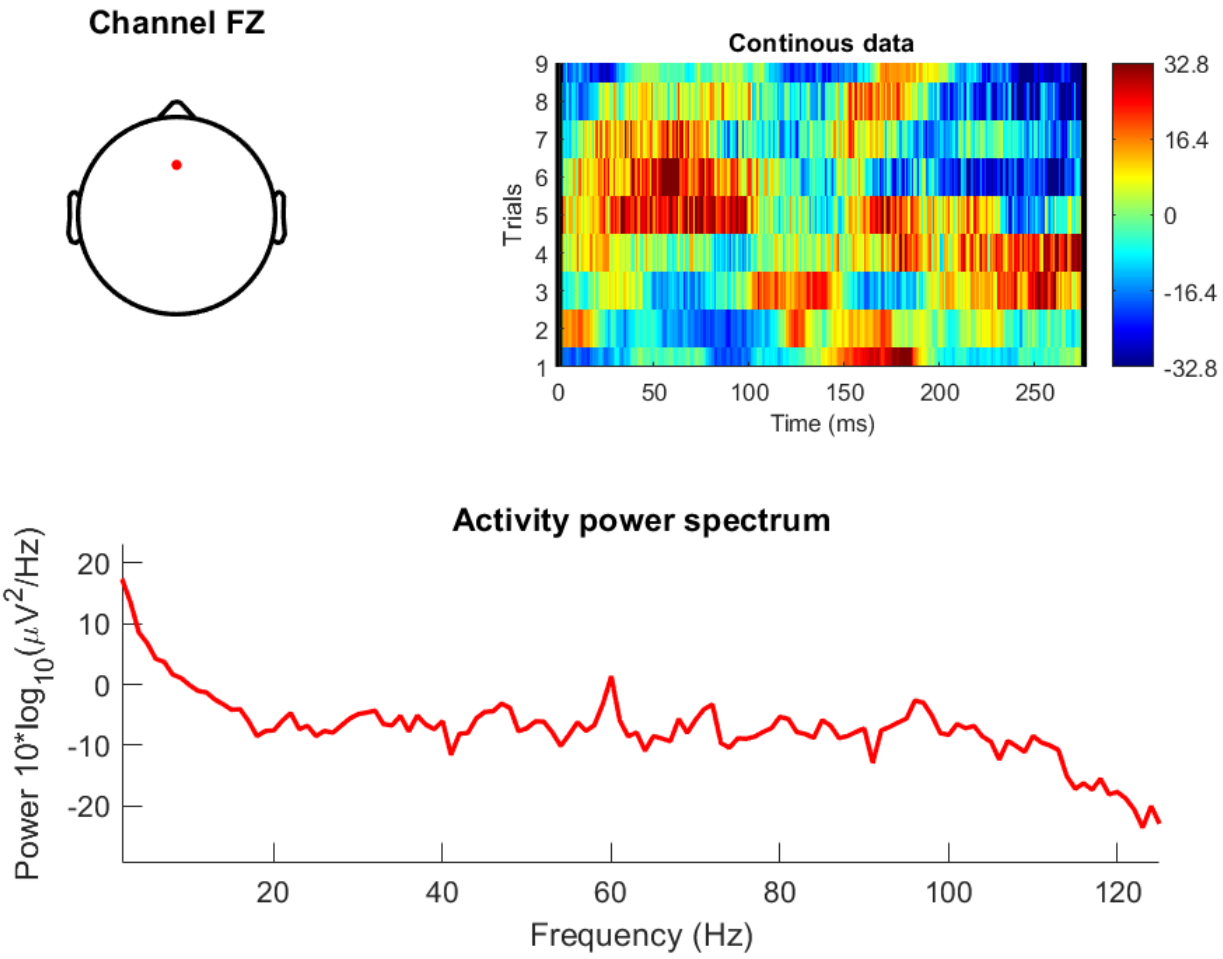


Figure 15: data 2 raw FZ channel

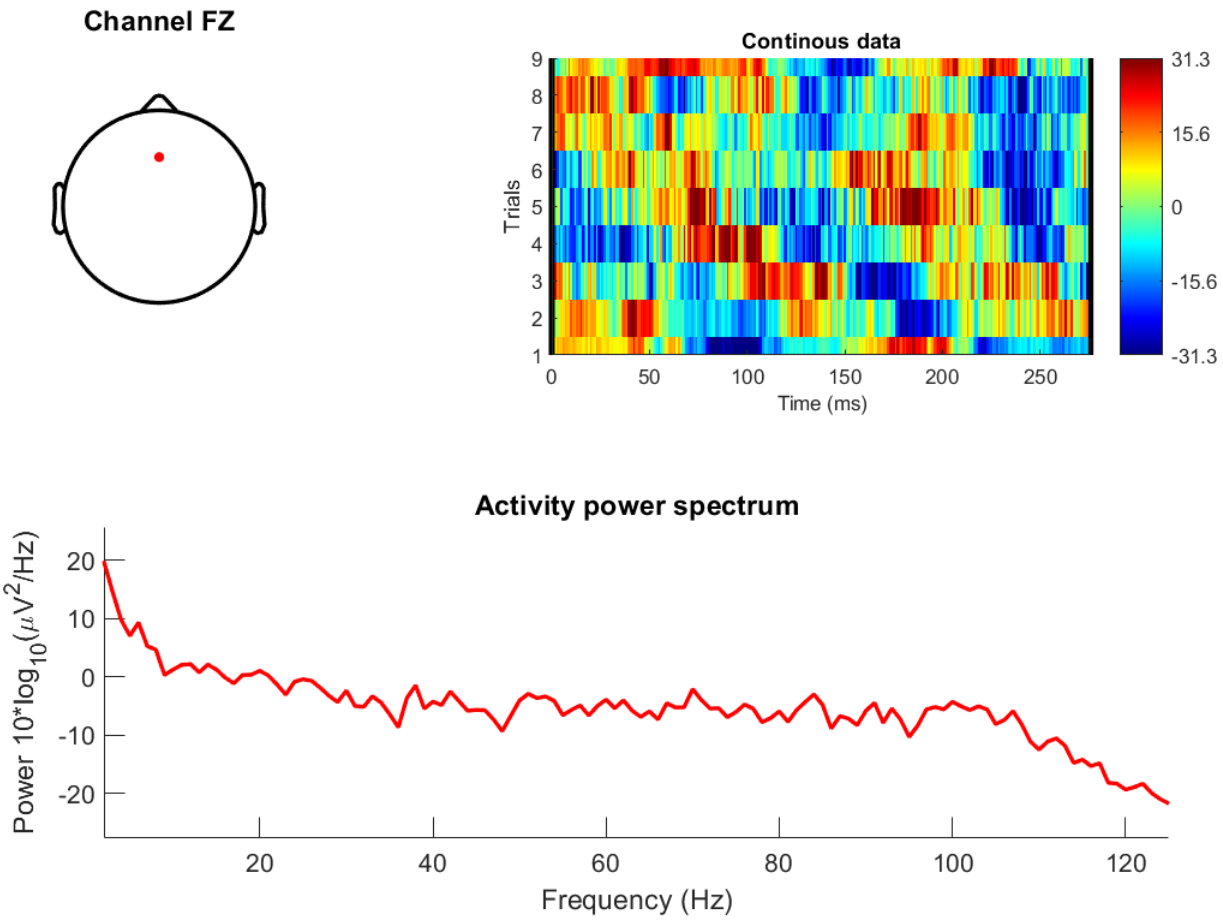


Figure 16: data 2 pre-processed FZ channel

ICA components:

here we present one of brain components that are present in each ICA:

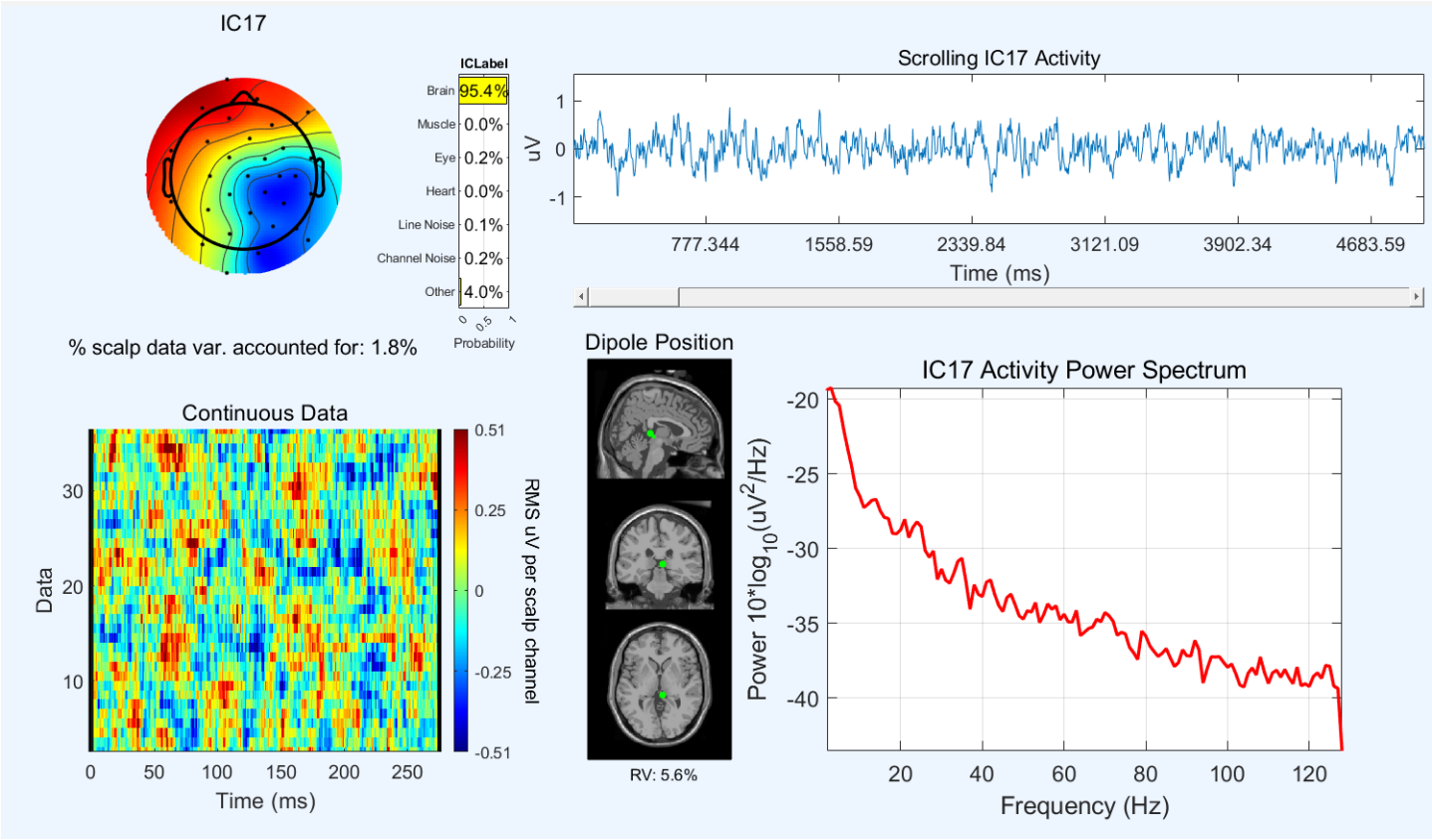


Figure 17: a brain component from data 1

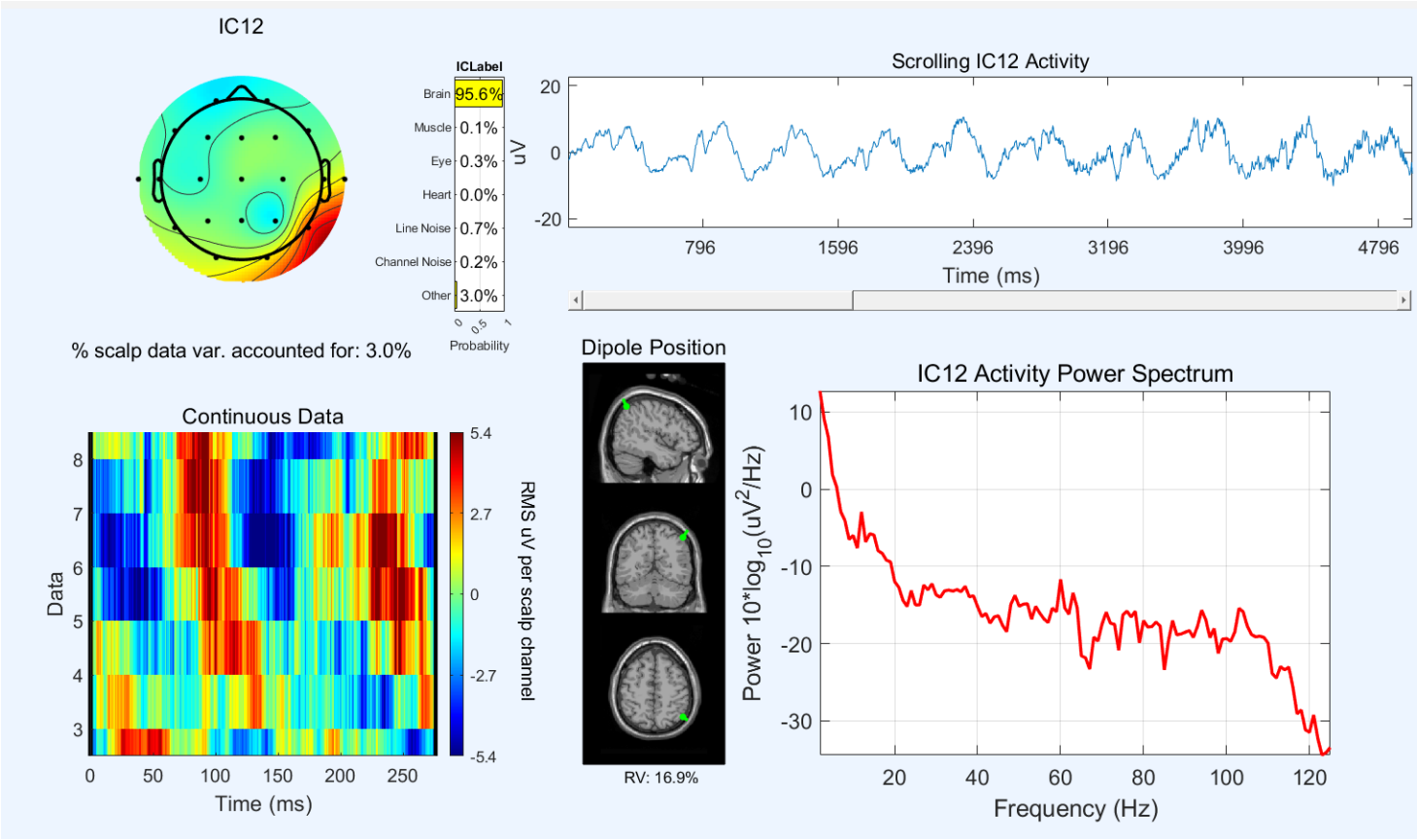


Figure 18: a brain component from data 2

5.1 Processed Data

the processed data is attached in a zip file presented alongside this PDF file.
the data is much cleaner and the frequency spectrum resembles a better $\frac{1}{f}$ shape which we expect a brain signal to be like. the spectrum seems much smoother and it can be seen that the line frequency which is like a spike atb 60 Hz in data 2 is gone here are some figures showing the pre-processed data:

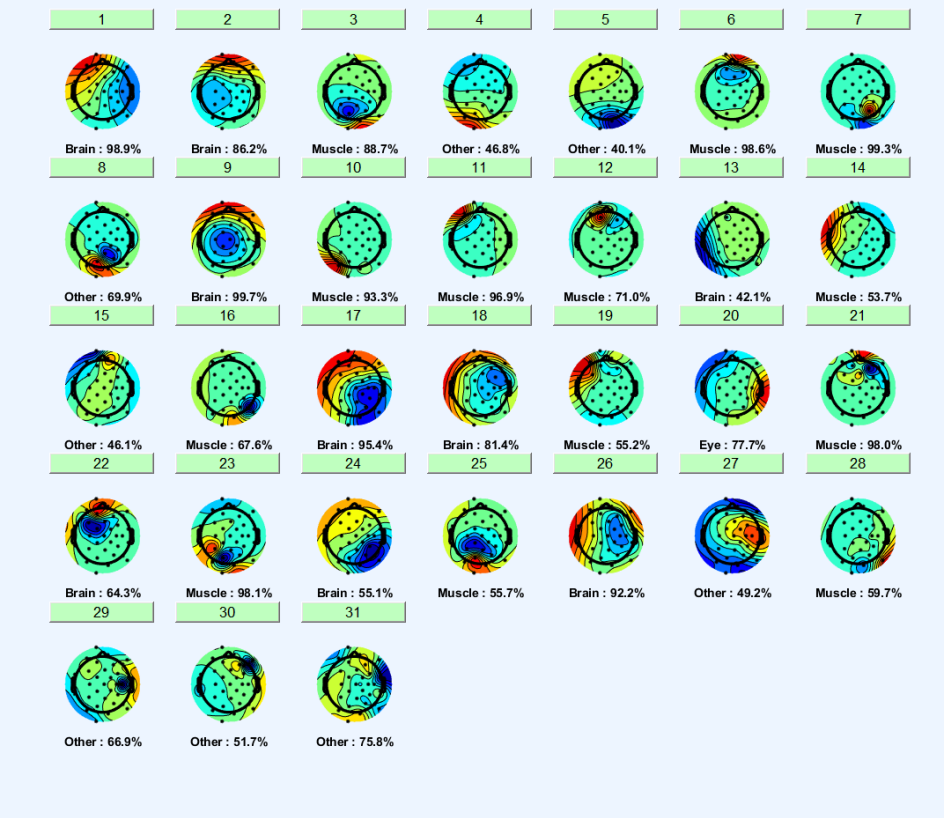


Figure 19: data 1 ICA

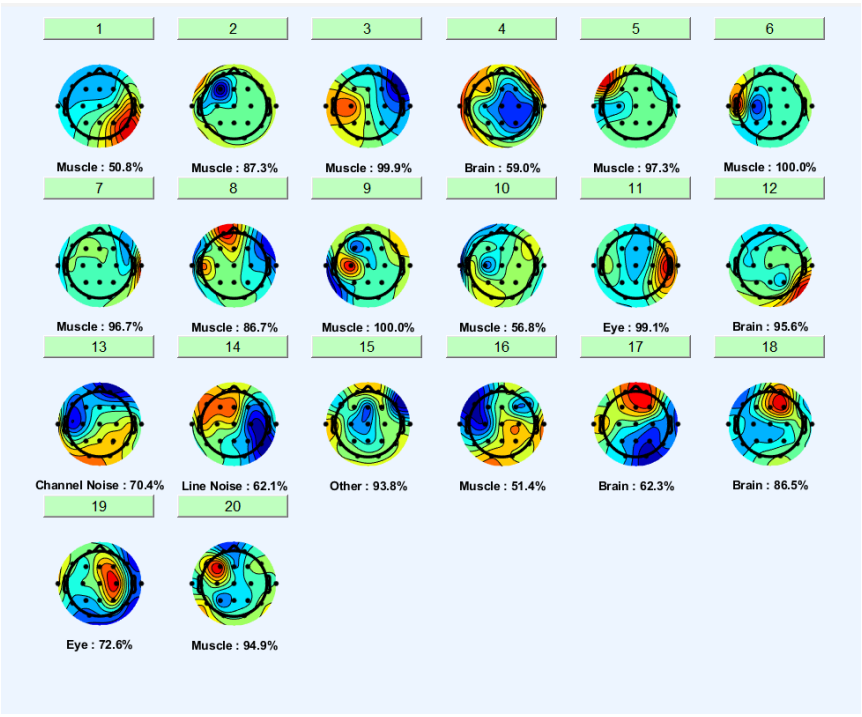


Figure 20: data 2 ICA

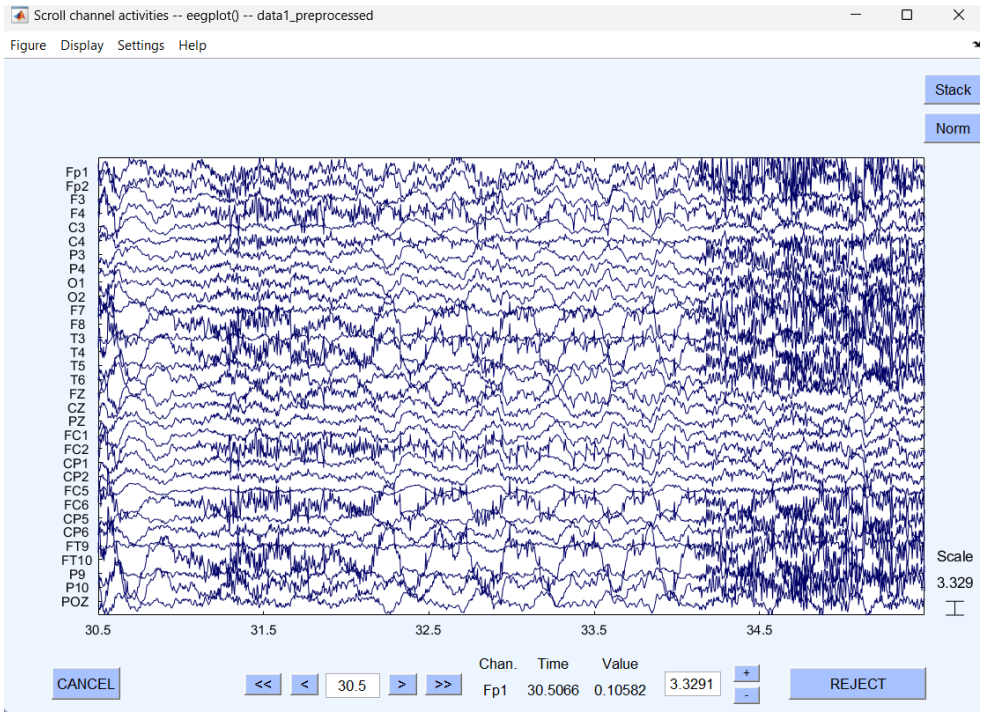


Figure 21: data 1 signal after Preprocessing(a change in signal can be seen that is likely the seizure)

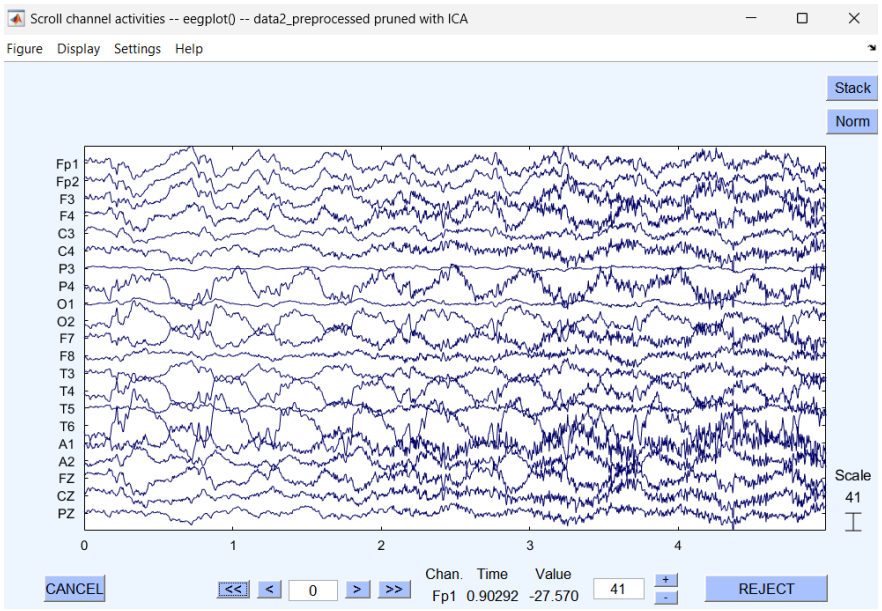


Figure 22: data 2 signal after Preprocessing

6 Phase2 Tasks

6.1 Loading Database

We load the data with the following code:

```
%% Loading data and setting
clc; clear all; close all;
mc02 = tt2Mat(edfread("chb01_02.edf"));
mc03 = tt2Mat(edfread("chb01_03.edf"));
mc04 = tt2Mat(edfread("chb01_04.edf"));
mc15 = tt2Mat(edfread("chb01_15.edf"));
mc16 = tt2Mat(edfread("chb01_16.edf"));
mc18 = tt2Mat(edfread("chb01_18.edf"));
mc26 = tt2Mat(edfread("chb01_26.edf"));
```

the function tt2Mat() creates a matrix out of a timeTable:

```
% the following function converts a timeTable to Matrix
% each column represents one EEG channel
% each row represents one sample in time
function data = tt2Mat(tt)
    cells = tt{:,:};
    data = cell2mat(cells);
end
```

6.2 PSD calculation Example

we calculated the PSD of the signal with pwelch() function of MATLAB with the following code:

```
%% PSD example
clc; clear all; close all;
tt = edfread("chb01_18.edf");
data = tt2Mat(tt);

pwelch(data)
```

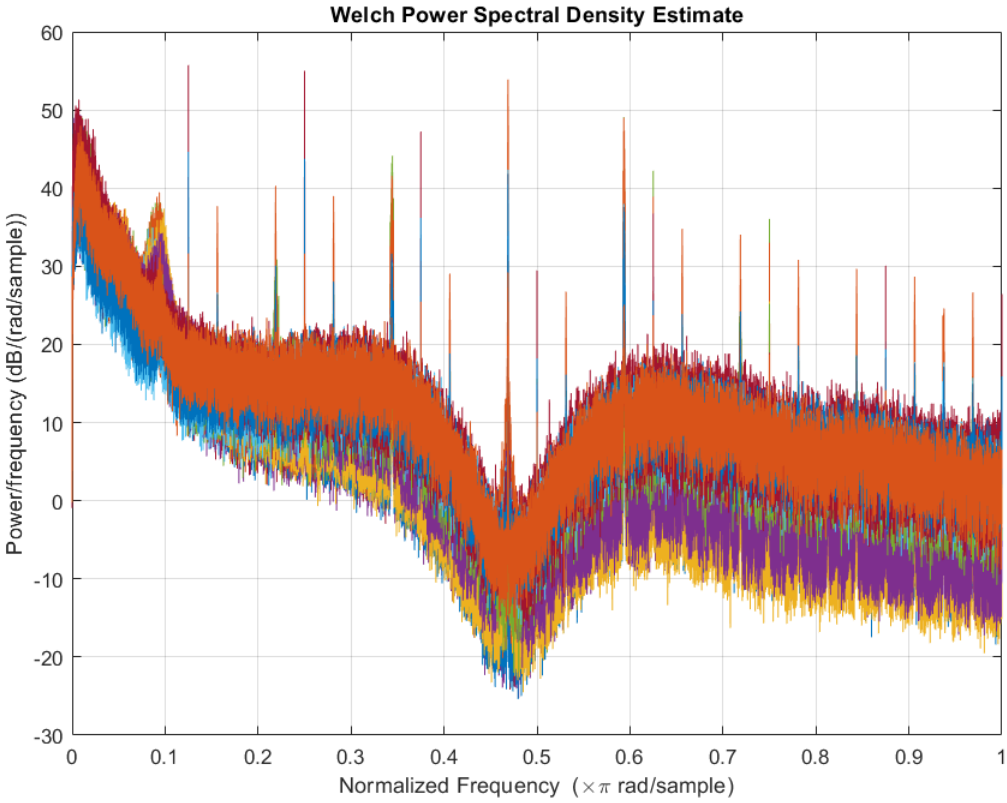


Figure 23: PSD spectrum of a chb01-18

6.3 Shannon Entropy calculation Example

In the following code Shannon Entropy is calculated on 16 second epochs:

```
%% shannon entropy
clc; clear all; close all;
mc02 = tt2Mat(edfread("chb01_02.edf"));
shannonE = shannonEpoch16(getDataBeforeTime(mc02,600,700,256), 16, 256);
plot(shannonE);
title('Shannon Entropy for 16 second Epochs on differnet Channels');
```

the function shannonEpoch16() is as following:

```
function y = shannonEpoch16(signal, lenSec, fs)
y = [];
n = fs*lenSec;
for i = 0:floor(length(signal)/n)-1
    e = signal(i*n+1:(i+1)*n,:);
    npsd = normalize(pwelch(e));
    shannon = transpose(wentropy(transpose(npsd)));
    y = [y; shannon];
end
```

```
10
11      if mod(length(signal),n) ~= 0
12          e = signal(i*n+1:end,:);
13          npsd = normalize(pwelch(e));
14          shannon = wentropy(npsd')';
15          y = [y; shannon];
16          end
17      end
```

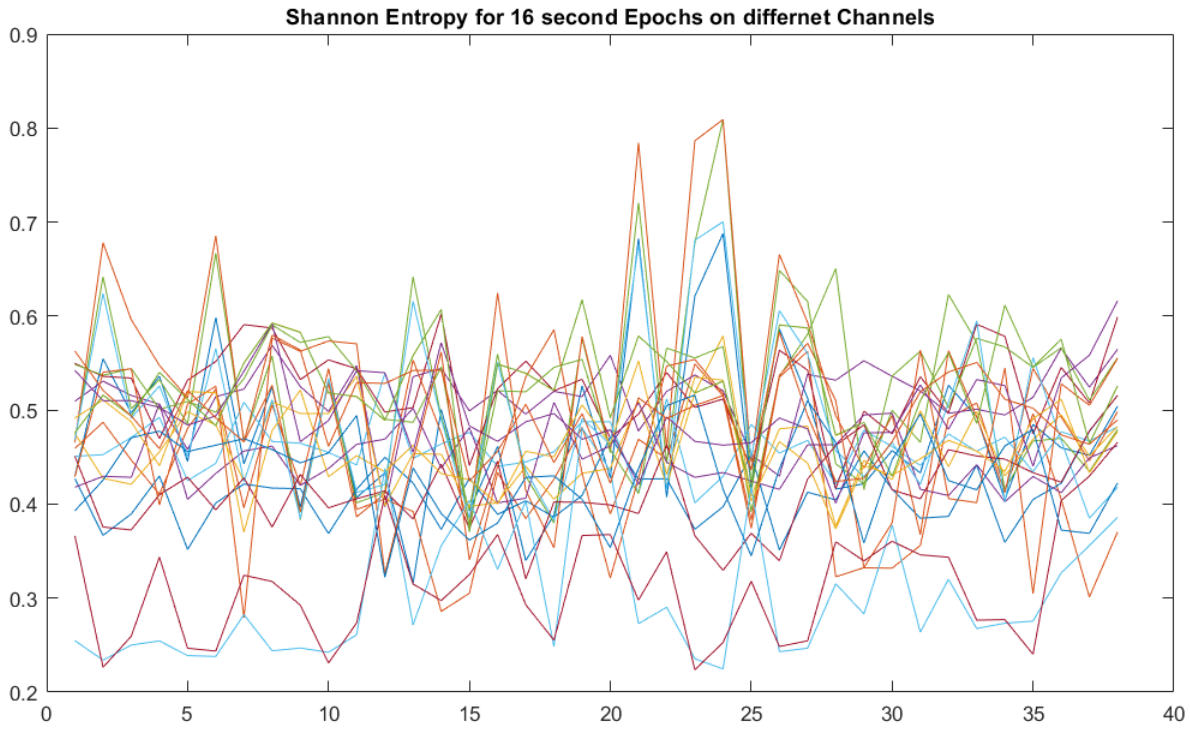


Figure 24: shannon Entropy versus time for 16 second epochs

6.4 Feature Extraction

Our feature Extraction method consists of finding Minimum of epoch, Maximum of epoch, Average value of epoch, standard deviation of epoch and the Shannon Entropy of the epoch and doing this for all of 23 the channels. we then put all these features for all the epochs in one vector. we also extract every data points meaning(for which channel and epoch it is and what type of feature it is) in a vector named info to later be able to find which data points were important for t-test. the feature extraction function is as the following:

```
1       function [f,info] = getFeature(signal, lenSec, fs)
2       info = [];
3       f = [];
4       n = fs*lenSec;
```

```
    for i = 0:floor(length(signal)/n)-1
        e = signal(i*n+1:(i+1)*n,:);
        normPSD = normalize(pwelch(e));
        shannon = transpose(wentropy(transpose(normPSD)));
        mu = mean(e);
        sigma = std(e);
        minVal = min(e);
        maxVal = max(e);
        f = [f shannon mu sigma maxVal minVal];
        for j = 1:23
            info = [info "ent" + string(i) + "-"+ string(j)];
        end

        for j = 1:23
            info = [info "mean" + string(i) + "-" + string(j)];
        end

        for j = 1:23
            info = [info "std" + string(i) + "-" + string(j)];
        end

        for j = 1:23
            info = [info "max" + string(i) + "-" + string(j)];
        end

        for j = 1:23
            info = [info "min" + string(i) + "-" + string(j)];
        end

    end

    if mod(length(signal),n) ~= 0
        e = signal(i*n+1:end,:);
        normPSD = normalize(pwelch(e));
        shannon = wentropy(normPSD')';
        mu = mean(e);
        sigma = std(e);
        minVal = min(e);
        maxVal = max(e);
        f = [f shannon mu sigma maxVal minVal];
        for j = 1:23
            info = [info "ent" + string(i) + "-"+ string(j)];
        end

        for j = 1:23
            info = [info "mean" + string(i) + "-" + string(j)];
        end

        for j = 1:23
            info = [info "std" + string(i) + "-" + string(j)];
```

```
        end

        for j = 1:23
            info = [info "max" + string(i) + "-" + string(j)];
        end

        for j = 1:23
            info = [info "min" + string(i) + "-" + string(j)];
        end
    end
end
```

now using this function in our own code we get:

```
    %% Feature extraction Section
    [A1,info] = getFeature(getDataBeforeTime(mc02,600,700,256), 16, 256);
    [A2,~] = getFeature(getDataBeforeTime(mc02,600,1400,256), 16, 256);
    [A3,~] = getFeature(getDataBeforeTime(mc02,600,2100,256), 16, 256);
    [A4,~] = getFeature(getDataBeforeTime(mc02,600,2600,256), 16, 256);
    [A5,~] = getFeature(getDataBeforeTime(mc02,600,3100,256), 16, 256);
    [A6,~] = getFeature(getDataBeforeTime(mc02,600,3600,256), 16, 256);
    [B1,~] = getFeature(getDataBeforeTime(mc03,600,2400,256), 16, 256);
    [B2,~] = getFeature(getDataBeforeTime(mc18,600,800,256), 16, 256);
    [C1,~] = getFeature(getDataBeforeTime(mc03,600,3100,256), 16, 256);
    [C2,~] = getFeature(getDataBeforeTime(mc04,600,1500,256), 16, 256);
    [C3,~] = getFeature(getDataBeforeTime(mc15,600,1800,256), 16, 256);
    [C4,~] = getFeature(getDataBeforeTime(mc16,600,1100,256), 16, 256);
    [C5,~] = getFeature(getDataBeforeTime(mc18,600,1850,256), 16, 256);
    [C6,~] = getFeature(getDataBeforeTime(mc26,600,2100,256), 16, 256);

    allFeatures = [ A1; A2; A3; A4; A5; A6; B1; B2; C1; C2; C3; C4; C5;
    C6];

    [~,p] = ttest2(allFeatures(1:8,:),allFeatures(9:end,:));
    pvalue = 0.005;
    features = [];
    finfo = [];
    for i = 1:length(p)
        if p(i) < pvalue
            features = [features allFeatures(:,i)];
            finfo = [finfo info(i)];
        end
    end

    label = [0; 0; 0; 0; 0; 0; 0; 0; 1; 1; 1; 1; 1; 1];
    trainLabel = [label(1:5); label(7); label(9:12)];
    trainData = [features(1:5,:); features(7,:); features(9:12,:)];
    testData = [features(6,:); features(8,:); features(13:14,:)];
```

at the start we had 4370 features (23 channles $\times$ 5 features $\times$ 38 epochs $=$ 4370) but filtering we extracted only 444 features for further analysis.

6.5 SVM classifier implementation

SVM classifier is a linear classifier used on n-dimensional classifier we have 444 features so out classifier works with 444 dimensions. here we present the code used for training and testing the classifier:

```
%% SVM
clc;
SVMclassifier = fitcsvm(trainData,trainLabel);
[labelSVM,scoreSVM] = predict(SVMclassifier, testData);
```

the SVM gives 100% right answers on this test.

6.6 KNN classifier implementation

The KNN classifier finds the nearest neighbors of the point in n-dimensional space and labels the point by looking at its neighbors.
below we used MATLAB built-in function for implementing KNN for our seizure detector code:

```
%% KNN
clc;
KNNmodel = fitcknn(trainData,trainLabel);
[labelKNN,scoreKNN] = predict(KNNmodel, testData);
```

the KNN gives 100% right answers on this test.

6.7 Performance

the following code tests Sensitivity(it gives 83.3% for KNN and 100% for SVM) and Specificity(it gives 100% for both classifiers):

```
% leave one out for knn and svm
function [percentileTrue, percentileFalse] = leaveOneOutKnn(features, label)
    knnTrueCorrect = 0;
    knnFalseCorrect = 0;
    trueMax = 0;
    falseMax = 0;

    for i = 1:size(features,1)
        f = features;
        f(i,:) = [];
        l = label;
        l(i) = [];
        knn = fitcknn(f,l);
        [knnp,~] = predict(knn, features(i,:));
        if label(i) == 1
            trueMax = trueMax + 1;
        end
```

```
18
19              if label(i) == 0
20                  falseMax = falseMax + 1;
21              end
22
23              if (label(i)==1) && (knnp == 1)
24                  knnTrueCorrect = knnTrueCorrect + 1;
25              end
26
27              if (label(i)==0) && (knnp == 0)
28                  knnFalseCorrect = knnFalseCorrect + 1;
29              end
30
31
32          end
33          percentileTrue = knnTrueCorrect*100/trueMax;
34          percentileFalse = knnFalseCorrect*100/falseMax;
35      end
36
37      function [percentileTrue, percentileFalse] = leaveOneOutSVM(features, label)
38          svmTrueCorrect = 0;
39          svmFalseCorrect = 0;
40          trueMax = 0;
41          falseMax = 0;
42          svmcorrect = 0;
43          for i = 1:size(features,1)
44              f = features;
45              f(i,:) = [];
46              l = label;
47              l(i) = [];
48              svm = fitcsvm(f,l);
49              [svmp,~] = predict(svm, features(i,:));
50              if label(i) == 1
51                  trueMax = trueMax + 1;
52              end
53
54              if label(i) == 0
55                  falseMax = falseMax + 1;
56              end
57
58              if (label(i)==1) && (svmp == 1)
59                  svmTrueCorrect = svmTrueCorrect + 1;
60              end
61
62              if (label(i)==0) && (svmp == 0)
63                  svmFalseCorrect = svmFalseCorrect + 1;
64              end
65
66
67          end
```

```
68          percentileTrue = svmTrueCorrect*100/trueMax;
69          percentileFalse = svmFalseCorrect*100/falseMax;
70      end
```

7 Deliverables for Phase 2

7.1 PSD plots

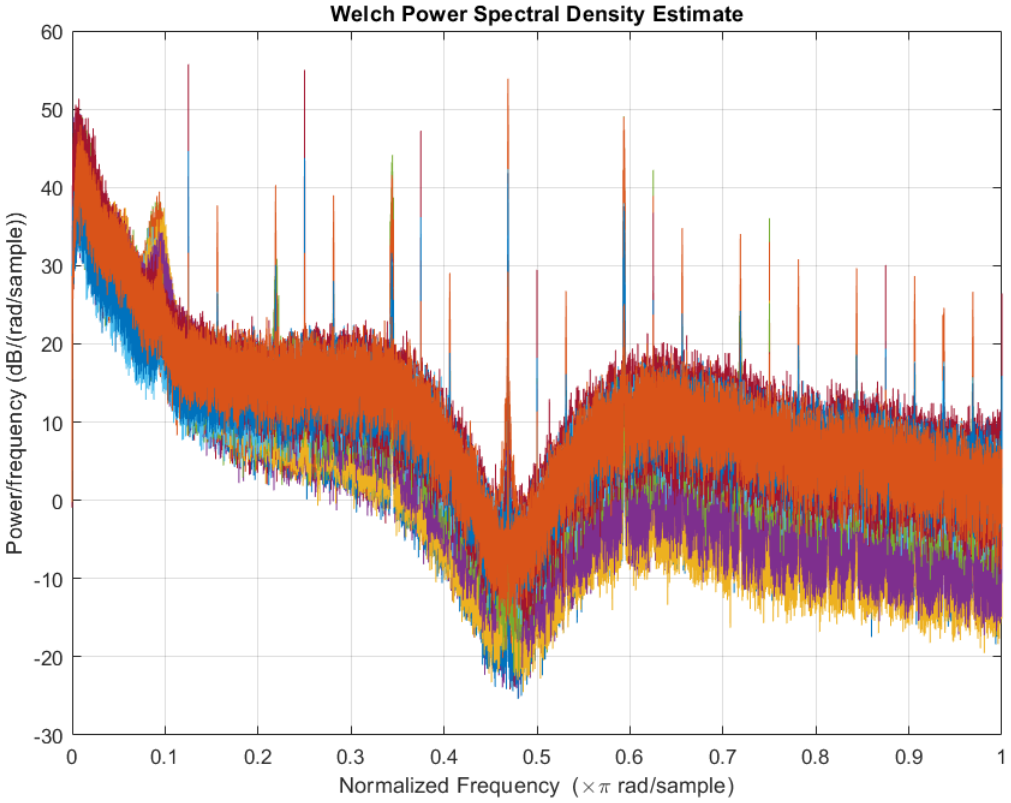


Figure 25: PSD spectrum of a chb01-18

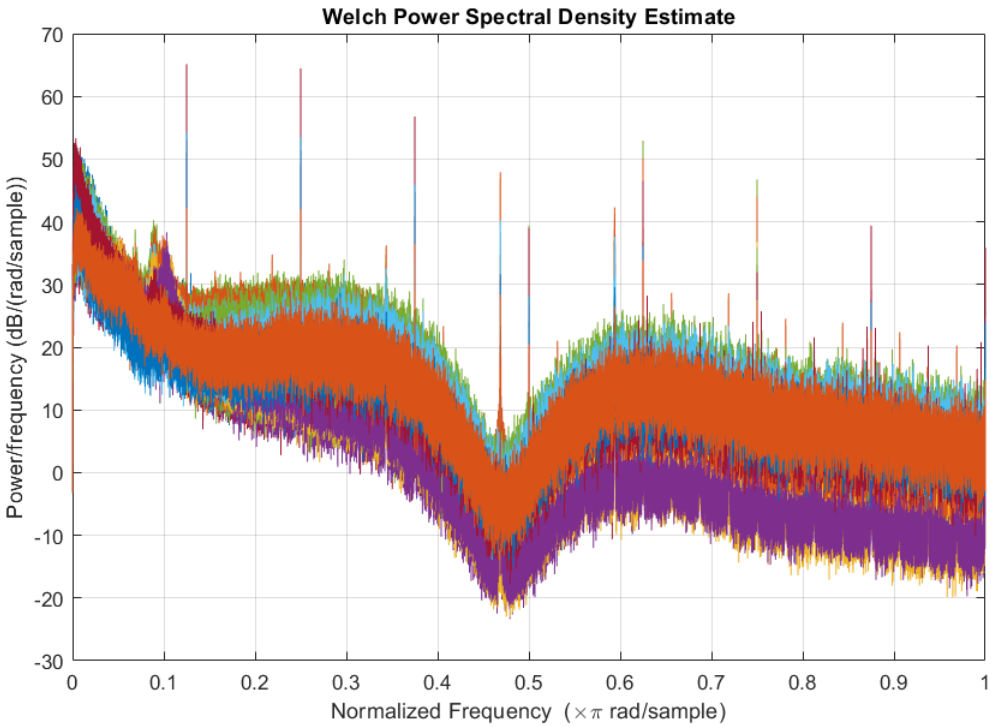


Figure 26: PSD spectrum of a chb01-02

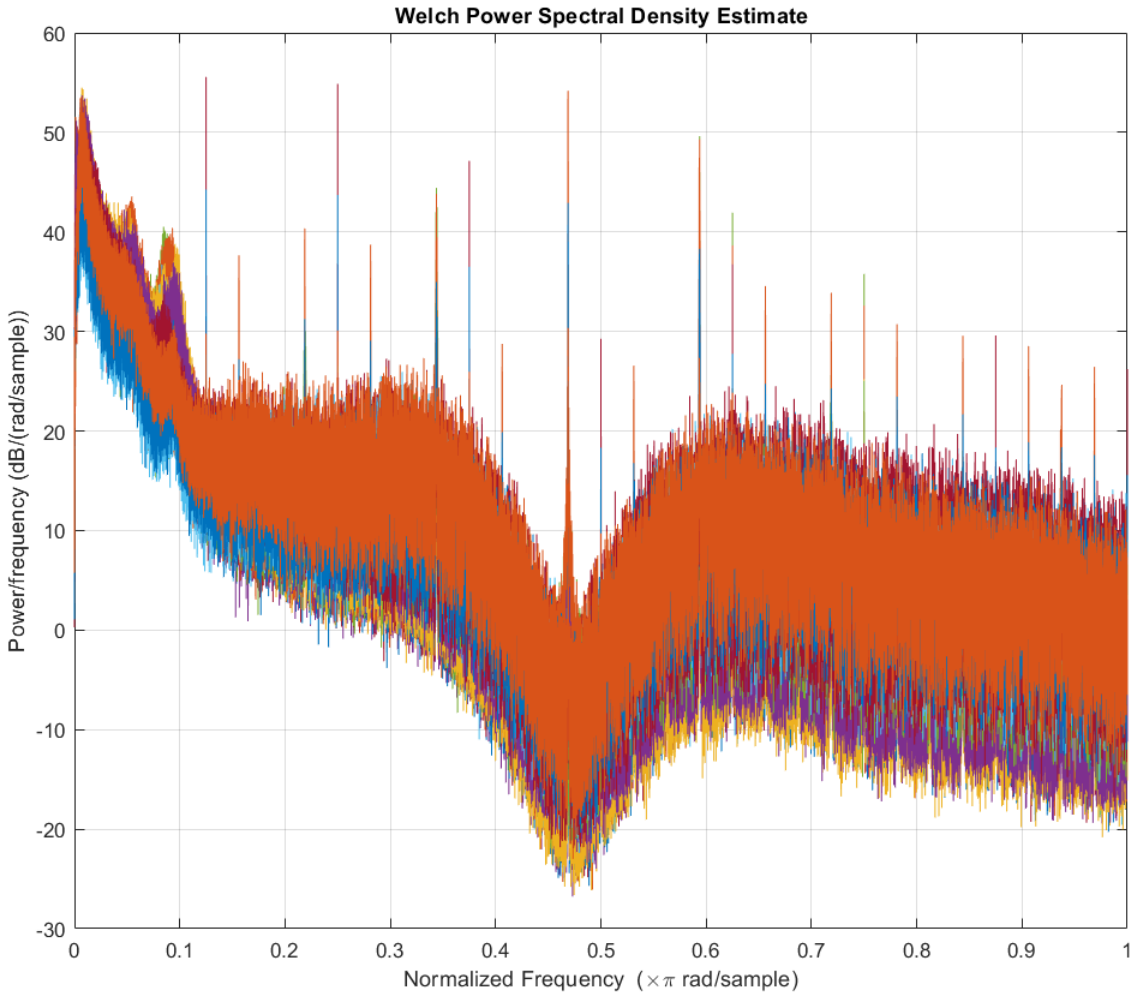


Figure 27: PSD spectrum of a chb01-15

7.2 Shannon Entropy Calculation

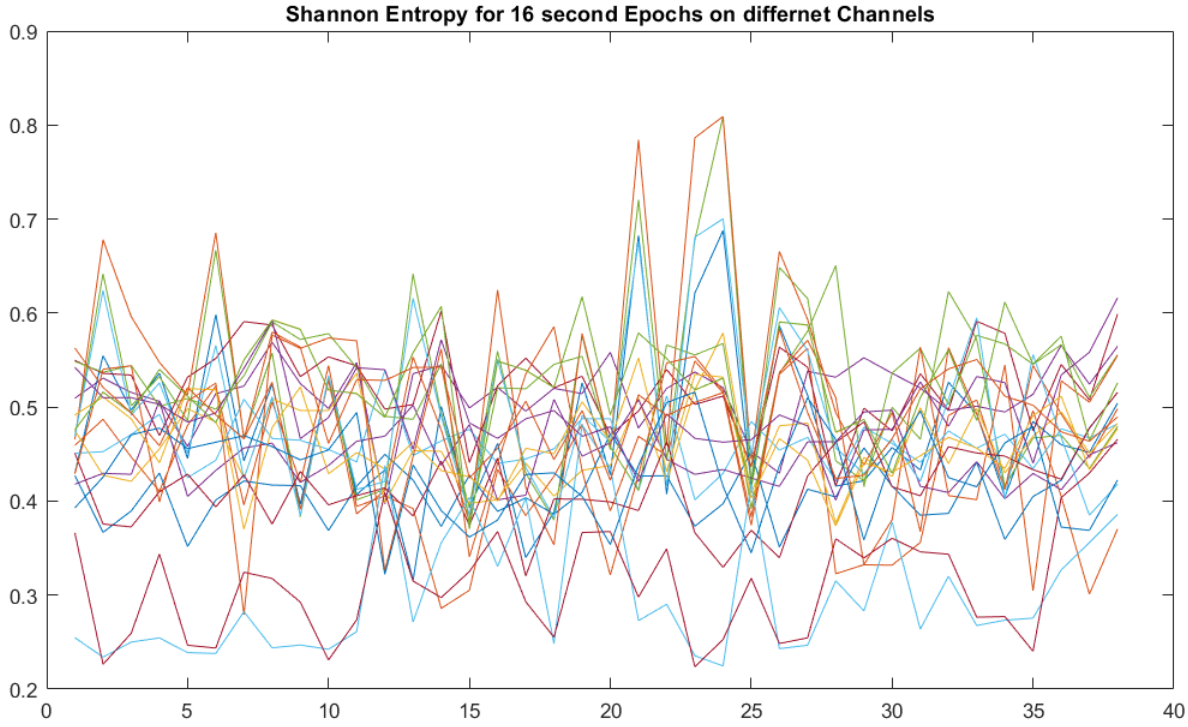


Figure 28: shannon Entropy versus time for 16 second epochs

7.3 Classifier Performances

both classifiers had a Performance of 100% when trained on different test and train data.

7.4 Selected Features

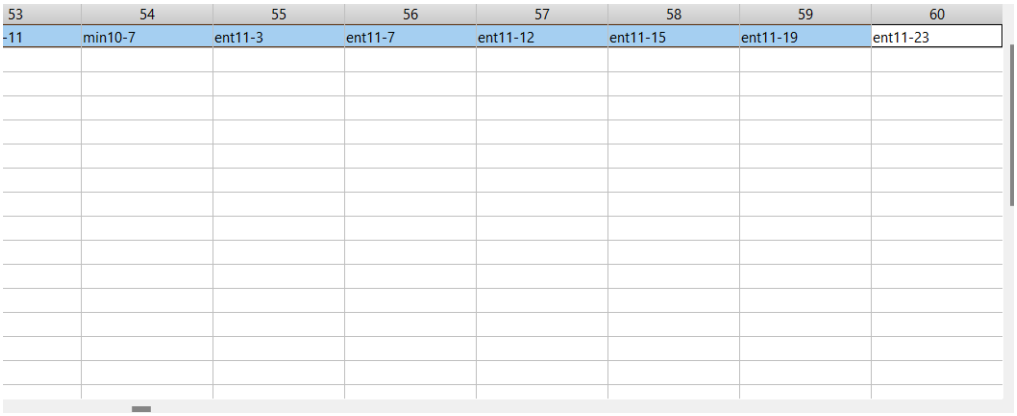
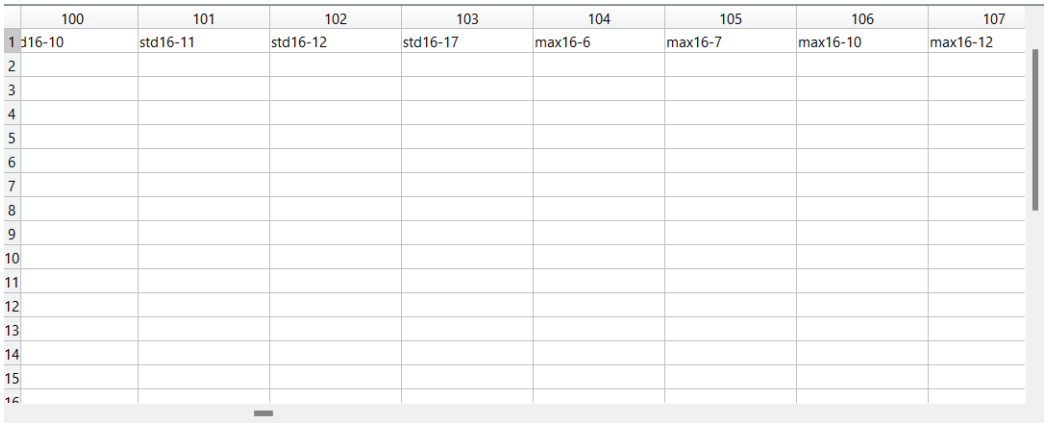
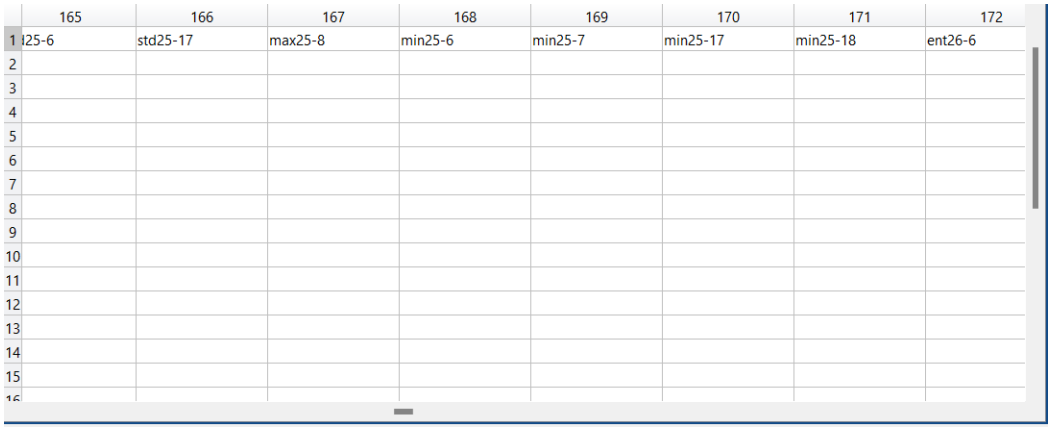


Figure 29: some Features



	100	101	102	103	104	105	106	107
1	d16-10	std16-11	std16-12	std16-17	max16-6	max16-7	max16-10	max16-12
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								

Figure 30: some Features



	165	166	167	168	169	170	171	172
1	d25-6	std25-17	max25-8	min25-6	min25-7	min25-17	min25-18	ent26-6
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								

Figure 31: some Features

our code also extract the name of the Selected features. it had 444 features so we cannot list all of them but we saw that Mean value was generally a less important feature and standard deviation and shannon entropy were the features with the lowest p-value and were the most useful features. all channels were important, but the later epochs of data were generally more important than earlier ones.

7.5 Cross-Validation Results

KNN true positivity was 83.3% and KNN true Negativity was 100%.
SVM true positivity was 100% and SVM true Negativity was 100%.